SIP Servlet API
Version 1.0 (和訳)

沖電気工業株式会社

2003 年 7 月 23 日

コピーライトについて

この資料は、SIP Servlet API の共同著作権者である Sun Microsystems 社の許可を得て、沖電気工業(株)が”SIP Servlet API Version 1.0”仕様書を日本語化したものです。本翻訳物の著作権は、沖電気工業(株)が保持しています。
以下に、許諾の文章を示します。

"Both Sun Microsystems and dynamicsoft have joint copyright of the specification, SIP Servlet API v1.0. So, this translation and distribution are permitted by the Java Community Process(SM) PMO of Sun Microsystems."
By Steven Grover. JAIN(TM) Project Manager, SUN Microsystems Limited.

これを訳した文章を示します。

Sun Microsystems 社と dynamicsoft 社は共同で SIP Servlet API v1.0 仕様のコピーライトを保有しています。 そして、翻訳および配付は、Sun Microsystems の Java Community Process(SM) PMO によって許可されています。
Steven Grover. JAIN(TM) プロジェクトマネージャ, SUN Microsystems 社.

また、一般的な注意を示します。

“The formal specification is only the original English one, so hope to use this translation for reference. If you have some questions and problems, please consult the original specification. This japanese translation are done by carefully, but the rightness is not guaranteed.”

これを訳した文章を示します。

正式な仕様書は、” SIP Servlet API v1.0”のオリジナルの英語の文書だけです。したがって、この翻訳の文書はあくまで参考(リファレンス)としてご利用ください。もし、疑問点や問題が生じた場合には、オリジナルの仕様書を参照して下さい。この日本語訳は、注意深く行ったつもりですが、正しいということは保証されていません。

また、以下にオリジナルの仕様書のコピーライトの文書を示します。

SIP Servlet API Specification ("Specification")
Version: 1.0
Status: Final Release
Specification Lead: dynamicsoft Inc. ("Specification Lead")
Release: February 4, 2003

NOTICE

The Specification is protected by copyright and the information described therein may be protected by one or more U.S. patents, foreign patents, or pending applications. Except as provided under the following license, no part of the Specification may be reproduced in any form by any means without the prior written authorization of the Specification Lead and its licensors, if any. Any use of the Specification and the information described therein will be governed by the terms and conditions of this license and the Export Control Guidelines as set forth in the Terms of Use on the Sun website. By viewing, downloading or otherwise copying the Specification, you agree that you have read, understood, and will comply with all of the terms and conditions set forth herein.

The Specification Lead hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under the Specification Lead's intellectual property rights that are essential to practice the Specification, to internally practice the Specification for the purpose of designing and developing your Java applets and applications intended to run on the Java platform or creating a clean room implementation of the Specification that: (i) includes a complete implementation of the current version of the Specification, without subsetting or supersetting; (ii) implements all of the interfaces and functionality of the Specification without subsetting or supersetting; (iii) includes a complete implementation of any optional components (as defined by the Specification) which you choose to implement, without subsetting or supersetting; (iv) implements all of the interfaces and functionality of such optional components, without subsetting or supersetting; (v) does not add any additional packages, classes or interfaces to the "java.*" or "javax.*" packages or subpackages or other packages defined by the Specification; (vi) satisfies all testing requirements available from the Specification Lead relating to the most recently published version of the Specification six (6) months prior to any release of the clean room implementation or upgrade thereto; (vii) does not derive from any of the Specification Lead's source code or binary code materials; and (viii) does not include any of the Specification Lead's source code or binary code materials without an appropriate and separate license from the Specification Lead. The Specification contains the proprietary information of the Specification Lead and may only be used in accordance with the license terms set forth herein. This license will terminate immediately without notice from the Specification Lead if you fail to comply with any provision of this license. Upon termination or expiration of this license, you must cease use of or destroy the Specification.

TRADEMARKS

No right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors, the Specification Lead or the Specification Lead's licensors is granted hereunder. Sun, Sun Microsystems, the Sun logo, Java, JAIN, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED "AS IS" AND IS EXPERIMENTAL AND MAY CONTAIN DEFECTS OR DEFICIENCIES WHICH CANNOT OR WILL NOT BE CORRECTED BY THE SPECIFICATION LEAD. THE SPECIFICATION LEAD MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER

EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE OR THAT ANY PRACTICE OR IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADE SECRETS OR OTHER RIGHTS. This document does not represent any commitment to release or implement any portion of the Specification in any product.

THE SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION THEREIN; THESE CHANGES WILL BE INCORPORATED INTO NEW VERSIONS OF THE SPECIFICATION, IF ANY. THE SPECIFICATION LEAD MAY MAKE IMPROVEMENTS AND/OR CHANGES TO THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THE SPECIFICATION AT ANY TIME. Any use of such changes in the Specification will be governed by the then-current license for the applicable version of the Specification.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL THE SPECIFICATION LEAD OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO ANY FURNISHING, PRACTICING, MODIFYING OR ANY USE OF THE SPECIFICATION, EVEN IF THE SPECIFICATION LEAD AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend the Specification Lead and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or clean room implementation; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

If this Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

You may wish to report any ambiguities, inconsistencies or inaccuracies you may find in connection with your evaluation of the Specification ("Feedback"). To the extent that you provide the Specification Lead with any Feedback, you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant the Specification Lead a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose related to the Specification and future versions, implementations, and test suites thereof.

