NSAPI プログラマーズガイド

iPlanet Web Server, Enterprise Edition

Version 6.0

2001 年 5 月

原典 : iPlanet Web Server, Enterprise Edition NSAPI Programmer's Guide

目 次

このマニュアルについて

このマニュアルの最終更新日 : 2001 年 5 月 15 日

このマニュアルでは、NSAPI (Netscape Server Application Programming Interface) を使用してサーバアプリケーション関数 (Server Application Functions: SAF) を定義するプラグインを構築し、iPlanet™ Web Server, Edition バージョン 6.0 を拡張および変更する方法について説明します。さらに、構成ファイル `obj.conf`、`magnus.conf`、`server.xml`、および `mime.types` の用途と使用方法についても説明し、これらの構成ファイルで使うことのできる指令および関数の一覧も掲載しています。また、新しいプラグインの定義に使うことのできる NSAPI 関数のリファレンスも記載しています。

このマニュアルは、次の章および付録から構成されています。

- 第 1 章「サーバの動作の基本」

  この章では、iPlanet Web Server が、初期化タスクの実行やクライアント要求の処理に構成ファイルをどのように使用するかについて説明します。

- 第 2 章「obj.conf の構文と使用法」

  この章では、構成ファイル `obj.conf` について詳しく説明します。また、サーバに要求の処理方法を指示するための、このファイル内での構文および指令の使用方法についても説明します。

- 第 3 章「事前定義済みの SAF および要求処理プロセス」

  この章では、要求処理プロセスの各段階について説明します。さらに、各段階で呼び出すことのできる SAF の API リファレンスも掲載しています。

- 第 4 章「カスタム SAF の作成」

  この章では、新しい SAF を定義するユーザ独自のプラグインを作成して、サーバが要求を処理する方法を変更または拡張する方法について説明します。

- 第 5 章「NSAPI 関数のリファレンス」

  この章では、NSAPI に含まれている関数のリファレンスを掲載しています。SAF を定義するには、NSAPI 関数を使います。

- 第 6 章「カスタム SAF の例」

この章では、要求処理プロセスの各段階で使うカスタム SAF の例について説明します。

- 第 7 章「magnus.conf の構文と使用法」

  この章では、初期化時に iPlanet Web Server を構成するために、構成ファイル `magnus.conf` に設定できる変数について説明します。

- 第 8 章「仮想サーバの構成ファイル」

  この章では、iPlanet Web Server で仮想サーバを構成するために、構成ファイル `server.xml` に設定できる変数について説明します。

- 付録 A「データ構造体のリファレンス」

  この付録では、よく使われる NSAPI データ構造体のいくつかを説明します。

- 付録 B「MIME タイプ」

  この付録では、ファイル拡張子をファイルタイプにマッピングする MIME タイプファイルについて説明します。

- 付録 C「ワイルドカードパターン」

  この付録には、`obj.conf`、各種の事前定義済みの SAF、および一部の NSAPI 関数で値を指定するときに使うことのできるワイルドカードパターンの一覧を掲載しています。

- 付録 D「時刻の書式」

  この付録には、時刻書式の一覧を掲載しています。

- 付録 E「HTTP (HyperText Transfer Protocol)」

  この付録では、HTTP の概要を示します。

- 付録 F「動的に結果をキャッシュする関数」

  この付録では、結果をキャッシュするプラグインを作成する方法について説明します。

- 付録 G「NSAPI 関数とマクロのアルファベット順リスト」
  付録 H「magnus.conf 内の指令のアルファベット順リスト」
  付録 I「事前定義済みの SAF のアルファベット順リスト」

  これらの付録には、NSAPI 関数、事前定義済みの SAF、および `magnus.conf` に含まれる変数が簡単に検索できる、アルファベット順の一覧を掲載しています。

注　このマニュアルでは、特に Linux と明記されていない限り、UNIX 固有の記述はすべて、Linux オペレーティングシステムにも当てはまります。

第 1 章

サーバの動作の基本

iPlanet Web Server の構成と動作は、1 連の構成ファイルによって決定されます。構成ファイルの設定値は、サーバマネージャインタフェースを使って、または手動でファイルを編集して変更できます。

クライアントからの要求を処理する方法をサーバに指示する命令が含まれている構成ファイルは、`obj.conf` と呼ばれます。`obj.conf` への命令の追加や変更により、要求処理プロセスを変更したり拡張したりできます。NSAPI (Netscape Server Application Programming Interface) を使って、`obj.conf` に含める命令で使う新しいサーバアプリケーション関数 (Server Application Function: SAF) を作成できます。

この章では、iPlanet Web Server が使う構成ファイルについて説明します。次に、要求の処理のためのサーバのプロセスの詳細を説明します。最後に、NSAPI を使って、要求処理プロセスを変更する新しい関数を定義する方法を紹介します。

この章は、次の節で構成されています。

- 構成ファイル
- 動的再構成
- サーバによるクライアントからの要求の処理方法
- 新しいサーバアプリケーション関数の作成

構成ファイル

iPlanet Web Server の構成と動作は、構成ファイルにより制御されます。構成ファイルは、ディレクトリ *server-root*/*server-id*/config/ にあります。このディレクトリには、各コンポーネントを制御するためのさまざまな構成ファイルがあります。構成ファイルの正確な個数や名前は、サーバで使用可能になっている、またはサーバに読み込まれているコンポーネントによって決まります。

ただし、このディレクトリにはサーバの動作に必要不可欠な 4 つの構成ファイルが必ず含まれています。それらのファイルを次に示します。

- magnus.conf - グローバルサーバ初期化情報が含まれている
- server.xml - 仮想サーバと待機ソケット用の初期化情報が含まれている
- obj.conf - クライアントからの要求を処理する方法を指示する命令が含まれている
- mime.types - 要求されたリソースの内容のタイプを判別するための情報が含まれている

magnus.conf

このファイルには、初期化時にサーバを構成する変数の値が設定されます。サーバは、起動時にこのファイルを参照し、設定値に従って実行します。サーバは、再起動されるまでこのファイルを再度参照することはありません。

magnus.conf に設定できるすべての変数と Init 指令の一覧は、第 7 章「magnus.conf の構文と使用法」を参照してください。

server.xml

このファイルでは、サーバが待機するアドレスとポートを設定し、それらの待機ソケットに仮想サーバクラスと仮想サーバを割り当てます。マスターファイルの server.dtd は、そのフォーマットと内容を定義します。

サーバが server.dtd および server.xml をどのように使用するかについては、第 8 章「仮想サーバの構成ファイル」を参照してください。

注	仮想サーバは、サーバインスタンスとは異なるものです。各サーバインスタンスは、1 つ以上の仮想サーバからなる完全に独立したサーバです。

obj.conf

このファイルには、ブラウザなどのクライアントからの要求を処理する方法をサーバに指示する命令が含まれています。サーバは、クライアントからの要求を処理するたびに、このファイルに定義されている構成情報を参照します。

仮想サーバクラス、または仮想サーバのグループごとに、`obj.conf` ファイルが 1 つあります。このマニュアルでは、「`obj.conf` ファイル」は、すべての `obj.conf` ファイル、または説明の対象となる仮想サーバクラス用の `obj.conf` ファイルのことを意味します。

すべての `obj.conf` ファイルは、*server_root*`/`*server_id*`/config` ディレクトリにあります。`obj.conf` ファイルには、通常 *vsclass*`.obj.conf` という名前が付けられます。この *vsclass* は、仮想サーバクラスの名前です。

`obj.conf` ファイルは、iPlanet Web Server の動作に必要不可欠です。サーバマネージャインタフェースによりサーバに変更を行なうと、システムは自動的に `obj.conf` を更新します。

`obj.conf` ファイルには、要求 - 応答プロセスの各段階で何をすべきかを iPlanet Web Server に伝える一連の命令 (指令) が含まれています。各指令は、サーバアプリケーション関数 (SAF) を呼び出します。SAF は、Netscape Server Application Programming Interface (NSAPI) を使って作成されています。iPlanet Web Server には事前定義済みの一式の SAF がありますが、NSAPI を使って、サーバが要求を処理する方法を変更する新しい命令を作成してユーザ独自の SAF を作成することもできます。

サーバが `obj.conf` をどのように使うかについては、第 2 章「obj.conf の構文と使用法」を参照してください。

mime.types

このファイルは、サーバが要求されたリソースの内容のタイプを判別できるように、ファイル拡張子を MIME タイプにマッピングします。たとえば、`.html` 拡張子が付いたリソースに対する要求は、クライアントが HTML ファイルを要求していることを示し、`.gif` 拡張子が付いたリソースに対する要求はクライアントが GIF フォーマットのイメージファイルを要求していることを示します。

サーバが `mime.types` をどのように使うかについては、付録 B「MIME タイプ」を参照してください。

動的再構成

サーバを再起動しなくても、obj.conf、mime.types、server.xml、および仮想サーバ固有のアクセス制御リスト (Access Control List, ACL) ファイルへの変更は有効になります。「Apply 」をクリックし、次に「Apply Changes」画面の「Load Configuration Files」ボタンをクリックするだけで、変更が適用されます。新しい構成情報のインストールの際にエラーが発生した場合は、前の構成情報が復元されます。

obj.conf を編集し、変更を適用すると、新しい構成情報が動的に構成可能なファイルからのすべての情報を含むメモリーに読み込まれます。

新たに確立された各接続は、最新の構成情報を参照します。構成情報を参照する最後のセッションが終了すると、使用されなくなった古い構成情報は削除されます。

サーバによるクライアントからの要求の処理方法

iPlanet Web Server は、HTTP (HyperText Transfer Protocol) 要求を受け取って応答する Web サーバです。Netscape Communicator などのブラウザは、HTTP、FTP、gopher などのプロトコルを使用して通信します。iPlanet Web Server は、特に HTTP を使用します。

HTTP プロトコルについては、付録 E「HTTP (HyperText Transfer Protocol)」、または最新の HTTP の仕様を参照してください。

HTTP の基本

要約すると、HTTP/1.1 プロトコルは次のように働きます。

- クライアント (通常はブラウザ) がサーバへの接続を開始し、要求を送信する
- サーバが要求を処理し、応答を生成し、Connection: Close ヘッダーを見つけた場合は、接続を終了する

要求は、GET や POST などのメソッド、要求されたリソースを示す URI (Universal Resource Identifier)、および HTTP プロトコルのバージョンを示す、空白文字で区切られた 1 つの行から構成されます。

要求には、通常、いくつかのヘッダー、ヘッダーの終わりを示す空行が続き、また時には本体のデータが続きます。ヘッダーは、要求またはクライアントのデータ本体についてのさまざまな情報を提供することができます。ヘッダーは、通常、POST メソッドと PUT メソッドの場合にだけ送信されます。

以下に示す要求の例は、Netscape ブラウザがサーバ foo.com に /index.html のリソースを返送するように要求するために送信します。この例では、メソッドが GET (要求の目的はデータを取得することで、送信することではない) であるため、データ本体は送信されません。

```
GET /index.html HTTP/1.0
User-agent: Mozilla
Accept: text/html, text/plain, image/jpeg, image/gif, */*
Host: foo.com
```

サーバは、要求を受け取り、処理します。サーバは、多数の要求を同時に処理することができますが、各要求は個別に取り扱います。各要求は、要求処理プロセスを構成する一連のステップに分割されます。

サーバは、HTTP プロトコルのバージョン、HTTP 状態コード、および原因を示す文字列から構成される、空白文字で区切られた応答を生成します。通常、この後にいくつかのヘッダーが続きます。ヘッダーの終わりは、空行で示されます。次に、応答のデータ本体が続きます。典型的な HTTP の応答は、次のようになります。

```
HTTP/1.0 200 OK
Server: Netscape-Enterprise/6.0
Content-type: text/html
Content-length: 83

<HTML>
<HEAD><TITLE>Hello World</Title></HEAD>
<BODY>Hello World</BODY>
</HTML>
```

状態コードと原因を示す文字列は、サーバがどのように要求を処理したかをクライアントに伝えます。通常、状態コードの 200 が返されます。このコードは、要求が正常に処理され、データ本体に要求されたものが含まれていることを示します。その他の結果コードは、別のサーバやブラウザのキャッシュへのリダイレクト、または「404 Not Found」などの各種の HTTP エラーを示します。

要求処理プロセスのステップ

サーバは、起動されると初期化を実行してから、ブラウザなどのクライアントからの HTTP 要求を待ちます。要求を受け取ると、まず仮想サーバを選択します。仮想サーバの決定方法の詳細は、298 ページの「要求処理のための仮想サーバの選択」を参照してください。

仮想サーバが選択されると、その仮想サーバクラス用の `obj.conf` ファイルが、以下のステップでどのように要求を処理すべきかを示します。

1. AuthTrans (承認変換)

   要求で送られてきた、(名前やパスワードなどの) 承認情報を検証する

2. NameTrans (名前変換)

   論理 URI をローカルファイルシステムのパスに変換する

3. PathCheck (パスの確認)

   ローカルファイルシステムのパスが有効であるかどうかを確認してから、ファイルシステム上の要求されたリソースに対するアクセス権限が要求元にあるかどうかを確認する

4. ObjectType (オブジェクトのタイプの判別)

   要求されたリソースの MIME (Multi-purpose Internet Mail Encoding) タイプ (たとえば、`text/html`、`image/gif` など) を判別する

5. Service (応答の生成)

   応答を生成し、クライアントに返す

6. AddLog (ログエントリの追加)

   ログファイルにエントリを追加する

7. Error (サービス)

   このステップは、前のステップでエラーが発生した場合にだけ実行される。エラーが発生した場合は、サーバはエラーメッセージを記録し、プロセスを中止する

要求の処理のための指令

`obj.conf` ファイルには、指令という一連の命令が含まれています。それらの指令は、要求処理プロセスの各段階で実行すべきことを iPlanet Web Server に指示します。各指令は、1 つ以上の引数を指定して SAF を呼び出します。各指令は、要求処理プロセスの特定の段階に適用されます。要求処理プロセスの段階には、`AuthTrans`、`NameTrans`、`PathCheck`、`ObjectType`、`Service`、および `AddLog` があります。

たとえば、NameTrans 段階では次の指令が適用されます。この指令は、root 引数に D:/Netscape/Server4/docs を設定して、document-root 関数を呼び出します。(document-root 関数は、URL の http://*server_name*/ の部分をドキュメントルートに変換します。この例ではドキュメントルートは、D:/Netscape/Server4/docs になっています。)

```
NameTrans fn="document-root" root="D:/Netscape/Server4/docs"
```

obj.conf に含まれる指令によって呼び出される関数は、サーバアプリケーション関数 (Server Application Functions: SAF) と呼ばれます。

新しいサーバアプリケーション関数の作成

iPlanet Web Server には、obj.conf にさらに指令を作成するために使うことのできる、事前定義済みのさまざまな SAF があります。NSAPI が提供する関数を使ってユーザ独自の SAF を作成することもできます。SAF を作成したら、サーバが新しい関数を適切なときに呼び出すように、指令を obj.conf に追加します。

各 SAF にはそれぞれの引数があり、obj.conf に含まれる指令により各 SAF に渡されます。各 SAF には、追加の引数も渡されます。追加の引数には、要求についての情報 (要求されたリソースが何であるかや、どのようなクライアントがそのリソースを要求したかなど) と、前に呼び出された指令によって呼び出された SAF によって作成または変更されたその他のサーバ変数が含まれます。各 SAF は、サーバ変数を検査、変更、または作成できます。

各 SAF は、処理が成功したか、何も行なわれなかったか、あるいは失敗したかをサーバに通知する結果コードを返します。

obj.conf についての詳細は、第 2 章「obj.conf の構文と使用法」を参照してください。

事前定義済みの SAF についての詳細は、第 3 章「事前定義済みの SAF および要求処理プロセス」を参照してください。

ユーザ独自の SAF の作成方法については、第 4 章「カスタム SAF の作成」を参照してください。

第 2 章

obj.conf の構文と使用法

obj.conf 構成ファイルには、クライアントからの要求を処理する方法を iPlanet Web Server に指示する指令が含まれています。この章では、obj.conf に含まれるサーバの命令、Object タグの使用法、変数の使用法、obj.conf 内の制御のフロー、obj.conf の編集上の構文規則、および指令の例について説明します。

この章は、次の節で構成されています。

- obj.conf に含まれるサーバの命令
- Object タグ
- server.xml に定義された変数
- obj.conf 内の制御のフロー
- obj.conf の編集上の構文規則
- obj.conf 指令の例について

obj.conf に含まれるサーバの命令

obj.conf ファイルには、ブラウザなどのクライアントから受け取った要求を処理する方法をサーバに指示する指令が含まれています。指令は、OBJECT タグ内に記述されます。

各指令は、関数の呼び出しの時期とそのための引数を指定して、関数を呼び出します。

各指令の構文を次に示します。

```
Directive fn=func-name name1="value1"...nameN="valueN"
```

次に例を示します。

```
NameTrans fn="document-root" root="D:/Netscape/Server4/docs"
```

Directive は、要求処理プロセスのどの時点でこの命令を実行するかを示します。値は、次のいずれかになります。AuthTrans、NameTrans、PathCheck、ObjectType、Service、Error、AddLog。

fn 引数の値には、実行対象のサーバアプリケーション関数 (Server Application Function: SAF) の名前を指定します。どの指令にも、fn パラメータの値を 1 つ指定する必要があります。関数がない場合は、命令は何も実行しません。

残りのパラメータは、関数に必要とされる引数であり、これは関数によって異なります。

iPlanet Web Server には、第 3 章「事前定義済みの SAF および要求処理プロセス」に説明されているように、obj.conf で指令の作成や変更を行うために使うことのできる、1 式の組み込みサーバアプリケーション関数 (SAF) があります。第 4 章「カスタム SAF の作成」に説明されているように、新しい SAF を定義することもできます。

magnus.conf ファイルには、サーバを初期化する Init 指令の SAF が含まれています。詳細は、第 7 章「magnus.conf の構文と使用法」を参照してください。

指令の要約

以下にサーバの指令のカテゴリを示し、各指令の用途について説明します。各カテゴリは、要求処理プロセスの特定の段階に対応します。32 ページの「obj.conf 内の制御のフロー」の節では、各段階で実行すべき指令をサーバがどのように判断するかを具体的に説明します。

- AuthTrans

  HTTP 要求に指定された承認情報 (通常、Authorization ヘッダーで送られる) を検証し、その情報をユーザまたはグループ、あるいはその両方に変換します。サーバのアクセス制御は、次の 2 つの段階からなります。AuthTrans は、ユーザの正当性を検証します。その後、PathCheck が、要求されたリソースに対するユーザのアクセス特権をテストします。

  ```
  AuthTrans fn=basic-auth userfn=ntauth auth-type=basic
  userdb=none
  ```

  この例では、basic-auth 関数を呼び出し、この関数はクライアントが送信してきた承認情報を検証するためにカスタム関数 (この場合 ntauth) を呼び出します。Authorization ヘッダーは、基本サーバ承認スキーマの一部として送られます。

- NameTrans

  要求に指定された URL を論理 URL から、要求されたリソースの物理ファイルシステムのパスに変換します。これは、別のサイトへのリダイレクトになることもあります。次に例を示します。

  ```
  NameTrans fn="document-root" root="D:/Netscape/Server4/docs"
  ```

この例では、root 引数に D:/Netscape/Server4/docs を設定して、document-root 関数を呼び出します。document-root 関数は、要求されたリソースの URL の http://*server_name*/ の部分をドキュメントルートに変換します。この場合のドキュメントルートは、D:/Netscape/Server4/docs です。したがって、http://*server-name*/doc1.html に対する要求は、D:/Netscape/Server4/docs/doc1.html に変換されます。

- PathCheck

  NameTrans ステップで得られた物理パスをテストします。一般に、このテストは、パスが有効であるかどうかと、要求したリソースにクライアントがアクセスを許可されているかどうかを判定します。次に例を示します。

  ```
  PathCheck fn="find-index" index-names="index.html,home.html"
  ```

  この例では、index-names 引数に index.html,home.html を設定して、find-index 関数を呼び出します。要求された URL がディレクトリである場合は、この関数は、要求されたディレクトリで index.html または home.html というファイルを検索するようにサーバに指示します。

- ObjectType

  要求されたリソースの MIME (Multi-purpose Internet Mail Encoding) タイプを判別します。MIME タイプには、属性として type (内容のタイプを示す)、encoding、および language があります。MIME タイプは、クライアントへの応答のヘッダーで送られます。MIME タイプは、サーバがどの Service 指令を実行すべきかを判断する際にも役立ちます。

  MIME タイプには、次のものがあります。

  - 一般的なドキュメントタイプの text/html や image/gif など (たとえば、ファイル名の拡張子の .gif は、MIME タイプの image/gif に変換される)
  - 内部サーバタイプ。内部タイプは、必ず magnus-internal で始まる

  次に例を示します。

  ```
  ObjectType fn="type-by-extension"
  ```

  この例では、type-by-extension 関数を呼び出します。この関数により、サーバは、要求されたリソースのファイル拡張子に基づいて MIME タイプを判別します。

- Service

  応答を生成し、クライアントに送信します。これには、HTTP 結果ステータスの設定、応答ヘッダー (内容のタイプや内容の長さなど) の設定、および応答データの生成と送信が含まれます。デフォルトの応答は、send-file 関数を呼び出して、要求されたファイルの内容を適切なヘッダーファイルと共にクライアントに送信します。

  デフォルトの Service 指令を次に示します。

  ```
  Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
  fn="send-file"
  ```

この指令は、メソッドが GET、HEAD、または POST であり、タイプ が magnus-internal/ で始まらない要求に対する応答として send-file 関数を呼び出すようにサーバに指示します。(特殊文字の *~ は、「一致しない」を示しています。)

別の例を次に示します。

```
Service method="(GET|HEAD)" type="magnus-internal/imagemap"
fn="imagemap"
```

この場合、要求のメソッドが GET または HEAD のどちらかであり、要求されたリソースのタイプが "magnus-internal/imagemap" である場合は、imagemap 関数が呼び出されます。

- AddLog

  トランザクションについての情報を記録するためにログファイルにエントリを追加します。次に例を示します。

  ```
  AddLog fn="flex-log" name="access"
  ```

  この例では、flex-log 関数を呼び出して、access という名前のログファイルに現在の要求についての情報を記録します。

- Error

  HTTP エラーに対処します。この指令は、前の指令がエラーになると、呼び出されます。通常、サーバは、問題点と考えられる解決策を示すカスタム HTML ドキュメントをユーザに送信することによりエラーに対処します。

  次に例を示します。

  ```
  Error fn="send-error" reason="Unauthorized"
  path="D:/netscape/server4/errors/unauthorized.html"
  ```

  この例では、サーバは、クライアントがアクセスを許可されていないリソースを要求するたびに、D:/netscape/server4/errors/unauthorized.html のファイルを送信します。

Object タグ

obj.conf ファイル内の指令は、<Object> タグで始まり、</Object> タグで終わるオブジェクトのグループに分けられます。デフォルトのオブジェクトは、要求を処理するデフォルトの方法をサーバに指示します。新しい各オブジェクトは、デフォルトのオブジェクトの動作を変更します。

Object タグには、name 属性または ppath 属性を指定できます。どちらのパラメータも、ワイルドカードパターンで指定できます。次に例を示します。

```
<Object name="cgi">
```

または

```
<Object ppath="/usr/netscape/server4/docs/private/*">
```

サーバは、常にデフォルトのオブジェクトの指令を処理することにより要求の処理を開始します。ただし、サーバは、次の条件のどちらかが真の場合、デフォルトのオブジェクトの NameTrans 段階の後に、別のオブジェクトの指令の処理に切り替えます。

- 成功した NameTrans 指令が、name 引数を指定する
- NameTrans 段階で得られた物理パス名が、別のオブジェクトの ppath 属性に一致する

デフォルトのオブジェクト以外のオブジェクトを使うようにサーバが指示されると、サーバはデフォルトのオブジェクトの指令の処理の前に、別のオブジェクトの指令を処理します。要求処理プロセスの一部のステップでは、サーバは、特定の段階の 1 つの指令の実行に成功するとただちにその特定の段階 (Service 段階など) の指令の処理を停止しますが、その他の段階ではサーバはその段階のすべての指令 (デフォルトのオブジェクト内および追加のオブジェクト内の指令を含む) を処理します。詳細は、32 ページの「obj.conf 内の制御のフロー」を参照してください。

name 属性を使うオブジェクト

デフォルトのオブジェクトの NameTrans 指令が name 引数を指定している場合は、サーバはデフォルトのオブジェクトの残りの指令を処理する前に、その名前のオブジェクトの指令を処理します。

たとえば、デフォルトのオブジェクトに含まれる次の NameTrans 指令は、名前 cgi を URL が http://*server_name*/cgi/ で始まる要求に割り当てます。

```
<Object name="default">
NameTrans fn="pfx2dir" from="/cgi"
dir="D:/netscape/server4/docs/mycgi" name="cgi"
...
</Object>
```

この NameTrans 指令が実行されると、サーバは cgi という名前のオブジェクトの指令の処理を開始します。

```
<Object name="cgi">
more directives...
</Object>
```

ppath 属性を使うオブジェクト

サーバがデフォルトのオブジェクトの NameTrans 指令の処理を終了したときは、すでに要求の論理 URL は物理パス名に変換されています。この物理パス名が、obj.conf 内の別のオブジェクトの ppath 属性に一致する場合は、サーバはデフォルトのオブジェクトの残りの指令を処理する前に、そのオブジェクトの指令の処理に切り替えます。

たとえば、次の NameTrans 指令は、要求された URL の http://*server_name*/ の部分を D:/Netscape/Server4/docs/ (ドキュメントルートのディレクトリ) に変換します。

```
<Object name="default">
NameTrans fn="document-root" root="D:/Netscape/Server4/docs"
...
</Object>
```

URL の http://*server_name*/internalplan1.html は、D:/Netscape/Server4/docs/internalplan1.html に変換されます。ただし、obj.conf に次の追加のオブジェクトが含まれているとします。

```
<Object ppath="*internal*">
more directives...
</Object>
```

この場合、部分的なパスの *internal* は、パス D:/Netscape/Server4/docs/internalplan1.html に一致します。したがって、サーバは、デフォルトのオブジェクトの残りの指令を処理する前に、このオブジェクトの指令の処理を開始します。

server.xml に定義された変数

server.xml ファイルで変数を定義し、それらの変数を obj.conf ファイルで参照できます。たとえば、次の server.xml コードでは、次のように docroot という変数を定義し、使用しています。

```
<!DOCTYPE SERVER SYSTEM "server.dtd" [
<!ATTLIST VARS
    docroot CDATA #IMPLIED
>
]>
...
       <VS id="a.com" connections="maingroup" urlhosts="a.com"
             mime="mime1" aclids="std">
          <VARS docroot="/u/server6/a/docs" />
       </VS>
...
```

次のようにして、この変数を obj.conf で参照できます。

```
NameTrans fn=document-root root="$docroot"
```

この docroot 変数を使用すると、obj.conf ファイルで仮想サーバクラス用のドキュメントルートを定義する手間が省けます。また、同じ仮想サーバクラス内で仮想サーバごとに異なるドキュメントルートを定義可能にします。

注	変数の代入は、obj.conf ファイルでのみ許可されています。その他の iPlanet Web Server 構成ファイルでは許可されていません。 obj.conf ファイルで参照される変数は、SERVER、VSCLASS、または VS レベルで server.xml ファイルに定義しておく必要があります。SERVER または VSCLASS レベルで変数をデフォルト値で定義し、それらの値を VS でオーバーライドすることをお勧めします。

詳細は、第 8 章「仮想サーバの構成ファイル」を参照してください。

obj.conf 内の制御のフロー

サーバは、要求を正しい仮想サーバに割り当ててからでなければ、要求を処理できません。仮想サーバの決定方法の詳細は、298 ページの「要求処理のための仮想サーバの選択」を参照してください。

仮想サーバが決定された後、サーバは、仮想サーバが属する仮想サーバクラス用の obj.conf ファイルを実行します。この節では、サーバが obj.conf に含まれているどの指令を実行すべきかを判断する方法について説明します。

AuthTrans

サーバが要求を受け取ると、デフォルトのオブジェクトの AuthTrans 指令を実行して、サーバへのアクセスをクライアントが許可されているかどうかを確認します。

複数の AuthTrans 指令がある場合は、サーバはそれらをすべて (いずれかがエラーにならないかぎり) 実行します。エラーが発生した場合は、サーバは Error 指令以外のその他のすべての指令の実行を省略します。

NameTrans

次に、サーバはデフォルトのオブジェクトの NameTrans 指令を実行して、要求されたリソースの論理 URL をサーバのファイルシステムの物理パス名にマッピングします。サーバは、適用できるものが見つかるまで、デフォルトのオブジェクトの各 NameTrans 指令を 1 つずつ確認していきます。

デフォルトのオブジェクトに複数の NameTrans 指令が含まれている場合は、サーバは、1 つが成功するまで各指令を確認します。

デフォルトのオブジェクトの NameTrans セクションには、document-root 関数を呼び出す指令が 1 つだけ含まれている必要があります。この関数は、要求された URL の http://*server_name*/ の部分を、サーバのドキュメントルートとして指定された物理ディレクトリに変換します。次に例を示します。

```
NameTrans fn="document-root" root="D:/Netscape/Server4/docs"
```

その他の NameTrans 指令が適用されない場合にだけ実行されるように、document-root を呼び出す指令は、NameTrans セクションの最後の指令にする必要があります。

pfx2dir (ディレクトリの接頭辞) 関数は、URL とディレクトリ間の追加のマッピングを設定するために使います。たとえば、次の指令は、URL http://*server_name*/cgi/ をディレクトリのパス名 D:/netscape/server4/docs/mycgi/ に変換します。

```
NameTrans fn="pfx2dir" from="/cgi"
dir="D:/netscape/server4/docs/mycgi"
```

この指令が document-root を呼び出す指令の後にくる場合は、この指令は実行されず、その結果のディレクトリのパス名は D:/netscape/server4/docs/cgi/ (mycgi ではなく) になります。これは、document-root を呼び出す指令を、NameTrans セクションの最後の指令にする必要がある理由を示しています。

その他のオブジェクトの処理についてサーバが判断する方法

NameTrans 指令を実行した結果、サーバが別のオブジェクトの指令の処理を開始することがあります。実行に成功した NameTrans 指令が別のオブジェクトの name 属性に一致する名前を指定しているか、別オブジェクトの ppath 属性に一致する部分的なパスを生成する場合に、そのようになります。

成功した NameTrans 指令が、name 引数を指定することにより名前を割り当てると、サーバは、残りの要求処理プロセスのデフォルトのオブジェクトの指令の処理の前に、指定されたオブジェクト (OBJECT タグで定義された) の指令の処理を開始します。

たとえば、デフォルトのオブジェクトに含まれる次の NameTrans 指令は、cgi という名前を URL が http://*server_name*/cgi/ で始まる要求に割り当てます。

```
<Object name="default">
...
NameTrans fn="pfx2dir" from="/cgi"
dir="D:/netscape/server4/docs/mycgi" name="cgi"
...
</Object>
```

この NameTrans 指令が実行されると、サーバは cgi という名前のオブジェクトの指令の処理を開始します。

```
<Object name="cgi">
more directives...
</Object>
```

NameTrans 指令の実行に成功すると、要求されたリソースに関連付けられた物理パス名が生成されます。生成されたパス名が別のオブジェクトの ppath (部分的なパス) 属性に一致する場合は、サーバは、残りの要求処理プロセスのデフォルトのオブジェクトの指令の処理の前に、他のオブジェクトの指令の処理を開始します。

たとえば、obj.conf に次のようなオブジェクトが含まれているとします。

```
<Object ppath="*internal*">
more directives...
</Object>
```

次に、成功した NameTrans 指令が、要求された URL をパス名 D:/Netscape/Server4/docs/internalplan1.html に変換するとします。この場合、部分的なパスの *internal* は、パス D:/Netscape/Server4/docs/internalplan1.html に一致します。したがって、これでサーバは、デフォルトのオブジェクトの残りの指令の処理の前に、このオブジェクトの指令の処理を開始します。

PathCheck

NameTrans ステップで要求されたリソースの論理 URL を物理パス名に変換した後、サーバは PathCheck 指令を実行して、要求したリソースにクライアントがアクセスを許可されているかどうかを検証します。

複数の PathCheck 指令がある場合は、サーバはいずれかの指令がアクセスを拒否しないかぎり、すべての指令を記述されている順に実行します。アクセスが拒否された場合は、サーバは Error セクションの指令の実行に切り替えます。

NameTrans 指令が、別のオブジェクトの name または ppath 属性に一致する名前を割り当てたか、あるいは物理パス名を生成した場合は、サーバはデフォルトのオブジェクトの指令を適用する前に、まず一致するオブジェクトの PathCheck 指令を適用します。

ObjectType

すべての PathCheck 指令がアクセスを許可すると仮定すると、サーバは次に ObjectType 指令を実行して、要求の MIME タイプを判別します。MIME タイプには、type、encoding、および language の 3 つの属性があります。サーバがクライアントに応答を送信するときに、type、language、および encoding の値が応答のヘッダーに入れられて送信されます。また、多くの場合、type は、クライアントへの応答を生成するためにどの Service 指令を実行すべきかをサーバが判断する際に役立ちます。

複数の ObjectType 指令がある場合は、サーバはすべての指令を記述されている順に適用します。ただし、指令が MIME タイプの属性を設定すると、それ以降に同じ属性の設定を試みても無視されます。すべての ObjectType 指令が適用されるのは、1 つの指令がたとえば type などの 1 つの属性を設定し、別の指令が language などの別の属性を設定することがあるためです。

PathCheck 指令と同様、NameTrans ステップの結果別のオブジェクトが要求に一致する場合は、サーバは、デフォルトのオブジェクトの ObjectType 指令の実行の前に、一致するオブジェクトの ObjectType 指令を実行します。

ファイル拡張子によるタイプの設定

通常デフォルトでは、サーバは、MIME タイプを判別するために type-by-extension 関数を呼び出します。この関数は、要求されたリソースのファイル拡張子に従って MIME タイプテーブルで MIME タイプを調べるようにサーバに指示します。このテーブルは、MIME タイプのファイル (通常、mime.types と呼ばれる) によって仮想サーバの初期化時に作成されています。

たとえば、拡張子 .html と .htm に対する MIME タイプテーブルのエントリは、通常次のようになっています。

```
type=text/html  exts=htm,html
```

これは、拡張子が .htm または html のファイルはすべて、HTML としてフォーマットされたテキストファイルであり、type が text/html であることを示しています。

MIME タイプファイルを変更する場合は、変更を有効にするためにサーバを再構成する必要があります。

MIME タイプについては、付録 B「MIME タイプ」を参照してください。

タイプの強制

前のどの ObjectType 指令もタイプを設定していない場合で、サーバが一致するファイル拡張子を MIME タイプテーブルで見つけることができない場合は、type-by-expression の実行後も type には値がありません。通常、サーバがファイル拡張子を認識できない場合は、タイプを text/plain に強制するのも一案です。そうすると、リソースの内容がプレーンテキストとして取り扱われます。また、指定された CGI ディレクトリ内のすべてのリソースに MIME タイプの magnus-internal/cgi を強制するなど、ファイル拡張子にかかわらずタイプを設定する必要がある場合もあります。

タイプを強制する関数は、force-type です。

たとえば、次の指令は、まず MIME タイプを MIME タイプテーブルで調べるようにサーバに指示し、次に type 属性が設定されていない (つまり、ファイル拡張子が MIME タイプテーブルに見つからなかった) 場合、type 属性を text/plain に設定します。

```
ObjectType fn="type-by-extension"
ObjectType fn="force-type" type="text/plain"
```

サーバがファイル abc.dogs に対する要求を受け取り、MIME タイプテーブルを調べたが、拡張子 .dogs に対応するものが見つからない場合は、type 属性は設定されません。type 属性がまだ設定されていないので、2 番目の指令は成功し、type 属性を強制的に text/plain にします。

次の例は、force-type の別の用途を示しています。この例では、サーバが MIME タイプテーブルを調べる前に、type が強制的に magnus-internal/cgi になります。この場合、http://*server_name*/cgi/ 内のリソースに対するすべての要求は、D:/netscape/server4/docs/mycgi/ ディレクトリ内のリソースに対する要求に変換されます。名前が要求に割り当てられるので、サーバは デフォルトのオブジェクトの指令の処理の前に、cgi という名前のオブジェクトの ObjectType 指令を処理します。このオブジェクトには、1 つの ObjectType 指令があり、この指令が強制的に type を magnus-internal/cgi にします。

```
NameTrans fn="pfx2dir" from="/cgi"
dir="D:/netscape/server4/docs/mycgi" name="cgi"
<Object name="cgi">
ObjectType fn="force-type" type="magnus-internal/cgi"
Service fn="send-cgi"
</Object>
```

サーバは引き続き、デフォルトのオブジェクトの指令を含むすべての ObjectType 指令を処理しますが、type 属性はすでに設定されているので、ほかの指令がこの属性を別の値に設定することはできません。

Service

次に、サーバは、クライアントへ送信する応答を生成するために、Service 指令を実行する必要があります。サーバは、type、method、および query string が一致する最初の指令を見つけるために、1 つずつ各 Service 指令を調べます。Service 指令が type、method、または query string を指定していない場合は、指定されていない属性は何にでも一致します。

複数の Service 指令がある場合は、サーバは要求の条件に最初に一致する指令を適用し、残りの Service 指令はすべて無視します。

PathCheck や ObjectType 指令と同様、NameTrans ステップの結果、別のオブジェクトが要求に一致する場合、サーバは、一致するオブジェクトの Service 指令に対処します。サーバは、一致するオブジェクトの Service 指令の実行に成功すると、デフォルトのオブジェクトの Service 指令は実行しません。これは、サーバが Service 指令を 1 つだけしか実行しないためです。

Service の例

Service 指令の働きを示す例として、サーバが URL D:/*server_name*/jos.html に対する要求を受け取ったらどうなるかを考えてみます。この場合、サーバが実行する指令はすべて、デフォルトのオブジェクトに含まれています。

- 次の NameTrans 指令は、要求された URL を D:/netscape/server4/docs/jos.html に変換する

  ```
  NameTrans fn="document-root" root="D:/Netscape/Server4/docs"
  ```

- PathCheck 指令がすべて成功すると仮定する
- 次の ObjectType 指令は、リソースの MIME タイプを MIME タイプテーブルで調べるようにサーバに指示する

  ```
  ObjectType fn="type-by-extension"
  ```

- サーバは、MIME タイプテーブルで次のエントリを見つける。このエントリは、type 属性を text/html に設定する

  ```
  type=text/html exts=htm,html
  ```

- サーバは、次の Service 指令を呼び出す。type パラメータの値は magnus-internal/ で始まらないものに一致する。(ワイルドカードパターンの一覧は、付録 C「ワイルドカードパターン」を参照。) この指令は、要求されたファイル jos.html をクライアントへ送信する

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file"
```

別のオブジェクトを使う例として、サーバが http://*server_name*/servlet/doCalculation.class に対する要求を受け取ったらどうなるかを考えてみます。この例では、サーブレットがアクティブになっており、ディレクトリ D://netscape/server4/docs/servlet/ がサーブレットディレクトリとして登録されている (つまり、サーバがそのディレクトリ内のすべてのファイルをサーブレットとして取り扱う) と仮定します。

- 次の NameTrans 指令は、要求された URL を D:netscape/Server4/docs/servlet/doCalculation.class に変換する。また、この指令は、名前 ServletByExt を要求に割り当てる

```
NameTrans fn="pfx2dir" from="/servlet"
dir="D:/Netscape/Server4/docs/servlet" name="ServletByExt"
```

- name の割り当ての結果、サーバは ServletByExt という名前のオブジェクトの指令の処理に切り替える。このオブジェクトは、次のように定義されている

```
<Object name="ServletByExt">
ObjectType fn="force-type" type="magnus-internal/servlet"
Service type="magnus-internal/servlet" fn="NSServletService"
</Object>
```

- ServletByExt オブジェクトには PathCheck 指令がないので、サーバはデフォルトのオブジェクトの PathCheck 指令を処理する。すべての PathCheck 指令が成功すると仮定する

- 次に、サーバは、ServletByExt オブジェクトに含まれるものから、ObjectType 指令の処理を開始する。この指令は、type 属性を magnus-internal/servlet に設定する

  ```
  ObjectType fn="force-type" type="magnus-internal/servlet"
  ```

  サーバは、デフォルトのオブジェクトのすべての ObjectType 指令の処理を続けるが、type 属性はすでに設定されているので、この値を変更することはできない

- Service 指令を処理するときには、サーバは ServletByExt オブジェクトの次の Service 指令から処理を開始する

  ```
  Service type="magnus-internal/servlet" fn="NSServletService"
  ```

- この指令の type 引数は、ObjectType 指令により設定された type の値に一致する。したがって、サーバは、NSServletService 関数を呼び出す、この Service 指令の実行に進む。この関数は、要求されたファイルをサーブレットとして呼び出し、サーブレットからの出力をクライアントへの応答として送信する。(要求されたリソースがサーブレットでない場合は、エラーが発生する。)

  Service 指令が実行されているので、サーバはその他の Service 指令を実行しない。(ただし、一致するオブジェクト内に実行された Service 指令がなかった場合は、サーバはデフォルトのオブジェクトの Service 指令の確認を続ける。)

デフォルトの Service 指令

ブラウザが送信した要求に一致する Service 指令がほかにない場合は、通常、デフォルトの操作 (ファイルの送信) を実行する Service 指令があります。このデフォルトの指令は、その他の Service 指令が成功していない場合にだけ呼び出されるように、デフォルトのオブジェクト内の Service 指令のリストの最後に記述する必要があります。デフォルトの Service 指令は、通常次のようになります。

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file"
```

この指令は、メソッドが GET、HEAD、または POST である要求に一致します。これは、ブラウザが送信するほとんどすべての要求に当てはまります。type 引数の値は、パターンマッチング用の特殊文字を使います。パターンマッチング用の特殊文字の詳細は、付録 C「ワイルドカードパターン」を参照してください。

文字 *~ は、「この後に続く文字に一致しないすべてのもの」を意味するので、*~magnus-internal/ は「magnus-internal/ に一致しないすべてのもの」を意味します。アスタリスク (*) 単独では、すべてのものに一致するので、*~magnus-internal/* 全体は、magnus-internal/ で始まらないすべてのものに一致します。

したがって、サーバがこの指令にたどり着くまでに Service 指令をまだ実行していない場合は、要求のメソッドが GET、HEAD、または POST であり、type 属性の値が magnus-internal/ で始まらないかぎり、この指令を実行します。呼び出される関数は send-file であり、この関数は要求されたファイルの内容をクライアントに送信するだけです。

AddLog

サーバは、応答を生成しクライアントへ送信した後、AddLog 指令を実行してログファイルにエントリを追加します。

すべての AddLog 指令が実行されます。サーバは、エントリを複数のログファイルに追加できます。

どのログファイルを使うか、またログファイルがどの書式を使うかにより、magnus.conf の Init セクションにはログを初期化するための指令が必要になる場合があります。たとえば、AddLog 指令の 1 つが拡張ログ書式を使う flex-log を呼び出す場合は、フレキシブルなログシステムを初期化するために flex-init を呼び出す指令が Init セクションに必要になります。

ログの初期化については、第 7 章「magnus.conf の構文と使用法」の flex-init および init-clf の説明を参照してください。

Error

PathCheck または AuthTrans 指令が要求したリソースへのアクセスを拒否したり、要求したリソースが存在しない場合など、要求の処理プロセス中にエラーが発生する場合は、サーバはただちにすべてのその他の指令の実行を停止し、Error 指令の実行を開始します。

obj.conf の編集上の構文規則

obj.conf ファイルでは、いくつかの規則が重要になります。このファイルの編集は、慎重に行なってください。単純な誤りでも、サーバが起動に失敗したり正しく動作しなくなることがあります。

指令の順序

サーバは指令を obj.conf に記述されている順に実行するので、指令の順序は重要です。一部の指令の結果が、ほかの指令の実行に影響を及ぼします。

PathCheck 指令の場合、サーバがすべての PathCheck 指令を実行するので、PathCheck セクション内の順序はさほど重要ではありません。しかし、ObjectType セクションでは順序は非常に重要です。これは、1 つの ObjectType 指令が属性値を設定すると、その他の ObjectType 指令はその値を変更できないからです。たとえば、デフォルトの ObjectType 指令が次の順序 (これは誤った逆の順序である) の場合は、すべての要求の type の値は text/plain に設定され、サーバには要求されたリソースの拡張子に従って type を設定する機会が与えられません。

```
ObjectType fn="force-type" type="text/plain"
ObjectType fn="type-by-extension"
```

同様に、Service セクション内の指令の順序は、非常に重要です。サーバは、現在の要求に一致する最初の Service 指令を実行し、その他は実行しません。

パラメータ

パラメータの個数と名前は、関数によって異なります。行の中でのパラメータの順序は、重要ではありません。

大文字、小文字の区別

obj.conf ファイル内の関数名、パラメータ名、多くのパラメータ値、パス名などの項目では、大文字と小文字が区別されます。

区切り文字

C 言語では、関数名には文字、数字、および下線だけを含めることができます。C コードの関数名の下線 (_) の代わりに、構成ファイルではハイフン (-) 文字を使うことができます。これは、関数名にだけ当てはまります。

引用符

引用符 (") は、文字列に空白文字が含まれるときにだけ文字列値のまわりに必要になります。そうでない場合は、省略可能です。各開始引用符には、対応する終了引用符が必要です。

空白文字

空白文字は、前の行から続いているとき以外は行の先頭に置くことはできません。空白文字は、名前と値を区切る等号 (=) の前後には置くことはできません。空白文字は、行の終わりや空行には置くことはできません。

行の継続

次の行を空白文字またはタブで始めることにより、長い行を次の行に続けることができます。

パス名

Windows NT では、パス名にはバックスラッシュ (\) ではなく必ずスラッシュ (/) を使います。バックスラッシュは、次の文字をエスケープします。

コメント

コメントは、シャープ (#) 記号で始まります。コメントを手動で `obj.conf` に追加し、その後サーバマネージャインタフェースを使ってサーバの設定を変更した場合には、サーバマネージャは、`obj.conf` を更新するときにコメントを削除します。

obj.conf 指令の例について

obj.conf ファイル内の行はすべて、以下のキーワードのいずれかで始まります。

```
AuthTrans
NameTrans
PathCheck
ObjectType
Service
AddLog
Error
<Object
</Object>
```

このマニュアルの例に出てきた行が上記以外の語で始まっている場合は、折り返されているためそうなっているだけであり、実際のファイルではそうはなりません。マニュアルの PDF および HTML 形式の行の長さの制限によって、このようになることがあります。

たとえば、次の指令は、実際の obj.conf ファイルではすべて 1 行に収まっています。

```
NameTrans fn="pfx2dir" from="/cgi"
dir="D:/netscape/server4/docs/mycgi" name="cgi"
```

第 3 章

事前定義済みの SAF および要求処理プロセス

この章では、obj.conf ファイル内でサーバに指示を与えるために使われる、標準の指令と事前定義済みのサーバアプリケーション関数 (SAF) を説明します。obj.conf の使い方と構文の詳細は、前の章、第 2 章「obj.conf の構文と使用法」を参照してください。

Init (初期設定) SAF のリストは、第 7 章「magnus.conf の構文と使用法」を参照してください。

この章には、iPlanet Web Server の主要機能の一部である関数も記載しています。この章では、サーブレットおよびサーバが構文解析する HTML などの追加のコンポーネントが有効な場合のみ利用できる関数については、記載していません。

この章では、各指令の節を設けて説明しています。それぞれの節には、該当の指令とともに使用できるすべての事前定義済みのサーバアプリケーション関数のリストがあります。

指令には次のものがあります。

- AuthTrans 段階
- NameTrans 段階
- PathCheck 段階
- ObjectType 段階
- Service 段階
- AddLog 段階
- Error 段階

事前定義済みの SAF のアルファベット順のリストは、付録 H「magnus.conf 内の指令のアルファベット順リスト」を参照してください。

表 3-1 は、各指令とともに使用できる SAF のリストです。

表 3-1　指令ごとの利用可能なサーバアプリケーション関数 (SAF)

AuthTrans 段階	basic-auth basic-ncsa get-sslid qos-handler
NameTrans 段階	assign-name document-root home-page pfx2dir redirect strip-params unix-home
PathCheck 段階	check-acl deny-existence find-index find-links find-pathinfo get-client-cert load-config nt-uri-clean ntcgicheck require-auth set-virtual-index ssl-check ssl-logout unix-uri-clean
ObjectType 段階	force-type set-default-type shtml-hacktype type-by-exp type-by-extension

表 3-1　指令ごとの利用可能なサーバアプリケーション関数 (SAF) (続き)

Service 段階	add-footer add-header append-trailer imagemap index-common index-simple key-toosmall list-dir make-dir query-handler remove-dir remove-file rename-file send-cgi send-file send-range send-shellcgi send-wincgi service-dump shtml_send stats-xml upload-file
AddLog 段階	common-log flex-log record-useragent
Error 段階	send-error qos-error

バケットパラメータ

iPlanet Web Server では、次のパフォーマンスバケットが事前に定義されています。

- `default-bucket` は、ユーザが定義したバケットまたは組み込みのバケットと関連しない関数の統計情報を記録します。
- `all-requests` バケットは、`default-bucket` の `.perf` 統計を含むすべての NSAPI SAF の `.perf` 統計を記録します。

`magnus.conf` ファイルに追加のパフォーマンスバケットを定義することができます (`perf-init` および `define-perf-bucket` 関数を参照)。

`bucket=cache-bucket` のように関数に `bucket=`*bucket-name* パラメータを追加することによって、`obj.conf` 内のすべての SAF のパフォーマンスを測定することができます。

パフォーマンス統計を一覧表示するには、`service-dump` Service 関数を使用します。

別の方法として、`stats-xml` Service 関数を使用してパフォーマンス統計を生成することができます。バケットは使用してもしなくてもかまいません。

パフォーマンスバケットの詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

AuthTrans 段階

`AuthTrans` は、承認変換 (Authorization Translation) を意味します。`AuthTrans` 指令は、クライアントにリソースへのアクセスを許可する前に承認を調べるため、命令をサーバに与えます。`AuthTrans` 指令は、`PathCheck` 指令との組み合わせで機能します。通常、`AuthTrans` 関数は、要求に関連付けられたユーザ名とパスワードが受け入れ可能かどうかを確認します。ただし、要求へのアクセスの許可や拒否は行なわず、これは `PathCheck` 関数が行ないます。

サーバは、次の 2 つのステップでクライアントユーザの承認を処理します。

- AuthTrans 指令 - Authorization ヘッダーにあるクライアントによって送信される承認情報の妥当性検査を行ないます。
- PathCheck 段階 - 承認されたユーザが要求されたリソースへのアクセスを許可されているかどうかを確認します。

承認プロセスは 2 つのステップに分かれているため、複数の承認スキーマを簡単に組み込むことができます。また、承認情報を記録してもそれを必要としないリソースを持つという、柔軟性も提供します。

AuthTrans 関数は、要求に関連付けられたヘッダーからユーザ名とパスワードを取得します。クライアントが最初に要求を行なうときは、ユーザ名とパスワードはまだ不明であり、AuthTrans 関数および PathCheck 関数はユーザ名とパスワードを検証できないため、共に作動して要求を拒否します。クライアントが拒否を受け取ったときの通常の対応は、適切なレルムを入力するためにユーザ名とパスワードを尋ねるダイアログボックスをポップアップすることです。その後クライアントは、ユーザ名とパスワードがヘッダーに入れられた状態で、再度要求を送信します。

obj.conf 内に複数の AuthTrans 指令がある場合、各関数は、いずれかがユーザの承認に成功するまで順番に実行されます。

この節では、次の AuthTrans クラス関数について詳しく説明します。

- basic-auth は、カスタム関数を呼び出してユーザ名とパスワードを検証します。任意 (省略可能) で、ユーザのグループを決定します。
- basic-ncsa は、ユーザ名とパスワードが NCSA スタイルまたはシステム DBM データベースに合っているかどうかを検証します。任意 (省略可能) で、ユーザのグループを決定します。
- get-sslid は、現在の SSL セッションに対して一意の文字列を検出し、それを Session->client パラメータブロック内に ssl-id 変数として格納します。
- qos-handler は、現在のサービス品質の統計を扱います。

basic-auth

AuthTrans クラス指令で適用可能です。

basic-auth 関数は、カスタム関数を呼び出して、クライアントが送信する承認情報を検証します。Authorization ヘッダーは、基本サーバ承認スキーマの一部として送信されます。

通常この関数は、PathCheck クラス関数 require-auth と組み合わせて使用されます。

パラメータ

auth-type	使用される承認のタイプを指定します。これは常に basic です。
userdb	(省略可能) ユーザの検証用に使われるユーザデータベースの絶対パスとファイル名を指定します。このパラメータは、ユーザ関数に渡されます。

userfn	承認を検証するためのユーザカスタム関数の名前です。この関数は、load-modules を使ってあらかじめ読み込んでおく必要があります。この関数のインタフェースはすべての SAF と同じです。ただし、pb パラメータにユーザ名 (user)、パスワード (pw)、ユーザデータベース (userdb)、およびもし指定されている場合には、グループデータベース (groupdb) を指定して呼び出されます。このユーザ関数は、データベースを使って名前とパスワードを確認し、それらが有効でなければ REQ_NOACTION を返します。名前とパスワードが有効であれば、REQ_PROCEED を返します。次に basic-auth 関数は、auth-type、auth-user (user)、auth-db (userdb)、および auth-password (pw、Windows NT のみ) を rq->vars pblock に追加します。
groupdb	(省略可能) グループデータベースの絶対パスとファイル名を指定します。このパラメータは、グループ関数に渡されます。
groupfn	(省略可能) グループカスタム関数の名前で、load-modules を使ってあらかじめ読み込んでおかなければなりません。この関数のインタフェースはすべての SAF と同じです。ただし、pb パラメータにユーザ名 (user)、パスワード (pw)、ユーザデータベース (userdb)、およびグループデータベース (groupdb) を指定して呼び出されます。また、この関数は、rq->vars pblock 内の auth-type、auth-user (user)、auth-db (userdb)、および auth-password (pw、Windows NT のみ) パラメータへもアクセス可能です。このグループ関数は、グループデータベースを使ってユーザのグループを決定し、それを auth-group として rq->vars に追加し、見つかった場合は REQ_PROCEED を返します。ユーザのグループが見つからなかった場合は、REQ_NOACTION を返します。
bucket	省略可能、すべての obj.conf 関数に共通

例

magnus.conf の場合 :

```
Init fn=load-modules shlib=/path/to/mycustomauth.so
funcs=hardcoded_auth
```

obj.conf の場合 :

```
AuthTrans fn=basic-auth auth-type=basic userfn=hardcoded_auth
PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
```

関連項目

require-auth

basic-ncsa

AuthTrans クラス指令で適用可能です。

basic-ncsa 関数は、クライアントが送信する承認情報がデータベースに合っているかどうか検証します。Authorization ヘッダーは、基本サーバ承認スキーマの一部として送信されます。

通常この関数は、PathCheck クラス関数 require-auth と組み合わせて使用されます。

パラメータ

auth-type	使用される承認のタイプを指定します。これは常に basic であるべきです。
dbm	(省略可能) サーバのネイティブ書式のユーザデータベースの絶対パスと基本ファイル名を指定します。ネイティブフォーマットはシステム DBM ファイルです。これは、膨大な数のユーザに瞬時にアクセスするための、ハッシュされたファイル書式です。このパラメータを使用する場合は、userfile パラメータを同時に使用しないでください。
userfile	(省略可能) NCSA スタイルの HTTPD ユーザファイル書式のユーザデータベースの絶対パス名を指定します。この書式は、*name:password* という書式の行で構成されます。ここで、*password* は暗号化されています。このパラメータを使用する場合、dbm は使用しないでください。
grpfile	(省略可能) 使用される NCSA スタイルの HTTPD グループファイルを指定します。グループファイルの各行は、*group* : *user1 user2 ... userN* で構成されます。ここで、各ユーザはスペースで区切られます。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
AuthTrans fn=basic-ncsa auth-type=basic
dbm=/netscape/server4/userdb/rs

PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
AuthTrans fn=basic-ncsa auth-type=basic
userfile=/netscape/server4/.htpasswd
grpfile=/netscape/server4/.grpfile

PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
```

関連項目

`require-auth`

get-sslid

`AuthTrans` クラス指令で適用可能です。

注	この関数は、下位互換性のためにのみ提供されているものです。`get-sslid` の機能は、SSL 接続の標準の処理に組み込まれています。

`get-sslid` 関数は、現在の SSL セッションに対して一意の文字列を検出し、それを `Session->client` パラメータブロック内に `ssl-id` 変数として格納します。

CGI が起動したときに変数 `ssl-id` が存在している場合、これが `HTTPS_SESSIONID` 環境変数として CGI に渡されます。

`get-sslid` 関数にはパラメータはなく、常に `REQ_NOACTION` が返されます。SSL が有効でない場合は無効です。

パラメータ

`bucket`	省略可能、すべての `obj.conf` 関数に共通

qos-handler

`AuthTrans` クラス指令で適用可能です。

qos-handler 関数は、仮想サーバ、仮想サーバクラス、およびグローバルサーバの現在のサービス品質の統計を検査し、統計情報のログをとり、エラーを返すことによって QOS パラメータを強化します。正しく機能させるためには、この関数は default オブジェクト内に構成されている最初の AuthTrans 関数でなければなりません。

この SAF 用のコードは、第 6 章「カスタム SAF の例」の例の 1 つにあります。

詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通

例

```
AuthTrans fn=qos-handler
```

関連項目

qos-error

NameTrans 段階

NameTrans は、名前変換 (Name Translation) を意味します。NameTrans 指令は、仮想 URL をサーバ上の物理ディレクトリに変換します。たとえば、URL

http://www.test.com/some/file.html

は、次のようなファイルシステムの絶対パスに変換されます。

/usr/netscape/server4/docs/some/file.html

NameTrans 指令は、デフォルトオブジェクトに存在します。1 つのオブジェクト内に複数の NameTrans 指令がある場合、サーバは、いずれかが成功するまで、それらの一つ一つを順番に実行します。

この節では、次の NameTrans クラス関数について詳しく説明します。

- assign-name は、名前付きオブジェクト内の指令を処理することを、サーバに伝えます。

- document-root は、要求されたリソースの http://*server-name*/ の部分をドキュメントルートディレクトリに置き換えることによって、URL をファイルシステムパスに変換します。
- home-page は、サーバのルートホームページ (/) に対する要求を特定のファイルに変換します。
- pfx2dir は、指定した接頭辞で始まるすべての URL をファイルシステムディレクトリに変換します。また任意 (省略可能) で、追加の名前付きオブジェクト内の指令を有効にします。
- redirect は、クライアントを別の URL にリダイレクトします。
- strip-params は、セミコロンで区切られた組み込みパラメータをパスから削除します。
- unix-home は、URL を、ユーザのホームディレクトリ内の指定したディレクトリに変換します。

assign-name

NameTrans クラス指令で適用可能です。

assign-name 関数は、現在の要求と一致する obj.conf 内のオブジェクトの名前を指定します。次にサーバは、デフォルトオブジェクト内の指令に優先して、名前付きオブジェクト内の指令を処理します。

例として、デフォルトオブジェクト内の次の指令について考えてください。

```
NameTrans fn=assign-name name=personnel from=/personnel
```

サーバが、http://*server-name*/personnel に対する要求を受け取ると仮定します。この NameTrans 指令を処理した後、サーバは obj.conf 内で personnel という名前のオブジェクトを探し、personnel オブジェクト内の指令を処理することにより続行します。

assign-name 関数は、常に REQ_NOACTION を返します。

パラメータ

from	影響を受けるパスを指定する、ワイルドカードパターンです。
name	指令がこの要求に適用される obj.conf 内の、追加の名前付きオブジェクトを指定します。

`find-pathinfo-forward`	(省略可能) これによりサーバは、パス内の ntrans-base 直後から順方向に PATHINFO を検索します。サーバ関数 `assign-name` がデフォルトで行なうように、パスの終わりから逆方向には検索しません。 このパラメータに割り当てる値は無視されます。このパラメータを使用したくない場合は、省略します。 `ntrans-base` パラメータが `rq->vars` に設定されていない場合、`find-pathinfo-forward` パラメータは無視されます。デフォルトでは、`ntrans-base` が設定されます。 この機能は、stat の実行数を減らすことによって、特定の URL のパフォーマンスを向上させます。
`nostat`	(省略可能) 可能なときは常に、サーバが指定した URL 上の stat を実行するのを禁止します。 `NameTrans` 関数 `assign-name` 内の `nostat="`*virtual-path*`"` の効果により、サーバは指定した *virtual-path* 上の stat が失敗するとみなします。このため、*virtual-path* のパスがシステム上に存在しないときのみ、たとえば、NSAPI プラグイン URL の場合、それらの URL の不要な stat を回避してパフォーマンスを向上させるために、`nostat` を使用します。 デフォルトの `PathCheck` サーバ関数を使用するとき、サーバは、*ntrans-base* が設定されている場合 (デフォルト状態) はパス */ntrans-base/virtual-path* および */ntrans-base/virtual-path/**` を stat しません。*ntrans-base* が設定されていない場合は、URL */virtual-path* および */virtual-path/**` を stat しません。
`bucket`	省略可能、すべての `obj.conf` 関数に共通

例

```
# This NameTrans directive is in the default object.
NameTrans fn=assign-name name=personnel from=/a/b/c/pers
...
<Object name=personnel>
...additional directives..
</Object>
NameTrans  fn="assign-name"  from="/perf"  find-pathinfo-forward=""
name="perf"
NameTrans fn="assign-name" from="/nsfc"  nostat="/nsfc"
name="nsfc"
```

document-root

NameTrans クラス指令で適用可能です。

document-root 関数は、サーバのルートドキュメントディレクトリを指定します。前の NameTrans 関数によって物理パスが設定されていない場合、パスの http://*server-name*/ 部分がドキュメントルートの物理パス名に置き換えられます。

サーバが http://*server-name*/somepath/somefile に対する要求を受け取るとき、document-root 関数は、http://*server-name*/ を root パラメータの値と置き換えます。たとえば、ドキュメントルートディレクトリが /usr/netscape/server4/docs の場合にサーバが http://*server-name*/a/b/file.html の要求を受け取ると、document-root 関数は、要求したリソースのパス名を /usr/netscape/server4/docs/a/b/file.html に変換します。

この関数は、常に REQ_PROCEED を返します。この後に存在する NameTrans 指令は決して呼び出されることはないため、必ず document-root を起動する指令が最後の NameTrans 指令となるようにします。

ルートドキュメントディレクトリは 1 つしか存在できません。追加のドキュメントディレクトリを指定するには、pfx2dir 関数を使って追加のパス名変換を設定します。

パラメータ

root	サーバのルートドキュメントディレクトリへのファイルシステムパスです。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
NameTrans fn=document-root root=/usr/netscape/server4/docs
NameTrans fn=document-root root=$docroot
```

関連項目

pfx2dir

home-page

NameTrans クラス指令で適用可能です。

home-page 関数は、サーバ用のホームページを指定します。クライアントがサーバのホームページ (/) を要求するときは常に、クライアントはその指定されたドキュメントを取得します。

パラメータ

path	ホームページファイルのパスと名前です。スラッシュ (/) で始まる場合、path はファイルへの絶対パスとみなされます。 この関数は、サーバの path 変数を設定し、REQ_PROCEED を返します。path が相対パスの場合、そのパスは URI に追加され、関数は REQ_NOACTION を返して、ほかの NameTrans 指令が続きます。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
NameTrans fn="home-page" path="homepage.html"
NameTrans fn="home-page" path="/httpd/docs/home.html"
```

pfx2dir

NameTrans クラス指令で適用可能です。

pfx2dir 関数は、要求された URL 内のディレクトリ接頭辞を実際のディレクトリ名と置き換えます。また、任意 (省略可能) で、現在の要求と一致するオブジェクトの名前を指定することもできます。(名前付きオブジェクトの使い方の詳細は、assign-name の説明を参照してください。)

パラメータ

from	変換する URI 接頭辞です。後ろにスラッシュ (/) をつけてはなりません。
dir	接頭辞の変換先のローカルのファイルシステムディレクトリパスです。後ろにスラッシュ (/) をつけてはなりません。
name	(省略可能) 指令がこの要求に適用される obj.conf 内の、追加の名前付きオブジェクトを指定します。

`find-pathinfo-forward`	(省略可能) これによりサーバは、パス内の ntrans-base 直後から順方向に PATHINFO を検索します。サーバ関数 `find-pathinfo` がデフォルトで行なうように、パスの終わりから逆方向には検索しません。 このパラメータに割り当てる値は無視されます。このパラメータを使用したくない場合は、省略します。 サーバ関数 `find-pathinfo` が呼び出されたときに `ntrans-base` パラメータが `rq->vars` に設定されていない場合、`find-pathinfo-forward` パラメータは無視されます。デフォルトでは、`ntrans-base` が設定されます。 この機能は、サーバ関数 `find-pathinfo` 内の stat の実行数を減らすことによって、特定の URL のパフォーマンスを向上させます。 NT では、この機能は、`PathCheck` サーバ関数 `find-pathinfo` が使われるときに、PATHINFO にサーバ URL 正規化プロセス ('\' を '/' に変更) を禁止するためにも使用できます。2 バイト文字によっては、\ または ~ のような URL 区切り文字として解析できる 16 進文字が含まれることがあります。場合によっては、`find-pathinfo-forward` パラメータを使用することで、2 バイト文字を含む URL の正しくない構文解析を防ぐことができます。
`bucket`	省略可能、すべての `obj.conf` 関数に共通

例

最初の例では、URL `http://`*server-name*`/cgi-bin/`*resource* (`http://x.y.z/cgi-bin/test.cgi` など) は物理パス名 `/httpd/cgi-local/`*resource* (`/httpd/cgi-local/test.cgi` など) に変換され、サーバも `cgi` という名前のオブジェクト内の指令の処理を開始します。

```
NameTrans fn=pfx2dir from=/cgi-bin dir=/httpd/cgi-local name=cgi
```

2 番目の例では、URL `http://`*server-name*`/icons/happy/`*resource* (`http://x.y.z/icons/happy/smiley.gif` など) は物理パス名 `/users/nikki/images/`*resource* (`/users/nikki/images/smiley.gif` など) に変換されます。

```
NameTrans fn=pfx2dir from=/icons/happy dir=/users/nikki/images
```

3 番目の例は、find-pathinfo-forward パラメータの使い方を示しています。URL http://*server-name*/cgi-bin/*resource* は、物理パス名 /export/home/cgi-bin/*resource* に変換されます。

```
NameTrans fn="pfx2dir" find-pathinfo-forward="" from="/cgi-bin"
dir="/export/home/cgi-bin" name="cgi"
```

redirect

NameTrans クラス指令で適用可能です。

redirect 関数により、URL を変更し、更新された URL をクライアントに送信することができます。クライアントが古いパスを使ってサーバにアクセスすると、サーバは要求を新しい URL の要求として扱います。

パラメータ

from	要求された URI の、一致する接頭辞を指定します。
url	(場合によって省略可能) クライアントに返す完全な URL を指定します。このパラメータを使用する場合、url-prefix は使用しないでください。逆の場合も同様です。
url-prefix	(場合によって省略可能) クライアントに返す新しい URL 接頭辞です。from 接頭辞は、単純にこの URL 接頭辞と置き換えられます。このパラメータを使用する場合、url は使用しないでください。逆の場合も同様です。
escape	(省略可能) 送信前に URL を util_uri_escape することをサーバに伝えるフラグです。yes または no でなければなりません。デフォルトは yes です。
bucket	省略可能、すべての obj.conf 関数に共通

例

最初の例では、http://*server-name*/*whatever* に対する要求はすべて、http://tmpserver/*whatever* に対する要求に変換されます。

```
NameTrans fn=redirect from=/ url-prefix=http://tmpserver
```

2 番目の例では、http://*server-name*/toopopular/*whatever* に対する要求はすべて、http://bigger/better/stronger/morepopular/*whatever* に対する要求に変換されます。

```
NameTrans fn=redirect from=/toopopular
url=http://bigger/better/stronger/morepopular
```

strip-params

NameTrans クラス指令で適用可能です。

strip-params 関数は、セミコロンで区切られた組み込みパラメータをパスから削除します。たとえば、/dir1;param1/dir2 の URI は /dir1/dir2 のパスになります。これを使用するとき、strip-params 関数は、リストされている最初の NameTrans 指令でなければなりません。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通

例

```
NameTrans fn=strip-params
```

unix-home

NameTrans クラス指令で適用可能です。

UNIX の場合のみ：unix-home 関数は、ユーザ名 (通常は、~username という形式) をサーバの UNIX マシン上のユーザのホームディレクトリに変換します。ユーザディレクトリをシグナルする URL 接頭辞を指定します。この接頭辞で始まるすべての要求は、ユーザのホームディレクトリに変換されます。

/etc/passwd ファイルまたは類似した構造のファイルのいずれかを使って、ユーザのリストを指定します。ファイル内の各行は、次のような構造になります (必要でない passwd ファイル内の要素は * で示されます)。

```
username:*:*:groupid:*:homedir:*
```

起動時に一度だけサーバにパスワードファイルを走査させたい場合は、Init クラス関数 init-uhome を magnus.conf 内で使用します。

パラメータ

from	変換する URL 接頭辞で、通常は「/~」です。
subdir	Web ドキュメントを含む、ユーザのホームディレクトリ内のサブディレクトリです。
pwfile	(省略可能) /etc/passwd と異なる場合は、パスワードファイルの絶対パスとファイル名です。
name	(省略可能) 指令がこの要求に適用される、追加の名前付きオブジェクトを指定します。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
NameTrans fn=unix-home from=/~ subdir=public_html
NameTrans fn=unix-home from=/~ pwfile=/mydir/passwd
subdir=public_html
```

関連項目

init-uhome, find-links

PathCheck 段階

PathCheck 指令は、NameTrans ステップの後に返されるローカルファイルシステムパスを検査します。このパスは、CGI パス情報などの項目や /./ および /../ および // などの危険要素を検査し、その後ですべてのアクセス制限が適用されます。

複数の PathCheck 指令がある場合、関数のそれぞれが順番に実行されます。

この節では、次の PathCheck クラス関数について詳しく説明します。

- check-acl は、承認のためアクセス制御リストを検査します。
- deny-existence は、リソースが見つからなかったことを示します。
- find-index は、ディレクトリが要求されたときにデフォルトファイルを検出します。
- find-links は、特定のファイルシステムへのリンクを持つディレクトリへのアクセスを拒否します。
- find-pathinfo は、PATH_INFO CGI 環境変数用にファイル名以外の特別なパス情報を検出します。
- get-client-cert は、SSL3 セッションから承認されたクライアントの証明書を取得します。
- load-config は、要求されたパス内のファイルから特別な構成情報を探して、読み込みます。
- nt-uri-clean は、「not found」を示すことで、安全でないパス名を持つリソースへのアクセスを拒否します。
- ntcgicheck は、指定した拡張子を持つ CGI ファイルを探します。
- require-auth は、承認されていないユーザまたはグループのアクセスを拒否します。
- set-virtual-index は、ディレクトリの仮想インデックスを指定します。
- ssl-check は、秘密鍵サイズを確認します。
- ssl-logout は、サーバの SSL セッションキャッシュ内の現在の SSL セッションを無効化します。
- unix-uri-clean は、「not found」を示すことで、安全でないパス名を持つリソースへのアクセスを拒否します。

check-acl

PathCheck クラス指令で適用可能です。

check-acl 関数は、クライアントが要求されたリソースへのアクセスを許可されているかどうかを確認するために使用する、アクセス制御リスト (Access Control List: ACL) を指定します。アクセス制御リストには、リソースへのアクセスが許可されている人や許可されていない人、どのような条件の下でアクセスが許可されているかに関する情報が含まれています。

オブジェクト内の PathCheck 指令の順番とは関係なく、check-acl 関数が最初に実行されます。指定した ACL が必要とする場合は、これによってユーザの認証が行なわれ、アクセス制御状態も更新されます。

パラメータ

acl	アクセス制御リストの名前です。
path	(省略可能) ACL を適用するパスを指定するワイルドカードパターンです。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
PathCheck fn=check-acl acl="*HRonly*"
```

deny-existence

PathCheck クラス指令で適用可能です。

deny-existence 関数は、クライアントが指定したパスにアクセスしようとしたときに、「not found」メッセージを送信します。サーバは「forbidden」ではなく「not found」を送信するため、パスが存在するかどうかをユーザが知ることができません。

パラメータ

path	(省略可能) 非表示にするファイルシステムパスのワイルドカードパターンです。パスが一致しない場合、この関数は何も行なわず、REQ_NOACTION を返します。パスが指定されていない場合は、一致するものとみなされます。
bong-file	(省略可能)「not found」メッセージで応答するのではなく、送信するファイルを指定します。これは、ファイルシステムの絶対パスです。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
PathCheck fn=deny-existence
path=/usr/netscape/server4/docs/private
PathCheck fn=deny-existence bong-file=/svr/msg/go-away.html
```

find-index

PathCheck クラス指令で適用可能です。

find-index 関数は、要求されたパスがディレクトリかどうかを調査します。要求されたパスがディレクトリであれば、この関数はディレクトリ内のインデックスファイルを検索してから、インデックスファイルをポイントするようにパスを変更します。インデックスファイルが見つからない場合、サーバはディレクトリのリストを生成します。

ファイル obj.conf に home-page を呼び出す NameTrans 指令があり、要求されたディレクトリがルートディレクトリの場合は、インデックスページではなくホームページがクライアントに返されることに注意してください。

照会文字列がある場合、HTTP メソッドが GET ではない場合、あるいはパスが有効なファイルのものである場合、find-index 関数は何も行ないません。

パラメータ

index-names	探索するインデックスファイル名のコンマで区切られたリストです。スペースは、それがファイル名の一部である場合のみ使用します。コンマの前または後にスペースを入れないでください。このリストは、ファイルシステムが大文字と小文字を区別する場合は、大文字と小文字を区別します。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
PathCheck fn=find-index index-names=index.html,home.html
```

find-links

PathCheck クラス指令で適用可能です。

UNIX の場合のみ：find-links 関数は、現在のパスに、ほかのディレクトリまたはファイルシステムへのシンボリックまたはハードリンクが存在するかどうか調べます。何も見つからないと、エラーが返されます。通常この関数は、信頼性のないディレクトリ (ユーザホームディレクトリなど) に対して使われます。この関数は、公開すべきでない情報をほかのユーザによってポイントされることのないようにします。

パラメータ

disable	使用不可にするリンクの文字列です。 • h はハードリンク • s はソフトリンク • o は、ユーザがリンクのターゲットを所有している場合のみ、ユーザホームディレクトリからのシンボリックリンクを許可します。
dir	検査を始めるディレクトリです。絶対パスを指定すると、指定したパスとそのサブディレクトリに対するすべての要求が、シンボリックリンクについて検査されます。部分パスを指定すると、その部分パスを含むすべての要求が、シンボリックリンクについて検査されます。たとえば、/user/ を使用するときに some/user/directory に対する要求が来ると、そのディレクトリはシンボリックリンクについて検査されます。
bucket	省略可能、すべての obj.conf 関数に共通
checkFileExistence	リンクされているファイルが存在しているかどうか検査し、その検査が失敗すると、403 (forbidden) で要求を中止します。

例

```
PathCheck fn=find-links disable=sh dir=/foreign-dir
PathCheck fn=find-links disable=so dir=public_html
```

関連項目

init-uhome, unix-home

find-pathinfo

PathCheck クラス指令で適用可能です。

find-pathinfo 関数は、URL 内のファイル名の後の追加のパス情報を探し、CGI 環境変数 PATH_INFO で使用するためにそれを格納します。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通

例

```
PathCheck fn=find-pathinfo
PathCheck fn=find-pathinfo find-pathinfo-forward=""
```

get-client-cert

PathCheck クラス指令で適用可能です。

get-client-cert 関数は、SSL3 セッションから承認済みクライアントの証明書を取得します。これは、すべての HTTP メソッド、または指定したパターンと一致するメソッドにのみ適用することができます。また、サーバ上で SSL が有効なときのみ機能します。

証明書が存在しているか、SSL3 セッションから取得された場合、この関数は REQ_NOACTION を返し、要求の続行を許可します。それ以外の場合、この関数は REQ_ABORTED を返し、プロトコル状態を 403 FORBIDDEN に設定します。これにより、要求は失敗し、クライアントは FORBIDDEN 状態になります。

パラメータ

dorequest	実際に証明書の取得を試みるか、またはそれが存在していることを単にテストするだけかを制御します。dorequest がない場合、デフォルト値は 0 です。 • 1 は、まだサーバにクライアントの証明書がない場合に、クライアント証明書を取得するために SSL3 ハンドシェークを再実行するよう関数に伝えます。一般的には、これによってクライアントは、ユーザに対してクライアント証明書を選択するダイアログボックスを表示します。最初のハンドシェークで要求されている場合、あるいはキャッシュされた SSL セッションが再開している場合は、サーバには既にクライアント証明書がある可能性があります。 • 0 は、まだサーバにクライアントの証明書がない場合に、SSL3 ハンドシェークを再実行しないよう関数に伝えます。 証明書がクライアントから取得され、サーバが検証に成功した場合、DER エンコードされた X.509 証明書の ASCII base64 エンコーディングは、Request->vars pblock 内のパラメータ auth-cert に配置され、この関数は、REQ_PROCEED を返して要求の続行を許可します。
require	クライアント証明書の取得の失敗が HTTP 要求を中止するかどうかを制御します。require がない場合、デフォルト値は 1 です。 • 1 は、dorequest が処理された後にクライアント証明書が存在しない場合、HTTP 要求を中止するよう関数に伝えます。この場合、HTTP 状態は PROTOCOL_FORBIDDEN に設定され、この関数は REQ_ABORTED を返します。 • 0 は、dorequest が処理された後にクライアント証明書が存在しない場合、REQ_NOACTION を返すよう関数に伝えます。
method	(省略可能) 関数が適用される HTTP メソッドのワイルドカードパターンを指定します。method がない場合、この関数はすべての要求に適用されます。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
# クライアント証明書をセッションから取得する
# 証明書がセッションに関連付けられていない場合は、証明書を要求する
# クライアントが有効な証明書を提示しなかった場合、要求は失敗する

PathCheck fn="get-client-cert" dorequest="1"
```

load-config

PathCheck クラス指令で適用可能です。

load-config 関数は、ドキュメントディレクトリ内で構成ファイルを検索し、そのファイルの内容をサーバの既存の構成に追加します。これらの構成ファイル (動的構成ファイルとも言う) は、要求されたリソースの追加のアクセス制御リストを指定します。サーバは、動的構成ファイルの規則に従って、要求されたリソースへのアクセスをクライアントに許可することも、許可しないこともあります。

load-config を起動する各指令は、ベースディレクトリに関連付けられています。ベースディレクトリは、basedir パラメータから明示的に示されるか、または要求されたリソースのルートディレクトリから派生します。ベースディレクトリは、次の 2 つのことを決定します。

- 要求がこの load-config 関数への呼び出しを起動する、最上位ディレクトリ。

  たとえば、ベースディレクトリが D:/Netscape/Server4/docs/nikki/ の場合、このディレクトリまたはそのサブディレクトリ (さらにそれらのサブディレクトリ) 内のリソースに対する要求のみが、動的構成ファイルの検索をトリガーします。要求されたリソースはベースディレクトリの親ディレクトリにあるため、リソース D:/Netscape/Server4/docs/somefile.html に対する要求はこのような検索をトリガーしません。

- サーバが要求されたリソースに適用する動的構成ファイルを検索する、最上位ディレクトリ。

  ベースディレクトリが D:/Netscape/Server4/docs/nikki/ の場合、サーバはこのディレクトリ内の動的構成ファイルの検索を開始します。ほかの要因によって、サブディレクトリを検索することもあれば、検索しないこともあります (ただし、親ディレクトリは検索しません)。

サーバマネージャのインタフェースから動的設定ファイルを有効にするとき、システムは obj.conf ファイルに、ppath パラメータを含む追加のオブジェクトを書き込みます。load-config を起動する指令を、デフォルトオブジェクトに手入力で追加する (それらを個々のオブジェクトに指定するのではなく) と、サーバマネージャのインタフェースはその変更を反映しないことがあります。

load-config を起動する PathCheck 指令を手入力でファイル obj.conf に追加する場合は、デフォルトオブジェクトにそれを指定するのではなく、追加のオブジェクト (<OBJECT> タグで作成したもの) にそれを指定します。動的構成ファイル内のアクセス規則が影響するリソースの部分パス名を指定するには、OBJECT タグの ppath 属性を使用します。この部分パス名は、パターンと一致する任意のパス名にすることができ、ワイルドカードを含むことができます。

たとえば、次の <OBJECT> タグは、ディレクトリ D:/Netscape/Server4/docs 内のリソースに対する要求がファイル my.nsconfig のアクセス規則に従うように指定しています。

```
<Object ppath="D:/Netscape/Server4/docs/*">
PathCheck fn="load-config" file="my.nsconfig" descend=1
basedir="D:/Netscape/Server4/docs"
</Object>
```

注 ppath が、ディレクトリツリーの中のベースディレクトリより上位 (あるいは、ツリーの別の分岐にある) のリソースまたはディレクトリへ解決する場合、load-config 関数は起動されません。これは、ベースディレクトリが、要求が load-config 関数を起動する最上位ディレクトリを指定するためです。

load-config 関数は、構成ファイルが読み込み済みであれば REQ_PROCEED を返し、エラーが発生したときは REQ_ABORTED を返し、構成ファイルがまだ読み込まれていなければ、REQ_NOACTION を返します。

パラメータ

file	(省略可能) 要求されたリソースに適用されるアクセス規則を含む、動的構成ファイルの名前です。これを指定しない場合、ファイル名は .nsconfig とみなされます。
disable-types	(省略可能) magnus-internal/cgi などの、ベースディレクトリを使用不可にするワイルドカードパターンタイプを指定します。これらのタイプと一致するリソースに対する要求は、中止されます。
descend	(省略可能) このパラメータがある場合は、サーバが動的構成ファイルについてこのディレクトリのサブディレクトリで検索することを指定します。たとえば、descend=1 は、サーバがサブディレクトリを検索しなければならないことを指定します。descend パラメータがなければ、この関数がベースディレクトリのみを検索しなければならないことを指定します。

basedir	(省略可能) ベースディレクトリを指定します。これは、要求が load-config 関数を起動する最上位ディレクトリであり、サーバが構成ファイルの検索を開始するディレクトリでもあります。 basedir が指定されていなければ、ベースディレクトリは、要求されたリソースの URL を物理パス名に変換した結果として生じる、ルートディレクトリとみなされます。たとえば、要求が http://*server-name*/a/b/file.html に対するものだった場合、物理ファイル名は次のようになります。 /*document-root*/a/b/file.html.
bucket	省略可能、すべての obj.conf 関数に共通

例

この例では、サーバが D:/Netscape/Server4/docs/nikki/ またはそのサブディレクトリにある部分文字列 secret を含むリソースに対する要求を受信するときは常に、checkaccess.nsconfig という構成ファイルが検索されます。

サーバはディレクトリ D:/Netscape/Server4/docs/nikki で検索を開始し、サブディレクトリも検索します。サーバは、要求されたリソースへのアクセスをクライアントに許可するか否かを決定するアクセス制御規則を適用するために、見つかった checkaccess.nsconfig の各インスタンスを読み込みます。

```
<Object ppath="*secret*">
PathCheck fn="load-config" file="checkaccess.nsconfig"
basedir="D:/Netscape/Server4/docs/nikki" descend="1"
</Object>
```

nt-uri-clean

PathCheck クラス指令で適用可能です。

Windows NT の場合のみ：nt-uri-clean 関数は、物理パスに \.\、\..\、または \\ (これらには潜在的にセキュリティ上の問題があります) が含まれているリソースへのアクセスを拒否します。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通
tildeok	存在する場合は、URL にチルド「~」文字が含まれることを許可します。これは、NT プラットフォームではセキュリティ上の危険性があります。longfi~1.htm は longfilename.htm を参照することもありますが、適切な ACL の検査は受けていません。
dotdirok	存在している場合は、"//" シーケンスは許可されます。

例

```
PathCheck fn=nt-uri-clean
```

関連項目

unix-uri-clean

ntcgicheck

PathCheck クラス指令で適用可能です。

Windows NT の場合のみ： ntcgicheck 関数は、拡張子を持たないファイル名にファイル名拡張子を追加するか、拡張子が .cgi のファイル名と置き換えることを指定します。

パラメータ

extension	置き換えるファイル拡張子です。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
PathCheck fn=ntcgicheck extension=pl
```

関連項目

init-cgi, send-cgi, send-wincgi, send-shellcgi

require-auth

PathCheck クラス指令で適用可能です。

require-auth 関数は、ユーザまたはグループが承認済みの場合のみ、リソースへのアクセスを許可します。この関数が呼び出される前は、承認関数 (basic-auth など) は AuthTrans 指令で呼び出す必要があります。

AuthTrans 指令でユーザが承認済みであり、auth-user パラメータが指定されている場合、ユーザの名前は auth-user ワイルドカード値と一致していなければなりません。また、auth-group パラメータが指定されている場合、承認済みユーザは承認済みグループに属している必要があり、この承認済みグループは auth-group ワイルドカード値と一致していなければなりません。

パラメータ

path	(省略可能) この関数が作動する、ワイルドカードのローカルのファイルシステムパスです。パスが指定されていない場合、この関数はすべてのパスに適用されます。
auth-type	使用される HTTP 承認のタイプであり、AuthTrans 内の前の承認関数の auth-type と一致していなければなりません。現在のところ、basic が唯一の定義されている承認タイプです。
realm	ユーザ名とパスワードが要求される、セキュリティ保護された領域 (またはレルム) を示すブラウザに送信された文字列です。
auth-user	(省略可能) アクセスが許可されているユーザのワイルドカードリストを指定します。このパラメータが指定されていない場合、承認関数によって承認されるすべてのユーザにアクセスが許可されます。
auth-group	(省略可能) アクセスが許可されているグループのワイルドカードリストを指定します。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
PathCheck fn=require-auth auth-type=basic realm="Marketing Plans"
auth-group=mktg auth-user=(jdoe|johnd|janed)
```

関連項目

basic-auth, basic-ncsa

set-virtual-index

PathCheck クラス指令で適用可能です。

set-virtual-index 関数は、URL の転送を決定する、ディレクトリの仮想インデックスを指定します。このインデックスは、LiveWire アプリケーション、それ自身のネームスペース内のサーブレット、Netscape Application Server applogic などがあてはまります。

from パラメータに現在の URI と一致する URI がリストされていない場合、REQ_NOACTION が返されます。virtual-index パラメータによって指定されているファイルがない場合、または現在の URI が見つからない場合は、REQ_ABORTED が返されます。現在の URI が from パラメータに記述されている URI のいずれかと一致する場合、または from パラメータがない場合は、REQ_RESTART が返されます。

パラメータ

virtual-index	ユーザが入力する URI 用のインデックスとして機能する、コンテンツジェネレータの URI です。
from	(省略可能) この virtual-index を適用できる、コンマで区切られた URI のリストです。from が指定されていない場合は、常に virtual-index が適用されます。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
# MyLWApp は LiveWire アプリケーションである
PathCheck fn=set-virtual-index virtual-index=MyLWApp
```

ssl-check

PathCheck クラス指令で適用可能です。

セキュリティの設定で指定されている現在の暗号化設定と整合性のない制限が選択されている場合、この関数は、秘密鍵サイズがより大きい暗号化を有効にする必要があることを警告する、ポップアップ ダイアログを開きます。この関数は、Client タグと一緒に使用して、エクスポートできないブラウザに対して特定のディレクトリへのアクセスを制限するよう設計されています。

SSL が有効でない場合、または secret-keysize パラメータが指定されていない場合、この関数は REQ_NOACTION を返します。現在のセッションの秘密鍵サイズが指定した secret-keysize より小さく、bong-file パラメータが指定されていない場合、この関数は、PROTOCOL_FORBIDDEN のステータスと共に REQ_ABORTED を返します。bong ファイルが指定されている場合、この関数は REQ_PROCEED を返し、bong ファイル名に path 変数が設定されます。また、鍵サイズの制限が満たされないときは、現在のセッションの SSL セッションキャッシュエントリは無効になるため、同じクライアントが次にサーバに接続するときにフル SSL ハンドシェークが発生します。

ssl-check が REQ_NOACTION 以外のものを返す場合、ssl-check を使用する要求はアクセラレータファイルのキャッシュではキャッシュできません。

パラメータ

secret-keysize	(省略可能) 秘密鍵に必要な最低ビット数です。
bong-file	(省略可能) 制限に合わない場合に機能させるファイルの名前 (URI ではない) です。
bucket	省略可能、すべての obj.conf 関数に共通

ssl-logout

PathCheck クラス指令で適用可能です。

ssl-logout は、サーバの SSL セッションキャッシュ内の現在の SSL セッションを無効化します。これは、現在の要求には影響しませんが、クライアントが次に接続するときに、新しい SSL セッションを作成します。SSL が有効な場合、この関数は、セッションキャッシュエントリを無効にした後に REQ_PROCEED を返します。SSL が有効でない場合は、REQ_NOACTION を返します。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通

unix-uri-clean

PathCheck クラス指令で適用可能です。

UNIX の場合のみ：unix-uri-clean 関数は、物理パスに /./、/../、または //(これらには潜在的にセキュリティ上の問題があります) が含まれているリソースへのアクセスを拒否します。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通
dotdirok	存在している場合は、"//" シーケンスは許可されます。

例

```
PathCheck fn=unix-uri-clean
```

関連項目

nt-uri-clean

ObjectType 段階

ObjectType 指令は、要求に応えてクライアントに送信されるファイルの、MIME タイプを決定します。現在、送信される MIME 属性は type、encoding、および language です。MIME タイプは、content-type ヘッダーの値としてクライアントに送信されます。

ObjectType 指令は type パラメータも設定します。このパラメータは、Service 指令が、要求された内容の種類に応じて要求を処理する方法を決定するために使用します。

1 つのオブジェクト内に複数の ObjectType 指令がある場合、現れる順番ですべての指令が適用されます。ある指令が属性を設定し、それ以降の指令がその属性を変更しようとした場合、最初の設定が使用され、それ以降の設定は無視されます。

ほとんどの場合 obj.conf ファイルには、type-by-extension 関数を呼び出す ObjectType 指令があります。この関数はサーバに、特定のファイル (MIME タイプのファイル) を確認して要求されたリソースの拡張子からコンテンツタイプを推測するよう、指示します。

この節では、次の ObjectType クラス関数について詳しく説明します。

- force-type は、特定のタイプへの応答の content-type ヘッダーを設定します。
- set-default-type を使って、クライアントに返信される応答のデフォルトの charset、content-encoding、および content-language を定義することができます。
- shtml-hacktype は、.htm および .html ファイルが、サーバが構文解析する html コマンドについて解析されるように要求します。
- type-by-exp は、要求されたパスに基づいて応答の content-type ヘッダーを設定します。
- type-by-extension は、ファイル拡張子と MIME タイプのデータベースに基づいて、応答の content-type ヘッダーを設定します。

force-type

ObjectType クラス指令で適用可能です。

force-type 関数は、まだ MIME タイプを持たない要求に、タイプを割り当てます。これを使用して、デフォルトのオブジェクトタイプを指定します。

ほかのすべての ObjectType 指令が MIME タイプを最初に設定する機会を持つように、この関数を呼び出す指令が ObjectType 指令のリストの最後に来るようにしてください。1 つのオブジェクト内に複数の ObjectType 指令がある場合、現れる順番ですべての指令が適用されます。ある指令が属性を設定し、それ以降の指令がその属性を変更しようとした場合、最初の設定が使用され、それ以降の設定は無視されます。

パラメータ

type	(省略可能) 一致する要求に割り当てられたタイプです (content-type ヘッダー)。
enc	(省略可能) 一致する要求に割り当てられたエンコーディングです (content-encoding ヘッダー)。
lang	(省略可能) 一致する要求に割り当てられた言語です (content-language ヘッダー)。
charset	(省略可能) rq->srvhdrs 内の magnus-charset パラメータの文字セットです。ブラウザが Accept-charset ヘッダーを送信した場合、またはその User-agent が mozilla/1.1 以上の場合は、; charset=*charset* を content-type に付加します。ここで、*charset* は、rq->srvhdrs 内の magnus-charset パラメータの値です。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
ObjectType fn=force-type type=text/plain
ObjectType fn=force-type lang=en_US
```

関連項目

type-by-extension, type-by-exp

set-default-type

ObjectType クラス指令で適用可能です。

この関数を使って、クライアントに返信される応答のデフォルトの charset、content-encoding、および content-language を定義することができます。

charset、content-encoding、および content-language が応答に設定されていない場合は、ヘッダーが送信される直前に、set-default-type で定義されているデフォルトが使用されます。この関数を obj.conf 内のさまざまなオブジェクトに置くことによって、ドキュメントツリーのさまざまな部分に対して、さまざまなデフォルトを定義することができます。

パラメータ

enc	(省略可能) 一致する要求に割り当てられたエンコーディングです (content-encoding ヘッダー)。
lang	(省略可能) 一致する要求に割り当てられた言語です (content-language ヘッダー)。
charset	(省略可能) rq->srvhdrs 内の magnus-charset パラメータの文字セットです。ブラウザが Accept-charset ヘッダーを送信した場合、またはその User-agent が mozilla/1.1 以上の場合は、; charset=*charset* を content-type に付加します。ここで、*charset* は、rq->srvhdrs 内の magnus-charset パラメータの値です。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
ObjectType fn="set-default-type" charset="iso_8859-1"
```

shtml-hacktype

ObjectType クラス指令で適用可能です。

shtml-hacktype 関数は、.htm または .html ファイルの content-type を magnus-internal/parsed-html に変更し、REQ_PROCEED を返します。これは、拡張子が .htm または .html のファイルをサーバ側に取り込むという、下位互換性を提供します。この関数は、UNIX システム上のファイルの実行ビットを検査することもできます。この関数の使用は推奨されていません。

パラメータ

exec-hack	(UNIX の場合のみ。省略可能) 関数に、実行ビットが有効な場合のみ content-type を変更することを伝えます。このパラメータの値は重要ではありません。指定するだけでかまいません。exec-hack=true を使用することもできます。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
ObjectType fn=shtml-hacktype exec-hack=true
```

type-by-exp

ObjectType クラス指令で適用可能です。

type-by-exp 関数は、現在のパスをワイルドカード拡張子と一致させます。この 2 つが一致する場合、type パラメータ情報がファイルに適用されます。これは、URL に指定されているファイルまたはディレクトリのワイルドカードパターンを使用することを除けば、type-by-extension と同じです。

パラメータ

exp	この関数を適用するパスのワイルドカードパターンです。
type	(省略可能) 一致する要求に割り当てられたタイプです (content-type ヘッダー)。
enc	(省略可能) 一致する要求に割り当てられたエンコーディングです (content-encoding ヘッダー)。

lang	(省略可能) 一致する要求に割り当てられた言語です (content-language ヘッダー)。
charset	(省略可能) rq->srvhdrs 内の magnus-charset パラメータの文字セットです。ブラウザが Accept-charset ヘッダーを送信した場合、またはその User-agent が mozilla/1.1 以上の場合は、; charset=*charset* を content-type に付加します。ここで、*charset* は、rq->srvhdrs 内の magnus-charset パラメータの値です。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
ObjectType fn=type-by-exp exp=*.test type=application/html
```

関連項目

type-by-extension, force-type

type-by-extension

ObjectType クラス指令で適用可能です。

この関数はサーバに、MIME タイプマッピングのテーブルを見て、要求されたリソースの拡張子に従って要求されたリソースの MIME タイプを探すよう指示します。MIME タイプは、クライアントに返信される content-type ヘッダーに追加されます。

MIME タイプマッピングのテーブルは、server.xml ファイル内の MIME 要素によって作成されます。このファイルは、MIME タイプのファイルまたはリストを読み込んでマッピングを作成します。server.xml の詳細は、第 8 章「仮想サーバの構成ファイル」を参照してください。MIME タイプファイルの詳細は、付録 B「MIME タイプ」を参照してください。

たとえば、次の 2 行は MIME タイプファイルの一部分です。

```
type=text/html     exts=htm,html
type=text/plain     exts=txt
```

要求されたリソースの拡張子が htm または html の場合、type-by-extension 関数はタイプを text/html に設定します。拡張子が .txt の場合、関数によりタイプは text/plain に設定されます。

パラメータ

bucket	省略可能、すべての obj.conf 関数に共通

例

```
ObjectType fn=type-by-extension
```

関連項目

type-by-exp, force-type

Service 段階

関数の Service クラスは、クライアントに応答データを送信します。

すべての Service 指令には、関数が実行されるかどうかを決定する、次のようなパラメータ (省略可能) があります。省略可能なすべてのパラメータは、実行される関数に対する現在の要求と一致している必要があります。

- type

  (省略可能) この関数が実行される MIME タイプのワイルドカードパターンを指定します。magnus-internal/* MIME タイプは、実行する Service 関数を選択するためにのみ使用されます。

- method

  (省略可能) この関数が実行される HTTP メソッドのワイルドカードパターンを指定します。一般的な HTTP メソッドは、GET、HEAD、および POST です。

- query

  (省略可能) この関数が実行される照会文字列のワイルドカードパターンを指定します。

- UseOutputStreamSize

(省略可能) クライアントに送信されるデータのデフォルトの出力ストリームのバッファサイズ (バイト単位) を決定します。このパラメータが指定されていない場合、デフォルトは 8192 バイトです。

注 UseOutputStreamSize パラメータを obj.conf ファイル内で 0 に設定して、出力ストリームバッファリングを使用不可にすることができます。magnus.conf ファイルの場合、UseOutputStreamSize を 0 に設定しても何にも影響は与えません。

- flushTimer

  (省略可能) バッファリングが有効である書き込み操作間の間隔の最大数を、ミリ秒で決定します。次に続く書き込み操作間の間隔がアプリケーションの flushTimer 値より大きい場合、それ以降のバッファリングは使用不可になります。これは、継続的に稼働し、定期的な状態更新レポートを生成する、CGI アプリケーションの状態を監視する際に必要です。このパラメータが指定されていない場合、デフォルトは 3000 ミリ秒です。

- ChunkedRequestBufferSize

  (省略可能)「チャンクしていない」要求データのデフォルトのバッファサイズ (バイト単位) を決定します。このパラメータが指定されていない場合、デフォルトは 8192 バイトです。

- ChunkedRequestTimeout

  (省略可能)「チャンクしていない」要求データのデフォルトのタイムアウト (秒単位) を決定します。このパラメータが指定されていない場合、デフォルトは 60 秒です。

複数の Service クラス関数がある場合、省略可能なワイルドカードパラメータ (type、method、および query) と一致する最初のものが実行されます。

UseOutputStreamSize、flushTimer、ChunkedRequestBufferSize、および ChunkedRequestTimeout パラメータの詳細は、326 ページの「バッファ化されたストリーム」を参照してください。UseOutputStreamSize、ChunkedRequestBufferSize、および ChunkedRequestTimeout パラメータも magnus.conf 指令と同様です。278 ページの「チャンクされたエンコーディング」を参照してください。obj.conf パラメータは、magnus.conf 指令を上書きします。

デフォルトでは、サーバは、send-file 関数を呼び出すことによって、要求されたファイルをクライアントに送信します。デフォルトを設定する指令は次のとおりです。

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file"
```

通常、この指令はほかのすべての Service 指令に起動の機会を与えるため、Service クラス指令のセットの最後に来ます。この指令は、要求のメソッドが GET、HEAD、または POST で、タイプが magnus-internal/ で始まらない場合に起動されます。*~ というパターンは「一致しない」ことを意味するので、注意してください。パターンで使用できる文字のリストは、付録 C「ワイルドカードパターン」を参照してください。

この節では、次の Service クラス関数について詳しく説明します。

- add-footer は、ファイル名または URL で指定されているフッターを HTML ファイルに付加します。
- add-header は、ファイル名または URL で指定されているヘッダーを HTML ファイルの前に付加します。
- append-trailer は、HTML ファイルの終わりにテキストを追加します。
- imagemap は、サーバ側のイメージマップを処理します。
- index-common は、要求されたディレクトリ内のファイルとディレクトリの拡張リストを生成します。
- index-simple は、要求されたディレクトリ内のファイルとディレクトリのシンプルなリストを生成します。
- key-toosmall は、提供されている証明書鍵サイズが受け入れるには小さすぎることを、クライアントに示します。
- list-dir は、ディレクトリの内容を一覧表示します。
- make-dir はディレクトリを作成します。
- query-handler は、HTML ISINDEX タグを処理します。
- remove-dir は空のディレクトリを削除します。
- remove-file はファイルを削除します。
- rename-file はファイルの名前を変更します。
- send-cgi は、環境変数を設定し、CGI プログラムを起動し、クライアントに応答を送信します。
- send-file は、ローカルファイルをクライアントに送信します。
- send-range は、バイト単位のファイルの範囲をクライアントに送信します。
- send-shellcgi は、環境変数を設定し、シェル CGI プログラムを起動し、クライアントに応答を送信します。
- send-wincgi は、環境変数を設定し、WinCGI プログラムを起動し、クライアントに応答を送信します。
- service-dump は、収集されたパフォーマンスバケットデータに基づいて、パフォーマンスレポートを作成します。

- shtml_send は、サーバが構文解析する html コマンドについて HTML ファイルを構文解析します。
- stats-xml は、XML フォーマットのパフォーマンスレポートを作成します。
- upload-file は、ファイルをアップ読み込んで保存します。

add-footer

Service クラス指令で適用可能です。

この関数は、クライアントに送信される HTML ファイルにフッターを付加します。このフッターはファイル名または URI のいずれかとして指定されます。つまり、フッターは動的に生成できます。フッターとして静的テキストを指定するには、append-trailer 関数を使用します。

パラメータ

file	(省略可能) フッターを含むファイルへのパス名。file または uri のいずれかを指定します。 デフォルトでは、パス名は相対です。パス名が絶対パス名の場合は、NSIntAbsFilePath パラメータを yes として渡します。
uri	(省略可能) フッターを含むリソースを指す URI。file または uri のいずれかを指定します。
NSIntAbsFilePath	(省略可能) file パラメータが指定されている場合、NSIntAbsFilePath パラメータは、ファイル名が絶対か相対かを決定します。デフォルトは相対です。絶対ファイルパスを示すには、値を yes に設定します。
type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type=text/html method=GET fn=add-footer
file="footers/footer1.html"
Service type=text/html method=GET fn=add-footer
file="D:/netscape/server4/footers/footer1.html"
NSIntAbsFilePath="yes"
```

関連項目

append-trailer, add-header

add-header

Service クラス指令で適用可能です。

この関数は、クライアントに送信される HTML ファイルの前にヘッダーを追加します。このヘッダーはファイル名または URI のいずれかとして指定されます。つまり、ヘッダーは動的に生成できます。

パラメータ

file	(省略可能) ヘッダーを含むファイルへのパス名。file または uri のいずれかを指定します。 デフォルトでは、パス名は相対です。パス名が絶対パス名の場合は、NSIntAbsFilePath パラメータを yes として渡します。
uri	(省略可能) ヘッダーを含むリソースをポイントする URI。file または uri のいずれかを指定します。
NSIntAbsFilePath	(省略可能) file パラメータが指定されている場合、NSIntAbsFilePath パラメータは、ファイル名が絶対か相対かを決定します。デフォルトは相対です。絶対ファイルパスを示すには、値を yes に設定します。
type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通

bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type=text/html method=GET fn=add-header
file="headers/header1.html"
Service type=text/html method=GET fn=add-header
file="D:/netscape/server4/headers/header1.html"
NSIntAbsFilePath="yes"
```

関連項目

add-footer, append-trailer

append-trailer

Service クラス指令で適用可能です。

append-trailer 関数は、HTML ファイルを送信し、その終わりにテキストを付加します。これは、HTML ファイルにテキストを付加するだけです。通常これは、作成者の情報と著作権のテキストに使われます。ファイルの最後の更新日を挿入できます。

必須パラメータがない場合、URL のファイル名の後に追加のパス情報がある場合、あるいはファイルが読み取り専用アクセスで開くことができない場合は、REQ_ABORTED を返します。

パラメータ

trailer	HTML ドキュメントに追加するテキストです。文字列は、送信される前に util_uri_unescape でアンエスケープ処理されます。テキストには HTML タグを入れることができ、データのアンエスケープ処理と挿入の後で最大 512 文字にすることができます。 文字列 :LASTMOD: (これは、ファイルが最後に更新された日付と置き換えられます) を使用する場合は、timefmt の時刻形式も指定する必要があります。
timefmt	(省略可能) :LASTMOD: の時刻形式の文字列です。時刻形式の詳細は、付録 D「時刻の書式」を参照してください。timefmt が指定されていない場合、:LASTMOD: は時刻とは置き換えられません。
type	省略可能、すべての Service クラス関数に共通

method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type=text/html method=GET fn=append-trailer
trailer="<hr><img src=/logo.gif> Copyright 1999"
# トレーラを MM/DD/YY 形式の日付付きで追加する : MM/DD/YY
Service type=text/html method=GET fn=append-trailer timefmt="%D"
trailer="<HR>File last updated on: :LASTMOD:"
```

関連項目

add-footer, add-header

imagemap

Service クラス指令で適用可能です。

imagemap 関数は、イメージマップの要求に応答します。イメージマップは複数領域に分割されたイメージであり、それぞれには関連する URL があります。どの URL がどの領域に関連付けられているかについての情報は、マッピングファイルに格納されます。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通

ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type=magnus-internal/imagemap method=(GET|HEAD)
fn=imagemap
```

index-common

Service クラス指令で適用可能です。

index-common 関数は、要求されたディレクトリ内のファイルの拡張 (または共通) リストを生成します。このリストはアルファベット順にソートされています。ピリオド (.) で始まるファイルは表示されません。各項目は 1 つの HTML リンクとして表示されます。この関数は、各ファイルのサイズ、最新の更新日付、およびアイコンを含む、index-simple よりも詳細な情報を表示します。また、リストにはヘッダーや readme ファイルも含むことができます。

magnus.conf 内の Init クラス関数 cindex-init は、イメージの検索場所を含むインデックスリストの形式を指定します。

obj.conf に Service 段階内の index-common への呼び出しが含まれている場合、magnus.conf は、Init 段階中に cindex-init を呼び出すことによって拡張 (または共通) インデックス処理を初期化しなければなりません。

インデックス処理は、要求されたリソースがインデックスファイルまたはホームページを含まないディレクトリのとき、あるいは、関数 find-index または home-page によってインデックスファイルやホームページが指定されていないときに発生します。

表示されるアイコンは .gif ファイルで、ファイルの content-type によって異なります。

"text/*"	text.gif
"image/*"	image.gif
"audio/*"	sound.gif
"video/*"	movie.gif
"application/octet-stream"	binary.gif

ディレクトリ	menu.gif
ほかのすべて	unknown.gif

パラメータ

header	(省略可能) ディレクトリリスティングの先頭にある、ディレクトリの内容を紹介するパス (インデックスされるディレクトリに対して相対) とファイルの名前 (HTML またはプレーンテキスト)。最初にこのファイルは、末尾に追加されている .html で検索されます。見つかった場合は、ディレクトリリストの先頭近くに HTML として組み込まれます。見つからない場合、このファイルは .html を除いた部分で検索され、事前フォーマット済みのプレーンテキスト (<PRE> で囲まれたもの) として組み込まれます。
readme	(省略可能) ディレクトリリスティングに付加するパス (インデックスされるディレクトリに対して相対) とファイルの名前 (HTML またはプレーンテキスト) です。このファイルは、著作権、作者、またはその他の情報など、ディレクトリの内容に関するより詳細な情報を提供することがあります。最初にこのファイルは、末尾に追加されている .html で検索されます。見つかった場合は、ディレクトリリストの最下部に HTML として組み込まれます。見つからない場合、このファイルは .html を除いた部分で検索され、事前フォーマット済みのプレーンテキスト (<PRE> と </PRE> で囲まれたもの) として組み込まれます。
type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn=index-common type=magnus-internal/directory
method=(GET|HEAD) header=hdr readme=rdme.txt
```

関連項目

cindex-init, index-simple, find-index, home-page

index-simple

Service クラス指令で適用可能です。

index-simple 関数は、要求されたディレクトリ内のファイルのシンプルなインデックスを生成します。これは、ディレクトリを走査し、そのディレクトリ内のファイルとディレクトリをビュレット付きのリストで HTML ページとしてブラウザに表示します。このリストはアルファベット順にソートされています。ピリオド (.) で始まるファイルは表示されません。各項目は 1 つの HTML リンクとして表示されます。

インデックス処理は、要求されたリソースがインデックスファイルまたはホームページのいずれかを含まないディレクトリのとき、あるいは、関数 find-index または home-page によってインデックスファイルやホームページが指定されていないときに発生します。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type=magnus-internal/directory fn=index-simple
```

関連項目

cindex-init, index-common

key-toosmall

Service クラス指令で適用可能です。

注	この関数は下位互換性のためにのみ提供されているもので、iPlanet Web Server 4.*x* では推奨されていませんでした。これは PathCheck クラス SAF ssl-check と置き換えられます。

key-toosmall 関数は、SSL 通信用の秘密鍵サイズが小さすぎることをクライアントに示すメッセージを返します。この関数は、Client タグと一緒に使用して、エクスポートできないブラウザに対して特定のディレクトリへのアクセスを制限するよう設計されています。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
<Object ppath=/mydocs/secret/*>
Service fn=key-toosmall
</Object>
```

list-dir

`Service` クラス指令で適用可能です。

`list-dir` 関数は、メソッドが INDEX である要求に応えて、一連のテキスト行をクライアントに返します。返される行の形式は次のとおりです。

name type size mtime

name フィールドは、ファイルまたはディレクトリの名前です。これは、インデックスされるディレクトリに対して相対です。これは URL エンコーディングされています。つまり、文字は `%xx` で表されます。`xx` は、文字の ASCII 番号の 16 進表現です。

type フィールドは、`text/html` のような MIME タイプです。ディレクトリはタイプ `directory` になります。サーバにタイプがないファイルのタイプは、`unknown` になります。

size フィールドは、バイト単位のファイルのサイズです。

mtime フィールドは、ファイルの最後の更新日時の数値表現です。この数は、epoch (1970 年 1 月 1 日、00:00、協定世界時) 以降の、ファイルの最後の更新日時までの秒数です。

サーバでリモートファイル操作が使用可能なとき、`obj.conf` ファイルには、メソッドが `INDEX` である要求に対する `list-dir` を呼び出す `Service` クラス関数が含まれます。

パラメータ

`type`	省略可能、すべての Service クラス関数に共通
`method`	省略可能、すべての Service クラス関数に共通
`query`	省略可能、すべての Service クラス関数に共通
`UseOutputStreamSize`	省略可能、すべての Service クラス関数に共通
`flushTimer`	省略可能、すべての Service クラス関数に共通
`ChunkedRequestBufferSize`	省略可能、すべての Service クラス関数に共通
`ChunkedRequestTimeout`	省略可能、すべての Service クラス関数に共通

bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn=list-dir method="INDEX"
```

make-dir

Service クラス指令で適用可能です。

make-dir 関数は、メソッドが MKDIR である要求をクライアントが送信するときに、ディレクトリを作成します。サーバがそのディレクトリに書き込みできない場合、この関数は失敗します。