目次

1. はじめに

SIP (Session Initiation Protocol) は、IP 電話/プレゼンス/インスタントメッセージ等のマルチメディアアプリケーションで利用する IP セッションを確立・変更・終了させるためのプロトコルです。このような通信インフラ (プロトコル) の重要な側面として、インフラ上にアプリケーションサービスをプログラミングできることが挙げられます。SIP Servlet API の目的は、SIP という通信プロトコル上にアプリケーションサービスを実現するための標準のプラットフォーム機能を提供することにあります。ここで言うプラットフォームには、Java の API だけではなく、パッケージングや配備等の環境も含まれています。

1.1. SIP Servlet API の目的

SIP Servlet API の重要な特徴を以下に示します。

- **SIP シグナリング**:SIP Servlet API を利用して開発するアプリケーションでは、ユーザエージェントクライアント (UAC)、ユーザエージェントサーバ (UAS) およびプロキシ (Proxy) の動作を含む、完全な SIP シグナリング処理を扱うことができます。
- **簡潔さ**:ネットワークの接続点/再送/Cseq/Call-ID/Via ヘッダやルート等の開発者にとって「本質的ではない」複雑な処理は、コンテナが制御を行うため、アプリケーション開発者は意識する必要がありません。
- **融合型アプリケーション**: Web/電話/プレゼンス等、複数のプロトコルやメディアタイプを統合した「融合型アプリケーション」を開発することができます。
- **サードパーティによるアプリケーション開発**: サーブレットモデルを利用することにより、サードパーティによるアプリケーション開発が容易になります。XML で記述される配備記述子 (deployment descriptor) により、アプリケーション情報を開発者からデプロイヤに伝えることができます。
- **アプリケーションの組合せ**: 送受信される一つのリクエスト(request)やレスポンス(response)に対して複数のアプリケーションを実行させることができます。 各アプリケーションはそれぞれルールを持ち、他のアプリケーションと独立して、決められた順序・規則の基で実行されます。
- **キャリアグレード**: サーブレットでは、アプリケーションデータをコンテナの管理するセッションオブジェクトに保存します。高可用性を実現するために、このデータを永続化することや、バックアップすることもできます。

1.2. SIP Servlet とは

SIP サーブレットは、Java でプログラミングされたアプリケーションコンポーネントです。SIP サーブレットは、SIP サーブレットコンテナによって管理され、SIP のシグナリング処理を実行します。他の Java のコンポーネントと同様に、SIP サーブレットはプラットフォーム独立な Java のクラスとして実行時にロードされ、Java 対応の SIP アプリケーションサーバ上で稼動します。

サーブレットコンテナはサーブレット機能を提供するサーバで、サーブレットエンジンとも呼ばれます。サーブレットはサーブレットコンテナを介して、SIP クライアントとの間で、SIP のリクエストメッセージとレスポンスメッセージを交換します。

1.3. SIP Servlet コンテナとは

サーブレットコンテナは、SIP のリクエストとレスポンスの送受信を行う、ネットワークサービスを提供するアプリケーションサーバの機能の一部です。サーブレットコンテナでは、アプリケーションの実行順序を決定すると共に、サーブレットの生成から消滅までを管理します。

サーブレットコンテナは SIP サーバ上に最初から組み込まれるか、またはサーバ固有の拡張 API を利用することによって、SIP サーバに追加コンポーネントとしてインストールすることができます。更に、サーブレットコンテナはサーブレット対応のアプリケーションサーバとして実現することもできます。

SIP サーブレットコンテナは、SIP トラヒックに関連するネットワーク情報（トランスポートのプロトコル、IP アドレスとポート番号の組合せ）を管理します。SIPの仕様では、すべての SIP エレメントが UDP と TCP の両方を必ずサポートし、オプションとして TLS や SCTP、その他のトランスポートセキュリティを提供することが必要となります。

サーブレットコンテナはサーブレットの実行環境にセキュリティ上の制限を委ねることになります。Java 2 Standard Edition1.2（J2SE）や Java 2 Enterprise Edition1.3(J2EE)では、Java 2 プラットフォームにより定義されたパーミッションアーキテクチャを使うことになります。例えば、ハイエンドのアプリケーションサーバでは、サーブレットコンテナ内の各種コンポーネントが、実行上の影響を受けないことを保証するために、「Thread」オブジェクトの生成に制限を掛けることもできます。

1.4. HTTP Servlet API との関係

Java Servlet API は Java Servlet 仕様[Servlet API]として定義されています。一般的な部分が javax.servlet パッケージとして、また HTTP に特化した部分が javax.servlet.http として定義されています。この仕様書では 一般的なものを “Servlet API”、HTTP 特化したものを “HTTP Servlet API”と表記します。SIP Servlet API は一般的な Servlet API をベースに HTTP Servlet API と同様に拡張されており、javax.servlet.sip パッケージにより定義されています。そこで、SIP サーブレットコンテナは、javax.servlet と javax.servlet.sip パッケージを提供することになります。

SIP Servlet 仕様書は、Java Servlet 仕様の構成に沿って書かれており、SIP Servlet API に特有でない内容も含んでいます。そのような部分は、Java Servlet 仕様書をベースに、HTTP とは異なる SIP サーブレットの特徴を反映した修正を行っています。例えば、SIP にも HTTP にも当てはまる場合には、「Web アプリケーション」の代りに、広い意味の「Servlet アプリケーション」という表現を使うことにします。

1.4.1. HTTP Servlet API との違い

SIP は HTTP から派生して規定されている範囲があり、二つのプロトコルには多くの共通の仕様を持っています。両プロトコルとも、リクエスト-レスポンス型のプロトコルであり、メッセージもとても良く似た構造やフォーマットを持ちますが、サービスを提供する上で、二つのプロトコルには以下のような重要な違いがあります。

- (HTTP Servlet アプリケーションを含む) HTTP サービスは、通常、サービスを実行する HTTP サーバ上に配置され、そのサーバがリクエストに対する最終のレスポンスを生成します。それに対して、SIP アプリケーションの重要な機能には知的なルーティング機能があり、自サーバがリクエストを処理せずに、プロキシとして動作することが必要となります。
- HTTP は SIP のように peer-to-peer (注:エンドエンドで可能な) のプロトコルではなく、Web アプリケーションが最初にリクエストを送信することはありません。一方、SIP アプリケーションは、自身でリクエストを開始できる必要があります。"incoming call" (入りの呼) を受け付けるアプリケーションは、それに続く「BYE」リクエストを発信側に送ることにより、呼を終了させられることが必要となります。"wakeup-call" (目覚まし型) アプリケーションは最初に「INVITE」リクエストを送り、通話セッションを確立できる必要があり、また、プレゼンスサーバアプリケーションは最初に「NOTIFY」を送信できる必要があります。更に、バックトゥバックユーザエージェント(B2BUA)はプロキシ機能だけでは実現できず、クライアントとしての機能を提供することにより、複数の呼に対する制御が実行できるようになります。このように、SIP サービスの基盤としては、クライアント側の機能も必要となり、SIP Servlet API にもクライアントとしての機能が提供されています。

受信するリクエストに応答する Servlet API から受け継いだ機能に加えて、SIP Servlet API では以下の機能をサポートする必要があります、

- 複数のレスポンスの生成機能 (例えば、一つ以上の 1xx レスポンス)
- 複数の宛先に対するリクエストのプロキシ機能
- リクエストの発信機能
- リクエストに対応したレスポンスの受信機能

1.4.1.1. Servlet.service() の使用

上記の機能を実現するために、SIP Servlet API はオリジナルの Servlet API インタフェースを HTTP Servlet API とは異なった方法で使用します。SIP サーブレットアプリケーションは登録済みのイベントが発生した時に起動されます。このイベントは、受信するリクエストでもレスポンスでも良く、イベントの受信時に、javax.servlet.Servlet インタフェースの service メソッドとして呼び出されます。

void service(ServletRequest request, ServletResponse response);

このメソッドは、HTTP サーブレットではアプリケーションの入口として利用されますが、SIP サーブレットでは少し異なった方法で利用します。SIP サーブレットで利用する場合は、「request」と「response」のどちらか一方が 「null」 となります。受信するリクエストを処理するために起動する時には、「response」が「null」となり、反対にレスポンスを処理するために起動する時には、「request」が「null」となります。

／＊ 注: このことは、SIP アプリケーションではリクエストとレスポンスの間に一対一の対応の必要が無いということを表わしています。＊／

1.4.1.2. 非同期

別の重要な違いに、HTTP API が同期モデルであるのに対して、SIP Servlet API のイベントモデルが非同期である点が挙げられます。これはアプリケーションがイベントを通知するリクエストに対して、すぐには応答しなくても良いこ

とを意味しています。アプリケーションは他のアクションを実行し、コンテナに制御を戻した後で、必要な時にレスポンスを返すことができます。なお、アプリケーションが時間内にリクエストに応答するのに失敗した場合でも、コンテナはアプリケーションインスタンスのタイムアウト監視により、リソースが最終的に開放されることを保証しています。

非同期機能は、イベント型サービスのプログラミングを簡単にし、長時間トランザクションが終了するのを待つ際に、スレッドを占有しない等の利点があり、B2BUA のようなアプリケーションの実現を可能にします。

1.4.1.3. アプリケーションの組合せ

HTTP Servlet API モデルは、受信したリクエストに対して一つのアプリケーションを実行しますが、SIP リクエストでは、複数のサービスを実行することの方が望ましい場合があります。この仕様書では、SIP サーブレットコンテナが複数のアプリケーションを起動する場合に満足する必要のある、アプリケーションの組合せモデルについて説明します。このモデルについては 2 章で説明することにします。

1.4.2. 融合アプリケーション

SIP Servlet API は HTTP Servlet API とは独立に実現することができます。しかし、多くの興味深いアプリケーションサービスが複数のコミュニケーションモデルを組合せることによって実現できます。例えば、電話、Web、電子メール、プレゼンスやインスタントメッセージ等があります。そこで、SIP Servlet API と HTTP Servlet API の両方の機能をサポートでき、アプリケーションが SIP と HTTP コンポーネントの両方を持つようなサーブレットコンテナを実装できることが重要な目標となります。この仕様書では、融合したコンテナのための要求条件を定義することにします。