サーバでリモートファイル操作が使用可能なとき、obj.conf ファイルには、要求メソッドが MKDIR のときに make-dir を起動する Service クラス関数が含まれます。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn="make-dir" method="MKDIR"
```

query-handler

Service クラス指令で適用可能です。

注	この関数は下位互換性のためにのみ提供されているもので、主に、古い ISINDEX タグをサポートするために使われます。可能であれば、代わりに HTML 形式を使用してください。

query-handler 関数は、要求されたパスを参照する代わりに、CGI プログラムを実行します。

パラメータ

path	実行する CGI プログラムの絶対パスとファイル名です。
type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service query=* fn=query-handler path=/http/cgi/do-grep
Service query=* fn=query-handler path=/http/cgi/proc-info
```

remove-dir

Service クラス指令で適用可能です。

remove-dir 関数は、メソッドが RMDIR である要求をクライアントが送信するときに、ディレクトリを削除します。ディレクトリは空 (中にファイルがない状態) でなければなりません。ディレクトリが空でない場合、またはサーバにディレクトリを削除する特権がない場合、この関数は失敗します。

サーバでリモートファイル操作が使用可能なとき、obj.conf ファイルには、要求メソッドが RMDIR のときに remove-dir を起動する Service クラス関数が含まれます。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn="remove-dir" method="RMDIR"
```

remove-file

Service クラス指令で適用可能です。

remove-file 関数は、メソッドが DELETE である要求をクライアントが送信するときに、ファイルを削除します。ユーザが承認されており、サーバが必要なファイルシステム特権を持っている場合に、URL で示されているファイルを削除します。

サーバでリモートファイル操作が使用可能なとき、obj.conf ファイルには、要求メソッドが DELETE のときに remove-file を起動する Service クラス関数が含まれます。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通

flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn="remove-file" method="DELETE"
```

rename-file

Service クラス指令で適用可能です。

rename-file 関数は、メソッドが MOVE の New-URL ヘッダーを持つ要求をクライアントが送信するときに、ファイルの名前を変更します。ユーザが承認されており、サーバが必要なファイルシステム特権を持っている場合は、URL で示されているファイルの名前を同じディレクトリ内で New-URL へ変更します。

サーバでリモートファイル操作が使用可能なとき、obj.conf ファイルには、要求メソッドが MOVE のときに rename-file を起動する Service クラス関数が含まれます。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn="rename-file" method="MOVE"
```

send-cgi

`Service` クラス指令で適用可能です。

`send-cgi` 関数は、CGI 環境変数を設定し、新しいプロセスでファイルを CGI プログラムとして実行し、その結果をクライアントに送信します。

CGI 環境変数とそれらに対応する NSAPI についての詳細は、133 ページの「CGI から NSAPI への変換」の節を参照してください。

CGI の詳細は、iPlanet Web Server の『管理者ガイド』および iPlanet Web Server の『プログラマーズガイド』を参照してください。

CGI バッファのフラッシュに使用するタイミングを変更する方法は、3 つあります。

- フラッシュ間の間隔を `flushTimer` パラメータを使って調整する
- バッファサイズを `UseOutputStreamSize` パラメータを使って調整する
- CGI スクリプトの中のバッファに強制的にスペースを加えて、iPlanet Web Server に強制的にそのバッファをフラッシュさせる

`flushTimer` および `UseOutputStreamSize` の詳細は、326 ページの「バッファ化されたストリーム」を参照してください。

パラメータ

`user`	(UNIX の場合のみ) CGI プログラムを実行するユーザの名前を指定します。
`group`	(UNIX の場合のみ) CGI プログラムを実行するグループの名前を指定します。
`chroot`	(UNIX の場合のみ) 実行を始める前に `chroot` するディレクトリを指定します。これは、`magnus.conf.` に定義された `chroot` に相当するものです。
`dir`	(UNIX の場合のみ) 実行を始める前に、`chroot` の後に `chdir` するディレクトリを指定します。

rlimit_as	(UNIX の場合のみ) CGI プログラムの最大アドレス空間 (バイト単位) を指定します。現在の (弱い) 制限値と最大の (強い) 制限値の両方を、コンマで区切って指定できます。最初に弱い制限値がリストされる必要があります。制限値を 1 つしか指定しない場合、両方の制限値がその値に設定されます。
rlimit_core	(UNIX の場合のみ) CGI プログラムの最大コアファイルサイズを指定します。値 0 はコアの書き込みを禁止します。現在の (弱い) 制限値と最大の (強い) 制限値の両方を、コンマで区切って指定できます。最初に弱い制限値がリストされる必要があります。制限値を 1 つしか指定しない場合、両方の制限値がその値に設定されます。
rlimit_nofile	(UNIX の場合のみ) CGI プログラムのファイル記述子の最大数を指定します。現在の (弱い) 制限値と最大の (強い) 制限値の両方を、コンマで区切って指定できます。最初に弱い制限値がリストされる必要があります。制限値を 1 つしか指定しない場合、両方の制限値がその値に設定されます。
nice	(UNIX の場合のみ) サーバに対する CGI プログラムの優先度を決定する増分値を受け入れます。通常、サーバは nice 値 0 で稼働し、nice 増分値は、0 (CGI プログラムはサーバと同じ優先度で実行する) と 19 (CGI プログラムはサーバよりかなり低い優先度で実行する) の間の値になります。nice 増分値 -1 を指定することによってCGI プログラムの優先度をサーバの優先度より大きくすることはできますが、これは推奨されません。
type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

次の例では、send-cgi パラメータとして server.xml ファイルに定義されている変数を使用します。変数の定義の詳細は、第 8 章「仮想サーバの構成ファイル」を参照してください。

```
<Object name="default">
...
NameTrans fn="pfx2dir" from="/cgi-bin"
dir="/home/foo.com/public_html/cgi-bin" name="cgi"
...
</Object>

<Object name="cgi">
ObjectType fn="force-type" type="magnus-internal/cgi"
Service fn="send-cgi" user="$user" group="$group" dir="$dir"
chroot="$chroot" nice="$nice"
</Object>
```

send-file

Service クラス指令で適用可能です。

send-file 関数は、要求されたファイルの内容をクライアントに送信します。これは、content-type、content-length、および last-modified ヘッダーを提供します。

ほとんどの要求は、次の指令を使ってこの関数で処理されます (通常この指令は、デフォルトとして機能するように、デフォルトオブジェクト内の Service クラス指令のリストの最後に来ます)。

```
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file"
```

この指令は、要求のメソッドが GET、HEAD、または POST で、タイプが magnus-internal/ で始まらない場合に起動されます。*~ というパターンは「一致しない」ことを意味するので、注意してください。パターンで使用できる文字のリストは、付録 C「ワイルドカードパターン」を参照してください。

パラメータ

nocache	(省略可能) サーバが静的ファイル要求への応答をキャッシュしないようにします。たとえば、特定のディレクトリ内のファイルがキャッシュされないように指定することができます。これは、頻繁にファイルが変わるディレクトリの場合に便利です。 このパラメータに割り当てる値は無視されます。このパラメータを使用したくない場合は、省略します。
type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type="*~magnus-internal/*" method="(GET|HEAD)"
fn="send-file"
```

次の例では、URL 接頭辞 /myurl で要求されたとき、サーバは静的ファイルを /export/mydir/ からキャッシュしません。

```
<Object name=default>
...
NameTrans fn="pfx2dir" from="/myurl" dir="/export/mydir",
name="myname"
...
Service method=(GET|HEAD|POST) type=*~magnus-internal/*
fn=send-file
...
</Object>
<Object name="myname">
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
nocache=""
</Object>
```

send-range

Service クラス指令で適用可能です。

クライアントが HTTP バイト範囲を指定することによってドキュメントの一部を要求すると、send-range 関数はその部分を返します。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn=send-range
```

send-shellcgi

Service クラス指令で適用可能です。

Windows NT の場合のみ： send-shellcgi 関数は、ファイルをシェル CGI プログラムとして実行し、その結果をクライアントに送信します。シェル CGI は、Windows NT のファイル関連付けセットを使って CGI アプリケーションを実行するための、サーバ構成です。シェル CGI プログラムの詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn=send-shellcgi
Service type=magnus-internal/cgi fn=send-shellcgi
```

send-wincgi

Service クラス指令で適用可能です。

Windows NT の場合のみ： send-wincgi 関数は、ファイルを Windows CGI プログラムとして実行し、その結果をクライアントに送信します。Windows CGI プログラムの詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通

query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn=send-wincgi
Service type=magnus-internal/cgi fn=send-wincgi
```

service-dump

Service クラス指令で適用可能です。

service-dump 関数は、収集されたパフォーマンスバケットデータに基づいて、パフォーマンスレポートを作成します (48 ページの「バケットパラメータ」を参照)。

レポートを読むには、次のブラウザをポイントします。

http://*server_id*:*port*/.perf

パラメータ

type	この関数の perf でなければなりません。
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
<Object name=default>
NameTrans fn="assign-name" from="/.perf" name="perf"
...
</Object>

<Object name=perf>
Service fn="service-dump"
</Object>
```

関連項目

`stats-xml`

shtml_send

`Service` クラス指令で適用可能です。

`shtml_send` 関数は、組み込みコマンドを走査して、HTML ドキュメントを構文解析します。これらのコマンドは、サーバからの情報の提供、ほかのファイルの内容の組み込み、または CGI プログラムの実行を行ないます。`shtml_send` 関数は、Shtml プラグイン (UNIX では `libShtml.so` 、Windows NT では `libShtml.dll`) が読み込まれているときにのみ利用可能です。サーバが構文解析する HTML コマンドについては、iPlanet Web Server の『プログラマーズガイド』を参照してください。

パラメータ

`ShtmlMaxDepth`	許可される include の入れ子の最大の深さ。デフォルト値は 10 です。
`addCgiInitVars`	(UNIX の場合のみ) 指定されていて `yes` (デフォルトは `no`) と等しい場合、`init-cgi` SAF に定義されている環境変数を、SHTML `exec` タグによって実行されるコマンドの環境に追加します。
`type`	省略可能、すべての Service クラス関数に共通
`method`	省略可能、すべての Service クラス関数に共通
`UseOutputStreamSize`	省略可能、すべての Service クラス関数に共通
`flushTimer`	省略可能、すべての Service クラス関数に共通
`ChunkedRequestBufferSize`	省略可能、すべての Service クラス関数に共通
`ChunkedRequestTimeout`	省略可能、すべての Service クラス関数に共通

query	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service type=magnus-internal/shtml_send method=(GET|HEAD)
fn=shtml_send
```

stats-xml

Service クラス指令で適用可能です。

stats-xml 関数は、パフォーマンスレポートを XML 形式で作成します。パフォーマンスバケットがすでに定義されていれば、それらがこのパフォーマンスレポートに取り込まれます。

ただし、magnus.conf 内の stats-init 関数を使ってこの関数を初期化してから、NameTrans 関数を使って要求を stats-xml 関数に送る必要があります。下記の例を参照してください。

レポートは次のように生成されます。

http://*server_id*:*port*/stats-xml/iwsstats.xml

関連する DTD ファイルは次のようになります。

http://*server_id*:*port*/stats-xml/iwsstats.dtd

iwsstats.xml ファイルの形式の詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

magnus.conf の場合：

```
Init fn="stats-init" update-interval="5" virtual-servers="2000"
profiling="yes"
```

obj.conf の場合：

```
<Object name="default">
...
NameTrans fn="assign-name" from="/stats-xml/*" name="stats-xml"
...
</Object>
...
<Object name="stats-xml">
Service fn="stats-xml"
</Object>
```

関連項目

service-dump, stats-init

upload-file

Service クラス指令で適用可能です。

upload-file 関数は、メソッドが PUT の要求をクライアントが送信したときに、ユーザが承認されておりサーバに必要なファイルシステム特権がある場合、新しいファイルをアップロードして保存します。

サーバでリモートファイル操作が使用可能なとき、obj.conf ファイルには、要求メソッドが PUT のときに upload-file を起動する Service クラス関数が含まれます。

パラメータ

type	省略可能、すべての Service クラス関数に共通
method	省略可能、すべての Service クラス関数に共通
query	省略可能、すべての Service クラス関数に共通
UseOutputStreamSize	省略可能、すべての Service クラス関数に共通
flushTimer	省略可能、すべての Service クラス関数に共通
ChunkedRequestBufferSize	省略可能、すべての Service クラス関数に共通
ChunkedRequestTimeout	省略可能、すべての Service クラス関数に共通
bucket	省略可能、すべての obj.conf 関数に共通

例

```
Service fn=upload-file
```

AddLog 段階

サーバが要求に応答した後、AddLog 指令は、トランザクションに関する情報の記録のために実行されます。

AddLog 指令が複数ある場合、すべてが実行されます。

この節では、次の AddLog クラス関数について詳しく説明します。

- common-log は、要求に関する情報を共通ログ形式で記録します。
- flex-log は、要求に関する情報を、柔軟で構成可能な形式で記録します。
- record-useragent は、クライアントの IP アドレスとユーザエージェントのヘッダーを記録します。

common-log

AddLog クラス指令で適用可能です。

この関数は、要求に固有のデータを共通ログ形式 (ほとんどの HTTP サーバで使用される) で記録します。ログアナライザが iPlanet Web Server の /extras/log_anly ディレクトリにあります。

共通ログは、init-clf 関数によってあらかじめ初期化されていなければなりません。ログのローテーションについての詳細は、flex-rotate-init を参照してください。

共通ログ形式用のフリーの統計ジェネレータも、多数あります。

パラメータ

name	(省略可能) ログファイルの名前を指定します。この名前は、magnus.conf 内の init-clf 関数へのパラメータとして指定されている必要があります。名前が指定されていない場合、エントリはグローバルログファイルに記録されます。
iponly	(省略可能) サーバに、DNS 名を調べてログをとるのではなく、リモートクライアントの IP アドレスのログをとるよう指示します。magnus.conf ファイルで DNS がオフの場合、これによってパフォーマンスが向上します。iponly の値は、存在しさえすれば特別な意味はなく、iponly=1 を使用することもできます。
bucket	省略可能、すべての obj.conf 関数に共通

例

```
# すべてのアクセスのログをグローバルログファイルに記録する
AddLog fn=common-log
# サブネット (198.93.5.*) の外部からのアクセスログを
# nonlocallog に記録する
<Client ip="*~198.93.5.*">
AddLog fn=common-log name=nonlocallog
</Client>
```

関連項目

`flex-init`, `init-clf`, `record-useragent`, `flex-log`, `flex-rotate-init`

flex-log

`AddLog` クラス指令で適用可能です。

この関数は、要求に固有のデータを、柔軟なログ形式で記録します。共通ログ形式で要求を記録することもできます。ログアナライザが、iPlanet Web Server の `/extras/flexanlg` ディレクトリにあります。

共通ログ形式用のフリーの統計ジェネレータも、多数あります。

このログ形式は、`flex-init` 関数呼び出しによって指定されます。ログのローテーションについての詳細は、`flex-rotate-init` を参照してください。

パラメータ

`name`	(省略可能) ログファイルの名前を指定します。この名前は、`magnus.conf` 内の `flex-init` 関数へのパラメータとして指定されている必要があります。名前が指定されていない場合、エントリはグローバルログファイルに記録されます。
`iponly`	(省略可能) サーバに、DNS 名を調べてログをとるのではなく、リモートクライアントの IP アドレスのログをとるよう指示します。`magnus.conf` ファイルで DNS がオフの場合、これによってパフォーマンスが向上します。`iponly` の値は存在しさえすれば特別な意味はなく、`iponly=1` を使用することもできます。
`bucket`	省略可能、すべての `obj.conf` 関数に共通

例

```
# すべてのアクセスログをグローバルファイルに記録する
AddLog fn=flex-log
# サブネット (198.93.5.*) の外部からのアクセスログを
# nonlocallog に記録する
<Client ip="*~198.93.5.*">
AddLog fn=flex-log name=nonlocallog
</Client>
```

関連項目

`flex-init`, `init-clf`, `common-log`, `record-useragent`, `flex-rotate-init`

record-useragent

`AddLog` クラス指令で適用可能です。

`record-useragent` 関数は、クライアントの IP アドレスと、その後に User-Agent HTTP ヘッダーを記録します。これは、このトランザクション用に使用された Netscape Navigator (またはほかのクライアント) のバージョンを示します。

ログのローテーションの詳細は、`flex-rotate-init` を参照してください。

パラメータ

`name`	(省略可能) ログファイルの名前を指定します。この名前は、`magnus.conf` 内の `init-clf` 関数へのパラメータとして指定されている必要があります。名前が指定されていない場合、エントリはグローバルログファイルに記録されます。
`bucket`	省略可能、すべての `obj.conf` 関数に共通

例

```
# クライアント IP アドレスと User-Agent を browserlog に記録する
AddLog fn=record-useragent name=browserlog
```

関連項目

`flex-init`, `init-clf`, `common-log`, `flex-log`, `flex-rotate-init`

Error 段階

サーバアプリケーション関数がエラーとなる場合、HTTP 応答の状態コードを設定し、値 `REQ_ABORTED` を返します。このような状況が発生すると、サーバは要求の処理を停止します。その代わりに、サーバは、HTTP 応答の状態コードまたはそれに関連する原因を示す句と一致する Error 指令を検索し、その指令の関数を実行します。一致する Error 指令を見つけられない場合、サーバはクライアントに応答状態コードを返します。

この節では、次の Error クラス関数について詳しく説明します。

- `send-error` は、HTML ファイルを特定の HTTP 応答状態の代わりにクライアントに送信します。
- `qos-error` は、エラーを発生させたサービス品質の制限と QOS 統計の値を示す、エラーページを返します。

send-error

`Error` クラス指令で適用可能です。

`send-error` 関数は、HTML ファイルを特定の HTTP 応答状態の代わりにクライアントに送信します。これによりサーバは、問題を記述するわかりやすいメッセージを提示することができます。HTML ページには、イメージや、サーバのホームページ、ほかのページへのリンクが含まれていることもあります。

パラメータ

`path`	クライアントに送信する HTML ファイルのファイルシステムの絶対パスを指定します。ファイルは、その名前や実際のタイプとは無関係に、`text/html` として送信されます。ファイルが存在しない場合、サーバは単純なデフォルトのエラーページを送信します。
`reason`	（省略可能） 原因を示す文字列 （「Unauthorized」や「Forbidden」など) のいずれかのテキストです。文字列は大文字と小文字で区別されません。

code	(省略可能) 401 や 407 などの、HTTP 応答の状態コードを表す 3 桁の数字です。 これは、HTTP 応答の状態コード、または HTTP の仕様に従った原因を示す文字列にすることができます。 次に、共通の HTTP 応答状態コードと原因を示す文字列のリストを示します。 • `401 Unauthorized.` • `403 Forbidden.` • `404 Not Found.` • `500 Server Error.`
bucket	省略可能、すべての `obj.conf` 関数に共通

例

```
Error fn=send-error code=401
path=/netscape/server4/docs/errors/401.html
```

qos-error

`Error` クラス指令で適用可能です。

`qos-error` 関数は、エラーを発生させたサービス品質の制限と QOS 統計の値を示す、エラーページを返します。

この SAF 用のコードは、第 6 章「カスタム SAF の例」の例の 1 つにあります。

詳細は、iPlanet Web Server の『管理者ガイド』のパフォーマンスについての章を参照してください。

パラメータ

code	(省略可能) 401 や 407 などの、HTTP 応答の状態コードを表す 3 桁の数字です。推奨される値は 503 です。 これは、HTTP 応答の状態コード、または HTTP の仕様に従った原因を示す文字列にすることができます。 次に、共通の HTTP 応答状態コードと原因を示す文字列のリストを示します。 • `401 Unauthorized`. • `403 Forbidden`. • `404 Not Found`. • `500 Server Error`.
bucket	省略可能、すべての `obj.conf` 関数に共通

例

```
Error fn=qos-error code=503
```

関連項目

`qos-handler`

第 4 章

カスタム SAF の作成

この章では、カスタムサーバアプリケーション関数 (Server Application Functions: SAF) を定義するユーザ独自の NSAPI プラグインを作成する方法について説明します。プラグインを作成すると、iPlanet Web Server に組み込まれている機能を変更したり拡張したりできます。たとえば、ユーザの承認を特別な方法で処理するように、またはデータベースの情報に基づいて動的な HTML ページを生成するように、サーバを変更できます。

この章には、次の節があります。

- SAF インタフェース
- SAF パラメータ
- 結果コード
- カスタム SAF の作成と使用
- NSAPI C 関数の概要
- 各指令用の SAF に要求される動作
- CGI から NSAPI への変換

カスタム SAF を作成する前に、第 1 章「サーバの動作の基本」に説明されている要求処理プロセスについて理解しておく必要があります。また、カスタム SAF を作成する前に、作成しようとしているカスタム SAF の機能と同じ機能が、すでに組み込まれている SAF にないかを確認します。

事前定義済みの Init SAF の一覧は、第 7 章「magnus.conf の構文と使用法」を参照してください。Init SAF 以外の事前定義済みの SAF の一覧は、第 3 章「事前定義済みの SAF および要求処理プロセス」を参照してください。

カスタム SAF を実現するための NSAPI ルーチンの完全な一覧は、第 5 章「NSAPI 関数のリファレンス」を参照してください。

SAF インタフェース

すべての (カスタムおよび組み込みの) SAF のインタフェースは、対応する要求処理ステップには関係なく、同じ C のインタフェースになります。SAF は、特定の要求 - 応答ステップ内での特定の目的のために設計された小さな関数です。SAF は、obj.conf ファイルで SAF を呼び出す指令から、サーバから、および前の SAF からパラメータを受け取ります。

次に SAF の C インタフェースを示します。

```
int function(pblock *pb, Session *sn, Request *rq);
```

次の節では、パラメータについて詳しく説明します。

SAF は、成功したかどうか、またどのように成功したかを示す結果コードを返します。サーバは、要求の処理をどのように進めるべきかを判断するために、各関数からの結果コードを使います。結果コードの詳細は、116 ページの「結果コード」を参照してください。

SAF パラメータ

この節では、SAF のパラメータについて詳しく説明します。SAF のパラメータを次に示します。

- pb (parameter block)- obj.conf ファイルに含まれている、SAF を呼び出す指令からのパラメータが入っている
- sn (session)- 1 つの TCP/IP セッションに関連する情報が入っている
- rq (request)- 現在の要求に関連する情報が入っている

pb (parameter block)

pb パラメータは、SAF を呼び出す指令が指定する値が含まれている pblock データ構造体へのポインタです。pblock データ構造体には、一連の名前 - 値ペアが含まれています。

たとえば、basic-nsca 関数を呼び出す指令は次のようになります。

```
AuthTrans fn=basic-ncsa auth-type=basic
dbm=/netscape/server4/userdb/rs
```

この場合、basic-ncsa に渡される pb パラメータには、auth-type=basic および dbm=/netscape/server4/userdb/rs に対応する名前 - 値ペアが含まれています。

NSAPI には、pblock データ構造体と共に作用するための一式の関数があります。たとえば、pblock_findval() は、pblock で指定された名前に対する値を返します。パラメータブロックと共に作用する最もよく使われる関数についての概要は、126 ページの「パラメータブロック操作ルーチン」を参照してください。

sn (session)

sn パラメータは、Session データ構造体へのポインタです。このパラメータには、1 つのセッションの間 (つまり、クライアントとサーバ間の TCP/IP 接続が開始されてから終了するまでの間) に関係する変数が含まれています。1 つのセッションの間、各要求内で呼び出される各 SAF には、同じ sn ポインタが渡されます。次に、このデータ構造体の最も重要なフィールドについて説明します。

Session データ構造体を操作するための NSAPI ルーチンについては、第 5 章「NSAPI 関数のリファレンス」を参照してください。

- sn->client

  クライアントの IP アドレス、DNS 名、または証明書などのクライアントについての情報が含まれている pblock へのポインタ。クライアントに DNS 名がないか、またはその DNS 名が見つからない場合は、-none に設定される

- sn->csd

  プラットフォームに依存しないクライアントソケット記述子。クライアントからの読み込みやクライアントへの書き込みのために、ルーチンにこの記述子を渡す

rq (request)

rq パラメータは、request データ構造体へのポインタです。このパラメータには、要求ヘッダー、URI、ローカルファイルシステムのパスなど現在の要求に関連する変数が含まれています。1 つの HTTP 要求の要求 - 応答プロセスでは、呼び出される各 SAF には、同じ request ポインタが渡されます。

次に、このデータ構造体の最も重要なフィールドについて説明します (Request データ構造体を操作するための NSAPI ルーチンについては、第 5 章「NSAPI 関数のリファレンス」を参照してください)。

- rq->vars

サーバの「作業用」変数が入っている pblock へのポインタ。このフィールドには、次の 3 つの pblock に明示的に示されていないものが入る。この pblock の内容は、特定の要求と SAF のタイプによって変わる。たとえば、AuthTrans SAF が auth-user パラメータを rq->vars に挿入し、次の PathCheck SAF でこのパラメータを使うことができる

- rq->reqpb

  HTTP 要求の要素が含まれている pblock へのポインタ。これには、HTTP のメソッド (GET、POST など)、URI、プロトコル (通常、HTTP/1.0)、照会文字列などがある。この pblock は、通常、要求 - 応答プロセスの間変化しない

- rq->headers

  クライアントからの HTTP 要求で受け取ったすべての要求ヘッダー (User-Agent、If-Modified-Since など) が含まれている pblock へのポインタ。要求ヘッダーについては、付録 E「HTTP (HyperText Transfer Protocol)」を参照。この pblock は、通常、要求 - 応答プロセスの間変化しない

- rq->srvhdrs

  クライアントへ HTTP 応答で送信される応答ヘッダー (Server、Date、Content-type、Content-length など) が含まれている pblock へのポインタ。応答ヘッダーについては、付録 E「HTTP (HyperText Transfer Protocol)」を参照

rq パラメータは、要求 - 応答プロセスで、情報の受け渡しをするための主要なメカニズムです。SAF への入力時には、rq には、前に実行された SAF によって挿入または変更された値がすべて含まれています。出力時には、rq には、その SAF が挿入した変更や追加情報が含まれています。一部の SAF は、プロセスの前のほうのステップによって提供される特定の情報の存在に依存します。たとえば、PathCheck SAF は、AuthTrans SAF によって前に挿入された rq->vars の値を取り出します。

結果コード

処理が完了すると、SAF は結果コードを返します。結果コードは、サーバが次に実行すべきことを示します。結果コードを次に示します。

- REQ_PROCEED

  SAF が目的を達成したことを示す。これは、一部の要求 - 応答ステップ (AuthTrans、NameTrans、Service、および Error) の場合、現在のステップのほかの SAF をスキップして、次の要求 - 応答ステップに進むようにサーバに指示する。ほかの要求 - 応答ステップ (PathCheck、ObjectType、および AddLog) の場合は、サーバは現在のステップの次の SAF に進む

- REQ_NOACTION

SAF が何も実行しなかったことを示す。サーバは、サーバの現在のステップの次の SAF から処理を続ける

- `REQ_ABORTED`

  エラーが発生し、エラーの原因を示すためにクライアントへ HTTP 応答を送信する必要があることを示す。`REQ_ABORTED` を返す SAF は、HTTP 応答状態コードも設定する必要がある。サーバが状態コードまたは説明句に一致する `Error` 指令を見つけた場合は、指定された SAF を実行する。そうでない場合は、サーバは状態コードと説明句から構成されるデフォルトの HTTP 応答、それにユーザのために状態コードと説明句を反映する短い HTML ページを送信する。次に、サーバは、最初の `AddLog` 指令に進む

- `REQ_EXIT`

  クライアントへの接続が失われたことを示す。これは、SAF がクライアントからの読み込みまたはクライアントへの書き込みに失敗すると、返される。次に、サーバは、最初の `AddLog` 指令に進む

カスタム SAF の作成と使用

カスタム SAF は、サーバによって読み込まれ呼び出される、共用ライブラリ内の関数です。次のステップに従って、カスタム SAF を作成します。

1. ソースコードを記述する

   NSAPI 関数を使って、ソースコードを記述します。各 SAF は、特定の指令用に作成されます。

2. コンパイルしリンクする

   ソースコードをコンパイルしリンクして、共用ライブラリ (`.so`、`.sl`、または `.dll`) ファイルを作成します。

3. SAF を読み込んで初期化する

   次の目的のために、`obj.conf` ファイルを編集して、SAF を読み込み初期化します。

   - カスタム SAF が含まれている共用ライブラリファイルを読み込む

   - 必要な場合は SAF を初期化する

4. サーバに SAF を呼び出すように指示する

   `obj.conf` を編集して、適切なときにカスタム SAF を呼び出すようにサーバに指示します。

5. サーバを再構成する

6. SAF をテストする

カスタム関数をトリガーする URL でブラウザからサーバにアクセスして、SAF をテストします。

以下の節では、これらのステップについてより詳しく説明します。

ソースコードを記述する

NSAPI 関数を使ってカスタム SAF を作成します。最もよく使われる一部の NSAPI 関数についての要約は、125 ページの「NSAPI C 関数の概要」の節を参照してください。第 5 章「NSAPI 関数のリファレンス」には、使用できるすべてのルーチンについての情報があります。

カスタム SAF の例については、サーバルートディレクトリの `nsapi/examples/`、および第 6 章「カスタム SAF の例」を参照してください。

すべての SAF の署名を次に示します。

```
int function(pblock *pb, Session *sn, Request *rq);
```

パラメータの詳細は、114 ページの「SAF パラメータ」の節を参照してください。

iPlanet Web Server は、マルチスレッド化された単一プロセスとして実行されます。UNIX プラットフォームでは、歴史的な理由から実際には 2 つのプロセス (親と子) が存在します。親プロセスは、初期化を一部実行し、子プロセスをフォークします。子プロセスは、さらに初期化を実行し、すべての HTTP 要求を処理します。

SAF を作成するときには、これらのことを念頭においてください。スレッド安全なコードを記述します。ブロック化は、パフォーマンスに影響を及ぼすことがあります。パラメータ付きの小さな関数を作成し、作成した関数を `obj.conf` に構成します。慎重にすべてのエラーを調べ、対処します。また、問題の原因を調べ修正できるように、エラーをログに記録します。

必要な場合は、新しい SAF が必要とする初期化タスクを実行する初期化関数を作成します。初期化関数の署名は、その他の SAF の署名と同じです。

```
int function(pblock *pb, Session *sn, Request *rq);
```

SAF は、パラメータから特定の種類の情報を入手できることを期待します。ほとんどの場合、パラメータブロック (pblock) データ構造体が、これらのパラメータの基本的な記憶メカニズムとなります。`pblock` には、データが名前 - 値ペアの集合として保持されます。`pblock` 構造体と共に作用するために最もよく使われる関数についての要約は、126 ページの「パラメータブロック操作ルーチン」を参照してください。

SAF を定義するときには、その SAF がどの指令用に作成されているかを特に示す必要はありません。ただし、各 SAF は、特定の指令 (`AuthTrans`、`Service` など) 用に作成する必要があります。各指令は、関連する SAF が特定のことを実行することを期待し、対応する指令の期待に従うようにカスタム SAF を作成する必要があります。各指令がその SAF に期待することについての詳細は、130 ページの「各指令用の SAF に要求される動作」を参照してください。

コンパイルしリンクする

ターゲットプラットフォームのネイティブコンパイラでコードをコンパイルし、リンクします。UNIX の場合は、`gmake` コマンドを使います。Windows NT の場合は、`nmake` コマンドを使います。Windows NT の場合は、Microsoft Visual C++ 6.0 以降を使います。サーバのバイナリからアクセスするためのすべてのグローバル変数および関数を指定するインポートリストを持っている必要があります。使用しているプラットフォームに適切なコンパイラおよびリンカーのフラグを使います。*server_root*`/plugins/nsapi/examples` ディレクトリにある Makefile の例を参照してください。

コンパイルおよびリンクについては、次のガイドラインに従います。

include ディレクトリと nsapi.h ファイル

server_root`/plugins/include` (UNIX) ディレクトリ、または *server_root*`\plugins\include` (Windows NT) ディレクトリを Makefile に追加して、`nsapi.h` ファイルを組み込みます。

ライブラリ

リンカーコマンドにライブラリディレクトリの *server_root*`/bin/https/lib` (UNIX)、または *server_root*`\bin\https\bin` (Windows NT) を追加します。

表 4-1 には、リンクする必要のあるライブラリを示します。

表 4-1 ライブラリ

プラットフォーム	ライブラリ
Windows NT	`ns-httpd40.dll` (標準の Windows ライブラリに加えて)
HPUX	`libns-httpd40.sl`
その他のすべての UNIX のプラットフォーム	`libns-httpd40.so`

共用オブジェクトの生成のためのリンカーのコマンドとオプション

共用ライブラリを生成するには、表 4-2 に示すコマンドやオプションを使います。

表 4-2　リンカーのコマンドおよびオプション

プラットフォーム	オプション
Solaris Sparc	`ld -G` または `cc -G`
Windows NT	`link -LD`
HPUX	`cc +Z -b -Wl,+s -Wl,-B,symbolic`
AIX	`cc -p 0 -berok -blibpath:$(LD_RPATH)`
Compaq	`cc -shared`
Linux	`gcc -shared`
IRIX	`cc -shared`

その他のリンカーフラグ

表 4-3 に示すリンカーフラグを使って、実行時にシンボルを解決するためにどのディレクトリで共用オブジェクトを検索するかを指定します。

表 4-3　リンカーフラグ

プラットフォーム	フラグ
Solaris Sparc	`-R` *dir*`:`*dir*
Windows NT	(フラグではなく、システム PATH 変数に `ns-httpd40.dll` ファイルを指定する必要がある)
HPUX	`-Wl,+b,`*dir*`,`*dir*
AIX	`-blibpath:`*dir*`:`*dir*
Compaq	`-rpath` *dir*`:`*dir*
Linux	`-Wl,-rpath,`*dir*`:`*dir*
IRIX	`-Wl,-rpath,`*dir*`:`*dir*

UNIX では、ライブラリの検索パスを、サーバを起動するときに設定する必要がある LD_LIBRARY_PATH 環境変数を使って設定することもできます。

コンパイラフラグ

表 4-4 には、ソースコードのコンパイルに使う必要のあるフラグと Defines 定義を示します。

表 4-4 コンパイラのフラグと Defines 定義

プラットフォーム	フラグ /Defines
Solaris Sparc	`-DXP_UNIX -D_REENTRANT -KPIC -DSOLARIS`
Windows NT	`-DXP_WIN32 -DWIN32 /MD`
HP-UX	`-DXP_UNIX -D_REENTRANT -DHPUX`
AIX	`-DXP_UNIX -D_REENTRANT -DAIX $(DEBUG)`
Compaq	`-DXP_UNIX -KPIC`
Linux	`-DLINUX -D_REENTRANT -fPIC`
IRIX	`-o32 -exceptions -DXP_UNIX -KPIC`
すべてのプラットフォーム	`-MCC_HTTPD -NET_SSL`

表 4-5 には、使用できる任意 (省略可能) のフラグと Defines 定義を示します。

表 4-5 フラグおよび Defines 定義 (省略可能)

フラグ /Defines 定義	プラットフォーム	説明
`-DSPAPI20`	すべて	プロキシユーティリティ関数の include ファイルの `putil.h` に必要

AIX での 3.x プラグインのコンパイル

AIX の場合のみ、サーバの 3.*x* バージョン用にビルドされたプラグインは、4.*x* および 6.*x* バージョンと動作させるために再リンクする必要があります。

server_root`/plugins/nsapi/examples/` ディレクトリにあるファイルで必要なものを次に示します。

- `Makefile` ファイルには、古い `-bM:SRE -berok -brtl -bnoentry` オプションの代わりに `-G` オプションが含まれています。

- `relink_36plugin` スクリプトは、サーバの 3.*x* バージョン用にビルドされたプラグインを変更して、4.*x* および 6.*x* バージョンで動作するようにします。このスクリプトのコメントが、その用途を示しています。

iPlanet Web Server 4.*x* および 6.*x* バージョンは、実行時リンクをネイティブにサポートする AIX 4.2 にビルドされています。このため、`ns-httpd` 主実行可能ファイル内のシンボルを参照する NSAPI プラグインは、シンボルを実行時に解決するように指定する、`-G` オプションを使ってビルドする必要があります。

ただし、旧バージョンの Netscape Enterprise Server は、ネイティブ実行時リンクをサポートしない AIX 4.1 にビルドされていました。Enterprise Server には、プラグインを可能にするために特別な補足ソフトウェア (IBM AIX 開発部門から Netscape に提供された) がありました。プラグインをビルドするために、特別な実行時リンク指令は必要ありませんでした。このため、AIX で旧バージョンのサーバ用にビルドされたプラグインは、iPlanet Web Server 4.*x* および 6.*x* バージョンと共には、同じように機能しません。

ただし、それらのプラグインを iPlanet Web Server 4.*x* および 6.*x* バージョンで機能するように簡単に再リンクすることができます。relink_36plugin スクリプトは、既存のプラグインを再リンクします。このスクリプトには、既存のプラグインだけが必要です。元のソースファイルや .o ファイルは必要ありません。より具体的なコメントは、スクリプト内に記述されています。4.2 以降のすべてのバージョンの AIX は実行時リンクをネイティブにサポートするので、バージョン 4.*x* 以降の iPlanet Web Server 用のプラグインは再リンクする必要はありません。

SAF を読み込んで初期化する

iPlanet Web Server に読み込まれるカスタム SAF が含まれている各共用ライブラリ (プラグイン) に対して、`load-modules` SAF を呼び出す `Init` 指令を 1 つ `magnus.conf` に追加します。

`load-modules` を呼び出す指令の構文を次に示します。

`Init fn=load-modules shlib=`[*path*]*sharedlibname* `funcs="`*SAF1,...,SAFn*"

- `shlib` は、共用ライブラリ (プラグイン) へのローカルファイルシステムのパスである
- `funcs` は、共用ライブラリから読み込まれる、コンマで区切られた関数のリストである。関数名では、大文字、小文字が区別される。関数名では、下線 (_) の代わりにダッシュ (-) を使うことができる。関数名リストには、空白文字を入れてはならない

  新しい SAF を初期化する必要がある場合は、初期化関数を `funcs` リストに入れるようにする。

たとえば、2 つの SAF do_small_anim() と do_big_anim() を定義し、また初期化関数の init_my_animations() を定義する共用ライブラリの animations.so を作成した場合は、次の指令を追加してこのプラグインを読み込みます。

```
Init fn=load-modules shlib=animations.so
funcs="do_small_anim,do_big_anim,init_my_animations"
```

必要な場合は、新たに読み込まれたプラグインのために初期化関数を呼び出す Init 指令も追加します。たとえば、maxAnimLoop パラメータの操作を実行する関数 init_my_new_SAF() を定義した場合は、次のような指令を magnus.conf に追加します。

```
Init fn=init_my_new_SAF maxAnimLoop=5
```

サーバに SAF を呼び出すように指示する

次に、指令を obj.conf に追加して、適切なときに各カスタム SAF を呼び出すようにサーバに指示します。この指令の構文を次に示します。

Directive fn=*function-name* [*name1*="*value1*"]...[*nameN*="*valueN*"]

- *Directive* は、AuthTrans、Service、などのサーバの指令のいずれかである
- *function-name* は、実行する SAF の名前である
- *nameN*="*valueN*" は、SAF に渡されるパラメータの名前と値である

新しい SAF が実行する内容によって、1 つだけ指令を obj.conf に追加すればよい場合と、新しい SAF を呼び出すのに必要な指示をすべて含んだ、複数の指令を追加する必要がある場合があります。

たとえば、新しい AuthTrans または PathCheck SAF を定義する場合は、デフォルトのオブジェクトに適切な指令を追加するだけですみます。しかし、要求されたリソースが特定のディレクトリにあるか、またはリソースのファイル拡張子の種類が新しいときにだけ新しい Service SAF を呼び出すように定義する場合は、さらに必要なステップがあります。

要求されたリソースのファイル拡張子が新しい種類の場合にだけ新しい Service SAF を呼び出す場合は、ObjectType 段階で type の値が適切に設定されるように MIME タイプファイルにエントリを追加する必要がある場合があります。そのようにすると、Service 指令を、必要な type 値を指定するデフォルトのオブジェクトに追加できます。

要求されたリソースが特定のディレクトリにある場合にだけ新しい Service SAF を呼び出す場合は、別のオブジェクトに一致する name または ppath 値を生成する NameTrans 指令を定義する必要がある場合があり、そのようにすると新しいオブジェクトで新しい Service 関数を呼び出すことができます。

たとえば、ユーザのプラグインが、パラメータとして speed が指定される 2 つの新しい SAF do_small_anim() と do_big_anim() を定義するとします。これらの関数は、アニメーションを実行します。小さなアニメーションとして扱われるファイルはすべて、ディレクトリ D:/Netscape/server4/docs/animations/small にあり、フルスクリーンアニメーションとして扱われるファイルはすべて、ディレクトリ D:/Netscape/server4/docs/animations/fullscreen にあります。

クライアントが小さなまたはフルスクリーンのアニメーションのどちらかに対する要求を送信するたびに新しいアニメーション関数が呼び出されるようにするには、NameTrans 指令をデフォルトのオブジェクトに追加して適切な URL を対応するパス名に変換し、また要求に名前を割り当てます。

```
NameTrans fn=pfx2dir from="/animations/small"
dir="D:/Netscape/server4/docs/animations/small" name="small_anim"
NameTrans fn=pfx2dir from="/animations/fullscreen"
dir="D:/Netscape/server4/docs/animations/fullscreen"
name="fullscreen_anim"
```

また、アニメーションを実行し、speed パラメータを指定する Service 指令が含まれているオブジェクトを定義する必要があります。

```
<Object name="small_anim">
Service fn=do_small_anim speed=40
</Object>
<Object name="fullscreen_anim">
Service fn=do_big_anim speed=20
</Object>
```

サーバを再構成する

obj.conf の変更後、サーバを再構成する必要があります。詳細は、20 ページの「動的再構成」を参照してください。

SAF をテストする

カスタム関数をトリガーする URL でブラウザからサーバにアクセスして、SAF をテストします。たとえば、`http://`*server-name*`/animations/small` にあるリソースへの要求によって新しい SAF がトリガーされる場合は、その URI で始まる有効なリソースの要求を試みます。

サーバが確実にアクセスされるように、ブラウザのキャッシュ機能を無効にする必要があります。Navigator では、シフトキーを押しながら「再読み込み (Reload)」ボタンをクリックして、キャッシュが使われないようにします。(シフト - 再読み込み方法では、イメージがすでにキャッシュにある場合は、クライアントがソースから常にイメージを取得するようになるとは限らないことに注意してください。)

`cache-init` SAF を使ってサーバのキャッシュを無効にすることもできます。

アクセスログとエラーログを調べると、デバッグに役立ちます。

NSAPI C 関数の概要

NSAPI には、SAF を実現するために使う一式の C 言語の関数があります。それらの C の関数には、いくつかの用途があります。また、iPlanet Server のオペレーティングシステムやハードウェアプラットフォームに依存しない、プラットフォームの独立を可能にします。さらに、パフォーマンスも向上させます。それらの C の関数は、スレッド安全であり、これは SAF の必要条件です。また、メモリーリークを防止します。SAF の実現に必要な機能も提供します。新しい SAF を定義するときには、必ずこれらの NSAPI ルーチンを使う必要があります。

この節では、使用できる関数のカテゴリとよく使われるルーチンの一部の概要を示します。すべての公開ルーチンの詳細は、第 5 章「NSAPI 関数のリファレンス」を参照してください。

NSAPI 関数の主なカテゴリを次に示します。

- パラメータブロック操作ルーチン
- Service SAF 用のプロトコルユーティリティ
- メモリーの管理
- ファイル入出力
- ネットワーク入出力
- スレッド
- ユーティリティ
- 仮想サーバ

パラメータブロック操作ルーチン

pblock データ構造体内のエントリの検索、追加、および削除のために、パラメータブロック操作関数が提供するルーチンには、次のものがあります。

- pblock_findval は、pblock に指定された名前の値を返す
- pblock_nvinsert は、pblock に新しい名前 - 値エントリを追加する
- pblock_remove は、pblock から pblock エントリを名前で削除する。エントリは廃棄されない。param_free を使って、エントリが使っていたメモリーを解放する
- param_free は、指定された pblock エントリのメモリーを解放する
- pblock_pblock2str は、"*name=value name=value*" の形式で pblock からすべての名前 - 値ペアが含まれる新しい文字列を作成する。これは、デバッグに役立つ関数である

Service SAF 用のプロトコルユーティリティ

プロトコルユーティリティは、Service SAF を実現するために必要な機能を提供します。

- request_header は、必要に応じてヘッダーを読み、指定された要求ヘッダー名に対する値を返す。この関数は、ブラウザヘッダーの pblock からエントリを要求するときに使う必要がある (rq->headers)
- protocol_status は、HTTP 応答状態コードと説明句を設定する
- protocol_start_response は、HTTP 応答とすべての HTTP ヘッダーをブラウザに送信する

メモリーの管理

メモリー管理ルーチンは、標準メモリー管理ルーチンの、高速かつプラットフォームに依存しないバージョンです。また、各要求に一時的なメモリー (「プールされた」メモリーと呼ばれる) から割り当て、各要求の後にプール全体を廃棄することにより、メモリーのリークを防止します。メモリー管理ルーチンは、固定メモリーを使うための、標準のメモリールーチンのラッパーです。デバッグのために、プールされたメモリーを無効にする方法については、第 7 章「magnus.conf の構文と使用法」の組み込み SAF の pool-init を参照してください。

- MALLOC
- FREE
- STRDUP

- REALLOC
- CALLOC
- PERM_MALLOC
- PERM_FREE
- PERM_STRDUP
- PERM_REALLOC
- PERM_CALLOC

ファイル入出力

ファイル入出力関数は、プラットフォームに依存しない、スレッド安全なファイル入出力ルーチンを提供します。

- system_fopenRO は、読み取り専用アクセスのためにファイルを開く
- system_fopenRW は、読み込み - 書き込みアクセスのためにファイルを開き、必要な場合はファイルを作成する
- system_fopenWA は、書き込み - 追加アクセスのためにファイルを開き、必要な場合はファイルを作成する
- system_fclose は、ファイルを閉じる
- system_fread は、ファイルから読み込む
- system_fwrite は、ファイルへ書き込む
- system_fwrite_atomic は、指定されたファイルに書き込む前に、そのファイルをロックする。このようにすると、複数のスレッドが同時に書き込むことによる干渉を防止できる

ネットワーク入出力

ネットワーク入出力関数は、プラットフォームに依存しない、スレッド安全なネットワーク入出力ルーチンを提供します。ネットワーク入出力ルーチンは、SSL (Secure Socket Layer) が有効な場合、SSL で動作します。

- netbuf_grab は、ネットワークのバッファのソケットからネットワークのバッファに読み込む
- netbuf_getc は、ネットワークのバッファから文字を取得する
- net_write は、ネットワークのソケットへ書き込む

スレッド

スレッド関数には、サーバのスレッドと互換性のあるユーザ独自のスレッドを作成するための関数などがあります。クリティカルセクションおよび条件変数用のルーチンもあります。

- `systhread_start` は、新しいスレッドを作成する
- `systhread_sleep` は、指定された期間スレッドを休眠させる
- `crit_init` は、新しいクリティカルセクション変数を作成する
- `crit_enter` は、クリティカルセクションの所有権を得る
- `crit_exit` は、クリティカルセクションの所有権を明け渡す
- `crit_terminate` は、新しいクリティカルセクション変数を廃棄する
- `condvar_init` は、新しい条件変数を作成する
- `condvar_notify` は、条件変数でブロックされたスレッドを呼び起こす
- `condvar_wait` は、条件変数でブロックする
- `condvar_terminate` は、条件変数を廃棄する
- `prepare_nsapi_thread` は、サーバによって作成されていないスレッドが、サーバにより作成されたスレッドのように動作することを可能にする

ユーティリティ

ユーティリティ関数には、多くの (文字列操作などの) 標準ライブラリ関数のプラットフォームに依存しない、スレッド安全なバージョンや、NSAPI に役立つ新しいユーティリティなどがあります。

- `daemon_atrestart` (UNIX のみ) は、サーバに再起動シグナル (HUP) が送信されるときに、またはシャツトダウン時に、呼びされるユーザ関数を登録する
- `util_getline` は、バッファから次の行 (LF または CRLF まで) を取得する
- `util_hostname` は、絶対パスによるドメイン名としてローカルホスト名を取得する
- `util_later_than` は、2 つの日付を比較する
- `util_sprintf` は、標準のライブラリルーチンの `sprintf()` と同じである
- `util_strftime` は、標準のライブラリルーチンの `strftime()` と同じである
- `util_uri_escape` は、文字列中の特殊文字を URI エスケープ処理された形式に変換する

- util_uri_unescape は、文字列中の URI エスケープ処理された文字を特殊文字に戻す

注	NSAPI 関数は、null が文字列の終わりであるとみなすので、文字列中に null を埋め込むことはできません。したがって、NSAPI プラグインを介して Unicode でエンコードされた内容を渡してもうまくいきません。

仮想サーバ

仮想サーバ関数は、仮想サーバについての情報を取得するためのルーチンを提供します。

- request_get_vs は、要求の送り先の仮想サーバを検索する
- vs_alloc_slot は、特定の仮想サーバに固有のデータへのポインタを記憶するための新しいスロットを割り当てる
- vs_get_data は、指定された仮想サーバとスロットのデータへのポインタの値を検索する
- vs_get_default_httpd_object は、仮想サーバの仮想サーバクラス構成からデフォルト (またはルート) のオブジェクトへのポインタを取得する
- vs_get_doc_root は、仮想サーバのドキュメントルートを検索する
- vs_get_httpd_objset は、指定された仮想サーバの仮想サーバクラス構成へのポインタを取得する
- vs_get_id は、仮想サーバの ID を検索する
- vs_get_mime_type は、指定された URI の Content-type: ヘッダーに返される MIME タイプを判別する
- vs_lookup_config_var は、指定された仮想サーバの構成変数の値を検索する
- vs_register_cb は、仮想サーバの初期化イベントおよび破棄イベントの通知を受ける関数をプラグインが登録できるようにする
- vs_set_data は、指定された仮想サーバおよびスロットのデータへのポインタの値を設定する
- vs_translate_uri は、特定の仮想サーバに対する要求の一部であるかのように URI を変換する

各指令用の SAF に要求される動作

新しい SAF を作成するときには、その SAF が要求処理プロセスのどの段階で呼び出されるかに応じて、特定の動作をするように定義する必要があります。たとえば、Init 段階で呼び出す SAF は、Service 段階で呼び出す SAF とは異なる要件を満たす必要があります。

rq パラメータは、要求 - 応答プロセスで、情報の受け渡しをするための主要なメカニズムです。SAF への入力時には、rq には、前に実行された SAF によって挿入または変更された値がすべて含まれています。出力時には、rq には、SAF が挿入した変更や追加情報が含まれています。一部の SAF は、要求 - 応答プロセスの前の方のステップによって提供される特定の情報の存在に依存します。たとえば、PathCheck SAF は、前に AuthTrans SAF によって挿入された rq->vars の値を取り出します。

この節では、要求処理プロセスの各段階で使われる SAF に期待される動作の概要を示します。

- Init SAF
- AuthTrans SAF
- NameTrans SAF
- PathCheck SAF
- ObjectType SAF
- Service SAF
- Error SAF
- AddLog SAF

Init SAF

- 目的：起動時に初期化する
- サーバの起動時と再起動時に呼び出される
- rq と sn は、NULL である
- ファイルやグローバル変数などの共用リソースを初期化する
- クリーンアップのために daemon_atrestart() でコールバック関数を登録できる
- エラーが発生した場合は、error パラメータを pb に挿入してエラーを記述し、REQ_ABORTED を返す
- 成功した場合は、REQ_PROCEED を返す

AuthTrans SAF

- 目的：承認情報を検証する。現在、HTTP/1.0 仕様では基本承認だけが定義されている
- rq->headers で、承認タイプと uu エンコードされたユーザおよびパスワード情報が含まれている Authorization ヘッダーを検査する。ヘッダーが送られてこなかった場合は、REQ_NOACTION を返す
- ヘッダーが存在する場合は、ユーザとパスワードの正当性を確認する
- 正当である場合は、PathCheck SAF が後で使用できるように、rq->vars に auth-type パラメータ、それに加えて auth-user または auth-group、あるいはその両方のパラメータを作成する
- ユーザの承認に成功した場合は、REQ_PROCEED を返し、失敗した場合は REQ_NOACTION を返す

NameTrans SAF

- 目的：論理 URI を物理パスに変換する
- 論理パス (rq->vars 内の ppath) に対して操作を実行して、論理パスをローカルファイルシステムの絶対パスに変換する
- rq->vars 内の ppath にローカルファイルシステムの絶対パスが含まれている場合は、REQ_PROCEED を返し、そうでない場合は、REQ_NOACTION を返す
- クライアントを別のサイトに転送するには、rq->vars 内の ppath を /URL に変更する。絶対パスによる URL (たとえば、http://home.netscape.com/) で url を rq->vars に追加する。REQ_PROCEED を返す

PathCheck SAF

- 目的：パスの有効性とユーザのアクセス権を検査する
- rq->vars 内の auth-type と、auth-user、および / または auth-group を検査する
- ユーザ (またはグループ) がこの領域 (rq->vars 内の ppath) へのアクセスを許可されている場合は、REQ_PROCEED を返す
- 許可されていない場合は、Basic; Realm=\"Our private area\" などの値で、WWW-Authenticate を rq->srvhdrs に挿入する。HTTP 応答ステータスを PROTOCOL_UNAUTHORIZED に設定するために protocol_status() を呼び出す。REQ_ABORTED を返す

ObjectType SAF

- 目的：データの内容のタイプを判別する
- rq->srvhdrs 内の content-type がすでに存在する場合は、REQ_NOACTION を返す
- MIME タイプを判別し、rq->srvhdrs に content-type を作成する
- content-type が作成された場合は、REQ_PROCEED を返し、そうでない場合は REQ_NOACTION を返す

Service SAF

- 目的：応答を生成し、クライアントに送信する
- Service SAF は、obj.conf の指令に指定されたオプションのパラメータの type、method、および query のそれぞれが要求に一致する場合にだけ呼び出される
- rq->srvhdrs から既存の content-type を削除する。rq->srvhdrs に正しい content-type を挿入する
- rq->srvhdrs にその他のヘッダーを作成する
- HTTP 応答ステータスを設定するために protocol_status を呼び出す
- HTTP 応答およびヘッダーを送信するために protocol_start_response を呼び出す
- net_write を使ってデータを生成しクライアントへ送信する
- 成功した場合は REQ_PROCEED を、書き込みエラーの場合は REQ_EXIT を、その他の失敗の場合は REQ_ABORTED を返す

Error SAF

- 目的：HTTP ステータスエラー状態に対応する
- Error SAF は、obj.conf の指令に指定されたオプションのパラメータの code と reason のそれぞれが現在のエラーに一致する場合にだけ呼び出される
- Error SAF は、Service SAF と同じ動作をするが、HTTP ステータスエラー状態への対応としてのみそのように動作する

AddLog SAF

- 目的：トランザクションをログファイルに記録する
- AddLog SAF は、pb、sn、または rq で使用可能なデータを使ってこのトランザクションを記録できる
- REQ_PROCEED を返す

CGI から NSAPI への変換

NSAPI を使って、CGI 変数を SAF に変換する必要がある場合があります。NSAPI には CGI 環境変数は利用できないので、それらの変数を NSAPI パラメータブロックから取得します。次の表に、各 CGI 環境変数を NSAPI で入手する方法を示します。

ユーザのコードは、NSAPI ではスレッド安全である必要があることを念頭においてください。スレッド安全な NSAPI 関数を使う必要があります。また、処理速度と、プラットフォームに依存しないために、NSAPI のメモリー管理およびその他のルーチンを使う必要があります。

表 4-6

CGI getenv()	NSAPI
AUTH_TYPE	pblock_findval("auth-type", rq->vars);
AUTH_USER	pblock_findval("auth-user", rq->vars);
CONTENT_LENGTH	pblock_findval("content-length", rq->headers);
CONTENT_TYPE	pblock_findval("content-type", rq->headers);
GATEWAY_INTERFACE	"CGI/1.1"
HTTP_*	pblock_findval("*", rq->headers); (* は小文字であり、ダッシュで下線を置き換える)
PATH_INFO	pblock_findval("path-info", rq->vars);
PATH_TRANSLATED	pblock_findval("path-translated", rq->vars);
QUERY_STRING	pblock_findval("query", rq->reqpb); (GET のみ。POST は、照会文字列 (query string) を本体のデータに含める)
REMOTE_ADDR	pblock_findval("ip", sn->client);
REMOTE_HOST	session_dns(sn) ? session_dns(sn) : pblock_findval("ip", sn->client);

表 4-6

CGI getenv()	NSAPI
REMOTE_IDENT	pblock_findval("from", rq->headers); (通常は使用できない)
REMOTE_USER	pblock_findval("auth-user", rq->vars);
REQUEST_METHOD	pblock_findval("method", req->reqpb);
SCRIPT_NAME	pblock_findval("uri", rq->reqpb);
SERVER_NAME	char *util_hostname();
SERVER_PORT	conf_getglobals()->Vport; (文字列として)
SERVER_PROTOCOL	pblock_findval("protocol", rq->reqpb);
SERVER_SOFTWARE	MAGNUS_VERSION_STRING
Netscape 固有 :	
CLIENT_CERT	pblock_findval("auth-cert", rq->vars)
HOST	char *session_maxdns(sn); (null である場合もある)
HTTPS	security_active ? "ON" : "OFF";
HTTPS_KEYSIZE	pblock_findval("keysize", sn->client);
HTTPS_SECRETKEYSIZE	pblock_findval("secret-keysize", sn->client);
QUERY	pblock_findval("query", rq->reqpb); (GET のみ。POST は照会文字列 (query string) をエンティティ本体のデータに含める)
SERVER_URL	http_uri2url_dynamic("","", sn, rq);

第 5 章

NSAPI 関数のリファレンス

この章では、NSAPI (Netscape Server Applications Programming Interface) の公開されているすべての C 言語の関数およびマクロをアルファベット順に示します。これらの関数は、独自のサーバアプリケーション関数 (Server Application Functions: SAF) を作成するときに使用することができます。

事前定義された Init SAF のリストは、第 7 章「magnus.conf の構文と使用法」を参照してください。Init SAF 以外の事前定義された SAF のリストは、第 3 章「事前定義済みの SAF および要求処理プロセス」を参照してください。

関数ごとに、名称、構文、パラメータ、戻り値、関数の用途の説明を提供し、また関数の使用例や関連する関数を示している場合もあります。

データ構造体については、付録 A「データ構造体のリファレンス」、また iPlanet Web Server 6.0 のビルドの `include` ディレクトリ内の `nsapi.h` ヘッダーファイルを参照してください。

NSAPI 関数 (アルファベット順)

アルファベット順の関数名のリストは、付録 G「NSAPI 関数とマクロのアルファベット順リスト」を参照してください。

C　D　F　L　M　N　P　R　S　U　V

C

CALLOC

CALLOC マクロは、C ライブラリルーチンの calloc の、プラットフォームに依存しないものです。CALLOC は、要求のメモリープールから num*size バイトを割り当てます。プールされたメモリーが (組み込み SAF の pool-init を使って) 構成ファイルで無効に設定されている場合は、PERM_CALLOC および CALLOC はシステムヒープからメモリーを取得します。

構文