融合された SIP/HTTP サーブレットアプリケーションは、SIP と HTTP の両方の部分に関する配備記述子を持ち、アプリケーションのインスタンスを表すアプリケーションセッションオブジェクトを通して状態を共有します（10 章を参照して下さい）。

1.5. アプリケーション例

1.5.1. 位置情報サービスの例

ルーティングは SIP の重要な機能であり、SIP サービスの一般的な機能となっています。ここでは、受信するリクエストの URI 要求に対してデータベースを参照し、リクエストに宛先アドレスを設定してプロキシする位置情報サービスの例を示します。このアプリケーションサービスは、以下のような手順で実行されます。

1. アリスは、ボブ（sip:bob@example.com）へ電話します。「INVITE」リクエストは、位置情報サービスを起動する Servlet コンテナが受信します。
2. 位置情報サービスは、SIP 仕様には依存しない方法を利用して、その宛先の位置を調べ、相手のボブ（callee）が SIP URI 情報で識別される二つの位置が登録されていることを確認します。
3. 位置情報サービスは、この二つの宛先に対して、レコード･ルーティングを行うことなく、同時にプロキシします。
4. 宛先の一方から「200 OK」が応答されると、他方へのプロキシ（分岐）はコンテナによってキャンセルされます。
5. 「200 OK」がアリスへ戻され、この発呼処理が普通通り正常に完了します。

この例では、アプリケーション（とアプリケーションがホストされるアプリケーションサーバ）は、SIP による会話の確立にのみ関わり、その後の会話内のシグナリング処理に関わることはありません。

1.5.2. マルチプロトコル Servlet アプリケーション

ここでは、融合型アプリケーションの例を示します。融合型アプリケーションは SIP と HTTP コンポーネントの両方から構成され、SIP はメディアのセッション確立だけでなく、プレゼンスの Subscription と Notification のプロトコル[SIMPLE]にも利用しています。

このサービスを CSBNA（話中と無応答時の呼び返し、Call Schedule on Busy or No Answer）と呼びます。前の例のように、サービスは発信者(caller)と着信者(callee)との間のネットワークサーバ上に存在します。着信者が話中や電話に出られないために呼設定が失敗した時には、着信者が対応可能になった時点で発信者に折り返し電話を掛けるか否かを選べるような Web ページの URL をアプリケーションが返します。アプリケーションはその人のプレゼンスに対して、「Subscribe」することにより、着信者の状態を知ることができます。着信者が対応可能になるとアプリケーションが通知を受取り、アプリケーションは 3PCC の手法を使って元の発信者と着信者の間でセッションを確立します。

図 1 は CSBNA アプリケーションのコールシーケンスを示します。

FIGURE 1. Call Flow for the CSBNA Application.

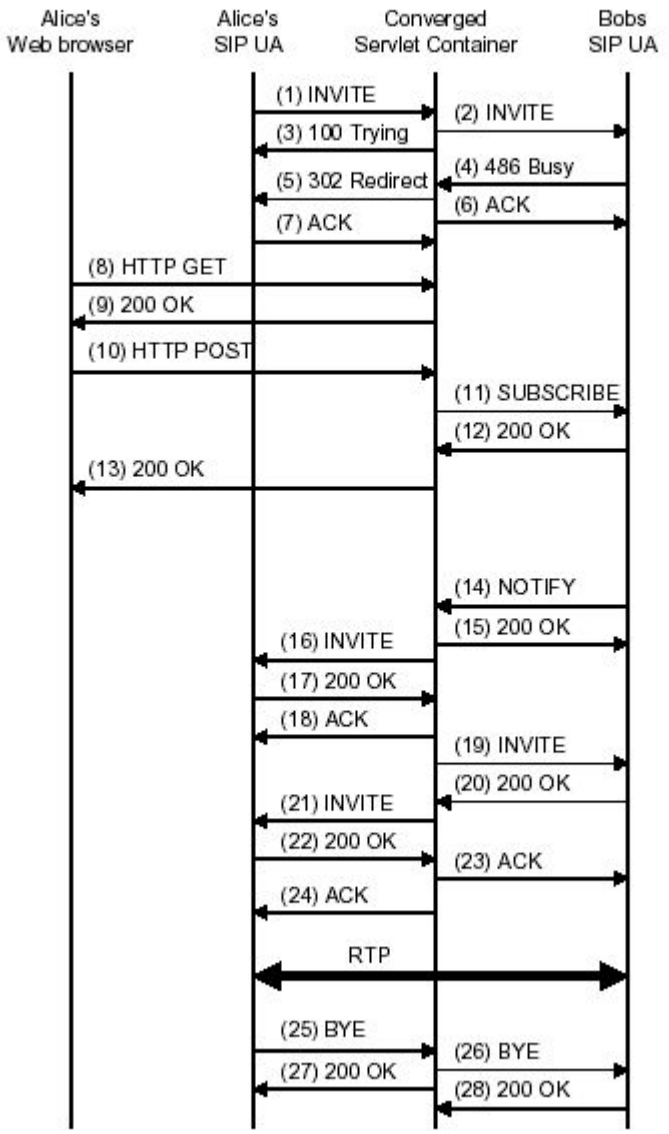


図1 CSBNA アプリケーションのコールシーケンス

1. アリスが、ボブに電話をかけます。「INVITE」リクエストが（SIP/HTTP を融合した）サーブレットコンテナに送信されます。

2. CSBNA SIP サーブレットは、B2BUA として振る舞い、「INVITE」をボブの SIP フォンに送信します。

3. （コンテナは）アリスの SIP フォン（UA）から「INVITE」の再送が行われないように、「100 Trying」を送信します。

4. ボブの SIP フォンは、電話中のため、「486 Busy Here」を返します。

5. CSBNA SIP サーブレットは、「302 Temporarily Unavalable」を返します。レスポンスには HTTP URL を含む Contact ヘッダが含まれており、その URL は、CSBNA アプリケーションの一部である HTTP サーブレットのアドレスを指しています。

6. SIP アプリケーションサーバは、SIP プロトコルに従って、ボブの SIP フォンに「ACK」を送信します。

7. アリスの SIP フォン（UA）は、アプリケーションサーバに「ACK」を送信します。

8. アリスの SIP フォン（UA）は、「302」レスポンスから HTTP の Contact URL を読み出して Web ブラウザを起動しま

す。
9. HTTP CSBNA サーブレットは、ボブの状態に基づいて、自動的に発呼処理を再開するかをアリスに尋ねる Web ページを表示します。このフォーム画面には、サーブレットにより、特にアリスとボブの SIP アドレスに関する必要な情報が設定されています。
10. アリスは、このサービスを見て、Web ページの Submit ボタンを押し、アプリケーションサーバに HTTP リクエストを送信します。
11. サーブレットのサービスは、SIP の「SUBSCRIBE」リクエストをボブの SIP フォン（UA）に送信します。これにより、ボブが通話可能な状態になった時に、その状態をサービスが知ることができます。
12. ボブの SIP フォン（UA）は、この申込みを受け付けます。
13.HTTP CSBNA サーブレットは、送信されたフォームを受取り、アリスが発呼のスケジュールを変更する、またはキャンセルするために使える Web ページを返します。
14. 暫く時間がたち、ボブが電話の受話器を置くと、UA から SIP の「NOTIFY」リクエストが送信され、CSBNA 加入者サーブレットは、ボブの電話が空いたことを知ります。
15. CSBNA アプリケーションは、「NOTIFY」に対して「200 OK」を返します。（アプリケーションは、この時点でボブのプレゼンスを「USUBSCRIBE」しますが、図では省略してあります。）
16-24. アプリケーションは、アリスとボブとの接続を、3PCC の手順で行います ［3PCC]。この場合、呼が接続され、メディアは RTP を使って送信されます。
25-28. 通話が終了します。

融合したコンテナ機能の他に、この例は「アプリケーションセッション」の必要性を表わしています。CSBNA アプリケーションのインスタンスのライフタイムを通じて、5 つのポイント-ポイント間の SIP シグナリング関係が作られています。これらの関係は SipSession インスタンスに対応します。また、コンテナと HTTP クライアントの間のシグナリング関係に対応する一つの HttpSessiion が作られます。このすべてのプロトコルセッションは関連しており、状態を共有することが必要となります。アプリケーションセッションはアプリケーションのインスタンスを表わし、個々のシグナリングの関係を表すプロトコルセッションと結び付けられています。これらのプロトコルセッションの間では、状態の共有が許されています。セッションモデルについては１０章で更に議論することにします。

2. リクエストのルーティング処理とアプリケーションの組合せ処理

SIP サーブレットコンテナの重要な機能に、アプリケーションにリクエストをルーティングすることがあります。アプリケーションの起動と順序を決定する際には、コンテナは初期（initial）と後続（subsequent）のリクエストを区別することが必要となります。初期リクエストはアプリケーションをコンテナに配備する時のルールをベースにルーティングされ、一方、後続リクエストは前の初期リクエストが取った経路に沿ってルーティングされます。

2.1. 初期リクエスト

各アプリケーションはそのアプリケーションが実行される条件に関するルールを、例えば、アプリケーションの実行対象となるリクエストに関する条件等として持っています。アプリケーションのルールは、明示的に配備記述子によって定義されます。（ルール言語は 11 章で定義します。）

コンテナは初期リクエストのルーティングに関する知識を持っていません。初期リクエストは SIP ダイアログを確立するものである必要はありません。コンテナはあらかじめ設定されたルールに対して初期リクエストを適合させ、適合したルールに基づいてアプリケーション（複数可）を起動します。

レスポンスは対応するリクエストの経路を逆に辿って送信されます。これは初期リクエストでも後続リクエストでも同じです。

2.2. 後続リクエスト

初期リクエストの結果、SIP ダイアログが始まり、少なくとも一つのアプリケーションがシグナリングの経路にある場合（それは UA でもレコードルーティングプロキシでも構いません）、コンテナは同じダイアログ内の他のリクエストをシグナリング経路にあるアプリケーションにルーティングすることができるような状態を設定します。コンテナ内でその状態に基づいて内部的にルーティングされるリクエストを「後続リクエスト」と呼びます。