```
void *CALLOC(int num, int size)
```

戻り値

メモリーのブロックへの void 型のポインタを返します。

パラメータ

int num は、割り当てる要素の個数です。

int size は、各要素のバイト単位のサイズです。

例

```
/* 100 個の char 型のポインタの配列に領域を割り当てる */
char *name;
name = (char *) CALLOC(100, sizeof(char *));
```

関連項目

FREE、REALLOC、STRDUP、PERM_MALLOC、PERM_FREE、PERM_REALLOC、PERM_STRDUP

cinfo_find

cinfo_find() 関数は、MIME タイプ情報を使って、URI (Universal Resource Identifier) またはローカルファイル名の拡張子に基づいてタイプ、コード化方法、または言語を判別します。この情報を使って、サーバから受け取るデータの content-type、content-encoding、および content-language を示すヘッダー (rq->srvhdrs) をクライアントへ送ります。

使われる名前は、最後のスラッシュ (/) 後のすべての文字、またはスラッシュがない場合は文字列全体になります。ファイル名拡張子には、大文字 - 小文字の区別がありません。名前には、タイプ、コード化方法、または言語を示すために、ピリオド (.) で区切られた複数の拡張子を含むことができます。たとえば、URI の a/b/filename.jp.txt.zip は、日本語で、タイプが text/plain であり、zip でコード化されたファイルを表します。

構文

```
cinfo *cinfo_find(char *uri);
```

戻り値

内容情報が見つかった場合は、新しく割り当てられた cinfo 構造体へのポインタを返し、内容が見つからない場合は NULL を返します。

割り当てられ、返された cinfo 構造体には、見つかった場合は、content-type、content-encoding、および content-language へのポインタが含まれています。それぞれ、types データベース内の static データへのポインタ情報であり、見つからない場合は NULL になります。これらのポインタを解放してはなりません。cinfo 構造体については、使い終わったら解放する必要があります。

パラメータ

char *uri は、URI またはローカルファイルの名前です。複数のファイル名拡張子は、ピリオド (.) で区切る必要があります。

condvar_init

condvar_init 関数は、指定されたクリティカルセクション変数に関連する新しい条件変数を初期化して返す、クリティカルセクション関数です。条件変数を使って、実行される 2 つのスレッドが干渉し合うのを防止できます。

構文

```
CONDVAR condvar_init(CRITICAL id);
```

戻り値

新しく割り当てられた条件変数 (CONDVAR) を返します。

パラメータ

CRITICAL id は、クリティカルセクション変数です。

関連項目

condvar_notify、condvar_terminate、condvar_wait、crit_init、crit_enter、crit_exit、crit_terminate

condvar_notify

condvar_notify 関数は、指定されたクリティカルセクション変数でブロックされたスレッドを呼び起こす、クリティカルセクション関数です。特定のクリティカルセクションの実行のスレッドを呼び起こすには、この関数を使います。まず、crit_enter を使って、クリティカルセクションの所有権を得ます。次に、返されたクリティカルセクション変数を使って condvar_notify を呼び出して、スレッドを呼び起こします。最後に、condvar_notify から復帰したら、crit_exit を呼び出してクリティカルセクションの所有権を返します。

構文

```
void condvar_notify(CONDVAR cv);
```

戻り値

void を返します。

パラメータ

CONDVAR cv は、条件変数です。

関連項目

condvar_init、condvar_terminate、condvar_wait、crit_init、crit_enter、crit_exit、crit_terminate

condvar_terminate

condvar_terminate 関数は、条件変数を解放するクリティカルセクション関数です。前に割り当てた条件変数を解放するには、この関数を使います。

警告

使用中の条件変数を終了させると、予期しない結果をまねくことがあります。

構文

```
void condvar_terminate(CONDVAR cv);
```

戻り値

void を返します。

パラメータ

CONDVAR cv は、条件変数です。

関連項目

condvar_init、condvar_notify、condvar_wait、crit_init、crit_enter、crit_exit、crit_terminate

condvar_wait

指定された条件変数でブロックするクリティカルセクション関数です。クリティカルセクション (条件変数引数で指定された) が利用できるようになるまで待つようにするには、この関数を使います。呼び出し元スレッドは、同じ条件変数引数で別のスレッドが condvar_notify を呼び出すまで、ブロックされます。呼び出し元は、condvar_wait を呼び出す前に、この条件変数に関連付けられたクリティカルセクションに入っている必要があります。

構文

```
void condvar_wait(CONDVAR cv);
```

戻り値

void を返します。

パラメータ

CONDVAR cv は、条件変数です。

関連項目

condvar_init、condvar_notify、condvar_terminate、crit_init、crit_enter、crit_exit、crit_terminate

crit_enter

クリティカルセクションへ入ることを試みるクリティカルセクション関数です。クリティカルセクションの所有権を得るには、この関数を使います。ほかのスレッドがすでにそのクリティカルセクションを所有している場合は、呼び出し元スレッドは、crit_exit の呼び出しによって最初のスレッドが所有権を返すまで、ブロックされます。

構文

```
void crit_enter(CRITICAL crvar);
```

戻り値

void を返します。

パラメータ

CRITICAL crvar は、クリティカルセクション変数です。

関連項目

crit_init、crit_exit、crit_terminate

crit_exit

クリティカルセクションの所有権を返すクリティカルセクション関数です。クリティカルセクションの所有権を返すには、この関数を使います。ほかのスレッドがブロックされクリティカルセクションを待っている場合は、ブロックは取り除かれ、待機していたスレッドにそのクリティカルセクションの所有権が与えられます。

構文

```
void crit_exit(CRITICAL crvar);
```

戻り値

void を返します。

パラメータ

CRITICAL crvar は、クリティカルセクション変数です。

関連項目

crit_init、crit_enter、crit_terminate

crit_init

新しいクリティカルセクション変数 (型が CRITICAL の変数) を作成し、返すクリティカルセクション関数です。実行される 2 つのスレッド間の干渉の防止の管理に使う、型が CRITICAL (クリティカルセクション変数) の変数の新しいインスタンスを取得するには、この関数を使います。スレッドの作成時には、どのスレッドもクリティカルセクションを所有していません。

警告

crit_terminate が呼び出される時は、スレッドがクリティカルセクションを所有していたり待機してはなりません。

構文

```
CRITICAL crit_init(void);
```

戻り値

新しく割り当てられたクリティカルセクション変数 (CRITICAL) を返します。

パラメータ

なし。

関連項目

crit_enter、crit_exit、crit_terminate

crit_terminate

前に割り当てたクリティカルセクション変数 (型が CRITICAL の変数) を削除する、クリティカルセクション関数です。crit_init を呼び出して前に取得したクリティカルセクション変数を解放するには、この関数を使います。

構文

```
void crit_terminate(CRITICAL crvar);
```

戻り値

void を返します。

パラメータ

CRITICAL crvar は、クリティカルセクション変数です。

関連項目

crit_init、crit_enter、crit_exit

D

daemon_atrestart

daemon_atrestart 関数を使って、サーバの終了時に使う、fn で指定されたコールバック関数を登録できます。初期化関数が割り当てたリソースをコールバック関数で割り当て解除する必要がある場合に、この関数を使います。daemon_atrestart 関数は、magnus_atrestart 関数を汎用化したものです。

magnus.conf 指令の TerminateTimeout と ChildRestartCallback も、NSAPI 関数のコールバックに影響します。

構文

```
void daemon_atrestart(void (*fn)(void *), void *data);
```

戻り値

void を返します。

パラメータ

void (* fn) (void *) は、コールバック関数です。

void *data は、サーバの再起動時にコールバック関数に渡されるパラメータです。

例

```
/* log_close 関数を登録し、サーバの再起動またはシャットダウン時に */
/* *ログファイルを閉じるために、*/
/* log_close に NULL を渡す */
daemon_atrestart(log_close, NULL);
NSAPI_PUBLIC void log_close(void *parameter)
{
system_fclose(global_logfd);
}
```

F

fc_close

fc_close 関数は、fc_open を使って開いたファイルを閉じます。この関数は、fc_open を使って開いたファイルでのみ呼び出す必要があります。

構文

```
void fc_close(PRFileDesc *fd, FcHdl *hDl);
```

戻り値

void を返します。

パラメータ

PRFileDesc *fd　fc_open への前への呼び出しから返された有効なポインタです。

FcHdl *hDl　は、型が FcHdl の構造体への有効なポインタです。このポインタは、fc_open への前の呼び出しで初期化されている必要があります。

fc_open

fc_open 関数は、開いているファイル (fileName) を指す PRFileDesc へのポインタを返します。fileName は、既存のファイルの絶対パス名である必要があります。ファイルは、読み取りモードでのみ開きます。この関数を呼び出すアプリケーションは、DUP_FILE_DESC もこの関数に渡される場合以外は、PRFileDesc * でポイントされるファイルの現在の位置を変更してはなりません。つまり、アプリケーションは (少なくとも)、PRFileDesc * の現在の位置を変更する、このポインタに基づいた読み取り操作を発行してはなりません。そのような読み取り操作 (PRFileDesc * の現在の位置を変更することがある) が必要な場合は、アプリケーションは、DUP_FILE_DESC 引数を指定してこの関数を呼び出す必要があります。

この関数への呼び出しが成功すると、PRFileDesc への有効なポインタが返され、ハンドル「FcHdl」が適切に初期化されます。ファイルのサイズの情報は、ハンドルの「fileSize」メンバーに格納されます。

構文

```
PRFileDesc *fc_open(const  char  *fileName, FcHdl *hDl, PRUint32  flags,
Session *sn, Request *rq);
```

戻り値

PRFileDesc へのポインタを返し、失敗した場合は NULL を返します。

パラメータ

const char *fileName は、開くファイルの絶対パス名です。

FcHdl*hDl は、型 FcHdl の構造体への有効なポインタです。

PRUint32 flags は、0 または DUP_FILE_DESC になります。

Session *sn は、セッションへのポインタです。

Request *rq は、要求へのポインタです。

filebuf_buf2sd

filebuf_buf2sd 関数は、ファイルバッファをソケット (記述子) に送信し、送信したバイト数を返します。

ファイル全体の内容をクライアントへ送信するには、この関数を使います。

構文

int filebuf_buf2sd(filebuf *buf, SYS_NETFD sd);

戻り値

成功した場合は、ソケットへ送信したバイトを返し、ファイルバッファを送信できなかった場合は、定数 IO_ERROR を返します。

パラメータ

filebuf *buf は、すでに開かれている必要がある、ファイルバッファです。

SYS_NETFD sd は、プラットフォームに依存しないソケット記述子です。通常、このソケット記述子は、sn (Session: セッション) 構造体の csd (client socket descriptor: クライアントソケット記述子) フィールドから得られます。

例

```
if (filebuf_buf2sd(buf, sn->csd) == IO_ERROR)
       return(REQ_EXIT);
```

関連項目

filebuf_close、filebuf_open、filebuf_open_nostat、filebuf_getc

filebuf_close

filebuf_close 関数は、ファイルバッファの割り当てを解除し、関連するファイルを閉じます。

通常は、まず filebuf_open を使ってファイルバッファを開き、次に filebuf_getc を使ってファイル内の情報にアクセスします。ファイルバッファを使用し終わったら、filebuf_close を使って閉じます。

構文

```
void filebuf_close(filebuf *buf);
```

戻り値

void を返します。

パラメータ

filebuf *buf は、filebuf_open で前に開いたファイルバッファです。

例

```
filebuf_close(buf);
```

関連項目

filebuf_open、filebuf_open_nostat、filebuf_buf2sd、filebuf_getc

filebuf_getc

filebuf_getc 関数は、現在のファイル位置から文字を取得し、その文字を整数として返します。次に、現在のファイルの位置を増分します。

バッファ化されたファイルから文字を順番に読み取るには、filebuf_getc を使います。

構文

```
filebuf_getc(filebuf b);
```

戻り値

取得した文字が含まれる整数を返し、ファイルの終わりまたはエラーの場合は定数の IO_EOF または IO_ERROR を返します。

パラメータ

filebuf b は、ファイルバッファの名前です。

関連項目

filebuf_close、filebuf_buf2sd、filebuf_open、filebuf_open_nostat

filebuf_open

filebuf_open 関数は、前に開いたファイルに対して新しいファイルバッファを開きます。新しいバッファ構造体を返します。バッファ化されたファイルは、オペレーティングシステムがバッファによる入出力をサポートしていない環境で、バッファによるファイルの入出力を可能にし、ファイルアクセスをより効率的にします。

構文

```
filebuf *filebuf_open(SYS_FILE fd, int sz);
```

戻り値

成功した場合はデータを入れる新しいバッファ構造体へのポインタを返し、バッファを開くことができない場合は NULL を返します。

パラメータ

SYS_FILE fd は、すでに開かれているファイルのプラットフォームに依存しないファイル記述子です。

int sz は、バッファに使用される、バイト単位のサイズです。

例

```
filebuf *buf = filebuf_open(fd, FILE_BUFFERSIZE);
if (!buf) {
       system_fclose(fd);
}
```

関連項目

filebuf_getc、filebuf_buf2sd、filebuf_close、filebuf_open_nostat

filebuf_open_nostat

filebuf_open_nostat 関数は、前に開いたファイルに対して新しいファイルバッファを開きます。新しいバッファ構造体を返します。バッファ化されたファイルは、オペレーティングシステムがバッファによる入出力をサポートしていない環境で、バッファによるファイルの入出力を可能にし、ファイルアクセスをより効率的にします。

この関数は、filebuf_open と同じですが、request_stat_path 関数を呼び出す必要がないので、より効率的です。この関数には、stat 情報を渡す必要があります。

構文

```
filebuf* filebuf_open_nostat(SYS_FILE fd, int sz,
       struct stat *finfo);
```

戻り値

成功した場合はデータを入れる新しいバッファ構造体へのポインタを返し、バッファを開くことができない場合は NULL を返します。

パラメータ

SYS_FILE fd は、すでに開かれているファイルのプラットフォームに依存しないファイル記述子です。

int sz は、バッファに使用されるバイト単位のサイズです。

struct stat *finfo は、ファイルのファイル情報です。filebuf_open_nostat 関数を呼び出す前に、request_stat_path 関数を呼び出してファイル情報を取得する必要があります。

例

```
filebuf *buf = filebuf_open_nostat(fd, FILE_BUFFERSIZE, &finfo);
if (!buf) {
       system_fclose(fd);
}
```

関連項目

filebuf_close、filebuf_open、filebuf_getc、filebuf_buf2sd

FREE

FREE マクロは、C ライブラリルーチンの free のプラットフォームに依存しないものです。この関数は、MALLOC、CALLOC、または STRDUP によって前に割り当てられた領域を、要求のメモリープールから割り当て解除します。

構文

```
FREE(void *ptr);
```

戻り値

void を返します。

パラメータ

void *ptr は、メモリーのブロックへの (void *) ポインタです。このポインタが MALLOC、CALLOC、または STRDUP で作成されたものでない場合は、動作は定義されていません。

例

```
char *name;
name = (char *) MALLOC(256);
...
FREE(name);
```

関連項目

MALLOC、CALLOC、REALLOC、STRDUP、PERM_MALLOC、PERM_FREE、PERM_REALLOC、PERM_STRDUP

func_exec

func_exec 関数は、指定された pblock 内の fn エントリで指定された関数を実行します。関数名が見つからない場合は、エラーをログに記録し、REQ_ABORTED を返します。

pblock 内に組み込み SAF を指定すると、この関数を使って、その SAF を実行できます。

構文

```
int func_exec(pblock *pb, Session *sn, Request *rq);
```

戻り値

実行された関数が返す値を返し、または関数が実行されなかった場合は定数の REQ_ABORTED を返します。

パラメータ

pblock *pb は、関数名 (function name: fn) とパラメータが入っている pblock です。

Session *sn は、Session です。

Request *rq は、Request です。

Session および Request パラメータは、カスタム SAF に渡されるものと同じです。

関連項目

log_error

func_find

func_find 関数は、name で指定された関数へのポインタを返します。関数が存在しない場合は、NULL を返します。

構文

```
FuncPtr func_find(char *name);
```

戻り値

間接参照に適した、選択した関数へのポインタを返します。関数が見つからない場合は、NULL を返します。

パラメータ

char *name は、関数の名前です。

例

```
/* このコードブロックは、func_exec と同じことを実行する */
char *afunc = pblock_findval("afunction", pb);
FuncPtr afnptr = func_find(afunc);
if (afnptr)
       return (afnptr)(pb, sn, rq);
```

関連項目

func_exec

L

log_error

log_error 関数は、エラーログにエントリを作成し、日付、重要度、および指定されたテキストを記録します。

構文

```
int log_error(int degree, char *func, Session *sn, Request *rq,
char *fmt, ...);
```

戻り値

ログエントリが作成された場合は 0、ログエントリが作成されなかった場合は -1 を返します。

パラメータ

int degree は、エラーの重要度を指定します。このパラメータは、次のいずれかの定数にする必要があります。

LOG_WARN— 警告
LOG_MISCONFIG— 構文エラーまたはアクセス権違反
LOG_SECURITY— 認証の失敗またはホストからの 403 エラー
LOG_FAILURE— 内部の問題
LOG_CATASTROPHE— サーバ回復不能のエラー
LOG_INFORM— 情報メッセージ

char *func は、エラーが発生した関数の名前です。

Session *sn は、Session です。

Request *rq は、Request です。

Session および Request パラメータは、SAF に渡されたものと同じです。

char *fmt は、メッセージを配信する printf 関数用の書式を指定します。

... は、printf 関数の一連のパラメータを表します。

例

```
log_error(LOG_WARN, "send-file", sn, rq,
       "error opening buffer from %s (%s)", path,
              system_errmsg(fd));
```

関連項目

func_exec

M

MALLOC

MALLOC マクロは、C ライブラリルーチンの malloc のプラットフォームに依存しないものです。MALLOC は、通常、要求のメモリープールから割り当てます。プールされたメモリーが構成ファイルで (組み込み SAF の pool-init を使って) 無効に設定されている場合は、PERM_MALLOC および MALLOC はどちらもシステムヒープからメモリーを取得します。

構文

```
void *MALLOC(int size)
```

戻り値

メモリーのブロックへの void 型のポインタを返します。

パラメータ

int size は、割り当てるバイト数です。

例

```
/* name に 256 バイトを割り当てる */
char *name;
name = (char *) MALLOC(256);
```

関連項目

FREE、CALLOC、REALLOC、STRDUP、PERM_MALLOC、PERM_FREE、PERM_CALLOC、PERM_REALLOC、PERM_STRDUP

N

net_ip2host

net_ip2host 関数は、テキスト形式の IP アドレスを絶対パスによるドメイン名に変換して返します。

注	この関数は、magnus.conf ファイルで DNS 指令が有効になっている場合にだけ機能します。詳細は、第 7 章「magnus.conf の構文と使用法」を参照してください。

構文

```
char *net_ip2host(char *ip, int verify);
```

戻り値

変換できた場合は、絶対パスによるドメイン名が含まれる新しい文字列を返し、変換できなかった場合は NULL を返します。

パラメータ

char *ip は、ドット付き 10 進表記の文字列形式 nnn.nnn.nnn.nnn の IP (Internet Protocol) アドレスです。

int verify は、ゼロでない場合、関数が絶対パスによるドメイン名を検証する必要があることを示します。これには、さらに照会が必要になりますが、アクセス制御を確認するときには使用する必要があります。

net_read

net_read 関数は、指定されたソケットから指定されたバッファにバイトを読み込みます。この関数は、ソケットで少なくとも 1 バイトが利用可能になるまで、または指定された期間が過ぎるまで、ソケットからのデータの受信を待ちます。

構文

```
int net_read (SYS_NETFD sd, char *buf, int sz, int timeout);
```

戻り値

読み込まれたバイト数を返し、この数は最大サイズの sz を超えません。エラーが発生した場合は負の値が返され、その場合、timeout 秒が経過する前に操作が完了しなかった場合は、定数 ETIMEDOUT に errno が設定されます。

パラメータ

SYS_NETFD sd は、プラットフォームに依存しないソケット記述子です。

char *buf は、バイトを受け取るバッファです。

int sz は、読み込む最大バイト数です。

int timeout は、復帰する前に読み込み操作に許可される秒数です。timeout の目的は、与えられた時間内に十分なバイトが読み込まれなかったために復帰することではなく、データを受け取るまで待つ時間を制限することです。

関連項目

net_write

net_write

net_write 関数は、指定されたバイト数を指定されたバッファから指定されたソケットに書き込みます。書き込まれたバイト数を返します。

構文

```
int net_write(SYS_NETFD sd, char *buf, int sz);
```

戻り値

書き込まれたバイト数を返し、エラーが発生した場合は要求されたサイズよりも小さいことがあります。

パラメータ

SYS_NETFD sd は、プラットフォームに依存しないソケット記述子です。

char *buf は、バイトが入っているバッファです。

int sz は、書き込むバイトの数です。

例

```
if (net_write(sn->csd, FIRSTMSG, strlen(FIRSTMSG)) == IO_ERROR)
       return REQ_EXIT;
```

関連項目

net_read

netbuf_buf2sd

netbuf_buf2sd 関数は、バッファをソケットへ送信します。この関数を使って、データを IPC パイプからクライアントへ送信することができます。

構文
```
int netbuf_buf2sd(netbuf *buf, SYS_NETFD sd, int len);
```

戻り値
成功した場合はソケットへ転送されたバイトの数を返し、失敗した場合は定数の IO_ERROR を返します。

パラメータ
netbuf *buf は、送信するバッファです。

SYS_NETFD sd は、プラットフォームに依存しないソケット識別子です。

int len は、バッファの長さです。

関連項目
netbuf_close、netbuf_getc, netbuf_grab、netbuf_open

netbuf_close
netbuf_close 関数は、ネットワークバッファを割り当て解除し、関連するファイルを閉じます。ネットワークバッファを割り当て解除し、ソケットを閉じる必要があるときに、この関数を使います。

Session 構造体内の netbuf パラメータは閉じてはなりません。

構文
```
void netbuf_close(netbuf *buf);
```

戻り値
void を返します。

パラメータ
netbuf *buf は、閉じるバッファです。

関連項目
netbuf_buf2sd、netbuf_getc, netbuf_grab、netbuf_open

netbuf_getc
netbuf_getc 関数は、b で指定されたネットワークバッファのカーソル位置から文字を取得します。

構文
```
netbuf_getc(netbuf b);
```

戻り値

文字を取得できた場合はその文字を表す整数を返し、ファイルの終わりまたはエラーの場合は定数の IO_EOF または IO_ERROR を返します。

パラメータ

netbuf b は、1 つの文字の取得先のバッファです。

関連項目

netbuf_buf2sd、netbuf_close、netbuf_grab、netbuf_open

netbuf_grab

netbuf_grab 関数は、ネットワークバッファ (buf) のソケットからネットワークバッファへ sz 個のバイトを読み込みます。バッファの大きさが十分でない場合は、大きさが調整されます。成功した場合、buf->inbuf からデータを取得できます。

netbuf_buf2sd 関数が、この関数を使います。

構文

```
int netbuf_grab(netbuf *buf, int sz);
```

戻り値

操作が成功した場合は実際に読み込んだバイトの数 (1 ～ sz) を返し、ファイルの終わりまたはエラーの場合は定数の IO_EOF または IO_ERROR を返します。

パラメータ

netbuf *buf は、データを読み込むバッファです。

int sz は、読み込むバイトの数です。

関連項目

netbuf_buf2sd、netbuf_close、netbuf_getc、netbuf_open

netbuf_open

netbuf_open 関数は、新しいネットワークバッファを開いて返します。netbuf_open を使って、netbuf 構造体を作成し、ソケットに対してバッファを利用した入出力の使用を開始できます。

構文

```
netbuf* netbuf_open(SYS_NETFD sd, int sz);
```

戻り値

新しい netbuf 構造体 (ネットワークバッファ) へのポインタを返します。

パラメータ

SYS_NETFD sd は、プラットフォームに依存しないソケット識別子です。

int sz は、ネットワークバッファに割り当てる文字の数です。

関連項目

netbuf_buf2sd、netbuf_close、netbuf_getc、netbuf_grab

P

param_create

param_create 関数は、指定された名前と値が含まれる pb_param 構造体を作成します。名前と値はコピーされます。pblock_pinsert などのように、pblock ルーチンへの呼び出しに使うために、この関数を使って pb_param 構造体を準備します。

構文

```
pb_param *param_create(char *name, char *value);
```

戻り値

新しい pb_param 構造体へのポインタを返します。

パラメータ

char *name は、名前が含まれる文字列です。

char *value は、値が含まれる文字列です。

例

```
pb_param *newpp = param_create("content-type","text/plain");
pblock_pinsert(newpp, rq->srvhdrs);
```

関連項目

param_free、pblock_pinsert、pblock_remove

param_free

param_free 関数は、pp で指定された pb_param 構造体とこの構造体に関連する構造体を解放します。pblock_remove で pblock から削除した後に pb_param を廃棄するには、param_free 関数を使います。

構文

```
int param_free(pb_param *pp);
```

戻り値

パラメータが解放された場合は 1、パラメータが NULL の場合は 0 を返します。

パラメータ

pb_param *pp は、pblock に格納された名前 - 値のペアです。

例

```
if (param_free(pblock_remove("content-type", rq-srvhdrs)))
       return; /* 削除した */
```

関連項目

param_create、pblock_pinsert、pblock_remove

pblock_copy

pblock_copy 関数は、コピー元の pblock のエントリをコピーし、コピー先の pblock に追加します。コピー先の pblock に前からあるエントリは、影響を受けません。

構文

```
void pblock_copy(pblock *src, pblock *dst);
```

戻り値

void を返します。

パラメータ

pblock *src は、コピー元である pblock です。

pblock *dst は、コピー先の pblock です。

元の pblock を解放したり、または元の pblock に影響を与えずに新しい pblock を変更したりできるように、名前と値が新たに割り当てられます。

関連項目

pblock_create、pblock_dup、pblock_free、pblock_find、pblock_findval、pblock_remove、pblock_nvinsert

pblock_create

pblock_create 関数は、新しい pblock を作成します。pblock は、名前 - 値ペアのより早い検索のために内部にハッシュテーブルを維持します。

構文

```
pblock *pblock_create(int n);
```

戻り値
新たに割り当てられた pblock へのポインタを返します。

パラメータ
int n は、pblock 用のハッシュテーブルのサイズ (名前 - 値ペアの数) です。

関連項目
pblock_copy、pblock_dup、pblock_find、pblock_findval、pblock_free、pblock_nvinsert、pblock_remove

pblock_dup
pblock_dup 関数は、pblock を複製します。この関数は、pblock_create に pblock_copy を続けて実行する場合と同じことを行ないます。

構文
```
pblock *pblock_dup(pblock *src);
```

戻り値
新たに割り当てられた pblock へのポインタを返します。

パラメータ
pblock *src は、コピー元の pblock です。

関連項目
pblock_create、pblock_find、pblock_findval、pblock_free、pblock_find、pblock_remove、pblock_nvinsert

pblock_find
pblock_find 関数は、pblock 内の指定された名前 - 値ペアのエントリを検索し、pb_param 構造体を返します。名前に関連する値だけが必要な場合は、pblock_findval 関数を使います。

この関数は、マクロとして実装されています。

構文
```
pb_param *pblock_find(char *name, pblock *pb);
```

戻り値
見つかった場合は pb_param 構造体へのポインタを返し、名前が見つからなかった場合は NULL を返します。

パラメータ

char *name は、名前 - 値ペアの名前です。

pblock *pb は、検索対象の pblock です。

関連項目

pblock_copy、pblock_dup、pblock_findval、pblock_free、pblock_nvinsert、pblock_remove

pblock_findval

pblock_findval 関数は、pblock 内に指定された名前の値を検索します。pblock の pb_param 構造体だけが必要な場合は、pblock_find 関数を使います。

返されるポインタは、pblock へのポインタです。このポインタを解放してはなりません。変更する必要がある場合は、STRDUP を実行し、コピーを変更します。

構文

```
char *pblock_findval(char *name, pblock *pb);
```

戻り値

名前に関連付けられた値が含まれる文字列を返し、一致するものが見つからなかった場合は NULL を返します。

パラメータ

char *name は、名前 - 値ペアの名前です。

pblock *pb は、検索対象の pblock です。

関連項目

pblock_create、pblock_copy、pblock_find、pblock_free、pblock_nvinsert、pblock_remove、request_header

pblock_free

pblock_free 関数は、指定された pblock とその中のすべてのエントリを解放します。その pblock 中の変数を保存する必要がある場合は、pblock_remove 関数を使ってその変数を削除し、その結果のポインタを保存します。

構文

```
void pblock_free(pblock *pb);
```

戻り値

void を返します。

パラメータ

pblock *pb は、解放する pblock です。

関連項目

pblock_copy、pblock_create、pblock_dup、pblock_find、pblock_findval、pblock_nvinsert、pblock_remove

pblock_nninsert

pblock_nninsert 関数は、指定された pblock に指定された名前と数値で新しいエントリを作成します。数値は、最初に文字列に変換されます。名前と値のパラメータがコピーされます。

構文

```
pb_param *pblock_nninsert(char *name, int value, pblock *pb);
```

戻り値

新しい pb_param 構造体へのポインタを返します。

パラメータ

char *name は、新しいエントリの名前です。

int value は、pblock に挿入される数値です。このパラメータは、整数である必要があります。割り当てた値が数値でない場合は、代わりに pblock_nvinsert 関数を使ってパラメータを作成します。

pblock *pb は、挿入先の pblock です。

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nvinsert、pblock_remove、pblock_str2pblock

pblock_nvinsert

pblock_nvinsert 関数は、指定された pblock に指定された名前と文字値で新しいエントリを作成します。名前と値のパラメータがコピーされます。

構文

```
pb_param *pblock_nvinsert(char *name, char *value, pblock *pb);
```

戻り値

新たに割り当てられた pb_param 構造体へのポインタを返します。

パラメータ

char *name は、新しいエントリの名前です。

char *value は、新しいエントリの文字列値です。

pblock *pb は、挿入先の pblock です。

例

```
pblock_nvinsert("content-type", "text/html", rq->srvhdrs);
```

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nninsert、pblock_remove、pblock_str2pblock

pblock_pb2env

pblock_pb2env 関数は、指定された pblock を指定された環境にコピーします。この関数は、pblock 内の名前 - 値のペアごとに、1 つ新しい環境エントリを作成します。この関数を使って、実行するプログラムに pblock エントリを送信します。

構文

```
char **pblock_pb2env(pblock *pb, char **env);
```

戻り値

環境へのポインタを返します。

パラメータ

pblock *pb は、コピーする pblock です。

char **env は、pblock のコピー先の環境です。

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nvinsert、pblock_remove、pblock_str2pblock

pblock_pblock2str

pblock_pblock2str 関数は、指定された pblock のすべてのパラメータを指定された文字列にコピーします。この関数は、必要に応じて、その文字列に対してヒープでない追加領域を割り当てます。

保存およびその他の目的で pblock をストリーミングするには、この関数を使います。

構文

```
char *pblock_pblock2str(pblock *pb, char *str);
```

戻り値

str パラメータの新しいバージョンです。str が NULL の場合は、新しい文字列です。NULL でない場合は、再度割り当てられた文字列です。どちらの場合も、要求のメモリープールから割り当てられます。

パラメータ

pblock *pb は、コピーされる pblock です。

char *str は、pblock のコピー先の文字列です。これは、PERM_MALLOC または PERM_REALLOC (システムヒープから割り当てられる) ではなく、MALLOC または REALLOC で割り当てられている必要があります。

文字列内の各名前 - 値ペアは、ほかのペアとは空白文字で区切られ、*name*="*value*" の形式になります。

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nvinsert、pblock_remove、pblock_str2pblock

pblock_pinsert

pblock_pinsert 関数は、pb_param 構造体を pblock に挿入します。

構文

```
void pblock_pinsert(pb_param *pp, pblock *pb);
```

戻り値

void を返します。

パラメータ

pb_param *pp は、挿入する pb_param 構造体です。

pblock *pb は、pblock です。

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nvinsert、pblock_remove、pblock_str2pblock

pblock_remove

pblock_remove 関数は、指定された pblock から指定された名前 - 値エントリを削除します。この関数を使う場合は、pb_param 構造体が使ったメモリーの割り当てを解除するために最終的に param_free を呼び出す必要があります。

構文

```
pb_param *pblock_remove(char *name, pblock *pb);
```

戻り値

見つかった場合は指定された pb_param 構造体へのポインタを返し、指定された pb_param が見つからなかった場合は NULL を返します。

パラメータ

char *name は、削除する pb_param の名前です。

pblock *pb は、名前 - 値エントリが削除される pblock です。

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nvinsert、param_create、param_free

pblock_str2pblock

pblock_str2pblock 関数は、文字列でパラメータペアを走査し、見つかったペアを pblock に追加して、追加したパラメータの数を返します。

構文

```
int pblock_str2pblock(char *str, pblock *pb);
```

戻り値

追加されたものがある場合は pblock に追加されたパラメータペアの数を返し、エラーが発生した場合は -1 を返します。

パラメータ

char *str は、走査対象の文字列です。

文字列内の名前 - 値ペアの形式は、*name*=*value* または *name*="*value*" です。

バックスラッシュ (\) にはすべて、リテラル文字を続ける必要があります。文字列値にアンエスケープされた = 記号がない (name= がない) 場合、この関数は文字列内の位置により名前が 1、2、3 などになるとみなします。たとえば、pblock_str2pblock は、"some strings together" を見つけると、この関数はこの文字列が 1="some" 2="strings" 3="together" の名前 - 値ペアとして現れたかのように扱います。

pblock *pb は、名前 - 値ペアが格納される pblock です。

関連項目

pblock_copy、pblock_create、pblock_find、pblock_free、pblock_nvinsert、pblock_remove、pblock_pblock2str

PERM_CALLOC

PERM_CALLOC マクロは、C ライブラリルーチンの calloc のプラットフォームに依存しないものです。この関数は、要求の処理の完了後も持続する、num*size バイトのメモリーを割り当てます。プールされたメモリーが構成ファイルで (組み込み SAF の pool-init を使って) 無効にされている場合は、PERM_CALLOC および CALLOC はどちらもシステムヒープからメモリーを取得します。

構文

```
void *PERM_CALLOC(int num, int size)
```

戻り値

メモリーのブロックへの void 型のポインタを返します。

パラメータ

int num は、割り当てる要素の数です。

int size は、各要素のバイト単位のサイズです。

例

```
/* 100 個の char 型のポインタの配列に領域を割り当てる */
char *name;
name = (char *) PERM_CALLOC(100, sizeof(char *));
```

関連項目

PERM_FREE、PERM_STRDUP、PERM_MALLOC、PERM_REALLOC、MALLOC、FREE、CALLOC、STRDUP、REALLOC

PERM_FREE

PERM_FREE マクロは、C ライブラリルーチンの free のプラットフォームに依存しないものです。この関数は、PERM_MALLOC、PERM_CALLOC、または PERM_STRDUP で前に割り当てられた持続領域の割り当てを解除します。プールされたメモリーが構成ファイルで (組み込み SAF の pool-init を使って) 無効にされている場合は、PERM_FREE および FREE はシステムヒープ内のメモリーの割り当てを解除します。

構文

```
PERM_FREE(void *ptr);
```

戻り値

void を返します。

パラメータ

void *ptr は、メモリーのブロックへの (void *) ポインタです。ポインタが PERM_MALLOC、PERM_CALLOC、または PERM_STRDUP で作成されたものでない場合は、動作は定義されていません。

例

```
char *name;
name = (char *) PERM_MALLOC(256);
...
PERM_FREE(name);
```

関連項目

FREE、MALLOC、CALLOC、REALLOC、STRDUP、PERM_MALLOC、PERM_CALLOC、PERM_REALLOC、PERM_STRDUP

PERM_MALLOC

PERM_MALLOC マクロは、C ライブラリルーチンの malloc のプラットフォームに依存しないものです。この関数は、要求の処理の完了後も持続する、メモリーの割り当てを可能にします。プールされたメモリーが構成ファイルで (組み込み SAF の pool-init を使って) 無効にされている場合は、PERM_MALLOC および MALLOC はどちらもシステムヒープからメモリーを取得します。

構文

```
void *PERM_MALLOC(int size)
```

戻り値

メモリーのブロックへの void 型のポインタを返します。

パラメータ

int size は、割り当てるバイト数です。

例

```
/* name に 256 バイトを割り当てる */
char *name;
name = (char *) PERM_MALLOC(256);
```

関連項目

PERM_FREE、PERM_STRDUP、PERM_CALLOC、PERM_REALLOC、MALLOC、FREE、CALLOC、STRDUP、REALLOC

PERM_REALLOC

PERM_REALLOC マクロは、C ライブラリルーチンの realloc のプラットフォームに依存しないものです。もともと PERM_MALLOC、PERM_CALLOC、または PERM_STRDUP で作成された、指定されたメモリーブロックのサイズを変更します。オブジェクトの内容は、古いサイズと新しいサイズのどちらか小さい方まで、変わりません。新しいサイズの方が大きい場合は、新しい領域は初期化されていません。

警告

MALLOC、CALLOC、または STRDUP で割り当てられたブロックに対して、PERM_REALLOC を呼び出しても動作しません。

構文

```
void *PERM_REALLOC(void *ptr, int size)
```

戻り値

メモリーのブロックへの void 型のポインタを返します。

パラメータ

void *ptr は、PERM_MALLOC、PERM_CALLOC、または PERM_STRDUP で作成されたメモリーのブロックへの void 型のポインタです。

int size は、メモリーのブロックの大きさをそれへ変更する、バイト数です。

例

```
char *name;
name = (char *) PERM_MALLOC(256);
if (NotBigEnough())
    name = (char *) PERM_REALLOC(name, 512);
```

関連項目

PERM_MALLOC、PERM_FREE、PERM_CALLOC、PERM_STRDUP、MALLOC、FREE、STRDUP、CALLOC、REALLOC

PERM_STRDUP

PERM_STRDUP マクロは、C ライブラリルーチンの strdup のプラットフォームに依存しないものです。この関数は、要求の処理の完了後も持続する、メモリー内の文字列の新しいコピーを作成します。プールされたメモリーが構成ファイルで (組み込み SAF の pool-init を使って) 無効にされている場合は、PERM_STRDUP および STRDUP はどちらもシステムヒープからメモリーを取得します。

PERM_STRDUP ルーチンの機能は、次の関数のシーケンスを実行する場合と同じです。

```
newstr = (char *) PERM_MALLOC(strlen(str) + 1);
strcpy(newstr, str);
```

PERM_STRDUP で作成した文字列は、PERM_FREE で破棄する必要があります。

構文

```
char *PERM_STRDUP(char *ptr);
```

戻り値

新しい文字列へのポインタを返します。

パラメータ

char *ptr は、文字列へのポインタです。

関連項目

PERM_MALLOC、PERM_FREE、PERM_CALLOC、PERM_REALLOC、MALLOC、FREE、STRDUP、CALLOC、REALLOC

prepare_nsapi_thread

prepare_nsapi_thread 関数は、サーバが作成していないスレッドが、サーバが作成したスレッドのように動作できるようにします。サーバが作成していないスレッドから他の NSAPI 関数を呼び出す前に、この関数を呼び出す必要があります。

構文

```
void prepare_nsapi_thread(Request *rq, Session *sn);
```

戻り値

void を返します。

パラメータ

Request *rq は、Request です。

Session *sn は、Session です。

Request および Session パラメータは、カスタム SAF に渡されるものと同じです。

関連項目

protocol_start_response

protocol_dump822

protocol_dump822 関数は、ヘッダーを指定された pblock から、特定のバッファへ、指定された位置に指定されたサイズでプリントします。たとえばメールメッセージなど、ヘッダーを直列化して送ることができるようにする場合は、この関数を使います。

構文

```
char *protocol_dump822(pblock *pb, char *t, int *pos, int tsz);
```

戻り値

必要に応じて再割り当てされる、バッファへのポインタを返します。

また、この関数は、バッファ内でヘッダーの終わりまで *pos を変更します。

パラメータ

pblock *pb は、pblock 構造体です。

char *t は、MALLOC、CALLOC、または STRDUP で割り当てられたバッファです。

int *pos は、ヘッダーがダンプされる、バッファ内の位置です。

int tsz は、バッファのサイズです。

関連項目

protocol_start_response、protocol_status

protocol_set_finfo

protocol_set_finfo 関数は、指定された stat 構造体から content-length および last-modified 日付を取得し、応答ヘッダー (rq->srvhdrs) に追加します。protocol_start_response を呼び出す前に、protocol_set_finfo を呼び出します。

構文

```
int protocol_set_finfo(Session *sn, Request *rq, struct stat
*finfo);
```

戻り値

要求の処理を正しく進めることができる場合は定数の REQ_PROCEED を返し、関数が要求を正しく処理する必要があるが、出力をクライアントへ送ってはならない場合は定数の REQ_ABORTED を返します。

パラメータ

Session *sn は、Session です。

Request *rq は、Request です。

Session および Request パラメータは、カスタム SAF に渡されるものと同じです。

stat *finfo は、ファイルの stat 構造体です。

stat 構造体には、ファイルシステムからのファイルについての情報が入っています。request_stat_path を使って、stat 構造体の情報を取得できます。

関連項目

protocol_start_response、protocol_status

protocol_start_response

protocol_start_response 関数は、指定されたセッションおよび要求に対して HTTP 応答を開始します。プロトコルのバージョンが HTTP/0.9 である場合は、この関数は何もしません。これは、このバージョンにはステータスの概念がないためです。プロトコルのバージョンが HTTP/1.0 である場合は、この関数はステータス行に応答ヘッダーを続けて送信します。この関数を使って、HTTP を設定し、クライアントおよびサーバが応答の本体 (つまり、データ) を受信できるように準備します。

構文

```
int protocol_start_response(Session *sn, Request *rq);
```

戻り値

操作が成功した場合は定数の REQ_PROCEED を返し、この場合は送信のために準備したデータを送信する必要があります。

操作が成功したが要求メソッドが HEAD であったため、データをクライアントに送らない場合は、定数の REQ_NOACTION を返します。

操作が失敗した場合は、定数の REQ_ABORTED を返します。

パラメータ

Session *sn は、Session です。

Request *rq は、Request です。

Session および Request パラメータは、カスタム SAF に渡されるものと同じです。

例

```
/* この関数からの noaction 応答は、要求が HEAD であったことを示す */
if (protocol_start_response(sn, rq) == REQ_NOACTION) {
       filebuf_close(groupbuf);  /* ファイルを閉じる */
       return REQ_PROCEED;
}
```

関連項目

protocol_status

protocol_status

protocol_status 関数は、エラー状態が発生したかどうかを示すセッションの状態を設定します。原因を示す文字列が NULL である場合は、サーバは、指定された状態コードに対する原因を示す文字列の検索を試みます。原因を示す文字列が見つからない場合は、"Unknown reason" を返します。原因を示す文字列は、HTTP 応答行でクライアントへ送られます。protocol_start_response 関数を呼び出す前に、この関数を使って、応答の状態を設定します。

すべての有効な状態コードの定数のリストは、サーバに付属する nsapi.h ファイルを参照してください。

構文

```
void protocol_status(Session *sn, Request *rq, int n, char *r);
```

戻り値

void を返します。しかし、この関数は状態コードと原因を示す文字列用の sn/rq で指定された Session/Request に値を設定します。

パラメータ

Session *sn は、Session です。

Request *rq は、Request です。

Session および Request パラメータは、カスタム SAF に渡されるものと同じです。

int n は、上記の状態コード定数の 1 つです。

char *r は、原因を示す文字列です。

例

```
/* 余分なパス情報が見つかった場合は、URL が適切でなかったため、*/
/* ブラウザに URL が見つからなかったと通知する */
if (t = pblock_findval("path-info", rq->vars)) {
        protocol_status(sn, rq, PROTOCOL_NOT_FOUND, NULL);
        log_error(LOG_WARN, "function-name", sn, rq, "%s not found",
              path);
        return REQ_ABORTED;
}
```

関連項目

protocol_start_response

protocol_uri2url

protocol_uri2url 関数は、指定された URI 接頭辞および URI 接尾辞が含まれている文字列を受け取り、http://(server):(port)(prefix)(suffix) の形式で新たに割り当てられる絶対パスによる URL を作成します。protocol_uri2url_dynamic を参照してください。

URI 接頭辞または接尾辞のどちらかを省略する場合は、どちらかのパラメータの値として NULL ではなく "" を使います。

構文

```
char *protocol_uri2url(char *prefix, char *suffix);
```

戻り値

URL が含まれている新しい文字列を返します。

パラメータ

char *prefix は、接頭辞です。

char *suffix は、接尾辞です。

関連項目

protocol_start_response、protocol_status、pblock_nvinsert、protocol_uri2url_dynamic

protocol_uri2url_dynamic

protocol_uri2url_dynamic 関数は、指定された URI 接頭辞および URI 接尾辞が含まれている文字列を受け取り、http://(server):(port)(prefix)(suffix) の形式で新たに割り当てられる絶対パスによる URL を作成します。

URI 接頭辞または接尾辞のどちらかを省略する場合は、どちらかのパラメータの値として NULL ではなく "" を使います。

protocol_uri2url_dynamic 関数は、protocol_uri2url 関数に似ていますが、Session および Request 構造体が利用できるときには常に使う必要があります。これにより、この関数が構築する URL が、確実にクライアントが指定したホストを参照するようになります。

構文

```
char *protocol_uri2url(char *prefix, char *suffix, Session *sn,
Request *rq);
```

戻り値

URL が含まれている新しい文字列を返します。

パラメータ

`char *prefix` は、接頭辞です。

`char *suffix` は、接尾辞です。

`Session *sn` は、Session です。

`Request *rq` は、Request です。

`Session` および `Request` パラメータは、カスタム SAF に渡されるものと同じです。

関連項目

`protocol_start_response`、`protocol_status`、`protocol_uri2url`

R

REALLOC

`REALLOC` マクロは、C ライブラリルーチンの `realloc` のプラットフォームに依存しないものです。もともと `MALLOC`、`CALLOC`、または `STRDUP` で作成された、指定されたメモリーのブロックのサイズを変更します。オブジェクトの内容は、古いサイズと新しいサイズのどちらか小さい方まで、変わりません。新しいサイズの方が大きい場合は、新しい領域は初期化されていません。

警告

`PERM_MALLOC`、`PERM_CALLOC`、または `PERM_STRDUP` で割り当てられたブロックに対して、`REALLOC` を呼び出しても動作しません。

構文

```
void *REALLOC(void *ptr, int size);
```

戻り値

要求を満たすことができる場合は、新しい領域へのポインタを返します。

パラメータ

`void *ptr` は、メモリーのブロックへの (void *) ポインタです。ポインタが `MALLOC`、`CALLOC`、または `STRDUP` で作成されたものでない場合は、動作は定義されていません。

`int size` は、割り当てるバイト数です。

例

```
char *name;
name = (char *) MALLOC(256);
if (NotBigEnough())
       name = (char *) REALLOC(name, 512);
```

関連項目

MALLOC、FREE、STRDUP、CALLOC、PERM_MALLOC、PERM_FREE、PERM_REALLOC、PERM_CALLOC、PERM_STRDUP

request_get_vs

request_get_vs 関数は、要求の送信先の VirtualServer* を検索します。

返される VirtualServer* は、現在の要求に対してだけ有効です。すべての要求に有効な仮想サーバ ID を取得するには、vs_get_id を使います。

構文

```
const VirtualServer* request_get_vs(Request* rq);
```

戻り値

要求の送信先の VirtualServer* を返します。

パラメータ

Request *rq は、VirtualServer* が返される Request です。

関連項目

vs_get_id

request_header

request_header 関数は、クライアントの HTTP 要求ヘッダー (rq->headers) が含まれている pblock 内のエントリを検索します。クライアントのヘッダーにアクセスする場合は、pblock_findval ではなく、この関数を使う必要があります。これは、サーバが、ヘッダーがすべて読み込まれる前に要求の処理を開始することがあるためです。

構文

```
int request_header(char *name, char **value, Session *sn, Request
*rq);
```

戻り値

ヘッダーが見つかった場合は REQ_PROCEED、ヘッダーが見つからない場合は REQ_ABORTED、クライアントからの読み込みでエラーが発生した場合は REQ_EXIT の結果コードを返します。

パラメータ

`char *name` は、ヘッダーの名前です。

`char **value` は、この関数が指定されたヘッダーの値を配置する場所のアドレスです。ヘッダーが見つからない場合は、NULL を格納します。

`Session *sn` は、Session です。

`Request *rq` は、Request です。

`Session` および `Request` パラメータは、カスタム SAF に渡されるものと同じです。

関連項目

`request_create`、`request_free`

request_stat_path

`request_stat_path` 関数は、指定されたパスのファイル情報構造体を返し、パスが指定されない場合は、指定された Request 構造体内の `vars` pblock 内の `path` エントリを返します。結果のファイル名が、サーバが読み取ることができるファイルを指している場合は、`request_stat_path` は新しいファイル情報構造体を返します。この構造体には、ファイルのサイズ、所有者、作成日時、および最終更新の日時の情報が含まれています。

現在アクセスしているファイルの情報を取得するには、(直接 `stat` を呼び出す代わりに) `request_stat_path` を使う必要があります。これは、この関数が、同じパスに対する前の呼び出しを追跡し、キャッシュに書き込んでおいた情報を返すからです。

構文

```
struct stat *request_stat_path(char *path, Request *rq);
```

戻り値

`path` パラメータで指定されたファイルのファイル情報構造体へのポインタを返します。この構造体を解放してはなりません。ファイルが無効、またはサーバがファイルを読み取ることができない場合は、NULL を返します。この場合、問題を説明するエラーメッセージを `rq->staterr` に残します。

パラメータ

`char *path` は、パスの名前が含まれる文字列です。`path` の値が NULL である場合は、この関数は `rq` が示す Request 構造体内の `vars` pblock 内の `path` エントリを使います。

`Request *rq` は、サーバアプリケーションの関数呼び出しに対する要求識別子です。

例

```
fi = request_stat_path(path, rq);
```

関連項目

request_create、request_free、request_header

request_translate_uri

request_translate_uri 関数は、指定されたセッションで指定された URI を仮想から物理へマッピングします。指定された URI がアクセスされたときに、どのファイルを送り返すべきか判断する際に、この関数を使います。

構文

```
char *request_translate_uri(char *uri, Session *sn);
```

戻り値

マッピングを実行した場合はパスの文字列を返し、マッピングを実行できなかった場合は NULL を返します。

パラメータ

char *uri は、URI の名前です。

Session *sn は、カスタム SAF に渡される Session パラメータです。

関連項目

request_create、request_free、request_header

S

session_dns

session_dns 関数は、指定されたセッションに関連するクライアントの IP アドレスを DNS 名に解釈処理します。この関数は、新たに割り当てられた文字列を返します。session_dns を使って、数字の IP アドレスを読みやすいものに変えることができます。

session_maxdns 関数は、クライアントが、本当に表明しているクライアントであるかを検証します。session_dns 関数は、この検証を実行しません。

注	この関数は、magnus.conf ファイルで DNS 指令が有効になっている場合にだけ機能します。詳細は、第 7 章「magnus.conf の構文と使用法」を参照してください。

構文

```
char *session_dns(Session *sn);
```

戻り値

ホスト名が含まれる文字列を返し、IP アドレスに対応する DNS 名が見つからない場合は NULL を返します。

パラメータ

`Session *sn` は、Session です。

`Session` は、カスタム SAF に渡されるものと同じです。

session_maxdns

`session_maxdns` 関数は、指定されたセッションに関連するクライアントの IP アドレスを DNS 名に解釈処理します。この関数は、新たに割り当てられた文字列を返します。`session_maxdns` を使って、数字の IP アドレスを読みやすい形式に変えることができます。

注	この関数は、`magnus.conf` ファイルで `DNS` 指令が有効になっている場合にだけ機能します。詳細は、第 7 章「magnus.conf の構文と使用法」を参照してください。

構文

```
char *session_maxdns(Session *sn);
```

戻り値

ホスト名が含まれる文字列を返し、IP アドレスに対応する DNS 名が見つからない場合は NULL を返します。

パラメータ

`Session *sn` は、Session です。

`Session` は、カスタム SAF に渡されるものと同じです。

shexp_casecmp

`shexp_casecmp` 関数は、指定されたシェル表現を検証し、指定された文字列と比較します。この関数は、一致、一致するものなし、および無効な比較を表す 3 つの値のどれか 1 つを返します。この比較では、(`shexp_cmp` 関数とは異なり) 大文字 - 小文字は区別されません。

*.netscape.com のようなシェル表現があり、foo.netscape.com などの文字列がそれに一致することを確認したい場合に、この関数を使います。

構文

```
int shexp_casecmp(char *str, char *exp);
```

戻り値

一致するものが見つかった場合は、0 を返します。

一致するものが見つからなかった場合は、1 を返します。

比較の結果、表現が無効とみなされた場合は、-1 を返します。

パラメータ

char *str は、比較される文字列です。

char *exp は、それに照らして比較されるシェル表現 (ワイルドカードパターン) です。

関連項目

shexp_cmp、shexp_match、shexp_valid

shexp_cmp

shexp_cmp 関数は、指定されたシェル表現を検証し、指定された文字列と比較します。この関数は、一致、一致するものなし、および無効な比較を表す 3 つの値のどれか 1 つを返します。この比較では、(shexp_casecmp 関数とは異なり) 大文字 - 小文字が区別されます。

*.netscape.com のようなシェル表現があり、foo.netscape.com などの文字列がそれに一致することを確認したい場合に、この関数を使います。

構文

```
int shexp_cmp(char *str, char *exp);
```

戻り値

一致するものが見つかった場合は、0 を返します。

一致するものが見つからなかった場合は、1 を返します。

比較の結果、表現が無効とみなされた場合は、-1 を返します。

パラメータ

char *str は、比較される文字列です。

char *exp は、それに照らして比較されるシェル表現 (ワイルドカードパターン) です。

例

```
/* このパスが必要としているパスであるかどうかを確認するために、ワイルドカードによるマッチングを使用する */
char *path;
char *match = "/usr/netscape/*";
if (shexp_cmp(path, match) != 0)
       return REQ_NOACTION;    /* 一致するものなし */
```

関連項目

shexp_casecmp、shexp_match、shexp_valid

shexp_match

shexp_match 関数は、指定された事前検証済みのシェル表現を指定された文字列と比較します。この関数は、一致、一致するものなし、および無効な比較を表す 3 つの値のどれか 1 つを返します。この比較では、(shexp_casecmp 関数とは異なり) 大文字 - 小文字が区別されます。

shexp_match 関数は、シェル表現の検証は実行しません。代わりに、この関数は、すでに shexp_valid が呼び出されたものとみなします。

*.netscape.com のようなシェル表現があり、foo.netscape.com などの文字列がそれに一致することを確認したい場合に、この関数を使います。

構文

```
int shexp_match(char *str, char *exp);
```

戻り値

一致するものが見つかった場合は、0 を返します。

一致するものが見つからなかった場合は、1 を返します。

比較の結果、表現が無効とみなされた場合は、-1 を返します。

パラメータ

char *str は、比較される文字列です。

char *exp は、それに照らして比較される、事前検証済みのシェル表現 (ワイルドカードパターン) です。

関連項目

shexp_casecmp、shexp_cmp、shexp_valid

shexp_valid

shexp_valid 関数は、exp で指定された、指定されたシェル表現を検証します。shexp_match 関数を使って表現を文字列と比較する前に、この関数を使ってシェル表現を検証します。

構文

```
int shexp_valid(char *exp);
```

戻り値

exp が標準の文字列である場合は、定数の NON_SXP を返します。

exp がシェル表現であるが無効な場合は、定数の INVALID_SXP を返します。

exp が有効なシェル表現である場合は、定数の VALID_SXP を返します。

パラメータ

char *exp は、それに照らして検証されるシェル表現 (ワイルドカードパターン) です。

関連項目

shexp_casecmp、shexp_match、shexp_valid

STRDUP

STRDUP マクロは、C ライブラリルーチンの strdup のプラットフォームに依存しないものです。この関数は、要求のメモリープールに文字列の新しいコピーを作成します。

STRDUP ルーチンの機能は、次の関数のシーケンスを実行する場合と同じです。

```
newstr = (char *) MALLOC(strlen(str) + 1);
strcpy(newstr, str);
```

STRDUP で作成された文字列は、FREE で破棄する必要があります。

構文

```
char *STRDUP(char *ptr);
```

戻り値

新しい文字列へのポインタを返します。

パラメータ

char *ptr は、文字列へのポインタです。

例

```
char *name1 = "MyName";
char *name2 = STRDUP(name1);
```

関連項目

MALLOC、FREE、CALLOC、REALLOC、PERM_MALLOC、PERM_FREE、PERM_CALOC、PERM_REALLOC、PERM_STRDUP

system_errmsg

system_errmsg 関数は、最後のシステムコールから発生した最後のエラーを返します。この関数は、グローバル配列 sys_errlist からのエントリを返すマクロとして実装されています。このマクロは、入出力エラーの診断に役立ちます。

構文

```
char *system_errmsg(int param1);
```

戻り値

システムコールの結果である最後のエラーメッセージのテキストが入っている文字列を返します。この文字列を解放してはなりません。

パラメータ

int param1 は予約済みであり、値は常に 0 である必要があります。

関連項目

system_fopenRO、system_fopenRW、system_fopenWA、system_lseek、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_fclose

system_fclose 関数は、指定されたファイル記述子を閉じます。system_fopen 関数で開いたすべてのファイル記述子に対して、system_fclose 関数を呼び出す必要があります。

構文

```
int system_fclose(SYS_FILE fd);
```

戻り値

閉じるのに成功した場合は 0、閉じるのに失敗した場合は定数の IO_ERROR を返します。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

例

```
SYS_FILE logfd;
system_fclose(logfd);
```

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_lseek、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_ulock

system_flock

system_flock 関数は、ほかのプロセスからの干渉を防止するために、指定されたファイルをロックします。現在開いてるファイルをほかのプロセスが使うことがないようにする場合は、system_flock を使います。ファイルロックを使いすぎると、パフォーマンスが低下し、デッドロックが生じることがあります。

構文

```
int system_flock(SYS_FILE fd);
```

戻り値

ロックが成功した場合は定数の IO_OKAY、ロックが失敗した場合は定数の IO_ERROR を返します。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_lseek、system_fread、system_fwrite、system_fwrite_atomic、system_ulock、system_fclose

system_fopenRO

system_fopenRO 関数は、path で示されたファイルを読み取り専用モードで開き、有効なファイル記述子を返します。ユーザのプログラムで変更されないファイルを開くには、この関数を使います。また、system_fopenRO を使って、filebuf_open を使用する新しいファイルバッファ構造体を開くことができます。

構文

```
SYS_FILE system_fopenRO(char *path);
```

戻り値

開くことに成功した場合はシステムに依存しないファイル記述子 (SYS_FILE)、開くことに失敗した場合は 0 を返します。

パラメータ

char *path は、ファイル名です。

関連項目

system_errmsg、system_fopenRW、system_fopenWA、system_lseek、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_fopenRW

system_fopenRW 関数は、path で示されたファイルを読み込み - 書き込みモードで開き、有効なファイル記述子を返します。ファイルがすでに存在する場合は、system_fopenRW はそれを切り捨てません。ユーザのプログラムでの読み込み - 書き込みの対象になるファイルを開くには、この関数を使います。

構文

```
SYS_FILE system_fopenRW(char *path);
```

戻り値

開くことに成功した場合はシステムに依存しないファイル記述子 (SYS_FILE)、開くことに失敗した場合は 0 を返します。

パラメータ

char *path は、ファイル名です。

例

```
SYS_FILE fd;
fd = system_fopenRO(pathname);
if (fd == SYS_ERROR_FD)
       break;
```

関連項目

system_errmsg、system_fopenRO、system_fopenWA、system_lseek、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_fopenWA

system_fopenWA 関数は、path で示されたファイルを書き込み - 付加モードで開き、有効なファイル記述子を返します。ユーザのプログラムがデータを付加するファイルを開くには、この関数を使います。

構文

```
SYS_FILE system_fopenWA(char *path);
```

戻り値

開くことに成功した場合はシステムに依存しないファイル記述子 (SYS_FILE)、開くことに失敗した場合は 0 を返します。

パラメータ

char *path は、ファイル名です。

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_lseek、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_fread

system_fread 関数は、指定されたバイトの数を指定されたファイルから指定されたバッファに読み込みます。読み込んだバイトの数を返します。system_fopenWA 以外のいずれかの system_fopen 関数を使ってファイルを開いてからでなければ、system_fread を使うことはできません。

構文

```
int system_fread(SYS_FILE fd, char *buf, int sz);
```

戻り値

読み込まれたバイトの数を返します。その数の文字を取得する前にエラーが発生したかファイルの終わりに達した場合は、要求されたサイズよりも小さいことがあります。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

char *buf は、バイトを受け取るバッファです。

int sz は、読み込むバイトの数です。

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_lseek、system_fwrite、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_fwrite

system_fwrite 関数は、指定されたバイトの数を指定されたバッファから指定されたファイルに書き込みます。

system_fopenRO 以外のいずれかの system_fopen 関数を使ってファイルを開いてからでなければ、system_fwrite を使うことはできません。

構文

```
int system_fwrite(SYS_FILE fd, char *buf, int sz);
```

戻り値

書き込みが成功した場合は定数の IO_OKAY、書き込みが失敗した場合は定数の IO_ERROR を返します。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

char *buf は、書き込み対象のバイトが入っているバッファです。

int sz は、ファイルへ書き込むバイトの数です。

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_lseek、system_fread、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_fwrite_atomic

system_fwrite_atomic 関数は、指定されたバイトの数を指定されたバッファから指定されたファイルに書き込みます。この関数は、書き込みの前にファイルをロックし、書き込みが完了したらロックを解除して、同時の書き込み操作による干渉を防止します。system_fopenRO 以外のいずれかの system_fopen 関数を使ってファイルを開いてからでなければ、system_fwrite_atomic を使うことはできません。

構文

```
int system_fwrite_atomic(SYS_FILE fd, char *buf, int sz);
```

戻り値

書き込み / ロックが成功した場合は定数の IO_OKAY、書き込み / ロックが失敗した場合は定数の IO_ERROR を返します。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

char *buf は、書き込み対象のバイトが入っているバッファです。

int sz は、ファイルへ書き込むバイトの数です。

例

```
SYS_FILE logfd;

char *logmsg = "An error occurred.";
system_fwrite_atomic(logfd, logmsg, strlen(logmsg));
```

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_lseek、system_fread、system_fwrite、system_flock、system_ulock、system_fclose

system_gmtime

system_gmtime 関数は、標準の gmtime 関数をスレッド安全にしたものです。グリニッチ標準時 (Greenwich Mean Time : GMT) に調整した現在の時刻を返します。

構文

```
struct tm *system_gmtime(const time_t *tp, const struct tm *res);
```

戻り値

GMT 時間を含む暦時間 (tm) 構造体へのポインタを返します。使用しているシステムにより、このポインタは、2 番目のパラメータが表すデータ項目を指すか、または静的に割り当てられた項目を指すことがあります。可搬性のためには、どちらか一方の状況のみを想定してはなりません。

パラメータ

time_t *tp は、算術時間です。

tm *res は、暦時間 (tm) 構造体へのポインタです。

例

```
time_t tp;
struct tm res, *resp;
tp = time(NULL);
resp = system_gmtime(&tp, &res);
```

関連項目

system_localtime、util_strftime

system_localtime

system_localtime 関数は、標準の localtime 関数をスレッド安全にしたものです。現在の時間を現地時間で返します。

構文

```
struct tm *system_localtime(const time_t *tp, const struct tm *res);
```

戻り値

現地時間が含まれている暦時間 (tm) 構造体へのポインタを返します。使用しているシステムにより、このポインタは、2 番目のパラメータが表すデータ項目を指すか、または静的に割り当てられた項目を指すことがあります。可搬性のためには、どちらか一方の状況のみを想定してはなりません。

パラメータ

time_t *tp は、算術時間です。

tm *res は、暦時間 (tm) 構造体へのポインタです。

関連項目

system_gmtime、util_strftime

system_lseek

system_lseek 関数は、ファイルのファイル位置を設定します。これは、system_fread または system_fwrite からのデータが読み込まれるか、または書き込まれる位置に影響します。

構文

```
int system_lseek(SYS_FILE fd, int offset, int whence);
```

戻り値

操作が成功した場合は、ファイルの先頭からの新しい位置のバイト単位のオフセット、操作が失敗した場合は -1 を返します。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

int offset は、whence に相対するバイトの数です。負数の場合もあります。

int whence は、次の定数のいずれかです。

SEEK_SET、ファイルの先頭から。

SEEK_CUR、現在のファイル位置から。

SEEK_SET、ファイルの終わりから。

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_ulock、system_fclose

system_rename

system_rename 関数は、ファイルの名前を変更します。古いディレクトリと新しいディレクトリが異なるファイルシステムにある場合、そのディレクトリでは、この関数が機能しないことがあります。

構文

```
int system_rename(char *old, char *new);
```

戻り値

操作が成功した場合は 0、操作が失敗した場合は -1 を返します。

パラメータ

char *old は、ファイルの古い名前です。

char *new は、ファイルの新しい名前です。

system_ulock

system_ulock 関数は、system_flock 関数でロックされた、指定されたファイルのロックを解除します。ロックについての詳細は、system_flock を参照してください。

構文

```
int system_ulock(SYS_FILE fd);
```

戻り値

操作が成功した場合は定数の IO_OKAY、操作が失敗した場合は定数の IO_ERROR を返します。

パラメータ

SYS_FILE fd は、プラットフォームに依存しないファイル記述子です。

関連項目

system_errmsg、system_fopenRO、system_fopenRW、system_fopenWA、system_fread、system_fwrite、system_fwrite_atomic、system_flock、system_fclose

system_unix2local

system_unix2local 関数は、指定された UNIX 形式のパス名をローカルファイルシステムのパス名に変換します。ファイル名が UNIX 形式 (スラッシュが含まれているものなど) になっている場合に、Windows NT などの別のシステム上のファイルにアクセスする必要があるときに、この関数を使います。system_unix2local を使って、UNIX 形式のファイル名を Windows NT が受け入れる形式に変換できます。UNIX 環境では、この関数は何も実行しませんが、可搬性のために呼び出すことができます。

構文

```
char *system_unix2local(char *path, char *lp);
```

戻り値

ローカルファイルシステムのパスの文字列へのポインタを返します。

パラメータ

char *path は、変換される、UNIX 形式のパス名です。

char *lp は、ローカルパス名です。

パラメータ lp を割り当てる必要があり、このパラメータにはローカルパス名を入れるのに十分な領域が必要です。

関連項目

system_fclose、system_flock、system_fopenRO、system_fopenRW、system_fopenWA、system_fwrite

systhread_attach

systhread_attach 関数は、既存のスレッドをプラットフォームに依存しないスレッドにします。

構文

```
SYS_THREAD systhread_attach(void);
```

戻り値
プラットフォームに依存しないスレッドへの SYS_THREAD ポインタを返します。

パラメータ
なし。

関連項目
systhread_current、systhread_getdata、systhread_init、
systhread_newkey、systhread_setdata、systhread_sleep、
systhread_start、systhread_timerset

systhread_current
systhread_current 関数は、現在のスレッドへのポインタを返します。

構文
```
SYS_THREAD systhread_current(void);
```

戻り値
現在のスレッドへの SYS_THREAD ポインタを返します。

パラメータ
なし。

関連項目
systhread_getdata、systhread_newkey、systhread_setdata、
systhread_sleep、systhread_start、systhread_timerset

systhread_getdata
systhread_getdata 関数は、現在のスレッド内で指定された鍵に関連付けられたデータを取得します。

構文
```
void *systhread_getdata(int key);
```

戻り値
呼び出しが成功した場合は、key の同じ値を使って、現在のスレッドからの、systhread_setkey 関数で前に使ったデータへのポインタを返します。たとえば、systhread_setkey 関数が、このセッション中に指定された鍵で呼び出されることがなかった場合など、呼び出しが失敗した場合は、NULL を返します。

パラメータ

int key は、systhread_setdata 関数で格納されたデータに関連付けられた値です。鍵は、systhread_newkey 関数で割り当てられます。

関連項目

systhread_current、systhread_newkey、systhread_setdata、systhread_sleep、systhread_start、systhread_timerset

systhread_newkey

systhread_newkey 関数は、スレッド専用データに新しい整数鍵 (識別子) を割り当てます。この鍵を使って、現在のスレッドにローカライズしたい変数を識別します。次に、systhread_setdata 関数を使って、値を鍵に関連付けます。

構文

```
int systhread_newkey(void);
```

戻り値

整数鍵を返します。

パラメータ

なし。

関連項目

systhread_current、systhread_getdata、systhread_setdata、systhread_sleep、systhread_start、systhread_timerset

systhread_setdata

systhread_setdata 関数は、データに現在のスレッドの指定された鍵番号を関連付けます。鍵は、systhread_newkey 関数で割り当てられます。

構文

```
void systhread_setdata(int key, void *data);
```

戻り値

void を返します。

パラメータ

int key は、スレッドの優先順位です。

void *data は、key の値に関連付けられるデータの文字列へのポインタです。

関連項目

systhread_current、systhread_getdata、systhread_newkey、
systhread_sleep、systhread_start、systhread_timerset

systhread_sleep

systhread_sleep 関数は、呼び出し元スレッドを指定された期間休眠させます。

構文

```
void systhread_sleep(int milliseconds);
```

戻り値

void を返します。

パラメータ

int milliseconds は、スレッドを休眠させるミリ秒単位の期間です。

関連項目

systhread_current、systhread_getdata、systhread_newkey、
systhread_setdata、systhread_start、systhread_timerset

systhread_start

systhread_start 関数は、指定された優先順位でスレッドを作成し、指定されたバイト数のスタックを割り当て、指定された引数で指定された関数を呼び出します。

構文

```
SYS_THREAD systhread_start(int prio, int stksz,
       void (*fn)(void *), void *arg);
```

戻り値

呼び出しが成功した場合は新しい SYS_THREAD ポインタ、呼び出しが失敗した場合は定数の SYS_THREAD_ERROR を返します。

パラメータ

int prio は、スレッドの優先順位です。優先順位は、システムに依存します。

int stksz は、バイト単位のスタックのサイズです。stksz が 0 の場合は、この関数はデフォルトのサイズを割り当てます。

void (*fn)(void *) は、呼び出す関数です。

void *arg は、fn 関数の引数です。

関連項目

systhread_current、systhread_getdata、systhread_newkey、
systhread_setdata、systhread_sleep、systhread_timerset

systhread_timerset

systhread_timerset 関数は、スレッドシステムに対する割り込みタイマー間隔を開始またはリセットします。

ほとんどのシステムではタイマー間隔の変更は許可されていないので、この関数は、必ず実行しなければならないものではなく、提案とみなしてください。

構文

```
void systhread_timerset(int usec);
```

戻り値

void を返します。

パラメータ

int usec は、ミリ秒単位の時間です。

関連項目

systhread_current、systhread_getdata、systhread_newkey、
systhread_setdata、systhread_sleep、systhread_start

U

util_can_exec

UNIX のみ

util_can_exec 関数は、指定されたファイルを実行できるかどうかを調べ、1 (実行可能) または 0 を返します。この関数は、指定されたユーザおよびグループ ID でユーザがファイルを実行できるかどうかを調べます。

exec システムコールを使ってプログラムを実行する前に、この関数を使います。

構文

```
int util_can_exec(struct stat *finfo, uid_t uid, gid_t gid);
```

戻り値

ファイルが実行可能である場合は 1、ファイルが実行可能でない場合は 0 を返します。

パラメータ

`stat *finfo` は、ファイルに関連付けられた stat 構造体です。

`uid_t uid` は、UNIX のユーザ ID です。

`gid_t gid` は、UNIX のグループ ID です。`uid` とともに、`gid_t gid` は、UNIX ユーザのアクセス権を判定します。

関連項目

`util_env_create`、`util_getline`、`util_hostname`

util_chdir2path

`util_chdir2path` 関数は、現在のディレクトリを、ファイルにアクセスする指定されたディレクトリに変更します。

Windows NT 下で実行する場合は、クリティカルセクションを使って、複数のスレッドがこの関数を同時に呼び出すことがないようにします。

ファイルアクセスを若干速くしたい場合は、絶対パスを使う必要がないので、`util_chdir2path` を使います。

構文

```
int util_chdir2path(char *path);
```

戻り値

ディレクトリを変更した場合は 0、ディレクトリを変更できなかった場合は -1 を返します。

パラメータ

`char *path` は、ディレクトリの名前です。

このパラメータは、書き込み可能な文字列である必要があります。

util_cookie_find

`util_cookie_find` 関数は、cookie 文字列で特定の cookie を検索し、その値を返します。

構文

```
char *util_cookie_find(char *cookie, char *name);
```

戻り値

成功した場合、cookie の `NULL` で終わっている値へのポインタを返します。失敗した場合は、`NULL` を返します。この関数は、名前と値を NULL で終わらせて、cookie 文字列パラメータを変更します。

パラメータ

`char *cookie` は、Cookie: 要求ヘッダーの値です。

`char *name` は、値が取得対象の cookie の名前です。

util_env_find

`util_env_find` 関数は、名前で示された文字列を指定された環境で探し、関連付けられた値を返します。環境内のエントリを探すには、この関数を使います。

構文

```
char *util_env_find(char **env, char *name);
```

戻り値

見つかった場合は環境変数の値、文字列が見つからない場合は NULL を返します。

パラメータ

`char **env` は、環境です。

`char *name` は、env 内の環境変数の名前です。

関連項目

`util_env_replace`、`util_env_str`、`util_env_free`、`util_env_create`

util_env_free

`util_env_free` 関数は、指定された環境を解放します。`util_env_create` 関数を使って作成した環境の割り当てを解除するには、この関数を使います。

構文

```
void util_env_free(char **env);
```

戻り値

`void` を返します。

パラメータ

`char **env` は、解放する環境です。

関連項目

`util_env_replace`、`util_env_str`、`util_env_find`、`util_env_create`

util_env_replace

util_env_replace 関数は、指定された環境内にある名前で示される変数を、指定された値で置き換えます。環境内の設定の値を変更するには、この関数を使います。

構文

```
void util_env_replace(char **env, char *name, char *value);
```

戻り値

void を返します。

パラメータ

char **env は、環境です。

char *name は、名前 - 値ペアの名前です。

char *value は、格納する新しい値です。

関連項目

util_env_str、util_env_free、util_env_find、util_env_create

util_env_str

util_env_str 関数は、環境エントリを作成し、そのエントリを返します。この関数は、名前に含まれる英数字でない記号 (等号 = など) を検査しません。この関数を使って、新しい環境エントリを作成できます。

構文

```
char *util_env_str(char *name, char *value);
```

戻り値

名前 - 値ペアが入っている新しく割り当てられた文字列を返します。

パラメータ

char *name は、名前 - 値ペアの名前です。

char *value は、格納される新しい値です。

関連項目

util_env_replace、util_env_free、util_env_find、util_env_create

util_getline

util_getline 関数は、改行、またはキャリッジリターン / 改行で終わる文字列を検索するために、指定されたファイルバッファを走査します。文字列が指定されたバッファにコピーされ、NULL で終わらせます。この関数は、文字列がバッファに格納されたか、エラーが検出されたか、またはファイルの終わりに達したかどうかを示す値を返します。

構成ファイルなどのテキストファイルからの行を走査するには、この関数を使います。

構文

```
int util_getline(filebuf *buf, int lineno, int maxlen, char *line);
```

戻り値

成功した場合は、0 を返します。line に文字列が入っています。

ファイルの終わりに達した場合は、1 を返します。line に文字列が入っています。

エラーが発生した場合は -1 を返します。line には、エラーの説明が入っています。

パラメータ

filebuf *buf は、走査されるファイルバッファです。

int lineno は、エラーが発生したときに、エラーメッセージに行番号をインクルードする _ ために使います。呼び出し元は、行番号が確実に正しくなるようにする必要があります。

int maxlen は、line に書き込むことのできる文字の最大個数です。

char *line は、文字列を格納するバッファです。line の割り当てと割り当て解除は、ユーザの責任において行ないます。

関連項目

util_can_exec、util_env_create、util_hostname

util_hostname

util_hostname 関数は、ローカルホスト名を取得し、そのホスト名を文字列として返します。この関数は、絶対パスによるドメイン名を見つけることができなかった場合は、NULL を返します。この文字列を割り当て直すか、または解放できます。使用しているシステムの名前を調べるには、この関数を使います。

構文

```
char *util_hostname(void);
```

戻り値
絶対パスによるドメイン名が見つかった場合は、その名前が入っている文字列を返し、見つからなかった場合は NULL を返します。

パラメータ
なし。

util_is_mozilla
util_is_mozilla 関数は、指定されたユーザ - エージェントのヘッダー文字列が、指定されたバージョン以上の Netscape ブラウザであるかどうかを調べ、そうである場合は 1、そうでない場合は 0 を返します。この関数は、1.56 > 1.5 のような曖昧さを避けるために、バージョンを指定するために文字列を使います。

構文
```
int util_is_mozilla(char *ua, char *major, char *minor);
```

戻り値
ユーザ - エージェントが Netscape ブラウザである場合は 1、そうでない場合は 0 を返します。

パラメータ
char *ua は、要求ヘッダーからのユーザ - エージェント文字列です。

char *major は、メジャーリリース番号 (小数点の左側) です。

char *minor は、マイナーリリース番号 (小数点の右側) です。

関連項目
util_is_url、util_later_than

util_is_url
util_is_url 関数は、文字列が URL であるかどうかを調べ、そうである場合は 1 を、そうでない場合は 0 を返します。英字で始まり後にコロンが続く場合は、文字列は URL です。

構文
```
int util_is_url(char *url);
```

戻り値
url で指定された文字列が URL である場合は 1、url で指定された文字列が URL でない場合は 0 を返します。

パラメータ

`char *url` は、調査される文字列です。

関連項目

`util_is_mozilla`、`util_later_than`

util_itoa

`util_itoa` 関数は、指定された整数を文字列に変換し、文字列の長さを返します。数のテキスト表現を作成するには、この関数を使います。

構文

```
int util_itoa(int i, char *a);
```

戻り値

作成された文字列の長さを返します。

パラメータ

`int i` は、変換される整数です。

`char *a` は、値を表す ASCII 文字列です。`a` の割り当てと割り当て解除はユーザの責任において行ない、また `a` の長さは少なくとも 32 バイトである必要があります。

util_later_than

`util_later_than` 関数は、time 構造体に指定された日付を、文字列内に指定された日付と比較します。文字列中の日付が time 構造体内の日付に等しいかそれ以降である場合は、この関数は 1 を返します。この関数は、RFC 822、RFC 850、および ctime 書式を処理するために使います。

構文

```
int util_later_than(struct tm *lms, char *ims);
```

戻り値

`ims` で表された日付が `lms` で表された日付と等しいかそれ以降である場合は 1、`ims` で表された日付が `lms` で表された日付よりも前の日付である場合は 0 を返します。

パラメータ

`tm *lms` は、日付が含まれている time 構造体です。

`char *ims` は、日付が含まれる文字列です。

関連項目

`util_strftime`

util_sh_escape

util_sh_escape 関数は、指定された文字列を構文解析し、シェル特殊文字の前にバックスラッシュ (\) を置き、その結果の文字列を返します。クライアントからの文字列が、シェルに予想外のことをさせないことを確実にするには、この関数を使います。

シェル特殊文字は、空白文字に以下の文字を加えたものです。

```
&;`'"|*?~<>^()[]{}$\#!
```

構文

```
char *util_sh_escape(char *s);
```

戻り値

新しく割り当てられた文字列を返します。

パラメータ

char *s は、構文解析される文字列です。

関連項目

util_uri_escape

util_snprintf

util_snprintf 関数は、printf 形式の構文を使い、範囲の確認を実行して、指定された文字列を指定された形式でフォーマットして、指定されたバッファに入れます。フォーマットされたバッファ内の文字の数を返します。

詳細は、使用しているコンパイラの実行時ライブラリの printf 関数についてのマニュアルを参照してください。

構文

```
int util_snprintf(char *s, int n, char *fmt, ...);
```

戻り値

フォーマットされバッファに入れられた文字の数を返します。

パラメータ

char *s は、フォーマットされた文字列を受け取るバッファです。

int n は、コピーできるバイトの最大数です。

char *fmt は、書式文字列です。この関数は、%d および %s 文字列のみを処理します。幅や精度の文字列は処理しません。

... は、printf 関数の一連のパラメータを表します。

関連項目

util_sprintf、util_vsnprintf、util_vsprintf

util_sprintf

util_sprintf 関数は、printf 形式の構文を使い、範囲の確認を実行しないで、指定された文字列を指定された形式でフォーマットして、指定されたバッファに入れます。フォーマットされたバッファ内の文字の数を返します。

util_sprintf は範囲の確認を実行しないので、文字列がバッファに収まることが確かな場合にのみ、この関数を使います。そうでない場合は、util_snprintf 関数を使います。詳細は、使用しているコンパイラの実行時ライブラリの printf 関数についてのマニュアルを参照してください。

構文

```
int util_sprintf(char *s, char *fmt, ...);
```

戻り値

フォーマットされバッファに入れられた文字の数を返します。

パラメータ

char *s は、フォーマットされた文字列を受け取るバッファです。

char *fmt は、書式文字列です。この関数は、%d および %s 文字列のみを処理します。幅や精度の文字列は処理しません。

... は、printf 関数の一連のパラメータを表します。

例

```
char *logmsg;
int len;

logmsg = (char *) MALLOC(256);
len = util_sprintf(logmsg, "%s %s %s\n", ip, method, uri);
```

関連項目

util_snprintf、util_vsnprintf、util_vsprintf

util_strcasecmp

util_strcasecmp 関数は、2 つの英数字文字列を比較し、どちらが大きいか、または同一であるかを示す -1、0、または 1 を返します。

比較では、大文字 - 小文字は区別されません。

構文
```
int util_strcasecmp(const char *s1, const char *s2);
```

戻り値
s1 が s2 よりも大きい場合は、1 を返します。

s1 が s2 に等しい場合は、0 を返します。

s1 が s2 よりも小さい場合は、-1 を返します。

パラメータ
char *s1 は、最初の文字列です。

char *s2 は、2 番目の文字列です。

関連項目
util_strncasecmp

util_strftime
util_strftime 関数は、システム時間を記述する構造体である tm 型の構造体をテキスト表現に変換します。この関数は、標準の strftime 関数をスレッド安全にしたものです。

構文
```
int util_strftime(char *s, const char *format, const struct tm *t);
```

戻り値
s に入れられた文字の数を返します。終わりの NULL 文字は、この数には含まれません。

パラメータ
char *s は、テキストを入れる文字列バッファです。範囲の確認は行なわれないので、必ずバッファが日付のテキストを入れるのに十分な大きさであるようにします。

const char *format は書式文字列であり、特定の %x 部分文字列付きのテキストから構成されているという点で、少し printf 文字列に似ています。定数の HTTP_DATE_FMT を使って、標準のインターネット形式で日付文字列を作成できます。詳細は、使用しているコンパイラの実行時ライブラリの printf 関数についてのマニュアルを参照してください。時刻形式の詳細は、付録 D「時刻の書式」を参照してください。

const struct tm *t は、暦時間 (tm) 構造体へのポインタであり、この構造体は通常、関数 system_localtime または system_gmtime で作成されたものです。

関連項目

system_localtime、system_gmtime

util_strncasecmp

util_strncasecmp 関数は、英数字文字列内の最初の n 文字を比較し、どちらが大きいか、または同一であるかを示す -1、0、または 1 を返します。

この関数の比較では、大文字 - 小文字は区別されません。

構文

```
int util_strncasecmp(const char *s1, const char *s2, int n);
```

戻り値

s1 が s2 よりも大きい場合は、1 を返します。

s1 が s2 に等しい場合は、0 を返します。

s1 が s2 よりも小さい場合は、-1 を返します。

パラメータ

char *s1 は、最初の文字列です。

char *s2 は、2 番目の文字列です。

int n は、比較する、最初の文字の数です。

関連項目

util_strcasecmp

util_uri_escape

util_uri_escape 関数は、URI に含まれている特殊文字を URI 形式 (%XX、ここで XX は ASCII 文字に対応する 16 進数) に変換し、エスケープされた文字列を返します。特殊文字には、%?#:+&*"<>、空白文字、キャリッジリターン、および改行があります。

URI をクライアントへ送り返す前に、util_uri_escape を使います。

構文

```
char *util_uri_escape(char *d, char *s);
```

戻り値

エスケープされた文字で置き換えられた文字列 (おそらく新たに割り当てられたもの) を返します。

パラメータ

`char *d` は、文字列です。`d` が NULL でない場合は、この関数はフォーマットされた文字列を `d` にコピーしてそれを返します。`d` が NULL である場合は、この関数は適切なサイズにされた文字列を割り当て、フォーマットされた特殊文字を新しい文字列にコピーしてそれを返します。

`util_uri_escape` 関数は、パラメータ `d` に対する範囲を確認しません。したがって、`d` が NULL でない場合、`d` は文字列 `s` の少なくとも 3 倍の大きさである必要があります。

`char *s` は、もとのエスケープされていない URI が含まれている文字列です。

関連項目

`util_uri_is_evil`、`util_uri_parse`、`util_uri_unescape`

util_uri_is_evil

`util_uri_is_evil` 関数は、指定された URI に不確実なパス文字が含まれていないかを調べます。不確実なパス文字には、URI の終わりにある `//`、`/./`、`/../` および `/.`、`/..` (または NT の場合は `./`) などがあります。クライアントが要求した URI が不確実でないかどうかを確認するには、この関数を使います。

構文

```
int util_uri_is_evil(char *t);
```

戻り値

URI が不確実な場合は 1、URI が OK である場合は 0 を返します。

パラメータ

`char *t` は、検査される URI です。

関連項目

`util_uri_escape`、`util_uri_parse`

util_uri_parse

`util_uri_parse` 関数は、指定された URI に含まれている `//`、`/./`、および `/*/../` (`*` は `/` 以外の任意の文字) を、`/` に変換します。この関数を使って、URI の不良シーケンスを有効なシーケンスに変換できます。まず、`util_uri_is_evil` 関数を使って、その関数に不良シーケンスがあるかどうかを調べます。

構文

```
void util_uri_parse(char *uri);
```

戻り値
void を返します。

パラメータ
char *uri は、変換される URI です。

関連項目
util_uri_is_evil、util_uri_unescape

util_uri_unescape
util_uri_unescape 関数は、URI のコード化された文字を対応する ASCII の文字に変換します。コード化された文字は %XX として現れます。ここで XX は、その文字に対応する 16 進数です。

注	NSAPI 関数は、NULL は文字列の終わりであるとみなすので、文字列中に NULL を埋め込むことはできません。したがって、NSAPI プラグインで Unicode でコード化された内容を渡しても動作しません。

構文
```
void util_uri_unescape(char *uri);
```

戻り値
void を返します。

パラメータ
char *uri は、変換される URI です。

関連項目
util_uri_escape、util_uri_is_evil、util_uri_parse

util_vsnprintf
util_vsnprintf 関数は、vprintf 形式の構文を使い、範囲の確認を実行して、指定された文字列を指定された形式でフォーマットして、指定されたバッファに入れます。フォーマットされたバッファ内の文字の数を返します。

詳細は、使用しているコンパイラの実行時ライブラリの vprintf 関数についてのマニュアルを参照してください。

構文

```
int util_vsnprintf(char *s, int n, register char *fmt, va_list
args);
```

戻り値

フォーマットされバッファに入れられた文字の数を返します。

パラメータ

`char *s` は、フォーマットされた文字列を受け取るバッファです。

`int n` は、コピーできるバイトの最大数です。

`register char *fmt` は、書式文字列です。この関数は、`%d` および `%s` 文字列のみを処理します。幅や精度の文字列は処理しません。

`va_list args` は、前の `va_start` の呼び出しで取得した STD 引数変数です。

関連項目

`util_sprintf`、`util_vsprintf`

util_vsprintf

`util_vsprintf` 関数は、`vprintf` 形式の構文を使い、範囲の確認を実行しないで、指定された文字列を指定された形式でフォーマットして、指定されたバッファに入れます。フォーマットされたバッファ内の文字の数を返します。

詳細は、使用しているコンパイラの実行時ライブラリの `vprintf` 関数についてのマニュアルを参照してください。

構文

```
int util_vsprintf(char *s, register char *fmt, va_list args);
```

戻り値

フォーマットされバッファに入れられた文字の数を返します。

パラメータ

`char *s` は、フォーマットされた文字列を受け取るバッファです。

`register char *fmt` は、書式文字列です。この関数は、`%d` および `%s` 文字列のみを処理します。幅や精度の文字列は処理しません。

`va_list args` は、前の `va_start` の呼び出しで取得した STD 引数変数です。

関連項目

`util_snprintf`、`util_vsnprintf`

V

vs_alloc_slot

vs_alloc_slot 関数は、特定の VirtualServer* に固有のデータへのポインタを格納するための新しいスロットを割り当てます。返されるスロット番号は、この後の vs_set_data および vs_get_data の呼び出しで使うことができます。返されるスロット番号は、どの VirtualServer* にも有効です。

ポインタの値 (vs_set_data への呼び出しで返される) は、どの VirtualServer* の場合も、デフォルトで NULL になります。

構文

```
int vs_alloc_slot(void);
```

戻り値

成功した場合はスロット番号、失敗した場合は -1 を返します。

関連項目

vs_get_data、vs_set_data

vs_get_data

vs_get_data 関数は、指定された VirtualServer* と slot のデータへのポインタの値を検索します。slot は、vs_alloc_slot または vs_set_data から返されたスロット番号である必要があります。

構文

```
void* vs_get_data(const VirtualServer* vs, int slot);
```

戻り値

vs_set_data が前に格納したポインタの値を返し、失敗した場合は NULL を返します。

パラメータ

const VirtualServer* vs は、ポインタを照会検索する仮想サーバを表します。

int slot は、ポインタの取得先のスロット番号です。

関連項目

vs_set_data、vs_alloc_slot

vs_get_default_httpd_object

vs_get_default_httpd_object 関数は、仮想サーバの httpd_objset (仮想サーバクラスの obj.conf ファイルで定義された構成) から、デフォルト (またはルート) の httpd_object へのポインタを取得します。デフォルトのオブジェクトの名前は、通常、default です。プラグインは、httpd_object を VSInitFunc 時にのみ変更できます (VSInitFunc 時については、vs_register_cb を参照してください)。

返されたオブジェクトを解放してはなりません。

構文

```
httpd_object* vs_get_default_httpd_object(VirtualServer* vs);
```

戻り値

デフォルトの httpd_object へのポインタを返し、失敗した場合は NULL を返します。このオブジェクトを解放してはなりません。

パラメータ

VirtualServer* vs は、仮想サーバを表し、このサーバのためにデフォルトのオブジェクトを検索します。

関連項目

vs_get_httpd_objset、vs_register_cb

vs_get_doc_root

vs_get_doc_root 関数は、仮想サーバのためにドキュメントルートを検索します。返される文字列は、ドキュメントルートへのオペレーティングシステムの絶対パスです。

呼び出し元は、返された文字列を使い終わったら解放する必要があります。

構文

```
char* vs_get_doc_root(const VirtualServer* vs);
```

戻り値

ドキュメントルートへのオペレーティングシステムの絶対パスを表す文字列へのポインタを返します。この文字列の解放は、呼び出し元の責任において行ないます。

パラメータ

const VirtualServer* vs は、仮想サーバを表し、このサーバのためにドキュメントのルートを検索します。

vs_get_httpd_objset

vs_get_httpd_objset 関数は、指定された仮想サーバの httpd_objset (仮想サーバクラスの obj.conf ファイルで定義された構成) へのポインタを取得します。プラグインは、httpd_objset を VSInitFunc 時にのみ変更できます (VSInitFunc 時については、vs_register_cb を参照してください)。

返された objset を解放してはなりません。

構文

```
httpd_objset* vs_get_httpd_objset(VirtualServer* vs);
```

戻り値

httpd_objset へのポインタを返し、失敗した場合は NULL を返します。この objset を解放してはなりません。

パラメータ

VirtualServer* vs は、仮想サーバを表し、このサーバの objset を検索します。

関連項目

vs_get_default_httpd_object、vs_register_cb

vs_get_id

vs_get_id 関数は、VirtualServer* の ID を検索します。

仮想サーバの ID は、構成間で一定の、一意の NULL で終わる文字列です。ID は構成間で一定ですが、VirtualServer* ポインタの値は変わります。

仮想サーバの ID 文字列を解放してはなりません。要求の処理中に呼び出された場合は、その文字列は現在の要求の間、有効です。VSInitFunc 処理中に呼び出された場合は、この文字列は対応する VSDestroyFunc 関数が復帰するまで有効です (vs_register_cb を参照)。

現在の要求にだけ有効な VirtualServer* を取得するには、request_get_vs を使います。

構文

```
const char* vs_get_id(const VirtualServer* vs);
```

戻り値

仮想サーバ ID を表す文字列へのポインタを返します。この文字列を解放してはなりません。

パラメータ

`const VirtualServer* vs` は、目的の仮想サーバを表します。

関連項目

`vs_register_cb`、`request_get_vs`

vs_get_mime_type

`vs_get_mime_type` 関数は、指定された URI 内の `Content-type:` ヘッダー内に返される MIME タイプを判別します。

呼び出し元は、返された文字列を使い終わったら解放する必要があります。

構文

```
char* vs_get_mime_type(const VirtualServer* vs, const char* uri);
```

戻り値

MIME タイプを表す文字列へのポインタを返します。この文字列の解放は、呼び出し元の責任において行ないます。

パラメータ

`const VirtualServer* vs` は、目的の仮想サーバを表します。

`const char* uri` は、MIME タイプが判別対象である URI です。

vs_lookup_config_var

`vs_lookup_config_var` 関数は、指定された仮想サーバの構成変数の値を検索します。

返された文字列を解放してはなりません。

構文

```
const char* vs_lookup_config_var(const VirtualServer* vs, const
char* name);
```

戻り値

成功した場合は変数名の値を表す文字列へのポインタ、変数名が見つからない場合は `NULL` を返します。この文字列を解放してはなりません。

パラメータ

`const VirtualServer* vs` は、目的の仮想サーバを表します。

`const char *name` は、構成変数の名前です。

vs_register_cb

vs_register_cb 関数は、仮想サーバの初期化イベントおよび廃棄イベントの通知を受ける関数をプラグインが登録できるようにします。vs_register_cb 関数は、通常、magnus.conf 内の Init SAF から呼び出されます。

新しい構成が読み込まれると、新しい構成から要求がサービスを受ける前に、登録されたすべての VSInitFunc (仮想サーバの初期化) コールバックが、各仮想サーバに対して呼び出されます。VSInitFunc コールバックは、登録された順序と同じ順序で呼び出されます。つまり、最初に登録されたコールバックが、最初に呼び出されます。

最後の要求が古い構成からサービスを受けたら、仮想サーバが廃棄される前に、各仮想サーバに対してすべての登録された VSDestroyFunc (仮想サーバの廃棄) コールバックが呼び出されます。VSDestroyFunc コールバックは、逆の順序で呼び出されます。つまり、最初に登録されたコールバックが、最後に呼び出されます。

呼び出し元が初期化または廃棄のためのコールバックを必要としない場合は、それぞれに対応する initfn または destroyfn を NULL にすることができます。

構文

```
int vs_register_cb(VSInitFunc* initfn, VSDestroyFunc* destroyfn);
```

戻り値

操作が成功した場合は、定数の REQ_PROCEED を返します。

操作が失敗した場合は、定数の REQ_ABORTED を返します。

パラメータ

VSInitFunc* initfn は、仮想サーバの初期化時に呼び出す関数へのポインタです。呼び出し元が仮想サーバの初期化イベントを必要としない場合は NULL です。

VSDestroyFunc* destroyfn は、仮想サーバの廃棄時に呼び出す関数へのポインタです。呼び出し元が仮想サーバの廃棄イベントを必要としない場合は NULL です。

vs_set_data

vs_set_data 関数は、指定された仮想サーバおよびスロットのデータへのポインタの値を設定します。*slot は、-1、または vs_alloc_slot から返されるスロット番号でなければなりません。*slot が -1 の場合は、vs_set_data は vs_alloc_slot を暗黙に呼び出し、新しいスロット番号を *slot に返します。

格納されたポインタは、ID ごとではなく、VirtualServer* ごとに維持されます。異なる構成からの、異なる VirtualServer* が同じ仮想サーバ ID で同時に存在できます。ただし、それらは異なる VirtualServer* であるため、それぞれに専用の VirtualServer* 固有のデータがあります。その結果、一般に、vs_set_data を VSInitFunc 処理外で呼び出してはなりません (VSInitFunc 処理については、vs_register_cb を参照してください)。

構文

```
void* vs_set_data(const VirtualServer* vs, int* slot, void* data);
```

戻り値

成功した場合はデータ、失敗した場合は NULL を返します。

パラメータ

const VirtualServer* vs は、仮想サーバを表し、このサーバのためにポインタを設定します。

int slot は、ポインタの格納先のスロット番号です。

void* data は、格納するポインタです。

関連項目

vs_get_data、vs_alloc_slot、vs_register_cb

vs_translate_uri

vs_translate_uri 関数は、特定の仮想サーバに対する要求の一部であるかのように URI を変換します。返される文字列は、オペレーティングシステムの絶対パスです。

呼び出し元は、返された文字列を使い終わったら解放する必要があります。

構文

```
char* vs_translate_uri(const VirtualServer* vs, const char* uri);
```

戻り値

指定された URI に対するオペレーティングシステムの絶対パスを表す文字列へのポインタを返します。この文字列を解放するのは、呼び出し元の責任です。

パラメータ

const VirtualServer* vs は、仮想サーバを表し、このサーバのために URI を変換します。

const char* uri は、オペレーティングシステムのパスに変換する URI です。

第 6 章

カスタム SAF の例

この章では、要求 - 応答プロセスの各指令用のカスタムサーバアプリケーション関数 (Sever Application Functions: SAF) の例について説明します。これらの例を基に、ユーザ独自のカスタム SAF を作成することができます。ユーザ独自のカスタム SAF の作成方法については、第 4 章「カスタム SAF の作成」を参照してください。

注 実際のソースファイル内のコメントはすべて英語ですが、この章では、説明のために日本語にしています。

カスタム SAF を作成する前に、要求 - 応答プロセス (第 1 章「サーバの動作の基本」で説明されている) と構成ファイル `obj.conf` (第 2 章「obj.conf の構文と使用法」で説明されている) の役割を理解しておく必要があります。

ユーザ独自の SAF を作成する前に、既存の SAF で同じ目的が果たせないかを確認します。事前定義済みの SAF の説明は、第 3 章「事前定義済みの SAF および要求処理プロセス」にあります。

新しい SAF を作成するための NSAPI 関数のリストは、第 5 章「NSAPI 関数のリファレンス」を参照してください。

この章は、次の節から構成されています。

- ビルドに含まれている例
- AuthTrans の例
- NameTrans の例
- PathCheck の例
- ObjectType の例
- Service の例
- AddLog の例
- サービス品質の例

ビルドに含まれている例

サーバインストールディレクトリ内の nsapi/examples/ または plugins/nsapi/examples サブディレクトリに、SAF のソースコードの例があります。

同じディレクトリ内の Makefile を使って、例をコンパイルし、すべての例のファイルに含まれる関数が含まれるライブラリを作成することができます。

例をテストするには、magnus.conf の Init セクションに次の指令を追加して、examples 共用ライブラリを iPlanet Web Server に読み込みます。

```
Init fn=load-modules shlib=examples.so/dll
funcs=function1,function2,function3
```

funcs パラメータは、共用ライブラリから読み込む関数を指定します。

例で初期化関数を使っている場合は、load-modules への funcs 引数に初期化関数を指定し、また初期化関数を呼び出すための Init 指令を追加します。

たとえば、PathCheck の例は、acf-init 関数で初期化される restrict-by-acf 関数を実装しています。次の指令は、この両方の関数を読み込みます。

```
Init fn=load-modules yourlibrary funcs=acf-init,restrict-by-acf
```

次の指令は、サーバの初期化時に acf-init 関数を呼び出します。

```
Init fn=acf-init file=extra-arg
```

新しい SAF を応答処理プロセスの適切なステップで呼び出すには、たとえば次のような適切な指令を、その指令が適用されるオブジェクトに追加します。

```
PathCheck fn=restrict-by-acf
```

新しい Init 指令を magnus.conf に追加した後、Init 指令はサーバの初期化時にだけ適用されるので、変更を読み込むために必ず iPlanet Web Server を再起動する必要があります。

AuthTrans の例

この AuthTrans 関数の簡単な例は、リモートクライアントが提供するユーザ名およびパスワードが正しいものであるかどうかを検証するために、ユーザ独自の方法をどのように使用するかを示します。このプログラムは、ハードコード化されたユーザ名およびパスワードのテーブルを使い、与えられたユーザのパスワードを静的なデータ配列にあるパスワードと突き合せて確認します。*userdb* パラメータは、この関数では使われません。

AuthTrans 指令は、PathCheck 指令と連係します。一般に、AuthTrans 関数は、要求に関連付けられたユーザ名およびパスワードを受け入れることができるかどうかを確認しますが、要求へのアクセスの許可や拒否は行ないません -- それについては、PathCheck 関数に任せます。

AuthTrans 関数は、要求に関連するヘッダーからユーザ名とパスワードを取得します。クライアントが初めて要求を発行するときには、ユーザ名とパスワードはわかっていないので、AuthTrans 関数と PathCheck 関数は、ユーザ名およびパスワードの妥当性を検査できず、連係して要求を拒否します。クライアントは拒否の知らせを受けると、通常、ユーザ名とパスワードをユーザに要求するダイアログボックスを表示し、今度はユーザ名とパスワードをヘッダーに入れて要求を再送します。

この例では、AuthTrans ステップで呼び出される hardcoded-auth 関数が、ユーザ名とパスワードが、ユーザ名とパスワードのハードコード化されたテーブルのエントリに対応するかどうかを確認します。

例のインストール

関数を iPlanet Web Server にインストールするには、コンパイル済みの関数を読み込むために次の Init 指令を magnus.conf に追加します。

```
Init fn=load-modules shlib=yourlibrary funcs=hardcoded-auth
```

obj.conf に含まれるデフォルトのオブジェクト内に、次の AuthTrans 指令を追加します。

```
AuthTrans fn=basic-auth auth-type="basic" userfn=hardcoded-auth
userdb=unused
```

この関数は、実際に承認要求を実行するのではなく、与えられた情報を受け取り、サーバにその情報が正しいかどうかを知らせるだけであることに注意してください。PathCheck 関数の require-auth が実際に承認を行なうので、次の PathCheck 指令も追加します。

```
PathCheck fn=require-auth realm="test realm" auth-type="basic"
```

ソースコード

この例のソースコードは、サーバルートディレクトリの nsapi/examples/ または plugins/nsapi/examples サブディレクトリ内の auth.c ファイルにあります。

```
#include "nsapi.h"

typedef struct {
    char *name;
    char *pw;
} user_s;

static user_s user_set[] = {
    {"joe", "shmoe"},
    {"suzy", "creamcheese"},
    {NULL, NULL}
};

#include "frame/log.h"

#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC int hardcoded_auth(pblock *param, Session *sn, Request
*rq)
{
    /* auth-basic によって与えられたパラメータ */
    char *pwfile = pblock_findval("userdb", param);
    char *user = pblock_findval("user", param);
    char *pw = pblock_findval("pw", param);

    /* 一時変数 */
    register int x;

    for(x = 0; user_set[x].name != NULL; ++x) {

        /* これが目的のユーザでない場合は、続行する
        if(strcmp(user, user_set[x].name) != 0) continue;
```

```
        /* パスワードを検証する _
        if(strcmp(pw, user_set[x].pw)) {
            log_error(LOG_SECURITY, "hardcoded-auth", sn, rq,
                "user %s entered wrong password", user);
            /* これにより、実行関数に、再度ユーザの入力を */
            /* 求めさせる */
            return REQ_NOACTION;
        }

        /* REQ_PROCEED を返した場合は、ユーザ名が受け入れられる */
        return REQ_PROCEED;
    }

    /* 一致しない場合は、再度入力を求めさせる */
    log_error(LOG_SECURITY, "hardcoded-auth", sn, rq,
        "unknown user %s", user);
    return REQ_NOACTION;
}
```

NameTrans の例

サーバルートディレクトリの nsapi/examples/ または plugins/nsapi/examples サブディレクトリの ntrans.c ファイルに、NameTrans 関数の次の 2 つの例のソースコードがあります。

- explicit_pathinfo

  この例では、URL 内で明示的な追加のパス情報を使うことを許可しています。

- https_redirect

  この例では、クライアントが特定のバージョンの Netscape Navigator である場合、URL をリダイレクトします。

この節では、最初の例について説明します。2 番目の例については、ntrans.c にあるソースコードを参照してください。

注　NameTrans 関数が通常主に実行することは、rq->vars 内の ppath の論理 URL を物理パス名に変換することです。ただし、ここで説明する例の、explicit_pathinfo は、URL を物理パス名に変換するのではなく、要求された URL の値を変更します。rq->vars の ppath の値を URL から物理パス名に変換する NameTrans 関数の例は、2 番目の例の ntrans.c の https_redirect を参照してください。

---

explicit_pathinfo の例は、CGI プログラムが使う追加のパス情報を URL に明示的に指定することを許可します。追加のパス情報は、URL の主な部分から、コンマなどの指定された区切り文字で区切られます。

次に例を示します。

```
http://server-name/cgi/marketing,/jan/releases/hardware
```

この場合、要求されたリソース (CGI プログラムになる) の URL は、http://*server-name*/cgi/marketing であり、CGI プログラムに与えられる追加のパス情報は /jan/releases/hardware です。

区切り文字を選択するときには、必ず実際の URL の一部として使われることのない文字を選択します。

explicit_pathinfo 関数は、URL を読み込み、コンマの後に続くものをすべて切り離し、request オブジェクト (rq->vars) の vars フィールドの path-info フィールドに入れます。CGI プログラムは、PATH_INFO 環境変数を介してこの情報にアクセスできます。

explicit_pathinfo の副次作用の 1 つは、SCRIPT_NAME CGI 環境変数の終わりに区切り文字が付加されることです。

通常、NameTrans 指令は、パスを変更するときには、REQ_PROCEED を返して、サーバがさらに NameTrans 指令を処理することがないようにします。ただし、この場合、URL を物理パス名にまだ変換していないので、パス情報を取り出した後も名前の変換を続ける必要があります。

例のインストール

関数を iPlanet Web Server にインストールするには、コンパイル済みの関数を読み込むために次の Init 指令を magnus.conf に追加します。

```
Init fn=load-modules shlib=yourlibrary funcs=explicit-pathinfo
```

obj.conf に含まれているデフォルトのオブジェクト内に、次の NameTrans 指令を追加します。

```
NameTrans fn=explicit-pathinfo separator=","
```

この NameTrans 指令は、デフォルトのオブジェクト内のほかの NameTrans 指令の前に指定する必要があります。

ソースコード

この例は、サーバルートディレクトリの nsapi/examples/ または plugins/nsapi/examples サブディレクトリ内の ntrans.c ファイルにあります。

```
#include "nsapi.h"

#include <string.h>           /* strchr */
#include "frame/log.h"        /* log_error */