後続リクエストを正しくルーティングするにはいくつもの方法があり、どの方法を選ぶかは実装に任されています。一つの方法は明示的にダイアログに関するルーティング情報を内部に保存し、受信するリクエストのダイアログ ID を使って検索するものです。別の方法では、ルーティング情報を（セッション ID の形式で）「Record-Route」ヘッダに入れてエンドポイントに送信します。その情報は、後続リクエストの「Route」ヘッダから得ることができます。

2.2.1. メッセージコンテキスト

アプリケーションにメッセージが渡される際に、SIP サーブレットリクエストとレスポンスのオブジェクトは、必ず「SipSession」オブジェクトと関連付けられます。「SipSession」オブジェクトは「SipApplicationSession」に属しています。この仕様書では、アプリケーションとアプリケーションセッションと「SipSession」の組合せを「メッセージコンテキスト」と呼びます。後続リクエストを届ける際には、コンテナはどのアプリケーションをどのコンテキストで呼び出すかを決定することになります。

複数のアプリケーションを同じリクエストの基で実行する時、それぞれのアプリケーションは別のメッセージコンテキスト上で実行されます。各々のアプリケーションは、他のアプリケーションとは独立の専用のセッションオブジェクトを持ちます。アプリケーションAがセッションオブジェクト内にデータを置いた場合でも、アプリケーションBはその情報を参照することはできません。

アプリケーションが後続リクエストを処理するために呼び出される時、メッセージコンテキストは初期リクエストと同じものとなります。例えば、アプリケーションは「INVITE」を処理するために呼び出され、レコードルーティングにより転送されると仮定します。同じダイアログから「BYE」を受取ると、同じアプリケーションが同じメッセージコンテキストで呼び出されます。つまり、「INVITE」と「BYE」リクエストに対して、更に対応するレスポンスに対して呼び出される時、「getApplicationSession」と「getSession」はそれぞれ同じセッションオブジェクトを返すことが要求されています。

注意. 論理的には、すべてのSIPメッセージはアプリケーションに渡され、アプリケーションは関連したアプリケーションセッションと「SipSession」オブジェクトを持ちます。ただし、性能上の理由から、コンテナはアプリケーションに影響がない時には、セッションオブジェクトの生成を遅らせることもできます。

2.2.2. アプリケーションパス

初期リクエストとは異なり、後続リクエストは設定されたルールに基づいてアプリケーションに送信されるのではなく、初期リクエストの経路（正確には経路の一部）に基づいて送信されます。あるいは、リクエストが着信者からのものであれば、逆の経路に基づいて送信されることになります。

基本的に、SIP ダイアログを開始する初期リクエストは、コンテナ内に「アプリケーションパス」を作ります。同じダイアログの後続リクエストは、その情報に基づいて送信されます。アプリケーションパスは（複数アプリケーションに対する）複数のメッセージコンテキストから構成され、後続リクエストを処理するコンテキストを規定します。アプリケーションパスは論理的な概念であり、コンテナ内で明示的に表現されても、表現されなくても構いません。

後続リクエストは、アプリケーションパス上のアプリケーションに順番に渡されます。リクエストが発信者から送信されたのか、着信者から送信されたのかにより、アプリケーションパスを辿る方向が変わります。発信者から送信された場合は、後続リクエストは初期リクエストと同じ順番に処理されます。着信者から送信された後続リクエストは、逆の経路に従うことになります。つまり、最後のアプリケーションに最初に渡され、最初のアプリケーションまで一つずつ溯ります。プロキシアプリケーションは、後続リクエストに対しては、初期リクエストを転送する時にレコードルートが設定されている場合にだけ、呼び出されます（8.1 章を参照して下さい）。

三つのアプリケーション A、B、C があり、「INVITE」初期リクエストを順番に処理する例を説明します。3つのアプリケーションはプロキシされますが、A と C だけがレコードルートとなった場合、アプリケーションパスは A と C とそれに関連するコンテキスト（セッション）から構成されます。着信者から受信される「BYE」後続リクエストは、このアプリケーションパスに基づいてルーティングされ、最初に C、次に A に渡されます。

／＊ 注意：初期リクエストに対して確立されたアプリケーションパスに基づいて後続リクエストがルーティングされるのは、2.4.1 章で議論するカスケードサービスのモデルと一貫したものとなっています。 ＊／

初期リクエストと後続リクエストの区別は、ローカルに開始されたリクエストにも適用されます。例えば、アプリケーション A が「INVITE」を最初に送り、これがアプリケーション B に渡るとします。B はレコードルートを設定し、アプリケーションサーバの外部の宛先に「INVITE」を送ります。同じダイアログで A から後続リクエスト「BYE」が B に渡され、さらに下流に送信されます。後続リクエストのプロキシについては、8.2.8 章で議論することにします。

「SipSession」が一つのアプリケーションパスにしか属さない点には、注意が必要となります。これは初期リクエストが新しい「SipSession」のコンテキストで処理され、アプリケーションパスと SIP ダイアログには一対一の対応があるためです。初期リクエストが一つ以上のダイアログを設定する場合、コンテナは二番目以降の経路のために派生した「SipSession」を作成します。これに関しては、10.2.3 章を参照して下さい。