#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC int explicit_pathinfo(pblock *pb, Session *sn, Request
*rq)
{
    /* パラメータ : パスを分割する文字 */
    char *sep = pblock_findval("separator", pb);

    /* サーバ変数 */
    char *ppath = pblock_findval("ppath", rq->vars);

    /* 一時変数 */
    char *t;

    /* 正しい使用法を検証する */
    if(!sep) {
        log_error(LOG_MISCONFIG, "explicit-pathinfo", sn, rq,
            "missing parameter (need root)");
        /* 中止した場合、デフォルトの状態コードは 500 Server
            Error である */
        return REQ_ABORTED;
    }

    /* 区切り文字を確認する。区切り文字がない場合は、何も操作を行なわない */
    t = strchr(ppath, sep[0]);
    if(!t)
        return REQ_NOACTION;

    /* 区切り文字の箇所でパスを切り捨てる */
    *t++ = '\0';
    /* パス情報を割り当てる */
    pblock_nvinsert("path-info", t, rq->vars);

    /* 通常、NameTrans 関数は、パスを変更したら
        REQ_PROCEED を返す。しかし、本関数終了後もまだ名前の変換を
        続行したい */
    return REQ_NOACTION;
}
```

```
#include "base/util.h"         /* is_mozilla */
#include "frame/protocol.h"    /* protocol_status */
#include "base/shexp.h"        /* shexp_cmp */

#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC int https_redirect(pblock *pb, Session *sn, Request
*rq)
{
    /* サーバ変数 */
    char *ppath = pblock_findval("ppath", rq->vars);
    /* パラメータ */
    char *from = pblock_findval("from", pb);
    char *url = pblock_findval("url", pb);
    char *alt = pblock_findval("alt", pb);
    /* 作業用変数 */
    char *ua;

    /* 使用法を確認する */
    if((!from) || (!url)) {
        log_error(LOG_MISCONFIG, "https-redirect", sn, rq,
            "missing parameter (need from, url)");
        return REQ_ABORTED;
    }
    /* ワイルドカードを使った突き合せで、このパスがリダイレクトする必要があるパ
スであるかどうかを
        確認する */
    if(shexp_cmp(ppath, from) != 0)
        return REQ_NOACTION;   /* 一致するものなし */

    /*  SSL の機能を確認するための唯一の方法は、UA を確認する
        こと */
    if(request_header("user-agent", &ua, sn, rq) == REQ_ABORTED)
        return REQ_ABORTED;

    /* is_mozilla 関数は、Mozilla バージョン 0.96 以降であるかを
        確認する */
    if(util_is_mozilla(ua, "0", "96")) {
        /* リターンコードを 302 Redirect に設定する */
        protocol_status(sn, rq, PROTOCOL_REDIRECT, NULL);
        /* エラー処理関数は、これを使用して Location: を設定する */
        pblock_nvinsert("url", url, rq->vars);
        return REQ_ABORTED;
    }

    /* 一致するものなし。古いクライアント */
```

```
    /* 指定された代わりのドキュメントがある場合は、それを使用する */
    if(alt) {
        pb_param *pp = pblock_find("ppath", rq->vars);
        /* 古い値を捨てる */
        FREE(pp->value);
        /* ライブラリが後でこの pblock を解放するため、それを複製しておく
           必要がある */
        pp->value = STRDUP(alt);
        return REQ_PROCEED;
    }
    /* そうでなければ、何も実行しない */
    return REQ_NOACTION;
}
```

PathCheck の例

この節の例は、パスの確認を実行するカスタム SAF の実現方法を示します。この例は、要求しているホストが、許可されているホストのリストにあるかどうかを確認するだけです。

`Init` 関数の `acf-init` は、各行に 1 つ IP アドレスがある、許可されている IP アドレスのリストが含まれているファイルを読み込みます。`PathCheck` 関数の `restrict-by-acf` は、要求を送信してきたホストの IP アドレスを取得し、リストにあるかどうかを確認します。ホストがリストにある場合は、アクセスが許可され、ない場合は拒否されます。

簡単にするために、ファイルからの IP アドレスの走査に stdio ライブラリを使っています。

例のインストール

ユーザの関数が含まれている共用オブジェクトを読み込むには、`magnus.conf` ファイルの `Init` セクションに次の行を追加します。

```
Init fn=load-modules yourlibrary funcs=acf-init,restrict-by-acf
```

許可されるホストのリストを読み込むために `acf-init` を呼び出すには、`magnus.conf` の `Init` セクションに次の行を追加します。この行は、`acf-init` が含まれているライブラリを読み込む行の後に、指定する必要があります。

```
Init fn=acf-init file=fileContainingHostsList
```

要求 - 応答プロセスで、あるオブジェクトに対してカスタム SAF を実行するには、`obj.conf` ファイル内でそのオブジェクトに次の行を追加します。

```
PathCheck fn=restrict-by-acf
```

ソースコード

この例のソースコードは、サーバルートディレクトリの nsapi/examples/ または plugins/nsapi/examples サブディレクトリ内の pcheck.c ファイルにあります。

```
#include "nsapi.h"

/* acf-init を呼び出していないユーザに対して問題が発生するのを避けるために、
NULL に設定する */
static char **hosts = NULL;

#include <stdio.h>
#include "base/daemon.h"
#include "base/util.h"      /* util_sprintf */
#include "frame/log.h"      /* log_error */
#include "frame/protocol.h" /* protocol_status */

/* アクセス制御ファイルで使用できる最も長い行 */
#define MAX_ACF_LINE 256

/* 再起動時に静的な配列を解放するために使用する */
#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC void acf_free(void *unused)
{
    register int x;

    for(x = 0; hosts[x]; ++x)
        FREE(hosts[x]);
    FREE(hosts);
    hosts = NULL;
}

#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC int acf_init(pblock *pb, Session *sn, Request *rq)
{
    /* パラメータ */
    char *acf_file = pblock_findval("file", pb);
```

```
    /* 作業用変数 */
    int num_hosts;
    FILE *f;
    char err[MAGNUS_ERROR_LEN];
    char buf[MAX_ACF_LINE];

    /* 使用法を確認する。Init 関数に特別なエラーロギング機能が備えられている
       ことに注意する */
    if(!acf_file) {
       util_sprintf(err, "missing parameter to acf_init
          (need file)");
       pblock_nvinsert("error", err, pb);
       return REQ_ABORTED;
    }

    f = fopen(acf_file, "r");

    /* 開いたか？ */
    if(!f) {
       util_sprintf(err, "can't open access control file %s (%s)",
          acf_file, system_errmsg());
       pblock_nvinsert("error", err, pb);
       return REQ_ABORTED;
    }

    /* ホスト配列を初期化する */
    num_hosts = 0;
    hosts = (char **) MALLOC(1 * sizeof(char *));
    hosts[0] = NULL;

    while(fgets(buf, MAX_ACF_LINE, f)) {
       /* stdio が便宜上そこに残しておいた改行を破棄します */
       buf[strlen(buf) - 1] = '\0';
       hosts = (char **) REALLOC(hosts, (num_hosts + 2) *
          sizeof(char *));
       hosts[num_hosts++] = STRDUP(buf);
       hosts[num_hosts] = NULL;
    }

    fclose(f);

    /* 再起動時に、ホスト配列を解放する */
    daemon_atrestart(acf_free, NULL);

    return REQ_PROCEED
}

#ifdef __cplusplus
extern "C"
#endif
```

```
NSAPI_PUBLIC int restrict_by_acf(pblock *pb, Session *sn, Request
*rq)
{
    /* パラメータなし */

    /* 作業用変数 */
    char *remip = pblock_findval("ip", sn->client);
    register int x;

    if(!hosts) {
        log_error(LOG_MISCONFIG, "restrict-by-acf", sn, rq,
            "restrict-by-acf called without call to acf-init");
        /* 中止した場合、デフォルトの状態コードは 500 Server
            Error */
        return REQ_ABORTED;
    }

    for(x = 0; hosts[x] != NULL; ++x) {
        /* リストにある場合は、許可される */
        if(!strcmp(remip, hosts[x]))
        return REQ_NOACTION;
    }

    /* 応答コードを forbidden に設定し、エラーを返す */
    protocol_status(sn, rq, PROTOCOL_FORBIDDEN, NULL);
    return REQ_ABORTED;
}
```

ObjectType の例

この節の例は、要求されたファイルの .shtml バージョンが存在する場合に、.html ファイルを .shtml ファイルとして扱うようにサーバに指示するカスタム SAF である html2shtml の実装方法を示します。

適切な処理を行なう ObjectType 関数は、内容のタイプがすでに設定されているかどうかを確認し、設定されている場合は何も実行しないで REQ_NOACTION を返します。

```
if(pblock_findval("content-type", rq->srvhdrs))
    return REQ_NOACTION;
```

ObjectType 指令の主な役割は、(内容のタイプが設定されていない場合に) 内容のタイプを設定することです。この例では、次に示す行で内容のタイプを magnus-internal/parsed-html に設定します。

```
/* content-type を magnus-internal/parsed-html に設定する */
pblock_nvinsert("content-type", "magnus-internal/parsed-html",
    rq->srvhdrs);
```

html2shtml 関数は、要求されたファイル名を確認します。ファイル名が .html で終わる場合は、この関数はベース部分の名前が同じで拡張子が .shtml のファイルを検索します。該当するファイルが見つかった場合、そのパスを使い、サーバにそのファイルが通常の HTML ではなく解析された HTML であることを知らせます。これには、アクセスされるすべての HTML ファイルに対して、追加の stat 呼び出しが必要であることに注意してください。

例のインストール

ユーザの関数が含まれている共用オブジェクトを読み込むには、magnus.conf ファイルの Init セクションに次の行を追加します。

```
Init fn=load-modules shlib=yourlibrary funcs=html2shtml
```

要求 - 応答プロセスで、あるオブジェクトに対してカスタム SAF を実行するには、obj.conf ファイル内でそのオブジェクトに次の行を追加します。

```
ObjectType fn=html2shtml
```

ソースコード

この例のソースコードは、サーバルートディレクトリの nsapi/examples/ または plugins/nsapi/examples サブディレクトリ内の otype.c ファイルにあります。

```
#include "nsapi.h"

#include <string.h>     /* strncpy */
#include "base/util.h"

#ifdef __cplusplus
extern "C"
#endif
```

```
NSAPI_PUBLIC int html2shtml(pblock *pb, Session *sn, Request *rq)
{
    /* パラメータなし */

    /* 作業用変数 */
    pb_param *path = pblock_find("path", rq->vars);
    struct stat finfo;
    char *npath;
    int baselen;

    /* タイプがすでに設定されている場合は、何も操作を行なわない */
    if(pblock_findval("content-type", rq->srvhdrs))
        return REQ_NOACTION;

    /* パスが .html で終了していない場合は、通常のオブジェクトタイプに処理を
       行なわせる */
    baselen = strlen(path->value) - 5;
    if(strcasecmp(&path->value[baselen], ".html") != 0)
        return REQ_NOACTION;

    /* 1 = html を shtml に変換するための領域 */
    npath = (char *) MALLOC((baselen + 5) + 1 + 1);
    strncpy(npath, path->value, baselen);
    strcpy(&npath[baselen], ".shtml");

    /* ない場合は何も操作しない */
    if(stat(npath, &finfo) == -1) {
        FREE(npath);
        return REQ_NOACTION;
    }

    /* 取得したら、入れ替えを行なう */
    FREE(path->value);
    path->value = npath;

    /* サーバが現在のパスの stat() をキャッシングする。
       それを更新する。*/
    (void) request_stat_path(NULL, rq);

    pblock_nvinsert("content-type", "magnus-internal/parsed-html",
        rq->srvhdrs);
    return REQ_PROCEED;
}
```

Service の例

この節では、simple-service という非常に単純な Service 関数について説明します。この関数が実行するのは、クライアントの要求に対する応答としてメッセージを送信することだけです。メッセージは、サーバの初期化時に simple-service-init 関数で初期化されます。

より複雑な例は、examples ディレクトリ内の service.c ファイルを参照してください。これについては、227 ページの「より複雑な Service の例」に説明があります。

例のインストール

ユーザの関数が含まれている共用オブジェクトを読み込むには、magnus.conf ファイルの Init セクションに次の行を追加します。

```
Init fn=load-modules shlib=yourlibrary
funcs=simple-service-init,simple-service
```

生成された出力を表すメッセージを初期化する simple-service-init 関数を呼び出すには、次の行を magnus.conf の Init セクションに追加します。この行は、simple-service-init が含まれているライブラリを読み込む行の後に、指定する必要があります。

```
Init fn=simple-service-init
generated-output="<H1>Generated output msg</H1>"
```

要求 - 応答プロセスで、あるオブジェクトに対してカスタム SAF を実行するには、obj.conf ファイル内でそのオブジェクトに次の行を追加します。

```
Service type="text/html" fn=simple-service
```

type="text/html" 引数は、content-type が text/html に設定されている場合にのみ、Service 段階でこの関数を呼び出すことを示します。

ソースコード

```
#include <nsapi.h>

static char *simple_msg = "default customized content";

/* これは、初期化関数である。
 * この関数は、magnus.conf ファイルの Init 指令に指定されている、
 * generated-output パラメータの値を取得する
*/
NSAPI_PUBLIC int init-simple-service(pblock *pb, Session *sn,
Request *rq)
{
    /* magnus.conf ファイルの指令にあるパラメータからメッセージを
     * 取得する
    */
    simple_msg = pblock_findval("generated-output", pb);
    return REQ_PROCEED;

}

/* これは、カスタマイズされた Service SAF である
 * これは、クライアントに「generated-output」メッセージを送信する
*/
NSAPI_PUBLIC int simple-service(pblock *pb, Session *sn, Request
*rq)
{
    int return_value;
    char msg_length[8];

    /* protocol_start_response を呼び出す前に、protocol_status 関数を使用
    * して応答の状態を設定する
    */
    protocol_status(sn, rq, PROTOCOL_OK, NULL);

    /* ObjectType 段階で content-type が設定されると期待していても、
    * ここで設定しておけば確実なので、content-type を text/html に
    * 設定する
    */
    param_free(pblock_remove("content-type", rq->srvhdrs));
    pblock_nvinsert("content-type", "text/html", rq->srvhdrs);

    /* keepalive を使用したい場合は、content-length ヘッダーを設定する必要が
    * ある。util_itoa 関数は、指定された整数を文字列に変換し、
    * 文字列の長さを返す。数のテキスト表現
    * を作成するには、この関数を使用する
    */

    util_itoa(strlen(simple_msg), msg_length);
    pblock_nvinsert("content-length", msg_length, rq->srvhdrs);
```

```
    /* クライアントにヘッダーを送信する */
    return_value = protocol_start_response(sn, rq);
    if (return_value == REQ_NOACTION) {
        /* GET の代わりに HTTP HEAD */
        return REQ_PROCEED;
    }

    /* net_write を使用して出力を書き込む */
    return_value = net_write(sn->csd, simple_msg,
        strlen(simple_msg));
    if (return_value == IO_ERROR) {
        return REQ_EXIT;
    }

    return REQ_PROCEED;
}
```

より複雑な Service の例

send-images 関数は、iPlanet ホームページから入手できる doit.cgi デモに代わる、カスタム SAF です。ファイルが /dir1/dir2/something.picgroup としてアクセスされると、send-images 関数は、そのファイルが Mozilla/1.1 ブラウザによってアクセスされているかどうかを確認します。そうでない場合は、短いエラーメッセージを送信します。ファイル something.picgroup には行のリストが含まれ、各行にはファイル名と、その後に content-type が指定されています (たとえば、one.gif image/gif)。

ユーザの関数が含まれている共用オブジェクトを読み込むには、magnus.conf ファイルの先頭に次の行を追加します。

```
Init fn=load-modules shlib=yourlibrary funcs=send-images
```

また、次の行を mime.types ファイルに追加します。

```
type=magnus-internal/picgroup exts=picgroup
```

要求 - 応答プロセスで、あるオブジェクトに対してカスタム SAF を実行するには、obj.conf ファイル内のそのオブジェクトに次の行を追加します。send-images は、任意 (省略可能) のパラメータである delay を受け入れますが、この例では使われていません。

```
Service method=(GET|HEAD) type=magnus-internal/picgroup
fn=send-images
```

ソースコードは、サーバルートディレクトリ内の nsapi/examples/ または plugins/nsapi/examples サブディレクトリの service.c ファイルにあります。

AddLog の例

この節の例は、要求についての情報である IP アドレス、メソッド、および URI (たとえば、198.93.95.99 GET /jocelyn/dogs/homesneeded.html) の 3 項目だけをログに記録するカスタム SAF である brief-log を実装する方法を示します。

例のインストール

ユーザの関数が含まれている共用オブジェクトを読み込むには、magnus.conf ファイルの Init セクションに次の行を追加します。

```
Init fn=load-modules shlib=yourlibrary funcs=brief-init,brief-log
```

ログファイルを開くために brief-init を呼び出すには、magnus.conf の Init セクションに次の行を追加します。この行は、brief-init が含まれているライブラリを読み込む行の後に、指定する必要があります。

```
Init fn=brief-init file=/tmp/brief.log
```

AddLog 段階で、あるオブジェクトに対してカスタム SAF を実行するには、obj.conf ファイル内でそのオブジェクトに次の行を追加します。

```
AddLog fn=brief-log
```

ソースコード

ソースコードは、サーバルートディレクトリ内の nsapi/examples/ または plugins/nsapi/examples サブディレクトリの addlog.c ファイルにあります。

```
#include "nsapi.h"

#include "base/daemon.h" /* daemon_atrestart */
#include "base/file.h"   /* system_fopenWA, system_fclose */
#include "base/util.h"   /* sprintf */

/* プロセス間で共有されるファイル記述子 */
static SYS_FILE logfd = SYS_ERROR_FD;

#ifdef __cplusplus
extern "C"
#endif
```

```
NSAPI_PUBLIC void brief_terminate(void *parameter)
{
    system_fclose(logfd);
    logfd = SYS_ERROR_FD;
}

#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC int brief_init(pblock *pb, Session *sn, Request *rq)
{
    /* パラメータ */
    char *fn = pblock_findval("file", pb);

    if(!fn) {
        pblock_nvinsert("error", "brief-init: please supply a
            file name", pb);
        return REQ_ABORTED;
    }

    logfd = system_fopenWA(fn);
    if(logfd == SYS_ERROR_FD) {
        pblock_nvinsert("error", "brief-init: please supply a
            file name", pb);
        return REQ_ABORTED;
    }

    /* サーバが再起動したら、ログファイルを閉じる */
    daemon_atrestart(brief_terminate, NULL);
    return REQ_PROCEED;
}_

#ifdef __cplusplus
extern "C"
#endif

NSAPI_PUBLIC int brief_log(pblock *pb, Session *sn, Request *rq)
{
    /* パラメータなし */

    /* サーバデータ */
    char *method = pblock_findval("method", rq->reqpb);
    char *uri = pblock_findval("uri", rq->reqpb);
    char *ip = pblock_findval("ip", sn->client);

    /* 一時変数 */
    char *logmsg;
    int len;
```

```
    logmsg = (char *)
    MALLOC(strlen(ip) + 1 + strlen(method) + 1 + strlen(uri) +
        1 + 1);
    len = util_sprintf(logmsg, "%s %s %s\n", ip, method, uri);
    /* 原子バージョンの場合は、干渉を防止するためにロックが使用される */
    system_fwrite_atomic(logfd, logmsg, len);
    FREE(logmsg);

    return REQ_PROCEED;
}
```

サービス品質の例

qos-handler および qos-error SAF のコードは、サービス品質の処理のためにユーザ独自の SAF を定義したい場合の例として用意されています。

詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

例のインストール

obj.conf に含まれているデフォルトのオブジェクト内に、次の AuthTrans 指令と Error 指令を追加します。

```
AuthTrans fn=qos-handler
...
Error fn=qos-error code=503
```

ソースコード

この例のソースコードは、サーバルートディレクトリ内の plugins/nsapi/examples サブディレクトリ内の qos.c ファイル内にあります。

```
#include "frame/log.h"
#include "frame/http.h"
#include "safs/qos.h"

/*----------------------------------------------------------------------------
 decode : pblock の QOS 値を解析するために使用される内部関数
```

```
------------------------------------------------------------------------------*/
void decode(const char* val, PRInt32* var, pblock* pb)
{
    char* pbval;
    if ((!var) || (!val) || (!pb))
        return;
    pbval = pblock_findval(val, pb);
    if (!pbval)
        return;
    *var = atoi(pbval);
}
/*------------------------------------------------------------------------------
qos_error
この関数は、HTTP 503 エラーコードのエラーハンドラの役割を果たす
このエラーコードは、QOS 制限値を超えていて、その実行が強制された場合に、qos_handler
によって返される
このサンプル関数は、どの制限値を超えているのかを示すメッセージを出力する
------------------------------------------------------------------------------*/
NSAPI_PUBLIC int qos_error(pblock *pb, Session *sn, Request *rq)
{
    char error[1024] = "";

    PRBool ours = PR_FALSE;

    PRInt32 vs_bw = 0, vs_bwlim = 0, vs_bw_ef = 0,
            vs_conn = 0, vs_connlim = 0, vs_conn_ef = 0,
            vsc_bw = 0, vsc_bwlim = 0, vsc_bw_ef = 0,
            vsc_conn = 0, vsc_connlim = 0, vsc_conn_ef = 0,
            srv_bw = 0, srv_bwlim = 0, srv_bw_ef = 0,
            srv_conn = 0, srv_connlim = 0, srv_conn_ef = 0;

    pblock* apb = rq->vars;

    decode("vs_bandwidth", &vs_bw, apb);
    decode("vs_connections", &vs_conn, apb);

    decode("vs_bandwidth_limit", &vs_bwlim, apb);
    decode("vs_bandwidth_enforced", &vs_bw_ef, apb);

    decode("vs_connections_limit", &vs_connlim, apb);
    decode("vs_connections_enforced", &vs_conn_ef, apb);

    decode("vsclass_bandwidth", &vsc_bw, apb);
    decode("vsclass_connections", &vsc_conn, apb);

    decode("vsclass_bandwidth_limit", &vsc_bwlim, apb);
    decode("vsclass_bandwidth_enforced", &vsc_bw_ef, apb);
```

```
decode("vsclass_connections_limit", &vsc_connlim, apb);
decode("vsclass_connections_enforced", &vsc_conn_ef, apb);

decode("server_bandwidth", &srv_bw, apb);
decode("server_connections", &srv_conn, apb);

decode("server_bandwidth_limit", &srv_bwlim, apb);
decode("server_bandwidth_enforced", &srv_bw_ef, apb);

decode("server_connections_limit", &srv_connlim, apb);
decode("server_connections_enforced", &srv_conn_ef, apb);

if ((vs_bwlim) && (vs_bw>vs_bwlim))
{
/* VS 帯域幅の制限値を超えた。それを表示する */
ours = PR_TRUE;
sprintf(error, "<P>Virtual server bandwidth limit of %d .
Current VS bandwidth : %d . <P>", &vs_bwlim, vs_bw);
};

if ((vs_connlim) && (vs_conn>vs_connlim))
{
/* VS 接続の制限値を超えた。それを表示する */
ours = PR_TRUE;
sprintf(error, "<P>Virtual server connection limit of %d .
Current VS connections : %d . <P>", &vs_connlim, vs_conn);
};

if ((vsc_bwlim) && (vsc_bw>vsc_bwlim))
{
/* VSCLASS 帯域幅の制限値を超えた。それを表示する */
ours = PR_TRUE;
sprintf(error, "<P>Virtual server class bandwidth limit of %d
. Current VSCLASS bandwidth : %d . <P>", &vsc_bwlim, vsc_bw);
};

if ((vsc_connlim) && (vsc_conn>vsc_connlim))
{
/* VSCLASS 接続の制限値を超えた。それを表示する */
ours = PR_TRUE;
sprintf(error, "<P>Virtual server class connection limit of
%d . Current VSCLASS connections : %d . <P>", &vsc_connlim,
vsc_conn);
};

if ((srv_bwlim) && (srv_bw>srv_bwlim))
```

```
    {
    /* SERVER 帯域幅の制限値を超えた。それを表示する */
    ours = PR_TRUE;
    sprintf(error, "<P>Global bandwidth limit of %d . Current
    bandwidth : %d . <P>", &srv_bwlim, srv_bw);
    };

    if ((srv_connlim) && (srv_conn>srv_connlim))
    {
    /* SERVER 接続の制限値を超えた。それを表示する */
    ours = PR_TRUE;
    sprintf(error, "<P>Global connection limit of %d . Current
    connections : %d . <P>", &srv_connlim, srv_conn);
    };

    if (ours)
    {
    /* これは実際に、QOS 障害だった。そのため、エラーページを送信する */
        pblock_nvreplace("content-type", "text/html", rq->srvhdrs);

        protocol_start_response(sn, rq);
        net_write(sn->csd, error, strlen(error));
        return REQ_PROCEED;
    }
    else
    {
    /* この 503 は QOS SAF 障害が原因ではない。ほかで処理を行なう必要がある */
    return _REQ_PROCEED;
    };
}

/*--------------------------------------------------------------------------
 qos_handler

 これは、NSAPI AuthTrans 関数である

この関数は、要求内の QOS 値を検査し、それを QOS 制限値と比較する

次のいくつかの処理を行なう :
1) QOS 制限値を超えている場合は、エラーをログに記録する
2) QOS 制限値を超えている場合は、REQ_ABORTED を 503 エラーコードとともに返し、
そして、QOS 制限を強制実行するように設定する。QOS 制限値を超えていない場合は、REQ_PROCEED を
返す

--------------------------------------------------------------------------*/

NSAPI_PUBLIC int qos_handler(pblock *pb, Session *sn, Request *rq)
```

```
{
    PRBool ok = PR_TRUE;

    PRInt32 vs_bw = 0, vs_bwlim = 0, vs_bw_ef = 0,
            vs_conn = 0, vs_connlim = 0, vs_conn_ef = 0,
            vsc_bw = 0, vsc_bwlim = 0, vsc_bw_ef = 0,
            vsc_conn = 0, vsc_connlim = 0, vsc_conn_ef = 0,
            srv_bw = 0, srv_bwlim = 0, srv_bw_ef = 0,
            srv_conn = 0, srv_connlim = 0, srv_conn_ef = 0;

    pblock* apb = rq->vars;

    decode("vs_bandwidth", &vs_bw, apb);
    decode("vs_connections", &vs_conn, apb);

    decode("vs_bandwidth_limit", &vs_bwlim, apb);
    decode("vs_bandwidth_enforced", &vs_bw_ef, apb);

    decode("vs_connections_limit", &vs_connlim, apb);
    decode("vs_connections_enforced", &vs_conn_ef, apb);

    decode("vsclass_bandwidth", &vsc_bw, apb);
    decode("vsclass_connections", &vsc_conn, apb);

    decode("vsclass_bandwidth_limit", &vsc_bwlim, apb);
    decode("vsclass_bandwidth_enforced", &vsc_bw_ef, apb);

    decode("vsclass_connections_limit", &vsc_connlim, apb);
    decode("vsclass_connections_enforced", &vsc_conn_ef, apb);

    decode("server_bandwidth", &srv_bw, apb);
    decode("server_connections", &srv_conn, apb);

    decode("server_bandwidth_limit", &srv_bwlim, apb);
    decode("server_bandwidth_enforced", &srv_bw_ef, apb);

    decode("server_connections_limit", &srv_connlim, apb);
    decode("server_connections_enforced", &srv_conn_ef, apb);

    if ((vs_bwlim) && (vs_bw>vs_bwlim))
    {
    /* 帯域幅の制限値を超えた。それをログに記録する */
    ereport(LOG_FAILURE, "Virtual server bandwidth limit of %d
    exceeded. Current VS bandwidth : %d", &vs_bwlim, vs_bw);

        if (vs_bw_ef)
        {
```

```
    /* 強制実行する */
    ok = PR_FALSE;
     };
};

if ((vs_connlim) && (vs_conn>vs_connlim))
{
/* 接続の制限値を超えた。それをログに記録する */
ereport(LOG_FAILURE, "Virtual server connection limit of %d
exceeded. Current VS connections : %d", &vs_connlim,
vs_conn);

     if (vs_conn_ef)
     {
    /* 強制実行する */
    ok = PR_FALSE;
     };
};

if ((vsc_bwlim) && (vsc_bw>vsc_bwlim))
{
/* 帯域幅の制限値を超えた。それをログに記録する */
ereport(LOG_FAILURE, "Virtual server class bandwidth limit of
%d exceeded. Current VSCLASS bandwidth : %d", &vsc_bwlim,
    vsc_bw);

     if (vsc_bw_ef)
     {
    /* 強制実行する */
    ok = PR_FALSE;
     };
};

if ((vsc_connlim) && (vsc_conn>vsc_connlim))
{
/* 接続の制限値を超えた。それをログに記録する */
ereport(LOG_FAILURE, "Virtual server class connection limit
of %d exceeded. Current VSCLASS connections : %d",
&vsc_connlim, vsc_conn);

     if (vsc_conn_ef)
     {
    /* 強制実行する */
    ok = PR_FALSE;
     };
};
```

```
    if ((srv_bwlim) && (srv_bw>srv_bwlim))
    {
    /* 帯域幅の制限値を超えた。それをログに記録する */
    ereport(LOG_FAILURE, "Global bandwidth limit of %d exceeded.
    Current global bandwidth : %d", &srv_bwlim, srv_bw);

        if (srv_bw_ef)
        {
       /* 強制実行する */
       ok = PR_FALSE;
        };
    };

    if ((srv_connlim) && (srv_conn>srv_connlim))
    {
    /* 接続の制限値を超えた。それをログに記録する */
    ereport(LOG_FAILURE, "Global connection limit of %d exceeded.
    Current global connections : %d", &srv_connlim, srv_conn);

        if (srv_conn_ef)
        {
       /* 強制実行する */
       ok = PR_FALSE;
        };
    };

    if (ok)
    {
    return REQ_PROCEED;
    }
    else
    {
    /* いずれかの制限値を超えた
    したがって、HTTP エラー 503「server too busy」を設定する */
    protocol_status(sn, rq, PROTOCOL_SERVICE_UNAVAILABLE, NULL);
    return REQ_ABORTED;
    };
}
```

第 7 章

magnus.conf の構文と使用法

iPlanet Web Server は、起動時に *server-id*/config ディレクトリにある magnus.conf というファイルを調べて、サーバの動作と構成に影響する 1 組のグローバル変数設定を確立します。iPlanet Web Server は、magnus.conf で定義されているすべての指令を実行します。

Init SAF を除けば、magnus.conf 内の指令は、次の例に示すように、変数と値を指定します。

```
ServerID https-boots.mcom.com
#ServerRoot d:/netscape/server4/https-boots.mcom.com
```

指令の順序は重要ではありません。

注 magnus.conf ファイルを編集するときは、変更内容を有効にするためにサーバを再起動する必要があります。

この章では、iPlanet Web Server 6.0 の magnus.conf で指定できるグローバル設定をリスト表示します。

次のカテゴリがあります。

- Init SAF
- サーバ情報
- 言語に関する問題
- DNS 検索
- スレッド、プロセス、および接続

- ネイティブスレッドプール
- CGI
- エラーログ作成と統計収集
- ACL
- セキュリティ
- チャンクされたエンコーディング
- その他

指令のアルファベット順リストについては、付録 H「magnus.conf 内の指令のアルファベット順リスト」を参照してください。

注	ファイルキャッシュの機能の多くは、nsfc.conf という構成ファイルによって制御されます。nsfc.conf については、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

Init SAF

Init 指令は、サーバを初期化します。たとえば、追加のモジュールとプラグインの読み込みと初期化、およびログファイルの初期化を行ないます。

Init 指令は obj.conf 指令と同様 SAF であり、ほかの magnus.conf 指令の単純な *variable value* 構文とは異なる SAF 構文を持っています。これらの指令は、ほかの magnus.conf 指令と同様にサーバ起動時に一回だけ実行されるため、magnus.conf に置かれています。

各 Init 指令には、省略可能な LateInit パラメータがあります。UNIX プラットフォームでは、LateInit が yes に設定されている場合、この関数は、親プロセスからフォークされた後に子プロセスによって実行されます。LateInit が no に設定されているか、何も指定されていない場合、この関数はフォークの前に親プロセスによって実行されます。サーバがユーザ root によって起動されたのに、別のユーザとして稼動するときは、ユーザ root として実行すべきアクティビティ (root が所有するファイルへの書き込みなど) はすべて、フォークの前に実行しなければなりません。thread-pool-init 以外の、スレッドを作成する関数は、フォークの後に実行する必要があります (つまり、関連する Init 指令には LateInit=yes の設定が必要です)。

どのプラットフォームでも、完全に構文解析された構成にアクセスする必要がある関数には、その Init 指令に LateInit=yes の設定が必要です。

障害発生時には、Init クラスの関数は REQ_ABORTED を返します。サーバでは、obj.conf にある Error 指令の命令に従ってエラーのログを記録して終了します。ほかの結果コードは、すべて成功とみなされます。

ここでは、次の Init クラス関数について詳しく説明します。

- cindex-init は、拡張インデックス化のデフォルト特性を変更します。
- define-perf-bucket は、パフォーマンスバケットを作成します。
- dns-cache-init は、DNS キャッシングを構成します。
- flex-init は、柔軟なログ作成システムを初期化します。
- flex-rotate-init は、柔軟なログのローテーションを有効にします。
- init-cgi は、CGI プログラムのデフォルトの設定を変更します。
- init-clf は、共通ログのサブシステムを初期化します。
- init-uhome は、ユーザのホームディレクトリ情報を読み込みます。
- load-modules は、共用ライブラリをサーバに読み込みます。
- nt-console-init は、NT コンソールを有効にします。NT コンソールは、標準の出力とエラーストリームを表示するコマンド行シェルです。
- perf-init は、パフォーマンスバケットを介したシステムパフォーマンスの測定を有効にします。
- pool-init は、プールされたメモリーの割り当てを構成します。
- register-http-method は、新しい HTTP メソッドを登録することによって HTTP プロトコルの拡張を可能にします。
- stats-init は、XML 形式のパフォーマンス統計のレポート作成を有効にします。
- thread-pool-init は、追加のスレッドプールを構成します。

cindex-init

Init クラス指令で適用可能。

関数 cindex-init は、共通のインデックス化のための、デフォルトの設定を行ないます。共通インデックス化 (拡張インデックス化ともいう) は、サービス関数 index-common によって実行されます。インデックス化は、要求された URL がインデックスファイルまたはホームページを含まないディレクトリに変換されるとき、またはインデックスファイルやホームページが指定されていないときに発生します。

共通 (拡張) インデックス化では、インデックス化されたファイルまたはディレクトリごとの名前、最終更新日、サイズ、および説明がディレクトリリストに表示されます。

パラメータ：

`opts` (省略可能) 有効化するオプションを指定する文字列です。現在のところ、使用できるオプションは 1 つだけです。

`s` は、説明フィールドに表示する HTML の `<TITLE>` タグの内容について、ディレクトリ内のインデックス化される各 HTML ファイルを走査するようサーバに伝えます。`<TITLE>` タグは、ファイルの最初の 255 文字内になければなりません。デフォルトでは、このオプションはオフになっています。

`<TITLE>` の検索では、大文字と小文字は区別されません。

`widths` (省略可能) インデックス表示内の各列の幅を指定します。文字列は、名前、最終更新日、サイズ、および説明の文字の列の幅をそれぞれ指定する、数字をコンマで区切ったリストです。

`widths` パラメータのデフォルトの値は、22,18,8,33 です。

最後の 3 つの値 (それぞれ、最終更新日、サイズ、説明に対応する) をそれぞれ 0 に設定すれば、その列の表示をオフにすることができます。名前の列はオフにすることはできません。列の最小サイズ (0 以外の値の場合) は、そのタイトルの長さによって指定されます。たとえば、「Date」列の最小サイズは 5 です (「Date」の長さにスペースを 1 つ加えた長さ)。列に対してタイトルの長さより短い 0 以外の値を設定すると、その幅がタイトルの表示に必要な最小の値にデフォルトとして設定されます。

`timezone` (省略可能) これは、最終更新日を現地時間とグリニッジ標準時のどちらで表示するかを示します。値は、`GMT` または `local` です。デフォルトは `local` です。

`format` (省略可能) このパラメータは、最終更新日の表示書式を決定します。これは、UNIX 関数 `strftime()` の書式仕様を使用します。

デフォルトは `%d-%b-%Y %H:%M` です。

`ignore` (省略可能) インデックス化中にサーバが無視すべきファイル名のワイルドカードパターンを指定します。ピリオド (.) で始まるファイル名は、常に無視されます。デフォルトでは、ピリオド (.) で始まるファイル名だけを無視します。

`icon-uri` (省略可能) ファイルのアイコン (`.gif` ファイル) の URL を生成するときに `index-common` 関数が使用する URI 接頭辞を指定します。デフォルトでは、`/mc-icons/` です。`icon-uri` がデフォルトとは異なる場合は、サーバがこれらのアイコンを見つけられるように、`NameTrans` 指令にある `pfx2dir` 関数を変更する必要があります。

例:

```
Init fn=cindex-init widths=50,1,1,0
Init fn=cindex-init ignore=*private*
Init fn=cindex-init widths=22,0,0,50
```

関連項目

index-common、find-index、home-page

define-perf-bucket

Init クラス指令で適用可能。

define-perf-bucket 関数は、obj.conf にある SAF のパフォーマンスの測定に使用できるパフォーマンスバケットを作成します (48 ページの「バケットパラメータ」および service-dump 関数を参照)。この関数は、perf-init 関数が有効な場合に限り機能します。

パフォーマンスバケットの詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

パラメータ

name	バケットの名前。たとえば、cgi-bucket。
description	バケットの測定対象の説明。たとえば、CGI Stats。

例:

```
Init fn="define-perf-bucket" name="cgi-bucket" description="CGI
Stats"
```

関連項目

perf-init

dns-cache-init

Init クラス指令で適用可能。

dns-cache-init 関数は、DNS 検索が有効になっているときに DNS 検索がキャッシュされるよう指定します。DNS 検索がキャッシュされている場合は、サーバがクライアントのホスト名情報を取得するときに、その情報を DNS キャッシュに格納します。将来、サーバがクライアントに関する情報を必要とする場合は、DNS キャッシュのこの情報を利用できます。

DNS キャッシュのサイズと、キャッシュエントリが無効になるまでの時間を指定することができます。DNS キャッシュには、32 ～ 32768 のエントリを含めることができ、デフォルト値は 1024 エントリです。キャッシュエントリの期限切れまでの時間の値 (秒単位で指定) は、1 秒から 1 年の範囲で指定でき、デフォルト値は 1200 秒 (20 分) です。

パラメータ

cache-size	(省略可能) キャッシュに含めるエントリの数を指定します。指定できる値は 32 ～ 32768 で、デフォルト値は 1024 です。
expire	(省略可能) キャッシュエントリの期限が切れるまでの時間 (秒単位) を指定します。指定できる値は 1 ～ 31536000 (1 年) で、デフォルトは 1200 秒 (20 分) です。

例 :

```
Init fn="dns-cache-init" cache-size="2140" expire="600"
```

flex-init

Init クラス指令で適用可能。

flex-init 関数は、柔軟なログ作成に使用する指定されたログファイルを開き、その記録の書式を確立します。ログ書式は、ログファイルの最初の行に記録されます。サーバがログファイルを使用している間は、ログ書式の変更はできません。

flex-log 関数は、要求処理プロセスの AddLog 段階中に、ログファイルにエントリを書き込みます。

サーバがシャットダウンまたは再起動するまで、ログファイルは開いたままになります。シャットダウンまたは再起動時に、すべてのログが閉じられ、再度開かれます。

注 flex-log を呼び出す AddLog 段階指令がサーバにある場合、柔軟なログファイルはサーバの初期化中に flex-init によって初期化されなければなりません。

複数のログファイル名を同じ flex-init 関数呼び出しで指定することができます。その後、flex-log 関数を持つ複数の AddLog 指令を使って、各ログファイルへのトランザクションを記録します。

flex-init 関数は、複数回呼び出すことができます。それぞれの新しいログファイル名と書式は、ログファイルのリストに追加されます。

現在アクティブなログファイルを、サーバをシャットダウンまたは再起動せずに移動、削除、または変更すると、クライアントのアクセスが記録されないことがあります。現在アクティブなログファイルを保存またはバックアップするには、ファイル名を変更してからサーバを再起動する必要があります。サーバは最初にログファイルを名前で検索し、見つからなければ、新しいログファイルを作成します (ユーザが使用できるように、名前を変更された元のログファイルは残っています)。

ログファイルのローテーションについては、flex-rotate-init を参照してください。

flex-init 関数には、次の 3 つのパラメータがあります。ログファイルに名前を付けるパラメータ、そのファイルの各レコードの書式を指定するパラメータ、およびログ作成のモードを指定するパラメータです。

パラメータ

logFileName

パラメータの名前は、ログファイルの名前です。パラメータの値は、ログファイルへの絶対パス、またはサーバの logs ディレクトリに対して相対パスとなるファイル名のいずれかを指定します。次に例を示します。

```
access="/usr/netscape/server4/https-servername/logs/access"
```

```
mylogfile = "log1"
```

このログファイル名は、後で flex-log 関数へのパラメータとして使用します。

format.*logFileName*

ログファイル内の各ログエントリの書式を指定します。

書式については、後述の「ログ書式の詳細」を参照してください。

buffer-size	グローバルログバッファのサイズを指定します。デフォルトは 8192 です。後述の 3 番目の flex-init の例を参照してください。
num-buffers	使用するログ作成バッファの最大数を指定します。デフォルトは 1000 です。後述の 3 番目の flex-init の例を参照してください。

ログ書式の詳細

flex-init 関数は、パーセント記号 (%) で囲まれた部分を、サーバのパラメータブロックに格納されている名前 - 値ペアの名前部分として認識します。(この規則の唯一の例外は、現在のシステム日付を配信する %SYSDATE% コンポーネントです。) %SYSDATE% は、時刻の書式 %d/%b/%Y:%H:%M:%S にグリニッジ標準時からのオフセットを加えたものを使って書式設定されます。

(パラメータブロックの詳細は 第 4 章「カスタム SAF の作成」を、pblock を操作する関数については 第 5 章「NSAPI 関数のリファレンス」を参照してください。)

追加のテキストはすべてリテラルテキストとして扱われるため、行に追加して読みやすくすることができます。書式設定パラメータの一般的なコンポーネントは、表 7-1 に一覧表示されています。コンポーネントによってはスペースが含まれることもあるため、エスケープされた引用符 (\") で区切る必要があります。

ログファイルの書式パラメータが全く指定されていない場合は、共通のログ書式が使われます。

```
"%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
\"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
```

これで、Req->headers.cookie.*name* コンポーネントのログを作成することによって、cookie のログを作成することができます。

次の表では、エスケープされた二重引用符 (\") で囲まれたコンポーネントは、空白を含む値へ解決する可能性のあるコンポーネントです。

表 7-1 flex-init 書式の一般的なコンポーネント

Flex-log オプション	コンポーネント
クライアントホスト名 (flex-log オプションで iponly が指定されていない限り、または DNS 名が使用不可でない限り) または IP アドレス	%Ses->client.ip%
クライアント DNS 名	%Ses->client.dns%

表 7-1　flex-init 書式の一般的なコンポーネント (続き)

Flex-log オプション	コンポーネント
システム日付	`%SYSDATE%`
完全な HTTP 要求行	`\"%Req->reqpb.clf-request%\"`
状態	`%Req->srvhdrs.clf-status%`
応答内容の長さ	`%Req->srvhdrs.content-length%`
応答内容のタイプ	`%Req->srvhdrs.content-type%`
Referer ヘッダー	`\"%Req->headers.referer%\"`
User-Agent ヘッダー	`\"%Req->headers.user-agent%\"`
HTTP メソッド	`%Req->reqpb.method%`
HTTP URI	`%Req->reqpb.uri%`
HTTP 照会文字列	`%Req->reqpb.query%`
HTTP プロトコルのバージョン	`%Req->reqpb.protocol%`
Accept ヘッダー	`%Req->headers.accept%`
Date ヘッダー	`%Req->headers.date%`
If-Modified-Since ヘッダー	`%Req->headers.if-modified-since%`
Authorization ヘッダー	`%Req->headers.authorization%`
任意のヘッダー値	`%Req->headers.`*headername*`%`
承認済みユーザの名前	`%Req->vars.auth-user%`
cookie の値	`%Req->headers.cookie.`*name*`%`
`Req->vars` 内の任意の変数の値	`%Req->vars.`*varname*`%`
仮想サーバ ID	`%vsid%`

例

次の最初の例では、柔軟なログ作成をファイル `/usr/netscape/server4/https-servername/logs/access` に行なうよう初期化します。

```
Init fn=flex-init
access="/usr/netscape/server4/https-servername/logs/access"
format.access="%Ses->client.ip% - %Req->vars.auth-user%
[%SYSDATE%] \"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
```

これにより、次のような項目が記録されます。

- 後ろに「 - 」という 3 文字が付いた ip またはホスト名
- 後ろに「 [」という 2 文字が付いたユーザ名
- 後ろに「] 」という 2 文字が付いたシステム日付
- 後ろに 1 つのスペースが付いた、引用符で囲まれた完全な HTTP 要求
- 後ろに 1 つのスペースが付いた、HTTP 結果状態
- 内容長

これはデフォルトの書式で、共通ログフォーマット (Common Log Format: CLF) に対応します。

多数のログアナライザが出力として予期しているため、ログの最初の 6 つの要素は、常に上記とまったく同じ書式にするようお勧めします。

2 番目の例では、柔軟なログ作成をファイル `/user/netscape/server4/https-servername/logs/extended` に行なうよう初期化します。

```
Init fn=flex-init
extended="/usr/netscape/server4/https-servername/logs/extended"
format.extended="%Ses->client.ip% - %Req->vars.auth-user%
[%SYSDATE%] \"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length% %Req->headers.referer%
\"%Req->headers.user-agent%\" %Req->reqpb.method% %Req->reqpb.uri%
%Req->reqpb.query% %Req->reqpb.protocol%"
```

3 番目の例は、要求処理スレッドが、ログファイルに書き込む時に、呼び出しをログフラッシュスレッドへ委託するのではなく、呼び出しをブロックしてしまうのを防ぐように、ログ作成を調整する方法を示しています。

buffer-size および num-buffers パラメータのサイズをデフォルト値の倍にして、LogFlushInterval magnus.conf 指令の値を 4 秒に下げる (第 7 章「magnus.conf の構文と使用法」を参照) と、要求処理スレッドが解放され、ログデータを迅速に書き込むことができます。

```
Init fn=flex-init buffer-size=16384 num-buffers=2000
access="/usr/netscape/server4/https-servername/logs/access"
format.access="%Ses->client.ip% - %Req->vars.auth-user%
[%SYSDATE%] \"%Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%
%Req->srvhdrs.content-length%"
```

関連項目

flex-rotate-init、flex-log

flex-rotate-init

Init クラス指令で適用可能。

flex-rotate-init 関数は、サーバ上のすべてのログファイルのログのローテーションを構成します。これには、エラーログ、common-log、flex-log、および record-useragent AddLog SAF が含まれます。flex-init を呼び出す前に、magnus.conf の Init セクションにあるこの関数を呼び出します。
flex-rotate-init 関数を使うと、ログファイルをローテーションさせる時間間隔を指定できます。サーバは、指定された時間間隔で、名前が移動の時間を示すファイルにログファイルを移動します。次に、obj.conf の AddLog 段階にあるログ関数が、新しいログファイル内にエントリのログ作成を開始します。ログファイルがローテーションしている間、サーバをシャットダウンしておく必要はありません。

注	サーバはすべてのローテーションされたログファイルを永久に保持するので、ディスクスペースの解放のために、必要に応じてそれらをクリーンアップする必要があります。

デフォルトでは、ログのローテーションは無効になっています。

パラメータ

rotate-start	ローテーションを開始する時刻を示します。この値は、時刻を 24 時制で示す 4 桁の文字列です。たとえば、0900 は午前 9 時を示し、2100 は午後 9 時を示します。
rotate-interval	ログローテーションとログローテーションの間の経過時間を分単位で示します。
rotate-access	(省略可能) common-log、flex-log、および record-useragent ログをローテーションさせるかどうかを決定します。値は、yes (デフォルト) と no です。
rotate-error	(省略可能) エラーログをローテーションさせるかどうかを決定します。値は、yes (デフォルト) と no です。
rotate-callback	(省略可能) ログファイルのローテーションの後に続けて実行する、ユーザが提供したプログラムのファイル名を指定します。このプログラムには、ローテーションされたログファイルのローテーション後の名前が、そのパラメータとして渡されます。

例

次の例は、ログのローテーションを、深夜 0 時に開始して 1 時間ごとに実施されるようにします。

```
Init fn=flex-rotate-init rotate-start=2400 rotate-interval=60
```

関連項目

flex-init、common-log、flex-log、record-useragent

init-cgi

Init クラス指令で適用可能。

init-cgi 関数は、CGI 実行のための特定の初期化タスクを実行します。次の 2 つのオプションがあります。CGI スクリプトの実行のタイムアウトと、環境変数の確立です。

パラメータ

`timeout`	(省略可能) サーバが CGI 出力を待機する時間 (秒) を指定します。その指定した時間内に CGI スクリプトが出力を配信しなかった場合、サーバはそのスクリプトを終了します。デフォルト値は 300 秒です。
`cgistub-path`	(省略可能) CGI スタブバイナリへのパスを指定します。指定しない場合、iPlanet Web Server は、サーバインスタンスの `config` ディレクトリに対して相対パスとなる次のディレクトリを次の順序で調べます。まず `../private/Cgistub` を調べ、次に `../../bin/https/bin/Cgistub` を調べます。 suid Cgistub (つまり、set-user-ID-on-exec ビットセットを持つ、root が所有する Cgistub) を収納するには、最初のディレクトリを使用します。suid 以外の Cgistub を収納するには、2 番目のディレクトリを使用します。2 番目のディレクトリは、iPlanet Web Server 4.*x* サーバが使用する場所です。 存在する場合、`../private` ディレクトリはサーバユーザが所有し、アクセス権 `d??x------` を持っていなければなりません。これにより、ほかのユーザ (たとえば、シェルアカウントまたは CGI アクセス権を持つユーザなど) が Cgistub を使って自分の uid を設定することを防ぎます。 suid Cgistub のインストールについては、iPlanet Web Server の『プログラマーズガイド』を参照してください。
env-variable	(省略可能) サーバが CGI の環境に置く環境変数の名前と値を指定します。環境変数は、1 つの `init-cgi` 関数内にいくつでも設定できます。

例

```
Init fn=init-cgi LD_LIBRARY_PATH=/usr/lib;/usr/local/lib
```

関連項目

`send-cgi`、`send-wincgi`、`send-shellcgi`

init-clf

`Init` クラス指令で適用可能。

init-clf 関数は、共通のログ作成に使われる、指定されたログファイルを開きます。common-log 関数は、要求処理プロセスの AddLog 段階中に、ログファイルにエントリを書き込みます。サーバがシャットダウンされるまで (シャットダウン時にログファイルは閉じられます)、または再起動されるまで (再起動時にログファイルは閉じられ、再度開かれます)、ログファイルは開いたままになります。

注 common-log を呼び出す AddLog 段階指令がサーバにある場合、共通ログファイルは初期化中に init-clf によって初期化されなければなりません。

注 この関数の呼び出しは一回のみにしてください。再度呼び出すと、以前のすべての呼び出しからのログファイル名が新しい呼び出しによって置き換えられます。

サーバをシャットダウンまたは再起動せずにログファイルを移動、削除、または変更すると、クライアントのアクセスが記録されないことがあります。ログファイルを保存またはバックアップするには、ファイル名を変更 (UNIX の場合はさらに、-HUP シグナルも送信) してからサーバを再起動する必要があります。サーバは最初にログファイルを名前で検索し、見つからなければ、新しいログファイルを作成します (ユーザが使用できるように、名前を変更された元のログファイルは残っています)。

ログファイルのローテーションについては、flex-rotate-init を参照してください。

パラメータ

logFileName パラメータの名前は、ログファイルの名前です。パラメータの値は、ログファイルへの絶対パス、またはサーバの logs ディレクトリに対して相対パスとなるファイル名のいずれかを指定します。次に例を示します。

```
access="/usr/netscape/server4/https-servername/logs/access"
mylogfile = "log1"
```

このログファイル名は、後で common-log 関数へのパラメータとして使用します。

例

```
Init fn=init-clf
access=/usr/netscape/server4/https-boots/logs/access
Init fn=init-clf templog=/tmp/mytemplog templog2=/tmp/mytemplog2
```

関連項目

common-log、record-useragent、flex-rotate-init

init-uhome

Init クラス指令で適用可能。

UNIX のみ：init-uhome 関数は、システムのユーザホームディレクトリに関する情報を内部ハッシュテーブルに読み込みます。これによりメモリ使用量が多少増えますが、ホームディレクトリへのトラフィックが大量にあるサーバの場合は、パフォーマンスが向上します。

パラメータ

pwfile	(省略可能) /etc/passwd 以外のファイルへのファイルシステムの絶対パスを指定します。指定しない場合は、デフォルトの UNIX パス (/etc/passwd) が使用されます。

例

```
Init fn=init-uhome
Init fn=init-uhome pwfile=/etc/passwd-http
```

関連項目

unix-home、find-links

load-modules

Init クラス指令で適用可能。

load-modules 関数は、共用ライブラリまたは動的リンクライブラリをサーバコードに読み込みます。ライブラリ中の指定された関数は、それ以降の指令から実行することができます。この関数を使用して、新しいプラグインまたは SAF を読み込みます。

ユーザ独自のサーバアプリケーション関数を定義する場合は、load-modules 関数を使用し、読み込む共用ライブラリまたは dll を指定することによって、サーバにそのサーバアプリケーション関数を読み込ませます。

パラメータ

shlib	共用ライブラリまたは動的リンクライブラリへの絶対パス、またはサーバ構成ディレクトリに対して相対パスとなるファイル名の、いずれかを指定します。
funcs	ほかの Init 指令または obj.conf の Service 指令による使用が可能となる、共用ライブラリまたは動的リンクライブラリ内の関数名の、コンマで区切られたリストです。このリストにはスペースを含めることはできません。関数名では、下線 (_) 文字の代わりにダッシュ (-) 文字を使用することができます。
NativeThread	(省略可能) 使用するスレッドモデルを指定します。 no の場合、ライブラリのルーチンはユーザレベルのスレッドを使用します。 yes の場合、カーネルレベルのスレッドが有効になります。デフォルトは yes です。
pool	thread-pool-init で指定される、カスタムスレッドプールの名前。

例

```
Init fn=load-modules shlib="C:/mysrvfns/corpfns.dll"
funcs="moveit"
Init fn=load-modules shlib="/mysrvfns/corpfns.so"
funcs="myinit,myservice"
Init fn=myinit
```

nt-console-init

Init クラス指令で適用可能。

nt-console-init 関数は、NT コンソールを有効にします。NT コンソールは、標準の出力とエラーストリームを表示するコマンド行シェルです。

パラメータ

stderr	エラーメッセージを NT コンソールに送信します。必須で唯一の値は、`console` です。
`stdout`	出力を NT コンソールに送信します。必須で唯一の値は、`console` です。

例

```
Init fn="nt-console-init" stdout=console stderr=console
```

perf-init

`Init` クラス指令で適用可能。

`perf-init` 関数は、パフォーマンスバケットを介したシステムパフォーマンスの測定を有効にします。

パフォーマンスバケットの詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

パラメータ

`disable`	パフォーマンスバケットを介したシステムパフォーマンス測定の使用を無効にするフラグ。値は true または false です。デフォルト値は true です。

例

```
Init fn=perf-init disable=false
```

関連項目

`define-perf-bucket`

pool-init

Init クラス指令で適用可能。

pool-init 関数は、プールされたメモリー設定のデフォルト値を変更します。空きブロックリストのサイズが変更されるか、プールされたメモリーがすべて使用不可になります。

メモリー割り当てプールを使って、サーバを大幅に高速に稼動させることができます。NSAPI を使ってプログラミングしている場合、プールされたメモリーを使用不可にすると、MALLOC、REALLOC、CALLOC、STRDUP、および FREE が若干異なる動作をするので注意してください。プールが有効な場合、サーバは、各要求が完了したときにこれらのルーチンによって割り当てられたすべてのメモリーを自動的にクリーンアップします。ほとんどの場合、これによってパフォーマンスが向上し、メモリーリークを防ぐことができます。プールが無効になっている場合、すべてのメモリーはグローバルになり、クリーンアップは行なわれません。

持続的なメモリー割り当てにしたい場合は、各ルーチン (PERM_MALLOC、PERM_REALLOC、PERM_CALLOC、PERM_STRDUP、および PERM_FREE) の名前に接頭辞 PERM_ を追加します。

注 Init クラス関数から割り当てるメモリーはすべて、MALLOC を使用する場合でも、持続的なメモリーとして割り当てられます。サーバは、要求の処理中に割り当てられるメモリーだけをクリーンアップします。また、Init クラス関数は要求を処理する前に実行されるため、それらのメモリーはグローバルに割り当てられます。

パラメータ

free-size	(省略可能) 空きブロックリストの最大サイズ (バイト単位)。1048576 より大きくすることはできません。
disable	(省略可能) プールされたメモリーの使用を無効にするフラグ。値は true または false です。デフォルト値は false です。

例

```
Init fn=pool-init disable=true
```

register-http-method

Init クラス指令で適用可能。

この関数は、新しい HTTP メソッドを登録することによって HTTP プロトコルの拡張を可能にします。(デフォルトの HTTP メソッドを登録する必要はありません。)

サーバは、接続を受け入れるときに、受信したメソッドがサーバにとって既知のものかどうかを確認します。サーバがそのメソッドを認識しない場合は、「501 Method Not Implemented」というエラーメッセージを返します。

パラメータ

methods	登録するメソッドの名前の、コンマで区切られたリストです。

例

次の例は、register-http-method と、メソッドの 1 つの Service 関数の使用法を示しています。

```
Init fn="register-http-method" methods="MY_METHOD1,MY_METHOD2"
Service fn="MyHandler" method="MY_METHOD1"
```

stats-init

Init クラス指令で適用可能。

この関数は、XML 形式のパフォーマンス統計のレポート作成を有効にします。実際のレポートは、obj.conf の stats-xml 関数によって生成されます。

パラメータ

update-interval	サーバ内での統計更新の間の期間 (秒単位)。パフォーマンスを良くするには大きめの値を設定し、更新の頻度を上げるには低めの値を設定します。最小値は 1 で、デフォルトは 5 です。
virtual-servers	統計の追跡対象になる仮想サーバの最大数。この数は、構成されている仮想サーバの数より大きく設定する必要があります。それより小さい数を設定すると、メモリ使用量が少なくなります。最小値は 1 で、デフォルトは 1000 です。

profiling	yes に設定すると、バケットを使った NSAPI パフォーマンスのプロファイルを有効にします。これは、perf-init Init SAF を使って有効にすることもできます。デフォルトは no で、この場合、サーバのパフォーマンスが多少向上します。

例

```
Init fn="stats-init" update-interval="5" virtual-servers="2000"
profiling="yes"
```

関連項目

stats-xml

thread-pool-init

Init クラス指令で適用可能。

この関数は、ユーザスレッドの新しいプールを作成します。プールは、使う前に宣言する必要があります。新しいプールを使用するようプラグインに指示するには、Init クラス関数 load-modules でプラグインを読み込むときに pool パラメータを指定します。

カスタムスレッドプールを作成する理由の 1 つは、プラグインがスレッドを意識しない場合に、プール内のスレッドの最大数を 1 に設定できることです。

古いパラメータ NativeThread=yes は、常に NativePool という 1 つのデフォルトのネイティブプールを専用使用します。

UNIX 上のネイティブプールは、すべてのスレッドが OS レベルのスレッドであるため、通常は専用使用されません。UNIX 上のネイティブプールを使用すると、追加のコンテキストスイッチが必要になるため、パフォーマンスのオーバーヘッドがわずかに発生することがありますが、jvm.stickyAttach の効果のローカライズや、リソースの制御と管理、プラグインのシングルスレッド動作のエミュレートなどのその他の目的に使用することができます。

Windows NT では、常にデフォルトのネイティブプールが使われ、iPlanet Web Server は最初の要求処理のためにファイバ (ユーザがスケジュールしたスレッド) を使用します。Windows NT でカスタムの追加プールを使用しても、余分なオーバーヘッドは発生しません。

さらに、ネイティブスレッドプールのパラメータは、利便性のために magnus.conf ファイルに追加することができます。詳細は、第 7 章「magnus.conf の構文と使用法」の 269 ページの「ネイティブスレッドプール」を参照してください。

パラメータ

name	スレッドプールの名前。
maxthreads	プール内のスレッドの最大数。
minthreads	プール内のスレッドの最小数。
queueSize	プールのキューのサイズ。プール内のスレッドがすべて使用中の場合、プールからスレッドを取得しようとしているそれ以降の要求処理スレッドは、プールキューで待機します。キューで待機できる要求処理スレッドの数は、キューのサイズによって制限されます。キューがいっぱいになると、キューに来る次の要求処理スレッドは返され、要求は受け付けられなくなります。ただし、要求処理スレッドは、キュー内でロックされるのではなく、別の要求を処理するために解放されたままです。
stackSize	ネイティブ (カーネル) スレッドプール内の各スレッドのスタックサイズ。

例

```
Init fn=thread-pool-init name="my-custom-pool" maxthreads=5
minthreads=1 queuesize=200

Init fn=load-modules shlib="C:/mydir/myplugin.dll"
funcs="tracker" pool="my-custom-pool"
```

関連項目

load-modules

サーバ情報

ここでは、サーバに関する情報を指定する magnus.conf の指令をリストします。それらの指令は次のとおりです。

- ExtraPath
- MtaHost
- NetSiteRoot
- ServerConfigurationFile
- ServerID
- ServerRoot
- TempDir
- TempDirSecurity
- User

ExtraPath

指定されたディレクトリ名を PATH 環境変数に付加します。これは、Windows NT での Java の構成に使われます。デフォルト値はないので、必ず値を指定する必要があります。

構文

ExtraPath *path*

MtaHost

サーバのエージェントが使用する SMTP メールサーバの名前を指定します。この値は、レポートがメールアドレスに送信される前に指定する必要があります。

NetSiteRoot

サーバインスタンスが下にある最上位ディレクトリへの絶対パス名を指定します。この指令は、Administration Server が使用します。デフォルト値はないので、必ず値を指定する必要があります。

構文

NetSiteRoot *path*

ServerConfigurationFile

仮想サーバの構成ファイルの位置を指定します。

構文

`ServerConfigurationFile` *path*

デフォルト

`ServerConfigurationFile` *server_root*`/`*server_id*`/config/server.xml`

ServerID

`https-boots.mcom.com` などのサーバ ID を指定します。

ServerRoot

サーバルートを指定します。この指令はインストール中に設定され、コメントアウトされます。ほかの指令とは異なり、サーバはこの指令が # で始まることを予期します。この指令は変更しないでください。変更すると、サーバマネージャが正しく機能しないことがあります。

構文

`#ServerRoot` *path*

例

```
#ServerRoot d:/netscape/server4/https-boots.mcom.com
```

TempDir

サーバが一時ファイル用に使用するローカルボリューム上のディレクトリを指定します。UNIX では、サーバを稼動するユーザが、このディレクトリを所有し、書き込み可能でなければなりません。User および TempDirSecurity 指令も参照してください。

構文

`TempDir` *path*

デフォルト

`/tmp` (UNIX)

`TEMP` (Windows NT の場合は環境変数)

TempDirSecurity

TempDir ディレクトリがセキュリティ保護されているかをサーバが確認するかどうかを決定します。UNIX では、`TempDirSecurity` に `off` を指定することによって、サーバが `/tmp` を一時ディレクトリとして使用できます。

警告 `TempDirSecurity` に `off` を指定したり、UNIX で `/tmp` を一時ディレクトリとして使用することはお勧めできません。`/tmp` を一時ディレクトリとして使用すると、潜在的なセキュリティ上のリスクがかなり増すことになります。

構文

```
TempDirSecurity [on|off]
```

デフォルト

```
on
```

User

Windows NT: `User` 指令は、サーバを稼動するためのユーザアカウントを指定します。特定のユーザアカウント (LocalSystem 以外) を使用することで、サーバに対してシステムの機能を制限したり有効にしたりできます。たとえば、ファイルを別のマシンからマウントできるようなユーザアカウントを使用できます。

UNIX: `User` 指令は、サーバ用の UNIX ユーザアカウントを指定します。スーパーユーザまたはルートユーザがサーバを起動すると、サーバはユーザが指定する Port にバインドし、次にそのユーザ ID を、`User` 指令で指定されたユーザアカウントに切り換えます。サーバが `root` として起動されなかった場合、この指令は無視されます。ユーザが指定するユーザアカウントには、サーバのルートとサブディレクトリに対する読み取り権が必要です。このユーザアカウントには、`logs` ディレクトリへの書き込みアクセス権と、すべての CGI プログラムに対する実行権が必要です。このユーザアカウントは、構成ファイルへの書き込みアクセス権は持つべきではありません。これにより、万一だれかがサーバを攻撃しても、構成ファイルを変更してマシンに対する広範なアクセス権を取得することはできません。`nobody` ユーザを使用することはできますが、お勧めはできません。

構文

`User` *name*

`name` は、ユーザアカウント用の 8 文字以下のログイン名です。

デフォルト

User 指令がない場合、サーバは起動に使われたユーザアカウントを使って稼動します。

例

```
User http
User server
User nobody
```

言語に関する問題

この節では、言語上の問題に関連した magnus.conf の指令をリストします。次の指令があります。

- AdminLanguage
- ClientLanguage
- DefaultCharSet
- DefaultLanguage

AdminLanguage

国際バージョンのサーバの場合、この指令はサーバマネージャ用の言語を指定します。値は、en (英語)、fr (フランス語)、de (ドイツ語)、または ja (日本語) です。

デフォルト

デフォルトは en です。

ClientLanguage

国際バージョンのサーバの場合、この指令はクライアントメッセージ (「File Not Found」など) 用の言語を指定します。値は、en (英語)、fr (フランス語)、de (ドイツ語)、または ja (日本語) です。

デフォルト

デフォルトは en です。

DefaultCharSet

国際バージョンのサーバの場合、この指令はサーバ用のデフォルトの文字セットを指定します。デフォルトの文字セットは、クライアントの応答と管理の両方に使われます。

デフォルト
デフォルトは iso-8859-1 です。

DefaultLanguage
国際バージョンのサーバの場合、この指令はサーバ用のデフォルトの言語を指定します。デフォルトの言語は、クライアントの応答と管理の両方に使われます。値は、en (英語)、fr (フランス語)、de (ドイツ語)、または ja (日本語) です。

デフォルト
デフォルトは en です。

DNS 検索

この節では、DNS 検索に影響する magnus.conf の指令をリストします。次の指令があります。

- AsyncDNS
- DNS

AsyncDNS
非同期 DNS を許可するかどうかを指定します。この指令を有効にするには、DNS 指令で on が設定されている必要があります。値は、on または off のいずれかです。DNS を有効に設定すると、非同期 DNS が可能となりサーバのパフォーマンスが向上します。

デフォルト
デフォルトは off です。

DNS
DNS 指令は、サーバにアクセスするクライアント上でサーバが DNS 検索を実行するかどうかを指定します。クライアントがサーバに接続するとき、サーバはクライアントの IP アドレスは知っていますがそのホスト名は知りません (たとえば、サーバはクライアントをホスト名 www.a.com ではなく 198.95.251.30 として認識します)。サーバは、アクセス制御、CGI、エラーレポート、アクセスログの作成などの操作のために、クライアントの IP アドレスをホスト名に解釈処理します。

サーバが 1 日に大量の要求に応答している場合は、ホスト名の解釈処理を中止したい (または中止する必要がある) ことがあります。そうすることにより、DNS または NIS サーバの負荷を削減できます。

構文

```
DNS [on|off]
```

デフォルト

DNS ホスト名解釈処理のデフォルトは on です。

例

```
DNS on
```

スレッド、プロセス、および接続

iPlanet Web Server 6.0 では、待機ソケット上の受け入れ側スレッドが接続を受け付けて、それらの接続を接続キューに入れます。次に、セッションスレッドがキューから接続を取り出して、要求にサービスを提供します。セッションスレッドは、要求の終わりで必要であれば、より多くのセッションスレッドを送信します。新しいスレッドの追加におけるポリシーは、接続キューの状態に基づいています。

- 新しい接続が返されるたびに、キューで待機している接続 (接続の未処理分) の数が、すでに作成されているセッションスレッドの数と比較されます。待機している接続の数がそのスレッドの数より多い場合は、次に要求が完了した時点でスレッドをさらに追加するようにスケジュールされます。
- 時間の経過につれて増加しているように見える場合、およびその増加が `ThreadIncrement` 値より大きく、セッションスレッドから未処理分を引いた数が `ThreadIncrement` 値より小さい場合は、さらに `ThreadIncrement` 数のスレッドを追加するようにスケジュールされるよう、前回の未処理分が追跡されます。
- 新しいセッションスレッドを追加するプロセスは、`RqThrottle` 値によって厳密に制限されます。
- 未処理分が急に増えたときに (たとえばベンチマーク読み込みの開始時など) 作成されるスレッドが多くなりすぎないようにするため、スレッドがさらに必要かどうかの決定がなされます。これは、すでに存在するセッションスレッドの数に基づいて、16 回または 32 回の接続ごとに一回だけ行なわれます。

ここでは、スレッド、プロセス、および接続の数とタイムアウトに影響する `magnus.conf` の指令をリストします。それらの指令は次のとおりです。

- ConnQueueSize
- HeaderBufferSize

- IOTimeout
- KeepAliveThreads
- KeepAliveTimeout
- KernelThreads
- ListenQ
- MaxKeepAliveConnections
- MaxProcs (UNIX のみ)
- PostThreadsEarly
- RcvBufSize
- RqThrottle
- RqThrottleMin
- SndBufSize
- StackSize
- StrictHttpHeaders
- TerminateTimeout
- ThreadIncrement
- UseNativePoll (UNIX のみ)

ネイティブカーネルスレッドのプールを制御する指令については、269 ページの「ネイティブスレッドプール」も参照してください。

ConnQueueSize

Web サーバが保持できる未処理の (まだサービスを受けていない) 接続の数を指定します。この値は、プロセスごとのオープンファイル記述子の最大数に対するオペレーティングシステムの制限値より常に大きくすることをお勧めします。

デフォルト

デフォルト値は 5000 です。

HeaderBufferSize

クライアントから要求データを読み取るために各要求処理スレッドが使用するバッファのサイズ (バイト単位)。要求処理スレッドの最大数は、RqThrottle の設定によって制御されます。

デフォルト
デフォルト値は `8192` (8 K バイト) です。

IOTimeout
クライアントからデータが届くのをサーバが待つ時間 (秒数) を指定します。タイムアウトが期限切れになる前にデータが届かない場合は、接続は閉じられます。この値をデフォルトの 30 秒より小さく設定すれば、早めにスレッドを解放することができます。ただし、低速で接続しているユーザが切り離されることがあります。

構文
IOTimeout *seconds*

デフォルト
ハードウェア暗号化デバイスを使用しないサーバの場合は 30 秒、使用するサーバの場合は 300 秒。

KeepAliveThreads
この指令は、キープアライブサブシステムのスレッドの数を決定します。この数は、システム上のプロセッサ数の小さい倍数にすることをお勧めします。(たとえば、CPU が 2 つのシステムは、キープアライブスレッドの数を 2 または 4 にします)。この設定の値を選ぶときは、許可されているキープアライブ接続の最大数 (`MaxKeepAliveConnections`) も考慮に入れる必要があります。

デフォルト
1

KeepAliveTimeout
この指令は、サーバが、クライアントとサーバ間で HTTP キープアライブ接続または持続接続を開いている最大時間を決定します。前のバージョンのサーバのキープアライブ機能では、サーバがクライアント要求を処理している間、クライアントとサーバの接続を開いたままにしておくことができます。デフォルトの接続は、サーバが接続を閉じるか、接続が `KeepAliveTimeout` で許可された時間を超えるまで接続が開いたままである、持続接続です。

タイムアウトのカウントダウンは、接続がキープアライブサブシステムに渡されるときに開始します。タイムアウトが期限切れになったときに接続に何も動作がない場合、接続は閉じられます。

デフォルト
デフォルト値は 30 秒です。最大値は 300 秒 (5 分) です。

KernelThreads

iPlanet Web Server では、オペレーティングシステムがカーネルレベルのスレッドをサポートしているときは常に、カーネルレベルとユーザレベルの両方のスレッドをサポートできます。ローカルスレッドはプロセス内の NSPR によってスケジュールされ、カーネルスレッドはホストのオペレーティングシステムによってスケジュールされます。通常、標準のデバッガとコンパイラは、カーネルレベルのスレッドの使用が意図されています。`KernelThreads` を `1` (オン) に設定すると、サーバがユーザレベルのスレッドではなく必ずカーネルレベルのスレッドのみを使用するようにできます。`KernelThreads` を `0` (オフ) に設定すると、サーバが必ずユーザレベルのスレッドのみを使用するようにし、これによりパフォーマンスが向上することがあります。

デフォルト
デフォルト値は 0 (オフ) です。

ListenQ

待機ソケット上の、未処理の接続の最大数を指定します。未処理のキューがいっぱいになっている待機ソケットでタイムアウトになった接続は、失敗します。

デフォルト
デフォルト値はプラットフォームに依存します。4096 (AIX)、200 (NT)、128 (その他すべて) です。

MaxKeepAliveConnections

サーバが同時に開いておくことができるキープアライブおよび持続接続の最大数を指定します。値の範囲は 0 ～ 32768 です。

デフォルト

MaxProcs (UNIX のみ)

サーバが同時に実行しておくことができるプロセスの最大数を指定します。`magnus.conf` ファイルに `MaxProcs` を含めない場合、サーバはデフォルトの単一プロセスの実行を採用します。

マルチプロセスモードで実行している場合は、プロセッサごとに 1 つのプロセスをお勧めします。iPlanet Web Server 6.0 では、この設定で指定するアクティブプロセス数のほかに、最初から存在するプロセスが常にあります。

上記に関して、およびその他のサーバ構成およびパフォーマンスの調整に関する問題の補足説明が、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』に記載されています。

デフォルト
1

PostThreadsEarly

この指令が 1 (オン) に設定されている場合、サーバは、接続を受け入れてから応答を要求に送るまでの間に、最小数のスレッドが待機ソケットで利用可能かどうかを確認します。長時間のデータベース接続など、処理に時間のかかる要求をサーバで処理するときは、この指令を使用します。

デフォルト
0 (オフ)

RcvBufSize

ソケットが使用する受信バッファのサイズ (バイト単位) を指定します。指定できる値は、オペレーティングシステムによって決定されます。

デフォルト
デフォルト値はオペレーティングシステムによって決定されます。通常のデフォルトは、4096 (4K バイト)、8192 (8K バイト) です。

RqThrottle

サーバが同時に処理できる要求処理スレッドの最大数を指定します。それぞれの要求は、自身のスレッドで実行されます。

上記に関して、およびその他のサーバ構成およびパフォーマンスの調整に関する問題の補足説明が、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』に記載されています。

デフォルト

RqThrottleMin

サーバの起動時に作成される要求処理スレッドの数を指定します。サーバでの負荷が増加するに従って、作成される要求処理スレッドも増えます (`RqThrottle` スレッドの最大数まで)。

デフォルト

SndBufSize

ソケットが使用する送信バッファのサイズ (バイト単位) を指定します。

デフォルト

デフォルト値はオペレーティングシステムによって決定されます。通常のデフォルトは、`4096` (4K バイト)、`8192` (8K バイト) です。

StackSize

各要求処理スレッドの最大スタックサイズを決定します。

デフォルト

最適なマシン固有のスタックサイズ。

StrictHttpHeaders

厳密な HTTP ヘッダー検査を制御します。厳密な HTTP ヘッダー検査がオンの場合、サーバは不適切に重複されたヘッダーが含まれている接続を拒否します。

構文

StrictHttpHeaders `[on|off]`

デフォルト

`on`

TerminateTimeout

サーバが、シャットダウンする前に、すべての既存の接続が終了するのを待つ時間を指定します。

デフォルト

30 秒

ThreadIncrement

サーバでの負荷の増加 (たとえば、未処理の接続 (要求処理キューにある) の数がアイドル状態の要求処理スレッドの数を超えたときなど) に対処するために作成される、追加の、または新しい要求処理スレッドの数。

サーバは起動時に、要求処理スレッドの RqThrottleMin 番号を作成します。負荷が増えるに従って、サーバは RqThrottle 個の要求処理スレッドが作成されるまで、ThreadIncrement 個の追加要求処理スレッドを作成します。

デフォルト
デフォルト値は 10 です。

UseNativePoll (UNIX のみ)

1 (オン) に設定されると、プラットフォーム固有のポーリングインタフェースを使用します。0 (オフ) に設定されると、KeepAlive サブシステムの NSPR ポーリングインタフェースを使用します。

デフォルト
1 (オン)

ネイティブスレッドプール

この節では、ネイティブカーネルのスレッドプールのサイズを制御する指令をリストします。システム変数 NSCP_POOL_STACKSIZE、NSCP_POOL_THREADMAX、および NSCP_POOL_WORKQUEUEMAX の設定により、ネイティブスレッドプールを制御することもできます。これらの値が環境変数として設定されていて、magnus.conf でも設定されている場合は、環境変数の値が優先されます。

UNIX 上のネイティブプールは、すべてのスレッドが OS レベルのスレッドであるため、通常は専用使用されません。UNIX 上のネイティブプールを使用すると、追加のコンテキストスイッチが必要になるため、パフォーマンスのオーバーヘッドがわずかに発生することがありますが、jvm.stickyAttach 効果のローカライズや、リソースの制御と管理、プラグインのシングルスレッド動作のエミュレートなどのその他の目的に使用することができます。

Windows NT では、常にデフォルトのネイティブプールが使われ、iPlanet Web Server は最初の要求処理のためにファイバ (ユーザがスケジュールしたスレッド) を使用します。Windows NT でカスタムの追加プールを使用しても、余分なオーバーヘッドは発生しません。

次の指令があります。

- NativePoolStackSize
- NativePoolMaxThreads
- NativePoolMinThreads

- NativePoolQueueSize

NativePoolStackSize

ネイティブ (カーネル) スレッドプール内の各スレッドのスタックサイズを決定します。

デフォルト

0

NativePoolMaxThreads

ネイティブ (カーネル) スレッドプール内のスレッドの最大数を決定します。

デフォルト

NativePoolMinThreads

ネイティブ (カーネル) スレッドプール内のスレッドの最小数を決定します。

デフォルト

1

NativePoolQueueSize

スレッドプール用にキューで待機できるスレッドの数を決定します。プール内のすべてのスレッドが使用中である場合、ネイティブプール内のスレッドを使用する必要のある次の要求処理スレッドは、そのキューで待機しなければなりません。キューがいっぱいの場合、キューに入ろうとする次の要求処理スレッドは拒否され、使用中 (ビジー) であるという応答がクライアントに返されます。その後スレッドは、キューで待機して拘束されるのではなく、別の着信要求を処理するために解放されます。

デフォルト

0

CGI

この節では、CGI プログラムに対する要求に影響を与える magnus.conf の指令のリストを示します。次の指令があります。

- CGIExpirationTimeout
- CGIStubIdleTimeout
- CGIWaitPid (UNIX のみ)
- MaxCGIStubs
- MinCGIStubs

CGIExpirationTimeout

この指令は、CGI プロセスが、強制終了される前に、実行することができる最大時間 (秒単位) を指定します。

CGIExpirationTimeout に設定する値が低くなりすぎないようにする必要があります。300 秒 (5 分) はほとんどの対話型 CGI に適した値ですが、それより長い時間かかる (誤った動作なく) ことが予想される CGI の場合は、CGI プログラムを正常に実行するのに予想される最大の時間を設定する必要があります。値 0 は、CGI の期限切れを無効にします。つまり、CGI プロセスの時間制限をなくします。

Windows NT プラットフォームでは、init-cgi タイムアウトは機能しないため、CGIExpirationTimeout を使用する必要があります。

デフォルト

0

CGIStubIdleTimeout

この指令により、サーバは、この指令で設定された秒数の間アイドル状態が続いたすべての CGIStub プロセスを強制終了します。いったんプロセスの数が MinCGIStubs の数になると、サーバはそれ以上のプロセスを強制終了しません。

デフォルト

30

CGIWaitPid (UNIX のみ)

UNIX プラットフォームの場合、CGIWaitPid が on に設定されているときは、SIGCHLD シグナルに対する動作は、そのシグナルに対するシステムのデフォルト動作です。NSAPI プラグインが子プロセスをフォークまたは実行 (exec) する時は、子プロセスの終了時に、「終了した」プロセスから退去することを避けるために CGIWaitPid が有効になっている場合、その子プロセスの pid で waitpid を呼び出す必要があります。CGIWaitPid が有効になっているとき、SHTML エンジンはその exec cmd 子プロセス上で明示的に待機します。この指令は CGI には影響をおよぼさないことに注意してください。

デフォルト

on

MaxCGIStubs

サーバが生成できる CGIStub プロセスの最大数を制御します。これは、実行中の同時 CGIStub プロセスの最大数であり、未処理の要求の最大数ではありません。デフォルトの値は、ほとんどのシステムに対して適切であると思われます。設定値が高すぎると、実際にはスループットが減ることがあります。

デフォルト

10

MinCGIStubs

デフォルトで開始されるプロセスの数を制御します。最初の CGIStub プロセスは、CGI プログラムがアクセスされるまで開始されません。magnus.conf ファイル内に init-cgi 指令がある場合、CGIStub プロセスの最小数は起動時に生成されることに注意してください。この値は、MaxCGIStubs 値より小さくなければなりません。

デフォルト

2

WincgiTimeout

この値より長くかかる WinCGI プロセスは、このタイムアウト (秒) に達すると終了させられます。

デフォルト

60

エラーログ作成と統計収集

この節では、エラーログ作成とサーバ統計の収集に影響する magnus.conf の指令をリストします。それらの指令は次のとおりです。

- ErrorLog
- ErrorLogDateFormat
- LogFlushInterval
- LogVerbose
- LogVsId
- PidLog

ErrorLog

ErrorLog 指令は、サーバがエラーのログを作成するディレクトリを指定します。エラーがファイルにレポートされる場合、ログを保持するファイルとディレクトリは、サーバを稼動するユーザアカウントが何であっても、常にそのユーザアカウントから書き込み可能でなければなりません。

UNIX: syslog 機能も使用できます。

構文

ErrorLog *logfile*

logfile は、絶対パスまたはファイル名のいずれかにすることができます。

UNIX システムでは、キーワード SYSLOG にすることができます (すべて大文字にする必要があります)。

デフォルト

デフォルトのエラーログはありません。

例

Windows NT:

```
ErrorLog C:\Netscape\ns-home\Logs\Errors
```

UNIX:

```
ErrorLog /var/ns-server/logs/errors
ErrorLog SYSLOG
```

ErrorLogDateFormat

ErrorLogDateFormat 指令は、サーバログが使用する日付の書式を指定します。

構文

ErrorLogDateFormat *format*

format は、C ライブラリ関数 strftime に対して有効な、任意の書式です。付録 D「時刻の書式」を参照してください。

デフォルト

%d/%b/%Y:%H:%M:%S

LogFlushInterval

この指令は、ログフラッシュスレッドのログフラッシュの間隔 (秒) を決定します。

デフォルト

30

LogVerbose

この指令は、詳細ログを作成するかどうかを決定します。この値が on の場合、サーバは、デフォルトでは記録されないものを含め、すべてのサーバメッセージのログを作成します。

デフォルト

off

LogVsId

この指令は、仮想サーバ ID をエラーログに表示するかどうかを決定します。複数の仮想サーバが同じログファイルを共有するときは、LogVsId を有効にする必要があります。

デフォルト

off

PidLog

PidLog は、ベースサーバプロセスのプロセス ID (pid) を記録するファイルを指定します。サーバサポートプログラムによっては、このログがサーバルートの logs/pid にあると仮定するものもあります。

サーバをシャットダウンするには、-TERM シグナルを使用して、pid ログファイルにリストされるベースサーバプロセスを強制終了します。構成ファイルを再度読み込んでログファイルを再度開くようサーバに指示するには、kill を -HUP シグナルとともに使用します。

サーバが使用しているユーザアカウントで PidLog ファイルが書き込み可能でない場合、サーバはそのプロセス ID のログをどこにも作成しません。サーバは、プロセス ID のログを作成できない場合、開始しません。

構文

PidLog *file*

file は、プロセス ID が格納される絶対パス名およびファイル名です。

デフォルト

デフォルトはありません。

例

```
PidLog /var/ns-server/logs/pid
PidLog /tmp/ns-server.pid
```

ACL

この節では、アクセス制御リスト (ACL) に関連する magnus.conf の指令をリストします。それらの指令は次のとおりです。

- ACLCacheLifetime
- ACLUserCacheSize
- ACLGroupCacheSize

ACLCacheLifetime

ACLCacheLifetime は、キャッシュエントリが期限切れになるまでの秒数を決定します。キャッシュ内のエントリが参照されるたびに、その経過時間が ACLCacheLifetime に照らして計算され、確認されます。経過時間が ACLCacheLifetime 以上の場合、このエントリは使用されません。この値を 0 に設定すると、キャッシュはオフになります。

この値に大きい数を指定する場合は、LDAP エントリに変更を加えるときに iPlanet Web Server を再起動する必要があることがあります。たとえば、この値を 120 秒に設定すると、iPlanet Web Server は 2 分間 LDAP サーバとの同期を失う可能性があります。LDAP が頻繁に変わる可能性がなさそうな場合は、大きい数を使用します。

デフォルト
120

ACLUserCacheSize

`ACLUserCacheSize` は、ユーザキャッシュ内のユーザ数を決定します。

デフォルト
200

ACLGroupCacheSize

`ACLGroupCacheSize` は、単一の UID/ キャッシュエントリ用にキャッシュできるグループ ID の数を決定します。

デフォルト
4

セキュリティ

この節では、iPlanet Web Server のサーバアクセスとセキュリティの問題に影響する `magnus.conf` の指令をリストします。それらの指令は次のとおりです。

- Security
- SSLCacheEntries
- SSLClientAuthDataLimit
- SSLClientAuthTimeout
- SSLSessionTimeout
- SSL3SessionTimeout

Security

Security 指令は、証明書をサーバインスタンスで利用できるようにすることによって、SSL をグローバルに有効または無効にします。SSL を使用する仮想サーバの場合、これは `on` でなければなりません。無効になっている場合、ユーザに対して管理者パスワードを求めるプロンプトが表示されます (証明書などにアクセスするため)。

注 セキュリティ保護された待機ソケットをサーバマネージャで作成するとき、セキュリティは自動的に `magnus.conf` でグローバルにオンになります。セキュリティ保護された待機ソケットを `server.xml` で手動で作成するときは、`magnus.conf` を編集してセキュリティをオンにする必要があります。

個々の仮想サーバに対する SSL の有効化についての詳細は、第 8 章「仮想サーバの構成ファイル」を参照してください。

構文
`Security [on|off]`

デフォルト
`off`

例
`Security off`

SSLCacheEntries
キャッシュできる SSL セッションの数を指定します。上限はありません。

構文
`SSLCacheEntries` *number*

number が 0 の場合、デフォルト値 10000 が使われます。

SSLClientAuthDataLimit
クライアント証明書のハンドシェーク段階でバッファされるアプリケーションデータの最大量 (バイト) を指定します。

デフォルト
デフォルト値は、`1048576` (1 M バイト) です。

SSLClientAuthTimeout
クライアント証明書のハンドシェーク段階がタイムアウトするまでの秒数を指定します。

デフォルト
`60`

SSLSessionTimeout

SSLSessionTimeout 指令は、SSL2 セッションのキャッシュ化を制御します。

構文

SSLSessionTimeout *seconds*

seconds 値は、キャッシュされた SSL2 セッションが無効になるまでの秒数です。SSLSessionTimeout 指令が指定されている場合、この秒数の値は暗黙的に 5 ～ 100 秒間に制限されます。

デフォルト

デフォルト値は 100 です。

SSL3SessionTimeout

SSL3SessionTimeout 指令は、SSL3 セッションのキャッシュ化を制御します。

構文

SSL3SessionTimeout *seconds*

seconds 値は、キャッシュされた SSL3 セッションが無効になるまでの秒数です。デフォルト値は 86400 (24 時間) です。SSL3SessionTimeout 指令が指定されている場合、この秒数の値は暗黙的に 5 ～ 86400 秒間に制限されます。

チャンクされたエンコーディング

この節では、チャンクされたエンコーディングを制御する指令をリストします。詳細は、326 ページの「バッファ化されたストリーム」を参照してください。

- UseOutputStreamSize
- ChunkedRequestBufferSize
- ChunkedRequestTimeout

これらの指令に対しては、obj.conf 内に同等の Service SAF パラメータがあります。obj.conf パラメータは、これらの指令をオーバーライドします。詳細は、80 ページの「Service 段階」を参照してください。

UseOutputStreamSize

UseOutputStreamSize 指令は、net_read および netbuf_grab NSAPI 関数用のデフォルトの出力ストリームバッファサイズを決定します。

注	UseOutputStreamSize パラメータを obj.conf ファイルでゼロに設定して、出力ストリームバッファリングを無効にすることができます。magnus.conf ファイルの場合は、UseOutputStreamSize をゼロに設定しても効果はありません。

構文

UseOutputStreamSize *size*

size 値は、バイト数です。

デフォルト

デフォルト値は 8192 (8 K バイト) です。

ChunkedRequestBufferSize

ChunkedRequestBufferSize 指令は、「チャンク解除」要求データのデフォルトのバッファサイズを決定します。

構文

ChunkedRequestBufferSize *size*

size 値は、バイト数です。

デフォルト

デフォルト値は 8192 です。

ChunkedRequestTimeout

ChunkedRequestTimeout 指令は、「チャンク解除」要求データのデフォルトのタイムアウトを決定します。

構文

ChunkedRequestTimeout *seconds*

seconds 値は、秒数です。

デフォルト

デフォルト値は 60 (1 分) です。

その他

この節では、magnus.conf のその他の指令をリストします。

- ChildRestartCallback
- HTTPVersion
- MaxRqHeaders
- Umask (UNIX のみ)

注	ブール値が記載されている指令には、次のような同等の値があります。on/yes/true および off/no/false。

ChildRestartCallback

この指令は、サーバが再起動またはシャットダウンされるときに、daemon_atrestart 関数を使用して登録された NSAPI 関数のコールバックを強制します。値は、on、off、yes、no、true、または false です。

デフォルト
no

HTTPVersion

m.n という書式の、サーバが使用する現在の HTTP バージョン。ここで、*m* はメジャーバージョンの番号で、*n* はマイナーバージョンの番号です。

デフォルト
デフォルト値は 1.1 です。

MaxRqHeaders

1 つの要求内のヘッダー行の最大数を指定します。値の範囲は 0 ～ 32 です。

デフォルト
32

Umask (UNIX のみ)

この指令は、さまざまなモードでファイルを開くために NSAPI 関数 `System_fopenWA()` および `System_fopenRW()` が使用する umask 値を指定します。この指令に有効な値は、UNIX の標準 umask 値です。

これらの関数の詳細は、第 5 章「NSAPI 関数のリファレンス」の `system_fopenWA` および `system_fopenRW` を参照してください。

第 8 章

仮想サーバの構成ファイル

server.xml ファイルは、仮想サーバを構成します。マスターファイル server.dtd は、server.xml ファイルの書式と内容を決定します。この章では、これら 2 つのファイルについて説明します。この章は、次の節で構成されています。

- server.dtd ファイル
- server.xml ファイル
- server.dtd および server.xml 内の要素
- 要求処理のための仮想サーバの選択
- ユーザデータベースの選択
- iPlanet LDAP スキーマ

server.dtd ファイル

server.dtd ファイルは、server.xml ファイルに含めることができる要素と、それらの要素に設定できる属性を定義します。server.dtd ファイルは、*server_root*/*server_id*/config ディレクトリにあります。

注 server.dtd ファイルは編集しないでください。このファイルの内容は、iPlanet Web Server のバージョンが更新されたときだけ変更されます。

たとえば、次のコードは、VSCLASS (または仮想サーバクラス) 要素を定義しています。最初の行では、VS 要素に VARS、VS、または QOSPARAMS 要素を含めることができるように指定しています (この要素にほかの要素を含めることができない場合は、カッコ () の中に要素名のリストではなく EMPTY が表示されます)。残りの行では、VSCLASS 要素に id、objectfile、rootobject、または acceptlanguage 属性を含めることができるが、id 属性のみが必須であることを指定しています。

```
<!ELEMENT VSCLASS (VARS?,VS*,QOSPARAMS?)>
<!ATTLIST VSCLASS
   id ID #REQUIRED
   objectfile CDATA #IMPLIED
   rootobject CDATA #IMPLIED
   acceptlanguage (yes|no|on|off|1|0) #IMPLIED
```

ID や CDATA などのラベルは、XML データ型です。XML の詳細は、次の場所にある XML の仕様を参照してください。

http://www.w3.org/TR/REC-xml

server.xml ファイル

server.xml ファイルは、サーバが待機するアドレスとポートを構成し、仮想サーバをそれらの待機ソケットに割り当てます。エンコーディングは、通常の UNIX テキストエディタと互換性を保つために、UTF-8 を使用します。server.xml ファイルは、*server_root*/https-*server_id*/config ディレクトリにあります。

次に示すのは、簡単な server.xml ファイルです。これには、2 つの待機ソケット (LS)、2 つの仮想サーバクラス (VSCLASS)、および 3 つの仮想サーバ (VS) が含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- declare any variables to be used in the server.xml file in the
ATTLIST below -->
<!DOCTYPE SERVER SYSTEM "server.dtd" [
<!ATTLIST VARS
   docroot CDATA #IMPLIED
   adminusers CDATA #IMPLIED
   webapps_file CDATA #IMPLIED
   webapps_enable CDATA #IMPLIED
   accesslog CDATA #IMPLIED
   user CDATA #IMPLIED
   group CDATA #IMPLIED
   chroot CDATA #IMPLIED
```

```
    dir CDATA #IMPLIED
    nice CDATA #IMPLIED
 >
]>

<SERVER legacyls="ls1">
    <VARS accesslog="/iws60/https-server.iplanet.com/logs/access"/>
    <LS id="ls1" ip="1.1.1.1" port="80" security="off"
    acceptorthreads="1">
        <CONNECTIONGROUP id="group1" matchingip="default"
        servername="server.iplanet.com"
        defaultvs="server.iplanet.com"/>
    </LS>
    <LS id="ls2" ip="any" port="80" security="off"
    acceptorthreads="1">
        <CONNECTIONGROUP id="group2" matchingip="default"
        servername="server2.iplanet.com"
        defaultvs="server2.iplanet.com"/>
    </LS>
    <MIME id="mime1" file="mime.types" />
    <ACLFILE id="acl1"
    file="/iws60/httpacl/generated.https-server.iplanet.com.acl" />
    <VSCLASS id="defaultclass" objectfile="obj.conf"
    rootobject="default" >
        <VARS docroot="/iws60/docs" />
        <VS id="server.iplanet.com" connections="group1" mime="mime1"
        aclids="acl1">
            <VARS webapps_file="web-apps.xml" webapps_enable="on" />
            <USERDB id="default" database="default" />
        </VS>
    </VSCLASS>
    <VSCLASS id="class2" objectfile="class2.obj.conf"
    rootobject="default" >
        <VARS docroot="/iws60/docs/class2" />
        <VS id="server2.iplanet.com" connections="group2"
        mime="mime1" aclids="acl1">
            <VARS webapps_file="web-apps.xml" webapps_enable="on" />
            <USERDB id="default" database="default" />
        </VS>
        <VS id="acme.com" connections="group2"
        mime="mime1" aclids="acl1">
            <VARS docroot="/iws60/docs/class2/acme"
            webapps_file="web-apps.xml" webapps_enable="on" />
            <USERDB id="default" database="default" />
        </VS>
    </VSCLASS>
</SERVER>
```

IP アドレスまたは Host ヘッダーと一致する仮想サーバ (VS) が見つからない場合、要求は CONNECTIONGROUP に定義されているデフォルトの VS を使って処理されます。この VS は、カスタマイズされたエラーメッセージを出力したり、特殊なドキュメントルートを使用して要求を処理するようにすることもできます。

変数

obj.conf ファイルで使用するために変数を定義することは、必須ではありませんが、定義すると役に立つこともあります。次のコードでは、docroot 変数を定義して使用しています。

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- declare any variables to be used in the server.xml file in the
ATTLIST below -->
<!DOCTYPE SERVER SYSTEM "server.dtd" [
<!ATTLIST VARS
    docroot CDATA #IMPLIED
    ...
 >
]>
...
       <VS id="acme.com" connections="group2"
       mime="mime1" aclids="acl1">
           <VARS docroot="/iws60/docs/class2/acme"
           webapps_file="web-apps.xml" webapps_enable="on" />
           <USERDB id="default" database="default" />
       </VS>
...
```

docroot 変数を使うことによって、それぞれの仮想サーバに、異なるドキュメントルートディレクトリを割り当てることができます。その後、この変数は obj.conf ファイルでも使用できます。次に例を示します。

```
NameTrans fn=document-root root="$docroot"
```

この docroot 変数を使うと、obj.conf ファイルに仮想サーバクラス用のドキュメントルートを個別に定義しなくてもすみます。また、同じ仮想サーバクラスの仮想サーバごとに、異なるドキュメントルートを定義することもできます。

注 変数の置換が許可されているのは、obj.conf ファイル内だけです。iPlanet Web Server のほかの構成ファイルでは許可されていません。

obj.conf ファイルで参照される変数はすべて、server.xml ファイル内に SERVER、VSCLASS、または VS レベルで定義する必要があります。デフォルト値を持つ変数を SERVER または VSCLASS レベルで定義し、それらを VS でオーバーライドすることをお勧めします。

変数の書式

次のような正規表現と一致する文字列が、変数であると認識されます。

```
\$[A-Za-z][A-Za-z0-9_]*
```

この式は、$ とそれに続く 1 つ以上の英数字を表しています。区切り文字で区切られたバージョン ("${VARS}") はサポートされていません。通常の $ 文字を取得するには、ファイル内で $$ を使用して変数の置換を行ないます。

id 変数

特殊変数 id は、常に VS 要素内で利用することができ、id 属性の値を参照します。これは事前に定義されており、オーバーライドはできません。id 属性は、仮想サーバを一意に識別します。次に例を示します。

```
<VARS
    docroot="/export/$id"
/>
```

包含する VS の id 属性が myserver の場合、docroot 変数は値 /export/myserver に設定されます。

インタフェースで使われる変数

次に示す変数は、Administration Server、サーバマネージャ、クラスマネージャ、および仮想サーバマネージャで使用されます。$id 変数とは違い、これらの変数はサーバで事前に定義されていないため、オーバーライドすることができます。

$docroot	仮想サーバのドキュメントルート。通常、obj.conf ファイル内の document-root パラメータの値として評価されます。
$webapps_file	Web アプリケーションの構成ファイルのパスおよび名前。通常は web-apps.xml です。web-apps.xml の詳細は、iPlanet Web Server の『サーブレットに関するプログラマーズガイド』を参照してください。

$webapps_enable	VS に対して Web アプリケーションが有効になっているかどうかを示すフラグ。指定できる値は、on と off です。webapps_file 変数に VS の値がある場合、この変数は定義する必要はなく、on とみなされます。
$accesslog	仮想サーバのログファイル。
$user	send-cgi SAF の user パラメータの値。
$group	send-cgi SAF の group パラメータの値。
$chroot	send-cgi SAF の chroot パラメータの値。
$dir	send-cgi SAF の dir パラメータの値。
$nice	send-cgi SAF の nice パラメータの値。

変数評価

変数は、個々の仮想サーバについて特定の objset を生成するときに評価されます。

評価は再帰的であり、変数値にほかの変数を含めることができます。次に例を示します。