2.3. アプリケーションパスの後始末

「SipSession」が終了すると、それが属していたアプリケーションパスから削除されます。「SipSession」はそれが属する「SipApplicationSession」が時間切れになるか、明示的に破棄されるか、または「SipSession」自体が明示的に破棄されるかすると終了します。コンテナ内で明示的にアプリケーションパスが表現された場合には、経路が空きになった時点で削除されます。

コンテナは、内部的にルーティング情報を持っていますが、既にこの情報が消去されているダイアログに対するリクエストを受信することもあります。このような場合、リクエストを拒否するか、アプリケーションを呼び出さずにプロキシするか、それを初期リクエストとして扱い、ルールに基づいてアプリケーションに送信するか等は、コンテナのポリシーに依存します。

2.4. アプリケーションの組合せ処理

SIP アプリケーションサーバは通常多くの異なったサービスを同時に管理します。一つ以上のアプリケーションが、同じ初期リクエストの処理に関係を持つ可能性があります。そのため、与えられたリクエストに対しても複数のルールが適用され、同じリクエストに対して複数のアプリケーションの起動を許すことが必要となります。従来の電話からの一般的な例としては、発着呼の番号制限（call-screening）や着信転送（call-forwarding）があります。これはどちらも、サービスの加入者宛てに送られる「INVITE」に対して実行されます。

コンテナはアプリケーションがお互いに不利な影響を与えることが無いように、また、アプリケーションサーバが SIP の仕様に適合して振る舞うように注意することが必要です。この章では、標準に準拠した SIP サーブレットコンテナがアプリケーションの組合せのために満足する必要のある条件について説明します。

2.4.1. カスケードサービスモデル

SIP Servlet API で採用されたアプリケーションの組合せのモデルは、カスケードサービスモデル[SERL]と呼ばれます。このモデルでは以下のような条件を満足する必要があります：

サービスアプリケーションの起動は同じホストからであっても、異なるホストからの起動と全く同じシーケンス

で実行されなければならない。

この規則は、それぞれ起動されたサービスが、実行されるコンテキストに依存しないことを保証しています。アプリケーション開発者は受信するリクエストに対して、自身が唯一のアプリケーションであるかのように SIP サービスを開発することができます。これはサービスが互いに独立に実行できることを意味するもので、アプリケーションの開発を単純にします。

SIP サーブレットコンテナでこの方針を最も単純に実現するには、受信するリクエストに対して呼び出されるアプリケーションを決定する際に、必ず一つのルールだけを選ぶことが挙げられます。この場合、アプリケーションの組合せは行うことはできませんが、振る舞いはカスケードサービスモデルと合致しており、正当な実装と言うことができます。

2.4.2. プロキシの役割

SIP において、別々のマシン上に配置された複数のアプリケーションが実行されるということは、リクエストがあるアプリケーションから、コンテナにより別のアプリケーションへと送信される時に、アプリケーションの組合せが発生することを意味しています。通常、アプリケーションは送信するリクエストが直接コンテナの外の世界に行くのか、ルールに一致した別のアプリケーションに送信されるかを知ることはできず、知る必要もありません。これは SIP Servlet API の非常に重要な特性です。これによりアプリケーションの組合せが実行可能な明確に定義されたプラットフォームが提供できます。

例えば、A と B の二つのアプリケーションが受信される「INVITE」に合致するルールと共に配備されると、コンテナは最初にアプリケーション A を起動するかもしれません。その際、A が「INVITE」をプロキシする時には、コンテナは B を起動しても構いません（B のルールが依然として合致するとして）。B は、A とは独立に、自由にリクエストに応答してもプロキシしても構いません。B がリクエストをどう扱うかに関わらず、A はトランザクションの一貫性を保証されます。例えば、A は SIP および SIP Servlet API の仕様に沿って、最終レスポンスを受信することができます。コンテナの実装では、合致する複数のルールの中から次に実行するものを決めるために、どのような仕組みを利用しても構いません。

フォーキング（forking）プロキシでは、論理的には同じマシン上で下流に複数のアプリケーションを起動することになります。例えば、ルールの設定に応じて、発信時の番号制限アプリケーションが Find-me アプリケーションによってフォークされ、送信された「INVITE」のすべてに対して起動するようなこともできます。

“プロキシによる組合せ”モデルによる別の影響は、前にも触れたように、レスポンスがリクエストの逆の順番で処理されることです。つまり、レスポンスは元のリクエストを論理的に上流方向に扱うために、最後のアプリケーションから、最初に起動されたアプリケーションに向かって渡されることになります。

2.4.3. ループの検出

これまでの要求により、関連するサーブレットが起動される際に、常に真となる評価ルールでは、実装によってはリクエストが転送される際に、ルールの再評価が起きる可能性があります。そのようなサーバ仕様では、A が B にプ

ロキシし、B が元の A にプロキシするような、起動するサービスがループしてしまう可能性もあります。そのため、このようなループを検出し、適切に処理することが必要となりますが、この仕様書では具体的な方式は提示しません。カスケードサービスモデルに一貫した一つの方式は、一つの SIP サーブレットコンテナ内でのループ検出に対して、複数の SIP サーブレットコンテナ間での検出手法を適用することです。具体的に、単純で効果的な方法は、リクエストが内部的に送信された時には、常に「Max-Forwards」ヘッダの値を減少させることです。8.2.9 章では、「Max-Forawards」ヘッダに基づいて 483(Too Many Hops)エラーを生成する場合について議論することにします。