```
...
<SERVER>
    <VARS docrootbase = "/export" />
    ...
    <VSCLASS ...>
        <VARS docroot = "$docrootbase/nonjava/$id" />
        ...
    </VSCLASS>
    <VSCLASS ...>
        <VARS docroot = "$docrootbase/java/$id" />
        ...
    </VSCLASS>
    ...
</SERVER>
...
```

ツリーの下側にある変数が、上にある変数をオーバーライドします。たとえば、仮想サーバのクラスに対して変数を設定し、個々の仮想サーバ内の同じ変数の定義でその変数をオーバーライドすることができます。

サーバマネージャとクラスマネージャの使用

iPlanet Web Server の『管理者ガイド』で説明されているように、サーバマネージャとクラスマネージャのインタフェースを介して、仮想サーバクラスと仮想サーバを iPlanet Web Server に追加することができます。

server.dtd および server.xml 内の要素

この節では、server.dtd および server.xml ファイル内の XML 要素について説明します。部分要素は、それらのリストされている順序で定義する必要があります。

SERVER

サーバを定義します。server.xml ファイルには、この要素のうちの 1 つだけを設定することができます。

部分要素：VARS、LS、MIME、ACLFILE、VSCLASS、QOSPARAMS

属性：

qosactive	サービス品質の機能を有効にします。これにより、サーバエンティティに制限を設定したり、帯域幅と接続のサーバ統計を表示したりできます。指定できる値は、yes、no、on、off、1、0 です。デフォルトは no です。
qosmetricsinterval	(省略可能) トラフィックを測定する間隔 (秒)。デフォルトは 30 です。
qosrecomputeinterval	(省略可能) すべてのサーバエンティティの帯域幅を再計算する期間 (ミリ秒)。デフォルトは 100 ms です。
legacyls	古いバージョン (4.*x*) のアプリケーションの待機ソケットの id 属性。この LS には、1 つの CONNECTIONGROUP だけを含める必要があります。さらに、CONNECTIONGROUP には、その defaultvs である 1 つの VS だけを指定する必要があります。古いバージョンのアプリケーションはすべてこの仮想サーバ上で稼動する必要があり、これがサーバ全体に対するデフォルトの仮想サーバです。

VARS

server.xml に値を指定することができ、obj.conf で参照できる変数を定義します。286 ページの「変数」を参照してください。server.xml で通常定義される変数のリストについては、287 ページの「インタフェースで使われる変数」を参照してください。

部分要素 : なし

属性 : なし

LS (待機ソケット)

待機ソケットを定義します。

注	セキュリティ保護された待機ソケットをサーバマネージャで作成するとき、セキュリティは自動的に magnus.conf でグローバルにオンになります。セキュリティ保護された待機ソケットを server.xml で手動で作成するときは、magnus.conf を編集してセキュリティをオンにする必要があります。

部分要素 : CONNECTIONGROUP

属性 :

id	(省略可能) ソケットファミリのタイプ。ソケットファミリのタイプは、数字で始めることはできません。 セキュリティ保護された待機ソケットを server.xml ファイルで作成するときは、magnus.conf で Security をオンにする必要があります。セキュリティ保護された待機ソケットをサーバマネージャで作成するとき、セキュリティは自動的に magnus.conf でグローバルにオンになります。待機ソケットの名前は、数字で始めることはできません。
ip	待機ソケットの IP アドレス。ドットで区切ったペアまたは IPv6 の表記法で指定できます。また、INADDR_ANY の any も指定できます。複数の CONNECTIONGROUP が指定されている場合は、any で待機するようにその待機ソケットを構成する必要があります。
port	待機ソケットを作成するポート番号。有効な値は、1 ～ 65535 です。UNIX の場合、ポート 1 ～ 1024 で待機するソケットを作成するには、スーパーユーザ特権が必要です。ポート 443 で待機するように SSL 待機ソケットを構成するようお勧めします。2 つの異なる IP アドレスが同じポートを使用することはできません。

security	(省略可能) 待機ソケットが SSL を実行するかどうかを決定します。有効な値は、on、off、yes、no、1、0 です。デフォルトは no です。SSL2 または SSL3 をオンまたはオフにし、CONNECTIONGROUP オブジェクト内の SSLPARAMS オブジェクトを使用して符号化方式を設定することができます。 magnus.conf ファイル内の Security 設定は、サーバインスタンスに対して証明書を利用できるようにすることで、SSL をグローバルに有効または無効にします。したがって、magnus.conf 内の Security は on にする必要があります。そうしないと、server.xml の security は機能しません。詳細は、第 7 章「magnus.conf の構文と使用法」を参照してください。
acceptorthreads	(省略可能) 待機ソケットの受け入れ側スレッドの数。推奨値は、マシン内のプロセッサの数です。デフォルトは 1 で、有効な値は 1 ～ 1024 です。
family	(省略可能) ソケットファミリのタイプ。デフォルトは inet で、有効な値は inet、inet6、および nca です。IPv6 の待機ソケットの場合は、値 inet6 を使用します。値 inet6 を使用すると、ログファイル内の IPv4 アドレスに接頭辞 ::ffff: が付きます。Solaris Network Cache および Accelerator を使用するよう nca を指定します。
blocking	(省略可能) 待機ソケットと受け入れられたソケットをブロックモードにするかどうかを決定します。ブロックモードを使用すると、ベンチマーク結果が向上する場合があります。有効な値は、on、off、yes、no、1、0 です。デフォルトは no です。

警告 ブロックモードのソケットは、現実の導入では使用しないでください。ブロックモードのソケットを使用すると、動的な再構成ができなくなり、サーバをサービス妨害攻撃の危険にさらすことになります。

CONNECTIONGROUP

仮想サーバを割り当てることができる接続プロパティのグループを定義します。詳細は、298 ページの「要求処理のための仮想サーバの選択」を参照してください。

部分要素 : SSLPARAMS

属性 :

id	接続グループの内部名。CONNECTIONGROUP 名は、数字で始めることはできません。
matchingip	関連付けられた仮想サーバが使用する IP アドレス、または値 default。ドットで区切った数字の組み合わせ、または IPv6 の表記法で指定できます。INADDR_ANY の any は指定できません。包含する LS に ip=any がない場合は、default でなければなりません。 包含する LS に ip=any がある場合は、特定の IP アドレスまたは default にすることができます。この場合、default は、ほかの LS または CONNECTIONGROUP 要素で指定されていない任意の IP アドレスを意味します。
defaultvs	この特定の接続グループのための、デフォルトの仮想サーバの id 属性。
servername	サーバがクライアントに送信する URL のホスト名セクションに指定する名前をサーバに伝えます。これは、サーバが自動的に生成する URL に影響を与えます。サーバに格納されるディレクトリおよびファイルの URL には影響を与えません。サーバがエイリアスを使用する場合は、この名前もエイリアス名にする必要があります。 コロンとポート番号を付加すると、サーバがクライアントに送信する URL でそのポートが使われます。

SSLPARAMS

接続グループの SSL パラメータを定義します。

1 つの SSLPARAMS 要素を、security 属性が on に設定されている待機ソケットに含まれる CONNECTIONGROUP 要素の内側に指定する必要があります (内側だけに指定できます)。

部分要素 : なし

属性 :

servercertnickname	証明書データベースまたは PKCS#11 トークン内のサーバ証明書のニックネーム。証明書では、名前の書式は *tokenname* : *nickname* です。この属性内の名前の *tokenname* : の部分は、省略可能です。

ssl2	(省略可能) SSL2 を有効にするかどうかを決定します。有効な値は、on、off、yes、no、1、0 です。デフォルトは no です。 1 つの仮想サーバに対して SSL2 と SSL3 の両方が有効になっている場合、サーバは最初に SSL3 暗号化を試みます。それに失敗すると、SSL2 暗号化を試みます。
ssl2ciphers	(省略可能) 使用される SSL2 符号化方式の、スペースで区切られたリスト。有効にするには接頭辞 + を付け、無効にするには - を付けます。たとえば、+rc4 のようになります。指定できる値は、rc4export、rc2export、idea、des です。
ssl3	(省略可能) SSL3 を有効にするかどうかを決定します。有効な値は、on、off、yes、no、1、0 です。デフォルトは yes です。 1 つの仮想サーバに対して SSL2 と SSL3 の両方が有効になっている場合、サーバは最初に SSL3 暗号化を試みます。それに失敗すると、SSL2 暗号化を試みます。
ssl3tlsciphers	(省略可能) 使用される SSL3 符号化方式の、スペースで区切られたリスト。有効にするには接頭辞 + を付け、無効にするには - を付けます。たとえば、+rsa_des_sha のようになります。指定できる SSL3 値は、rsa_des_sha、rsa_rc4_40_md5、rsa_rc2_40_md5、rsa_null_md5 です。指定できる TLS 値は、rsa_des_56_sha、rsa_rc4_56_sha です。
tls	(省略可能) TLS を有効にするかどうかを決定します。有効な値は、on、off、yes、no、1、0 です。デフォルトは on です。
tlsrollback	(省略可能) TLS ロールバックを有効にするかどうかを決定します。有効な値は、on、off、yes、no、1、0 です。デフォルトは on です。Microsoft Internet Explorer 5.0 および 5.5 の場合は、TLS ロールバックを有効にする必要があります。詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。
clientauth	(省略可能) ACL ベースのアクセス制御から独立して、すべての要求で SSL3 クライアント認証を実行するかどうかを決定します。有効な値は、on、off、yes、no、1、0 です。デフォルトは no です。

MIME

MIME タイプを定義します。

要求された、リソースの MIME タイプをサーバが決定するもっとも一般的な方法は、`obj.conf` ファイルの `ObjectType` セクションにある `type-by-extension` 指令を呼び出すことです。`type-by-extension` 関数は、`SERVER` 要素で `MIME` 要素が定義されていない場合は機能しません。

部分要素 : なし

属性 :

`id`	MIME タイプのリストの内部名。仮想サーバが使用する MIME タイプを定義するために、`VS` 要素で使用されます。MIME タイプ名は、数字で始めることはできません。
`file`	MIME タイプのファイルの名前。このファイルの書式の詳細は、付録 B「MIME タイプ」を参照してください。

ACLFILE

1 つ以上の ACL ファイルを参照します。

部分要素 : なし

属性 :

`id`	ACL ファイルのリストの内部名。仮想サーバが使用する ACL ファイルを定義するために、`VS` 要素で使用されます。ACL ファイルのリスト名は、数字で始めることはできません。
`file`	ACL ファイルの、スペースで区切られたリスト。各 ACL ファイルの名前は一意でなければなりません。ACL ファイルの書式の詳細は、iPlanet Web Server の『管理者ガイド』を参照してください。 デフォルトの ACL ファイルの名前は、`generated.https-`*server_id*`.acl` で、このファイルは *server_root*`/`*server_id*`/httpacl` ディレクトリにあります。このファイルを使用するには、`server.xml` 内でこのファイルを参照する必要があります。

VSCLASS

仮想サーバクラスを定義します。

部分要素：VARS、VS、QOSPARAMS

属性：

id	仮想サーバクラス ID。これは、特定の仮想サーバクラスの検索を可能にする一意な ID です。仮想サーバクラス ID は、数字で始めることはできません。
objectfile	仮想サーバのこのクラスのための obj.conf ファイルのファイル名。VS 要素内でオーバーライドすることはできません。
rootobject	(省略可能) obj.conf ファイルから読み込まれるオブジェクトのうちどれがデフォルトかをサーバに伝えます。デフォルトのオブジェクトには、その仮想サーバのためのすべての名前変換 (NameTrans) 指令があることが期待されており、デフォルトのオブジェクト内に構成されているサーバの動作はすべてサーバ全体に影響を与えます。デフォルト値は default です。 存在しないオブジェクトを指定すると、サーバはクライアントがドキュメントを取得しようとするまでエラーを報告しません。サーバマネージャでは、デフォルトは default という名前のオブジェクトであるとみなします。サーバマネージャを使用する (または使用を予定している) 場合は、この規則から外れないようにしてください。
acceptlanguage	(省略可能) on の場合、サーバは Accept-Language ヘッダーを構文解析し、クライアントが受け入れ可能な言語に基づいて、適切な言語のバージョンを送信します。サーバが複数の言語をサポートする場合にのみ、この値を on にすべきです。デフォルトは off です。VS 要素内でオーバーライドすることができます。

VS (仮想サーバ)

仮想サーバを定義します。

部分要素：VARS、QOSPARAMS、USERDB

属性：

id	仮想サーバ ID。これは、特定の仮想サーバの検索を可能にする一意な ID です。obj.conf ファイル内の変数 $id として参照することもできます。仮想サーバ ID は、数字で始めることはできません。
connections	(省略可能) 仮想サーバが使用する接続 (1 つまたは複数) を指定する、スペースで区切られた CONNECTIONGROUP id のリスト。CONNECTIONGROUP の defaultvs ではない VS の場合に限り必要です。
urlhosts	現在の仮想サーバを選択するために Host 要求ヘッダーで許可される値の、スペースで区切られたリスト。同じ CONNECTIONGROUP に指定されている各 VS には、そのグループに対して一意な urlhosts 値がなければなりません。
mime	仮想サーバが使用する MIME 要素の id。
state	(省略可能) VS がアクティブ (on) か非アクティブ (off、disable) かを決定します。デフォルトは on (アクティブ) です。非アクティブのとき、VS は要求にサービスを提供しません。 VS が disable の場合、これを on にできるのはグローバルサーバ管理者だけです。
aclids	(省略可能) スペースで区切られた、ACLFILE 要素の 1 つ以上の id 属性。仮想サーバが使用する ACL ファイル (1 つまたは複数) を指定します。
errorlog	(省略可能) 仮想サーバ固有のエラーメッセージ用のログファイルを指定します。
acceptlanguage	(省略可能) on の場合、サーバは Accept-Language ヘッダーを構文解析し、クライアントが受け入れ可能な言語に基づいて、適切な言語のバージョンを送信します。サーバが複数の言語をサポートする場合にのみ、この値を on にすべきです。デフォルトは off です。

QOSPARAMS

SERVER、VSCLASS、または VS のサービス品質のパラメータを定義します。

SERVER 要素の属性は、サービス品質の機能を有効にします。さらに、qos-handler および qos-error SAF は、obj.conf ファイルに含まれている必要があります。

詳細は、『Performance Tuning, Sizing, and Scaling Guide for iPlanet Web Server』を参照してください。

部分要素 : なし

属性 :

maxbps	(省略可能) SERVER、VSCLASS、または VS の帯域幅の最大制限値 (単位 : bps)。
enforcebandwidth	(省略可能) 帯域幅制限を強制するかどうかを指定します。指定できる値は、yes、no、on、off、1、0 です。デフォルトは no です。
maxconn	(省略可能) SERVER、VSCLASS、または VS の同時接続の最大数。
enforceconnections	(省略可能) 接続制限を強制するかどうかを指定します。指定できる値は、yes、no、on、off、1、0 です。デフォルトは no です。

USERDB

仮想サーバが使用するユーザデータベースを定義します。

指定された仮想サーバに対してユーザデータベースが選択される方法の詳細は、299 ページの「ユーザデータベースの選択」を参照してください。

部分要素 : なし

属性 :

id	仮想サーバの ACL ファイル内のユーザデータベース名。ユーザデータベース名は、数字で始めることはできません。
database	dbswitch.conf ファイル内のユーザデータベース名。
basedn	(省略可能) dbswitch.conf ファイル内のベース DN 検索をオーバーライドします。ただし、basedn 値は、依然として dbswitch.conf エントリからのベース DN 値に対して相対的です。
certmaps	(省略可能) LDAP エントリマッピング (certmap.conf で定義されている) に対して使用する証明書を指定します。存在しない場合、すべてのマッピングが使われます。certmap.conf 内のマッピングに基づくすべての検索は、VS の最終ベース DN に対して相対的です。

要求処理のための仮想サーバの選択

サーバは、要求を処理する前に、待機ソケットを介して要求を受け入れてから、適切な接続グループと仮想サーバにその要求を送信する必要があります。この節では、仮想サーバがどのように決定されるかについて説明します。

仮想サーバが決定されたら、サーバは、その仮想サーバが属する仮想サーバクラスの `obj.conf` ファイルを実行します。サーバが `obj.conf` で実行する指令を決定する方法の詳細は、32 ページの「obj.conf 内の制御のフロー」を参照してください。

接続グループは、最初に次のように選択されます。

- 待機ソケットが特定の IP アドレスで待機するよう構成されている場合は、1 つの接続グループだけを含めることができ、そのグループが選択されます。
- 待機ソケットが `any` で待機するよう構成されている場合、クライアントが接続した IP アドレスは、その待機ソケットに含まれる接続グループの `matchingip` 属性とマッチングされます。一致する `matchingip` 属性がない場合、`matchingip=default` を持つ接続グループが選択されます。

その後、仮想サーバは次のように選択されます。

- 接続グループがデフォルトの仮想サーバのみに指定されている場合は、その仮想サーバが選択されます。
- 複数の仮想サーバが指定されている接続グループの場合、要求 `Host` ヘッダーは仮想サーバの `urlhosts` 属性とマッチングされます。`Host` ヘッダーがないか、一致する `urlhosts` 属性がない場合、その接続グループのデフォルトの仮想サーバが選択されます。

仮想サーバが SSL 待機ソケットに指定されている場合、その `urlhosts` 属性はサーバ起動時に証明書のサブジェクトパターンに照らして検査され、一致しなければ警告が生成されてエラーログに書き込まれます。

ユーザデータベースの選択

USERDB オブジェクトは、包含する仮想サーバのユーザデータベースを選択します。この選択は、仮想サーバの ACL ファイルと dbswitch.conf ファイルによって決まります。

ACL ファイルの書式は、以前のバージョンの iPlanet Web Server から変更されていません。ただし、iPlanet Web Server 6.0 では次のような変更が行なわれました。

- server.xml 内の仮想サーバは、ACL ファイルを参照します。magnus.conf ファイルは、ACL ファイルを参照しなくなりました。
- ACL ファイルの database 属性は、直接 dbswitch.conf エントリにマッピングされません。代わりに、USERDB 要素の id 属性にマッピングされます。その後、USERDB 要素の database 属性は、dbswitch.conf エントリにマッピングされます。ACL ファイルと dbswitch.conf ファイルの間のこのような特別な層により、サーバ管理者は、仮想サーバ管理者およびユーザがどのデータベースに対してアクセス権を持つかを完全に制御できます。

iPlanet Web Server 6.0 では、dbswitch.conf ファイルと LDAP データベースに次のような変更が導入されています。

- dbswitch.conf 内の LDAP URL 内のベース DN は、これ以降のすべての DN 仕様のためのルートオブジェクトを定義します。このため、ほとんどの新しいインストールではこれを空にすることができます。これは、最終ベース DN が別の方法、つまり、DC ツリー検索または USERDB タグ内の明示的な「basedn」値のいずれかを使う方法で決定されるためです。
- LDAP データベースの新しい dbswitch.conf 属性である dcsuffix は、DC ツリーのルートを定義します。このルートは、LDAP URL 内のベース DN に対して相対となります。データベースが *schema compliant* の場合は、dcsuffix を使用できます。スキーマに準拠するための要件は、300 ページの「iPlanet LDAP スキーマ」にリストされています。

ユーザデータベースは、仮想サーバに対して次のように選択されます。

- VS に USERDB 部分要素がない場合、ユーザベースまたはグループベースの ACL は失敗します。
- 仮想サーバの ACL 定義に database 属性がない場合、VS には default の id 属性を持つ USERDB 部分要素が必要です。その後、USERDB の database 属性が、dbswitch.conf 内のデータベースをポイントします。「database」属性がない場合は、「default」が使用されます。
- LDAP データベースがスキーマに準拠している場合、アクセスのベース DN は、CONNECTIONGROUP の servername 属性の DC ツリー検索を使って計算されます。DC ツリー検索は、dcsuffix DN に基づいています。結果には、ベース DN が含まれている inetDomainBaseDN 属性が含まれている必要があります。このベース DN は現状のまま使用され、どのベース DN 値とも相対的ではありません。

- USERDB 要素の basedn 属性がなく、データベースがスキーマに準拠していない場合、アクセスは以前のバージョンの iPlanet Web Server の場合と同じように、dbswitch.conf エントリ内のベース DN に基づいて行なわれます。

iPlanet LDAP スキーマ

LDAP データベースがこの節で概説されている要件を満たす場合は、dbswitch.conf ファイル内の dcsuffix 属性を使用できます。

ISP エントリ (たとえば、o=isp) でルートになっているサブツリーは、「コンバージェンスツリー」と呼ばれます。これには、ISP がサービスを提供する組織 (顧客) に関連するすべてのディレクトリデータが含まれます。

o=internet でルートになっているサブツリーは、「ドメインコンポーネントツリー」または「dc ツリー」と呼ばれます。これには、サービスを受ける顧客ドメイン用のエントリを持つ空白 DNS ツリーが含まれます。これらのエントリは、そのドメイン用のデータが置かれているコンバージェンスツリー内の適切な位置へのリンクです。

ディレクトリツリーは単一でルートになることができ、それが推奨されています (たとえば、o=root には、その下に o=isp と o=internet を持つことができます)。または、コンバージェンスツリー用と dc ツリー用の 2 つの別々のルートを持つことができます。

コンバージェンスツリー

コンバージェンスツリーの最上位には、顧客 (または組織) ごとに 1 つの組織エントリと、ISP 自体に 1 つの組織エントリが必要です。

各組織の下には、次の 2 つの organizationalUnit エントリが必要です。ou=People および ou=Groups です。デバイスデータをその組織用に格納する場合は、3 番目の ou=Devices の存在も可能です。

各ユーザエントリには、指定された組織内で一意な uid 値が必要です。このサブツリーの下のネームスペースは、ユーザエントリを便利なグループにまとめる (たとえば、ou=eng、ou=corp など)、さまざまな ou エントリに分割することができます。ユーザ uid 値は、People サブツリー内全体では一意なままでなければなりません。

コンバージェンスツリー内のユーザエントリのタイプは、inetOrgPerson です。cn、sn、および uid 属性がなければなりません。uid 属性は有効な電子メール名でなければなりません (特に、RFC822 に定義されている有効なローカル部でなければなりません)。cn には *name initial sn* を含めるようお勧めします。ユーザエントリの RDN を uid 値にするようお勧めします。サービスに対して使用可能になっている、または有効であるとみなされる場合、ユーザエントリには補助クラス inetUser を含める必要があります。

ユーザエントリには補助クラス inetSubscriber も含めることができます。これは、アカウント管理のために使われます。エントリに inetUserStatus 属性があり、値が inactive または deleted の場合、そのエントリは無視されます。

グループは Groups サブツリーの下にあり、タイプ groupOfUniqueNames の LDAP エントリで構成されます。

ドメインコンポーネント (dc) ツリー

dc ツリーには階層的な domain エントリが含まれており、それぞれは DNS 名前コンポーネントです。

顧客のドメイン名を表すエントリは、LDAP 補助クラス inetDomain とオーバーレイされます。たとえば、2 つの LDAP エントリ
dc=customer1,dc=com,o=Internet,o=root および
dc=customer2,dc=com,o=Internet,o=root には inetDomain クラスが含まれますが、dc=com,o=Internet,o=root には含まれません。後者は、ツリーに構造を提供するためだけに存在しています。

inetDomain 属性を持つエントリは、仮想ドメインと呼ばれます。これらには、最上位の組織エントリの DN が入った属性 inetDomainBaseDN が必要です。その最上位の組織エントリで、このドメインのデータがコンバージェンスツリーに格納されます。たとえば、dc=cust2,dc=com,o=Internet,o=root 内の仮想ドメインエントリには、値 o=cust2,o=isp,o=root を持つ属性 inetDomainBaseDN が含まれます。

エントリに inetDomainStatus 属性があり、値が inactive または deleted の場合、そのエントリは無視されます。

データ構造体のリファレンス

NSAPI は、nsapi.h ヘッダーファイルに定義されている多数のデータ構造体を使用します。このヘッダーファイルは、*server-root*/plugins/include ディレクトリ内にあります。

第 5 章「NSAPI 関数のリファレンス」に記載されている NSAPI 関数は、ほとんどのデータ構造体とデータフィールドへのアクセスを提供します。naspi.h 内のデータ構造体に直接アクセスする前に、アクセス関数が存在しているかどうかを確認してください。

iPlanet Web Server 4.*x* のいくつかのデータ構造体の非公開化については、304 ページの「いくつかのデータ構造体の非公開化」を参照してください。

この章のその他の部分では、参照用に、頻繁に使用される nsapi.h 内の公開データ構造体のいくつかを説明します。ここでは、各データ構造体のもっとも一般的に使われるフィールドだけが記載されていることに注意してください。完全な詳細は nsapi.h 内にあります。

- session
- pblock
- pb_entry
- pb_param
- Session->client
- request
- stat
- shmem_s
- cinfo

注 実際のソースファイル内のコメントはすべて英語ですが、この章では、説明のために日本語にしています。

いくつかのデータ構造体の非公開化

iPlanet Web 4.*x* では、いくつかのデータ構造体が nsapi.h から nsapi_pvt.h に移動されました。nsapi_pvt.h 内のデータ構造体は、現在は非公開 (私設) データ構造体とみなされており、それらに直接アクセスするコードを書くべきではありません。代わりに、アクセス関数を使用します。これらのデータ構造体に直接アクセスするプラグインを書いている人はほとんどいないと思われるため、この変更がカスタマ定義のプラグインに与える影響はごくわずかと思われます。公開ドメインから削除されているデータ構造体と、今後アクセスに使用できるアクセス関数を見るには、nsapi_pvt.h 内を見てください。

nsapi_pvt.h に定義されているデータ構造体の内容にアクセスする、Enterprise Server 3.*x* 用に書かれたプラグインは、iPlanet Web Server 4.*x* および 6.*x* のものとはソースレベルでの互換性はなくなります。つまり、このようなプラグインをソースから構築するためには #include "nsapi_pvt.h" の指定が必要になります。nsapi_pvt.h 内のデータ構造体のいくつかはサイズが変更されているため、これらのプログラムが iPlanet Web Server 4.*x* および 6.*x* とバイナリレベルでの互換性がなくなる可能性も若干あります。特に、directive 構造体はより大きくなっています。つまり、dtable にある指令を使ってインデックス処理を行なうプラグインは、再構築 (組み込まれている nsapi_pvt.h を使用して) しなければ動作しません。

プラグインの大半は nsapi_pvt.h 内のデータ構造体の内部を参照しないため、ほとんどの既存の NSAPI プラグインは iPlanet Web Server 6.0 とのバイナリレベルでの互換性もソースレベルでの互換性もあるものと見込んでいます。

session

session は、クライアントとサーバ間の接続を開いてから閉じるまでの時間です。Session データ構造体は、次に示すように、送信される要求にかかわらず、全体に適用される変数を保持します。

```
typedef struct {
/* リモートクライアントに関する情報 */
    pblock *client;

    /* リモートクライアントへのソケット記述子 */
    SYS_NETFD csd;

    /* ソケット記述子のための入力バッファ */
    netbuf *inbuf;

    /* リモートクライアントに関する */
    /* 生のソケット情報 （内部使用） */
    struct in_addr iaddr;
} Session;
```

pblock

パラメータブロックとは、pb_entry 構造体を保持するハッシュテーブルです。その内容は、ほとんどのコードに対して透過です。このデータ構造体は NSAPI で頻繁に使われます。これは、パラメータと値をパッケージ化するための基本的なメカニズムを提供します。パラメータブロックを作成および管理したり、エントリを抽出、追加、削除するための、多くの関数があります。第 5 章「NSAPI 関数のリファレンス」で、名前が pblock_ で始まる関数を参照してください。pblock データフィールドに直接アクセスするコードを書く必要はないはずです。

```
typedef struct {
    int hsize;
    struct pb_entry **ht;
} pblock;
```

pb_entry

pb_entry は、パラメータブロック内の単一の要素です。

```
struct pb_entry {
    pb_param *param;
    struct pb_entry *next;
};
```

pb_param

pb_param は、pb_entry に格納されているような、「名前－値」のペアを表します。

```
typedef struct {
    char *name,*value;
} pb_param;
```

Session->client

Session->client パラメータブロック構造体には、次の 2 つのエントリがあります。

- ip エントリは、クライアントマシンの IP アドレスです。
- dns エントリは、リモートマシンの DNS 名です。このメンバーは、session_dns 関数呼び出しからアクセスする必要があります。

```
/*
* session_dns は、このセッションにクライアントの DNS ホスト名を返し、
* それをクライアントの pblock ブロックに挿入する。使用できない場合は、
* NULL を返す
*/
char *session_dns(Session *sn);
```

request

HTTP プロトコルのもとでは、セッションに対応する要求は 1 つだけです。Request 構造体は、該当のセッションにある要求に適用される変数があります (たとえば、変数にはクライアントの HTTP ヘッダーが含まれます)。

```
typedef struct {
    /* サーバの動作している変数 */
    pblock *vars;

    /* このリクエストのメソッド、URI、プロトコルバージョン */
    block *reqpb;

    /* プロトコル固有のヘッダー */
    int loadhdrs;
    pblock *headers;

    /* サーバの応答ヘッダー */
    pblock *srvhdrs;

    /* この要求を満たすために構築されたオブジェクトセット */
    httpd_objset *os;

    /* request_stat_path が最後に返した stat */
    char *statpath;
    struct stat *finfo;
} Request;
```

stat

プログラムが指定されたファイルのために stat( ) 関数を呼び出すとき、システムは、そのファイルに関する情報を提供する構造体を返します。この構造体に固有の詳細情報はプラットフォームの実装から入手できるはずですが、構造体の基本的な概要は次のとおりです。

```
struct stat {
    dev_t       st_dev;      /* i ノードのデバイス */
    inot_t      st_ino;      /* i ノード番号 */
    short       st_mode;     /* モードビット */
    short       st_nlink;    /* ファイルへのリンク数 /*
    short       st_uid;      /* 所有者のユーザ ID */
    short       st_gid;      /* 所有者のグループ ID */
    dev_t       st_rdev;     /* 特定のファイル用 */
    off_t       st_size;     /* ファイルサイズ（文字数）*/
    time_t      st_atime;    /* 最後にアクセスした時間 */
    time_t      st_mtime;    /* 最後に更新した時間 */
    time_t      st_ctime;    /* i ノードが最後に変更された時間 */
}
```

サーバのプラグイン API アクティビティのためにもっとも重要な要素は、st_size、st_atime、st_mtime、および st_ctime です。

shmem_s

```
typedef struct {
    void        *data;    /* データ */
    HANDLE      fdmap;
    int         size;     /* 最大データ長 */
    char        *name;    /* 内部使用： 公開される場合、リンクを解除する
ファイル名 */
    SYS_FILE    fd;       /* 内部使用： 領域のためのファイル記述子 */
} shmem_s;
```

cinfo

cinfo データ構造体は、ファイルの内容情報を記録します。

```
typedef struct {
    char    *type;
            /* ファイル内のデータの種類を識別する */
    char    *encoding;
            /* encoding は、uuencode など、ファイルに適用されている /*
            /* 圧縮方式またはその他の内容に依存しない変換方式を識別する /*
    char    *language;
            /* テキストドキュメントの言語を識別する */
} cinfo;
```

cinfo

付録 B

MIME タイプ

この付録では、MIME タイプファイルについて説明します。次の節があります。

- はじめに
- MIME タイプを判別する
- タイプが応答へどのように影響するか
- クライアントが MIME タイプを使って行なう作業
- MIME タイプファイルの構文
- MIME タイプファイルの例

はじめに

config ディレクトリ内の MIME タイプファイルには、MIME (Multipurpose Internet Mail Extensions) タイプとファイル拡張子が含まれています。たとえば、MIME タイプファイルは、拡張子 .html と .htm をタイプ text/html にマッピングします。

```
type=text/html exts=htm,html
```

iPlanet Web Server は、リソースに対する要求をクライアントから受け取ったとき、MIME タイプのマッピングを使用して、要求されているリソースの種類を判別します。

MIME タイプは次の 3 つの属性によって定義されます。言語 (lang)、エンコーディング (enc)、およびコンテンツのタイプ (type) です。各タイプには、これらの属性のうち少なくとも 1 つがなければなりません。もっともよく使われる属性は、type です。サーバは、クライアントへの応答の生成方法を決定するときに、しばしば type を検討します (enc および lang 属性が使われることはまれです)。

デフォルトの MIME タイプファイルは、mime.types と呼ばれます。

MIME タイプを判別する

要求処理プロセスの ObjectType ステップの間に、サーバは、クライアントが要求したリソースの MIME タイプ属性を判別します。MIME タイプの判別に使用できるサーバアプリケーション関数 (SAF) はいくつかありますが、もっともよく使われるものは type-by-extension です。この関数は、MIME タイプテーブルから、要求されたリソースのファイル拡張子に従って MIME タイプを探すよう、サーバに指示します。

obj.conf 内の指令のうち、拡張子に従って MIME タイプを調べるようサーバに指示する指令は、次のとおりです。

```
ObjectType fn=type-by-extension
```

サーバが type の判別のために force-type などの別の SAF を使用する場合は、MIME タイプテーブルはその特定の要求のためには使用されません。

ObjectType ステップの詳細は、第 2 章「obj.conf の構文と使用法」を参照してください。

タイプが応答へどのように影響するか

サーバは、クライアントへの応答の生成に使用する obj.conf 内の Service 指令を決定するとき、type 属性の値を考慮します。

デフォルトでは、type が magnus-internal/ で始まらない場合、サーバは単に要求されたファイルをクライアントに送信します。この命令が含まれている obj.conf 内の指令は、次のとおりです。

```
Service method=(GET|HEAD|POST) type=*~magnus-internal/* fn=send-file
```

ここで使われている特殊文字 *~ は「一致しない」ことを意味しています。特殊文字の詳細は、付録 C「ワイルドカードパターン」を参照してください。

要求されたリソースを単にクライアントに送信するだけでなくサーバによる何らかの処理を要求する場合の type の値は、規則によって、すべて magnus-internal/ で始まります。

次の例では、要求されたリソースのファイル拡張子が .map の場合、このタイプは magnus-internal/imagemap にマッピングされます。拡張子が .cgi、.exe、または .bat の場合、タイプは magnus-internal/cgi に設定されます。

```
type=magnus-internal/imagemap       exts=map
type=magnus-internal/cgi            exts=cgi,exe,bat
```

type が magnus-internal/ で始まる場合、サーバは、obj.conf 内にある指定したタイプに一致した Service 指令を実行します。たとえば、タイプが magnus-internal/imagemap の場合、サーバは、次の指令に示されるように、imagemap 関数を使用してクライアントへの応答を生成します。

```
Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
```

タイプが magnus-internal/servlet の場合、サーバは、次の指令に示されるように、NSServletService 関数を使用してクライアントへの応答を生成します。

```
Service type="magnus-internal/servlet" fn="NSServletService"
```

クライアントが MIME タイプを使って行なう作業

Service 関数は、データを生成し、要求を行なったクライアントに生成したデータを送信します。サーバは、クライアントにデータを送信するとき、ヘッダーも一緒に送信します。このときのヘッダーには、わかっている MIME タイプ属性 (通常は type) が含まれています。

クライアントはデータを受信したときに、MIME タイプを使ってそのデータの処理方法を決定します。ブラウザクライアントの場合、通常は、データをブラウザウィンドウに表示します。

要求されたリソースをブラウザに表示できず、別のアプリケーションで処理する必要がある場合、その type は application/ で始まります。たとえば、application/octet-stream (ファイル拡張子が .bin の場合) または application/x-maker (ファイル拡張子が .fm の場合) のようになります。クライアントには、ユーザによる編集が可能な、クライアント独自のマッピングのセットがあります。そこでは、どのタイプのデータに対してどのアプリケーションで処理をするのかが、マッピングのセットによって指示されています。

たとえば、タイプが application/x-maker の場合、一般にクライアントは、Adobe FrameMaker を使用してファイルを表示します。

MIME タイプファイルの構文

MIME タイプファイルの最初の行はファイル書式を示し、次のようになっている必要があります。

```
#--Netscape Communications Corporation MIME Information
```

コメント行以外の他の行は、次の書式になります。

```
type=type/subtype exts=[file extensions]
```

- `type/subtype` はタイプとサブタイプです
- `exts` は、このタイプと関連付けられているファイル拡張子です。

MIME タイプファイルの例

MIME タイプファイルの一例を次に示します。

```
#--Netscape Communications Corporation MIME Information
# 上記の行を削除しないでください。上記の行は、ファイルタイプの識別に使用され
# ています
type=application/octet-stream      exts=bin,exe
type=application/oda               exts=oda
type=application/pdf               exts=pdf
type=application/postscript        exts=ai,eps,ps
type=application/rtf               exts=rtf
type=application/x-mif             exts=mif,fm
type=application/x-gtar            exts=gtar
type=application/x-shar            exts=shar
type=application/x-tar             exts=tar
type=application/mac-binhex40      exts=hqx

type=audio/basic                   exts=au,snd
type=audio/x-aiff                  exts=aif,aiff,aifc
type=audio/x-wav                   exts=wav

type=image/gif                     exts=gif
type=image/ief                     exts=ief
type=image/jpeg                    exts=jpeg,jpg,jpe
type=image/tiff                    exts=tiff,tif
type=image/x-rgb                   exts=rgb
type=image/x-xbitmap               exts=xbm
type=image/x-xpixmap               exts=xpm
type=image/x-xwindowdump           exts=xwd

type=text/html                     exts=htm,html
type=text/plain                    exts=txt
type=text/richtext                 exts=rtx
type=text/tab-separated-values     exts=tsv
type=text/x-setext                 exts=etx

type=video/mpeg                    exts=mpeg,mpg,mpe
type=video/quicktime               exts=qt,mov
type=video/x-msvideo               exts=avi

enc=x-gzip                         exts=gz
enc=x-compress                     exts=z
enc=x-uuencode                     exts=uu,uue

type=magnus-internal/imagemap      exts=map
type=magnus-internal/parsed-html   exts=shtml
type=magnus-internal/cgi           exts=cgi,exe,bat
type=magnus-internal/jsp           exts=jsp
```

付録 C

ワイルドカードパターン

この付録では、iPlanet Web Server が使用するワイルドカードパターンの書式について説明します。

ワイルドカードは次の箇所で使われます。

- 構成ファイル `obj.conf` 内の指令 (第 2 章「obj.conf の構文と使用法」を参照)
- さまざまな組み込み型 SAF (第 3 章「事前定義済みの SAF および要求処理プロセス」を参照)
- いくつかの NSAPI 関数 (第 5 章「NSAPI 関数のリファレンス」を参照)

ワイルドカードパターンには、特殊文字を使用します。これらの特殊文字を特殊な意味を持たせずに使用したい場合は、1 つのバックスラッシュ (\) を該当する特殊文字の前に付けます。

ワイルドカードパターン

表 C-1　ワイルドカードパターン

パターン	使用法
*	0 個以上の文字に相当します。
?	任意の文字 1 個に相当します。
|	OR 式を構成します。この演算子とともに使用する部分文字列では、* または $ のような他の特殊文字を含むことができます。部分文字列は (a|b|c) などのようにカッコで囲む必要があります。しかし、( ) を入れ子にすることはできません。
$	文字列の末尾に相当します。これを OR 式で使用すると便利です。
[abc]	文字 a、b、または c の 1 回の出現に相当します。この式の中で特殊文字として扱う必要のある文字は] だけで、それ以外のものは特殊文字ではありません。

表 C-1　ワイルドカードパターン (続き)

パターン	使用法
[a-z]	a から z までの文字の 1 回の出現に相当します。
[^az]	a でも z でもない、任意の 1 個の文字に相当します。
*~	この式は、後ろに別の式が続き、2 番目の式と一致するパターンをすべて削除します。

ワイルドカードの例

表 C-2　ワイルドカードの例

パターン	結果
*.netscape.com	.netscape.com という綴りで終わる任意の文字列に相当します。
(quark|energy).netscape.com	quark.netscape.com または energy.netscape.com のいずれかに相当します。
198.93.9[23].???	198.93.92 または 198.93.93 のいずれかで始まり、任意の 3 文字で終わる数値文字列に相当します。
.	ピリオドが含まれる任意の文字列に相当します。
~netscape-	文字列の最初の部分が netscape- という綴りではない、任意の文字列に相当します。
*.netscape.com~quark.netscape.com	単一のホスト quark.netscape.com を除く、ドメイン netscape.com からの任意のホストに相当します。
*.netscape.com~(quark|energy|neutrino).netscape.com	ホスト quark.netscape.com、energy.netscape.com、および neutrino.netscape.com を除く、ドメイン .netscape.com の任意のホストに相当します。
.com~.netscape.com	サブドメイン netscape.com のホストを除く、ドメイン .com の任意のホストに相当します。
type=*~magnus-internal/*	magnus-internal/ という綴りでは始まらない、任意のタイプに相当します。 このワイルドカードパターンは、catch-all Service 指令内のファイル obj.conf で使われます。

付録 D

時刻の書式

この付録では、日付と時刻を表す書式文字列について説明します。これらの書式は、NSAPI 関数 `util_strftime`、`append-trailer` などのいくつかの組み込み型 SAF や、サーバが構文解析する HTML (`parse-html`) で使われます。

この書式は `strftime` C ライブラリルーチンが使用するものと似ていますが、必ずしもまったく同じというわけではありません。

表 D-1

記号	意味
%a	曜日名の省略形 (3 文字)
%d	10 進数による、月内の日付 (01 ～ 31)
%S	10 進数による、秒数 (00 ～ 59)
%M	10 進数による、分数 (00 ～ 59)
%H	24 時間の書式による時刻 (00 ～ 23)
%Y	10 進数による、年 (西暦) (最大 2099)
%b	月名の省略形 (3 文字)
%h	月名の省略形 (3 文字)
%T	時刻 HH:MM:SS
%X	時刻 HH:MM:SS
%A	曜日名 (省略なし)
%B	月名 (省略なし)
%C	%a %b %e %H:%M:%S %Y
%c	日付と時刻 %m/%d/%y %H:%M:%S
%D	日付 %m/%d/%y

表 D-1

記号	意味
%e	前に 0 が付かない、10 進数による、月内の日付 (1 ～ 31)
%I	12 時間の書式による時刻 (01 ～ 12)
%j	10 進数による、年内の通算日付 (001 ～ 366)
%k	前に 0 が付かない、24 時間の書式による時刻 (0 ～ 23)
%l	前に 0 が付かない、12 時間の書式による時刻 (1 ～ 12)
%m	10 進数による月 (01 ～ 12)
%n	改行
%p	12 時間時計の午前 (A.M.)/ 午後 (P.M.) の表示
%R	時刻 %H:%M
%r	時刻 %I:%M:%S %p
%t	タブ
%U	週の始まりが日曜日の、10 進数による、年内の週 (00 ～ 51)
%w	10 進数による曜日 (0 ～ 6、日曜は 0)
%W	週の始まりが月曜日の、10 進数による、年内の週 (00 ～ 51)
%x	日付 %m/%d/%y
%y	10 進数による、西暦の末尾 2 桁 (00 ～ 99)
%%	パーセント記号

HTTP (HyperText Transfer Protocol)

ハイパーテキスト転送プロトコル (HyperText Transfer Protocol: HTTP) は、Web ブラウザなどのクライアントと Web サーバが、互いに通信するためのプロトコル (情報の交換方法を記述した規則のセット) です。

HTTP は、要求 / 応答モデルに基づいています。ブラウザはサーバへの接続を開いて、サーバに要求を送信します。

サーバは、要求を処理して、ブラウザに送信する応答を生成します。その後、サーバは接続を閉じます。

この付録では、HTTP の基礎のいくつかについて簡単に説明します。HTTP についての詳細は、次の場所にある IETF のホームページを参照してください。

`http://www.ietf.org/home.html`

この付録では次の節について説明します。

- 準拠
- 要求
- 応答
- バッファ化されたストリーム

準拠

iPlanet Web Server 6.0 は HTTP 1.1 をサポートします。以前のバージョンのサーバも HTTP 1.0 をサポートしていました。このサーバは、IESG (Internet Engineering Steering Group) および IETF (Internet Engineering Task Force) の HTTP ワーキンググループ承認の HTTP 1.1 規格案に、条件付きで準拠しています。

条件付き準拠の基準については、次の場所にある「Hypertext Transfer Protocol--HTTP/1.1 specification (RFC 2068)」を参照してください。

http://www.ietf.org/rfc/rfc2068.txt?number=2068

要求

ブラウザからサーバへの要求には、次のような情報が含まれています。

- 要求メソッド、URI、およびプロトコルのバージョン
- 要求ヘッダー
- 要求データ

要求メソッド、URI、およびプロトコルのバージョン

ブラウザは、多数のメソッドを使って情報を要求することができます。一般的に使われるメソッドには、次のものがあります。

- GET— 指定したリソース (ドキュメントやイメージなど) を要求します
- HEAD— ドキュメントのヘッダー情報のみを要求します
- POST— CGI プログラムが処理するためのフォームからの入力など、サーバが受け入れるデータをブラウザに要求します
- PUT— サーバのドキュメントの内容を、ブラウザからのデータで置き換えます

要求ヘッダー

ブラウザはサーバにヘッダーを送信できます。ただし、ほとんどのものは任意 (省略可能) です。一般的に使われる要求ヘッダーのいくつかを、表 E-1 に示します。

表 E-1 一般的な要求ヘッダー

要求ヘッダー	説明
Accept	ブラウザが受け入れ可能なファイルタイプ
Authorization	ブラウザがサーバで自身を認証したい場合に使われます。ユーザ名やパスワードなどの情報が含まれます
User-agent	ブラウザソフトウェアの名前とバージョン
Referer	ユーザがリンクをクリックしたドキュメントの URL

表 E-1　一般的な要求ヘッダー (続き)

要求ヘッダー	説明
Host	要求されているリソースのインターネットホストとポート番号

要求データ

ブラウザは、POST または PUT 要求を実行した場合、要求ヘッダーに続く空白行の後にデータを送信します。ブラウザが GET または HEAD 要求を送信した場合は、送信するデータはありません。

応答

サーバの応答には次のものが含まれます。

- HTTP プロトコルのバージョン、状態コード、および原因を示す文字列
- 応答ヘッダー
- 応答データ

HTTP プロトコルのバージョン、状態コード、および原因を示す文字列

サーバは状態コードを返信します。状態コードは 3 桁の数値コードです。状態コードには、次の 5 つのカテゴリがあります。

- 100 ～ 199 は、暫定的な応答です。
- 200 ～ 299 は、成功したトランザクションです。
- 300 ～ 399 は、要求されたリソースは別の場所で検索する必要があることを示します。
- 400 ～ 499 は、ブラウザの原因によるエラーが発生したことを示します。
- 500 ～ 599 は、サーバで重大なエラーが発生したことを示します。

いくつかの一般的な状態コードを表 E-2 に示します。

表 E-2　一般的な HTTP 状態コード

状態コード	意味
200	OK。使用したメソッド (GET、POST、HEAD) に対する要求が成功しました。
201	要求によって、返された URI による新しいリソース参照が作成されました。
206	サーバがバイト範囲要求に応答を送信しました。
302	見つかりました。新しい URL へ、リダイレクトします。元の URL は移動しました。これはエラーではありません。ほとんどのブラウザでは、新しいページを取得できます。
304	ローカルコピーを使用します。ブラウザのキャッシュにすでにページがあり、そのページが再度要求されている場合、ブラウザ (Netscape Navigator など) の種類によっては、ブラウザのキャッシュされたコピーにある「最終更新の (last-modified)」タイムスタンプを Web サーバに中継することがあります。このサーバ上のコピーがブラウザのコピーよりも古い場合、サーバは、不要なネットワークトラフィックを減らすために、そのページを返すのではなく 304 コードを返します。これはエラーではありません。
400	要求が、有効な HTTP/1.0 または HTTP/1.1 要求でない場合に送信されます。たとえば、HTTP/1.1 では、ホストを「Host」ヘッダー内または要求行の URI の一部として指定する必要があります。
401	承認されていません。ユーザがドキュメントを要求しましたが、有効なユーザ名およびパスワードが指定されていませんでした。
403	禁止。この URL へのアクセスは禁止されています。
404	見つかりませんでした。要求されたドキュメントがサーバ上にありません。このコードは、承認されていないユーザにドキュメントが存在しないと伝えることによって、サーバがドキュメントを保護するよう指示されている場合にも、送信されます。
408	クライアントが要求を開始したが、サーバに設定されているキープアライブタイムアウト内でそれが完了しない場合に、この要求が送信されて接続が閉じられます。この要求は、開いている別の接続で再度行なうことができます。
411	クライアントが、可変長のチャンクされたエンコーディングで POST 要求を送信しました。しかし、サーバ上のリソースやアプリケーションは、固定長の「content-length (内容長)」ヘッダーの存在を要求しています。このコードは、内容長とともにその要求を再度送信するよう、クライアントに指示します。

表 E-2　一般的な HTTP 状態コード (続き)

状態コード	意味
413	アプリケーション (特定の NSAPI プラグインなど) によっては、大量のデータを処理できないものもあります。その場合、このコードが返されます。
414	URI の長さが、Web サーバがサービスできる最大長より長くなっています。
416	ファイル容量の範囲を越えるデータが要求されました。
500	サーバエラー。サーバに関連するエラーが発生しました。サーバ管理者がサーバのエラーログを確認して、何が起こったかを確認する必要があります。
503	「サービス品質」のメカニズムが有効になっており、帯域幅または接続の限界に達した場合に、送信されます。その後、サーバは該当のコードを使って要求に対処します。「サービス品質」の節を参照してください。

応答ヘッダー

応答ヘッダーには、サーバと応答データに関する情報が指定されています。一般的な応答ヘッダーを、表 E-3 に示します。

表 E-3　一般的な応答ヘッダー

応答ヘッダー	説明
Server	Web サーバの名前とバージョン。
Date	現在の日付 (グリニッジ標準時)。
Last-modified	ドキュメントが最後に更新された日付。
Expires	ドキュメントの有効期限切れの日付。
Content-length	次に続くデータの長さ (バイト)。
Content-type	次に続くデータの MIME タイプ。
WWW-authenticate	認証中に使われ、ブラウザソフトウェアに認証のために何が必要か (ユーザ名およびパスワードなど) を伝える情報が含まれています。

応答データ

サーバは最後のヘッダーの後に空白行を送信します。その後、イメージや HTML ページなどの応答データを送信します。

バッファ化されたストリーム

バッファ化されたストリームは、特に動的なコンテンツを生成する場合に、ネットワーク入出力 (たとえば、HTTP 要求と応答の交換など) の効率を高めます。バッファ化されたストリームは、透過 NSPR I/O 層として実装されます。これは、既存の NSAPI モジュールでも変更なしでこれらを使用できることを意味します。

バッファ化されたストリーム層は、iPlanet Web Server に次のような機能を追加します。

- キープアライブのサポートの拡張 : 応答がバッファサイズより小さいとき、バッファリング層が `content-length` ヘッダーを生成し、クライアントが応答の終わりを検出して、その接続を以降の要求のために再使用できるようにします。
- 応答の長さの決定 : バッファリング層は、応答の長さを決定できない場合、`content-length` ヘッダーではなく HTTP 1.1 チャンクされたエンコーディングを使って、概要情報を転送します。クライアントが HTTP 1.0 しか理解しない場合、サーバは、応答の終わりを示すために接続を閉じる必要があります。
- ヘッダーの書き込みを遅らせる : サーブレットがそれ自体のヘッダー (たとえば、セッション管理ヘッダー `set-cookie` など) を生成する機会を与えるために、応答ヘッダーの書き込みをできる限り遅らせます。
- チャンクされたエンコーディングでの要求エンティティ本体を理解する機能 : 一般のクライアントは `POST` 要求データの送信にチャンクされたエンコーディングを使用しませんが、HTTP 1.1 に準拠するには、この機能が必須となります。

バッファ化されたストリームが提供する改良された接続処理と応答長ヘッダーの生成は、応答長ヘッダーの不在がカテゴリ 1 障害とみなされる、HTTP 1.1 プロトコル準拠の問題にも対応します。Enterprise Server の以前のバージョンでは、長さヘッダーの送信は動的コンテンツ生成プログラムがその責任を担っていました。CGI スクリプトが `content-length` ヘッダーを生成しない場合は、応答の終わりを示すために、サーバがキープアライブメカニズムを中断して接続を閉じなければなりませんでした。しかし、CGI スクリプトやサーブレットの応答長を追跡し続けるのは非常に不便なこともあり、アプリケーションプラットフォームのプロバイダとしては、このような低いレベルのプロトコルの問題は Web サーバが処理することを期待しています。

出力バッファリングは、net_write (第 5 章「NSAPI 関数のリファレンス」を参照) のようなデータを送信する関数に組み込まれました。次のような、ストリームバッファリングに影響する Service SAF パラメータを指定することができます。詳細は 第 3 章「事前定義済みの SAF および要求処理プロセス」に記載されています。

- UseOutputStreamSize
- flushTimer
- ChunkedRequestBufferSize
- ChunkedRequestTimeout

UseOutputStreamSize、ChunkedRequestBufferSize、および ChunkedRequestTimeout パラメータには、対応する magnus.conf 指令も存在します。278 ページの「チャンクされたエンコーディング」を参照してください。
obj.conf パラメータは、magnus.conf 指令を上書きします。

注 UseOutputStreamSize パラメータを obj.conf ファイル内で 0 に設定して、出力ストリームバッファリングを使用不可にすることができます。magnus.conf ファイルの場合は、UseOutputStreamSize を 0 に設定しても効果はありません。

関数 net_read または netbuf_grab のいずれかを使用する SAF を呼び出すときにデフォルトの動作を上書きするために、obj.conf 内のパラメータの値を指定できます。たとえば、次のようにします。

```
Service fn="my-service-saf" type=perf UseOutputStreamSize=8192
```

動的に結果をキャッシュする関数

この付録で説明する関数を使って、iPlanet Web Server 用の結果をキャッシュするプラグインを作成できます。`Service` SAF である結果をキャッシュするプラグインは、Web サーバのアドレス空間にあるデータ、ページ、またはページの一部をキャッシュします。これらのものは、要求に応じて定期的に Web サーバによってリフレッシュすることができます。`Init` SAF は、リフレッシュを行なうコールバック関数を初期化します。

結果をキャッシュするプラグインは、次の 3 つの部分からなる要求に対するページを生成することができます。

- ページバナーなどのヘッダー。要求ごとに変わります。
- 本体。あまり頻繁には変わりません。
- フッター。これも、要求ごとに変わります。

この機能がなければ、プラグインは要求ごとにページ全体を生成しなければならなくなります (IFRAME を使用しない限り。IFRAME を使用する場合は、ヘッダーまたはフッターがその本体をポイントする IFRAME とともに最初の応答で送信されます。この場合ブラウザは、IFRAME に対する別の要求を送信しなければなりません)。

ページの本体が変更されていなければ、プラグインはヘッダーとフッターのみを生成して、次のような引数を持つ `dr_net_write` 関数 (`net_write` ではなく) を呼び出す必要があります。

- ヘッダー
- フッター
- キャッシュへのハンドル
- キャッシュされたオブジェクトを識別する鍵

Web サーバは、キャッシュから本体を取り出すことで、ページ全体を構築します。キャッシュの有効期限が切れている場合は、リフレッシュ関数を呼び出して、リフレッシュされたページをクライアントに送信します。

プラグインから見える Init SAF は、キャッシュへのハンドルを作成します。Init SAF は、dr_cache_init 関数に次のようなパラメータを渡す必要があります。

- RefreshFunctionPointer
- FreeFunctionPointer
- KeyComparatorFunctionPtr
- RefershInterval

RefershInterval 値は PrIntervalTime タイプでなければなりません。詳細は、次の場所にある NSPR リファレンスを参照してください。

http://www.mozilla.org/projects/nspr/reference/html/index.html

別のケースとして、本体が Web サーバシステムのマシン内のディレクトリにあるファイルの場合、プラグインは、ヘッダーとフッターを生成して、ファイル名とともに fc_net_write 関数を呼び出すことができます。

この付録では、結果をキャッシュするプラグインが使用できるもっとも重要な関数をリストしています。詳細については次のファイルを参照してください。

server_root/plugins/include/drnsapi.h

dr_cache_destroy

dr_cache_destroy 関数は、以前に作成され使用されたキャッシュハンドルに関連するリソースを削除して、解放します。別の dr_cache_init を実行しない限り、このハンドルは、前述の関数へのこの後の呼び出しには使用できなくなります。

構文

```
void dr_cache_destroy(DrHdl *hdl);
```

パラメータ

DrHdl *hdl は、キャッシュへの以前に初期化したハンドルへのポインタです (dr_cache_init を参照)。

戻り値

```
void
```

例

```
dr_cache_destroy(&myHdl);
```

dr_cache_init

dr_cache_init 関数は、キャッシュへの持続性ハンドル、つまり障害時の NULL を作成します。これは Init SAF によって呼び出されます。

構文

```
PRInt32 dr_cache_init(DrHdl *hdl, RefreshFunc_t ref, FreeFunc_t fre,
CompareFunc_t cmp, PRUint32 maxEntries, PRIntervalTime maxAge);
```

戻り値

1 成功した場合。

0 エラーが発生した場合。

パラメータ

DrHdl hdl は、割り当てられていないハンドルへのポインタです。

RefreshFunc_t ref は、キャッシュリフレッシュ関数へのポインタです。これは NULL にすることができます。dr_net_write の DR_CHECK フラグと DR_EXPIR 戻り値を参照してください。

FreeFunc_t fre は、エントリを解放する関数へのポインタです。

CompareFunc_t cmp は、鍵比較関数へのポインタです。

PRUint32 maxEntriesp は、指定した hdl のキャッシュに入れることができるエントリの最大数です。

PRIntervalTime maxAgep は、エントリが有効な最大時間です。0 の場合、キャッシュが期限切れになることはありません。

例

```
if(!dr_cache_init(&hdl, (RefreshFunc_t)FnRefresh,
(FreeFunc_t)FnFree, (CompareFunc_t)FnCompare, 150000,
PR_SecondsToInterval(7200)))
{
    ereport(LOG_FAILURE, "dr_cache_init() failed");
    return(REQ_ABORTED);
}
```

dr_cache_refresh

dr_cache_refresh 関数は、プラグインが要求したときにキャッシュエントリをリフレッシュする方法を提供します。これは、dr_cache_init 内の ref パラメータには NULL を渡し、dr_net_write 呼び出し内に DR_CHECK を渡すことで、実行することができます。DR_CHECK が dr_net_write に渡され、DR_EXPIR が返される場合、プラグインは、応答を送信するために、再度 dr_net_write を呼び出す前に、エントリに新しい内容を生成し、そのエントリで dr_cache_refresh を呼び出す必要があります。

プラグインは、まだ期限切れになっていなくても、キャッシュされたエントリを単に置き換えることを決定することもあります (他のビジネスロジックに従って)。この場合、dr_cache_refresh 関数が便利です。このようにプラグインは、自身でキャッシュリフレッシュ管理をアクティブに行ないます。

構文

```
PRInt32 dr_cache_refresh(DrHdl hdl, const char *key, PRUint32 klen,
PRIntervalTime timeout, Entry *entry, Request *rq, Session *sn);
```

戻り値

1 成功した場合。

0 エラーが発生した場合。

パラメータ

DrHdl hdl は、dr_cache_init 関数によって作成される持続性ハンドルです。

const char *key は、キャッシュ、検索、またはリフレッシュへの鍵です。

PRUint32 klen は、鍵の長さ (バイト) です。

PRIntervalTime timeout は、このエントリの期限切れの時間です。値 0 が渡されると、dr_cache_init に渡された maxAge 値が使用されます。

Entry *entry は、キャッシュされる非 NULL エントリです。

Request *rq は、要求へのポインタです。

Session *sn は、セッションへのポインタです。

例

```
Entry entry;
char *key = "MOVIES"
GenNewMovieList(&entry.data, &entry.dataLen);  // Implemented by
                                               // plugin developer
if(!dr_cache_refresh(hdl, key, strlen(key), 0, &entry, rq, sn))
{
    ereport(LOG_FAILURE, "dr_cache_refresh() failed");
    return REQ_ABORTED;
}
```

dr_net_write

dr_net_write 関数は、hdr、本体としてキャッシュされたエントリの内容 (key を使って配置されたもの)、および ftr から成るページ全体を構築したあと、要求者に応答を返信します。hdr、ftr、または hdl は NULL にすることができますが、これらのすべてを NULL にすることはできません。hdl が NULL の場合、キャッシュ検索は行なわれません。呼び出し元はフラグとして DR_NONE を渡さなければなりません。

デフォルトでは、キャッシュエントリが dr_cache_init に渡された ref 関数への呼び出しによって期限切れになった場合に、この関数がキャッシュエントリをリフレッシュします。指定した key でキャッシュエントリが見つからない場合は、応答を送信する前にこの関数が ref 関数を呼び出して、新しいキャッシュエントリを追加します。ただし、DR_CHECK フラグが flags パラメータで渡され、キャッシュエントリが期限切れになったか、key に相当するキャッシュエントリが存在しない場合は、dr_net_write はデータを送信しません。代わりに、DR_EXPIR とともに返されます。

ref (dr_cache_init に渡される) が NULL の場合、DR_CHECK フラグは flags パラメータには渡されず、key に相当するキャッシュエントリは期限切れになっているか存在せず、dr_net_write は DR_ERROR で失敗します。しかし、ref が NULL ではなく、DR_CHECK が渡されない場合は、dr_net_write はキャッシュをリフレッシュします。

ref (dr_cache_init に渡される) が NULL で、DR_CHECK フラグが渡されなくても DR_IGNORE が渡され、キャッシュにエントリがある場合、エントリが期限切れになっていても dr_net_write は応答を送信します。ただし、エントリが見つからない場合は、dr_net_write は DR_ERROR を返します。

ref (dr_cache_init に渡される) が NULL ではなく、DR_CHECK フラグが渡されなくても DR_IGNORE が渡され、キャッシュにエントリがある場合、エントリが期限切れになっていても dr_net_write は応答を送信します。ただし、エントリが見つからない場合、dr_net_write は、応答を送信する前に ref 関数を呼び出して、ref から返された新しいエントリを格納します。

構文

```
PRInt32 dr_net_write(DrHdl hdl, const char *key, PRUint32 klen,
const char *hdr, const char *ftr, PRUint32 hlen, PRUint32 flen,
PRIntervalTime timeout, PRUint32 flags, Request *rq, Session *sn);
```

戻り値

IO_OKAY 成功した場合。

IO_ERROR エラーが発生した場合。

DR_ERROR キャッシュ処理でエラーが発生した場合。

DR_EXPIR キャッシュの期限が切れている場合。

パラメータ

DrHdl hdl は、dr_cache_init 関数によって作成される持続性ハンドルです。

const char *key は、キャッシュ、検索、またはリフレッシュへの鍵です。

PRUint32 klen は、鍵の長さ (バイト) です。

const char *hdr は任意のヘッダーデータです (NULL にすることもできます)。

const char *ftr は任意のフッターデータです (NULL にすることもできます)。

PRUint32 hlen は、ヘッダーデータの長さ (バイト) です (0 にすることもできます)。

PRUint32 flen は、フッターデータの長さ (バイト) です (0 にすることもできます)。

PRIntervalTime timeout は、この関数を強制終了するまでのタイムアウトです。

PRUint32 flags は、この関数の ORed 指令です (「フラグ」を参照)。

Request *rq は、要求へのポインタです。

Session *sn は、セッションへのポインタです。

フラグ

DR_NONE は、キャッシュを使用しないことを指定します。このため、関数は net_write と同じように機能します。DrHdl は NULL にすることができます。

DR_FORCE は、キャッシュが期限切れになっていない場合でも、強制的にキャッシュをリフレッシュさせます。

DR_CHECK は、キャッシュの期限が切れている場合に DR_EXPIR を返します。呼び出し元の関数がリフレッシュ関数を提供しておらず、このフラグが使われない場合は、DR_ERROR が返されます。

DR_IGNORE は、キャッシュの期限が切れている場合でもキャッシュの期限切れを無視して、キャッシュエントリを送信します。

DR_CNTLEN は、「Content-length」ヘッダーを提供し、PROTOCOL_START_RESPONSE を実行します。

DR_PROTO は PROTOCOL_START_RESPONSE を実行します。

例

```
if(dr_net_write(Dr, szFileName, iLenK, NULL, NULL, 0, 0, 0,
DR_CNTLEN | DR_PROTO, rq, sn) == IO_ERROR)
{
    return(REQ_ABORTED);
}
```

fc_net_write

fc_net_write 関数は、ヘッダー、フッターのどちらかまたはその両方と、システム内のどこかに存在するファイルを送信するために使われます。fileName は、ファイルへの絶対パスでなければなりません。

構文

```
PRInt32 fc_net_write(const char *fileName, const char *hdr, const
char *ftr, PRUint32 hlen, PRUint32 flen, PRUint32 flags,
PRIntervalTime timeout, Session *sn, Request *rq);
```

戻り値

IO_OKAY 成功した場合。

IO_ERROR エラーが発生した場合。

FC_ERROR ファイル処理でエラーが発生した場合。

パラメータ

const char *fileName は挿入されるファイルです。

const char *hdr は任意のヘッダーデータです (NULL にすることもできます)。

const char *ftr は任意のフッターデータです (NULL にすることもできます)。

PRUint32 hlen は、ヘッダーデータの長さ (バイト) です (0 にすることもできます)。

PRUint32 flen は、フッターデータの長さ (バイト) です (0 にすることもできます)。

PRUint32 flags は、この関数の ORed 指令です (「フラグ」を参照)。

PRIntervalTime timeout は、この関数を強制終了するまでのタイムアウトです。

Request *rq は、要求へのポインタです。

Session *sn は、セッションへのポインタです。

フラグ

FC_CNTLEN は、「Content-length」ヘッダーを提供し、PROTOCOL_START_RESPONSE を実行します。

FC_PROTO は PROTOCOL_START_RESPONSE を実行します。

例

```
const char *fileName = "/docs/myads/file1.ad";
char *hdr = GenHdr(); // Implemented by plugin
char *ftr = GenFtr(); // Implemented by plugin

if(fc_net_write(fileName, hdr, ftr, strlen(hdr), strlen(ftr),
```

```
    FC_CNTLEN, PR_INTERVAL_NO_TIMEOUT, sn, rq) != IO_OKEY)
{
    ereport(LOG_FAILURE, "fc_net_write() failed");
    return REQ_ABORTED;
}
```

NSAPI 関数とマクロのアルファベット順リスト

C

D

F

L

M

N

P

R

S

U

V

magnus.conf 内の指令のアルファベット順リスト

A

C

D

E

F

H

I

K

L

M

N

P

R

S

T

U

W

事前定義済みの SAF のアルファベット順リスト

`Init` SAF については、付録 H「magnus.conf 内の指令のアルファベット順リスト」を参照してください。

A

B

C

D

F

G

H

I

K

L

M

N

P

Q

R

S

T

U

索引

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

ち

つ

て

と

も

ゆ

よ

り

れ

ろ

わ