2.4.4. モデルの制限

Servlet API では、アプリケーション開発者、システム構築者、デプロイヤの論理的な役割を定義します。そこには、サーブレットコンテナに配備されるアプリケーションは、それぞれ独立であるという暗黙の了解があります。このように、単純、明解、きれい、説明が簡単であるという理由から、カスケードサービスモデルは SIP Servlet API に良く適合していると言えます。

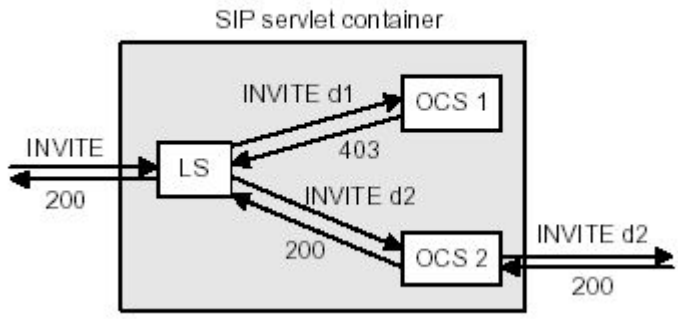
しかし、複数のサービスが同じ人か同じ会社によって開発され、互いに依存している可能性もあります。このような密接に組合されたサービスでは、アプリケーションの組合せが別のモデルにより実現される方が望ましいかもしれません。SIP Servlet API の将来の版では、そのような別のモデルを定義する可能性もあります。

2.5. 例

アプリケーションの組合せとリクエストのルーティング処理を簡単に説明するために、1.5.1 章の位置情報サービス(LS)と発信用の番号制限アプリケーション（OCS）を考えます。OCS アプリケーションは着信先のアドレスに基づいて呼の発信を制限します。説明のために、この OCS アプリケーションはレコードルートの設定を行うものとします。

図 2 は初期リクエスト「INVITE」のルーティング処理を示しています。以下のステップ毎のコールフロー説明では、セッションコンテキストを (app, as, ss)の三つ組で示します。アプリケーション名が app、アプリケーションセッションが as、「SipSession」が ss に相当します。

FIGURE 2. Example of Application Composition.



1. コンテナは「INVITE」リクエストを受信します。その「INVITE」は既にある SIP のダイアログには属していないため、コンテナはルールセットに基づいてアプリケーションに「INVITE」を送信します。
2. LS のルールが適合し、コンテナによって選ばれます。LS アプリケーションは、(LS, as1, ss1)というコンテキストで呼び出されます。
3. LS アプリケーションはデータベースを参照し、二つの宛先 d1 と d2 にプロキシします。

4. 送信された二つのリクエストはコンテナに戻り、再度ルール評価を行います。今回は、両方のリクエストに OCS ルールが適合します。「INVITE」d1 リクエストは(OCS, as2, ss2)コンテキストで処理され、「INVITE」d2 リクエストは(OCS, as3, ss3)コンテキストで処理されます。二つの OCS の起動は、別のセッションオブジェクトが関連付けられます。また、これらのリクエストは、リクエスト URI の d1/d2 を含む別のリクエストオブジェクトとして表現されています。
5. OCS アプリケーションの最初のインスタンスは、d1 が規制リスト内にあることを見つけて、「403 Forbidden」レスポンスによってリクエストを拒否します。
6. 「403」レスポンスはリクエストと反対方向に「トランザクションパス」に沿って上流方向に渡されます。これは最適なレスポンスを探している LS アプリケーションによって受信されます。この時点ではアプリケーションには渡されません。
7. OCS アプリケーションの二番目のインスタンスは、d2 が規制されるべきでないと分かり、リクエストを送信します（リクエスト URI は変更なし）。
8. 「INVITE」d2 リクエストに対して、合致するルールが無いため、コンテナは「INVITE」d2 リクエストをアプリケーションサーバ外の宛先の d2 に向けて送信します。
9. コンテナは「INVITE」d2 の分岐に対する「200 OK」レスポンスを受信します。再び、レスポンスは「トランザクションパス」を逆に辿ります。つまり、コンテナはレスポンスを最初に OCS 2 アプリケーションに渡し、(OCS, as3, ss3)コンテキストで処理され、次に LS アプリケーションに(LS, as1, ss1)コンテキストに渡します。
10. LS アプリケーションが受信する中で、「200」は最適なレスポンスであるため、LS アプリケーションは「200」レスポンスを受信します。コンテナは発信者に向けて「200」レスポンスを転送します。
11. 「200」レスポンスが SIP のダイアログを確立すると、アプリケーションパスが作られます。これは、そのダイアログの後続リクエストがシグナリングの経路上のアプリケーションに送信されるのを、コンテナが保証することを意味しています。この場合、アプリケーションパスは、(OCS, as3, ss3)だけから構成されます。
12. 「200」に対する「ACK」リクエストを受信します。これは、後続リクエストとして認識され、前に作られたアプリケーションパスと関係付けられます。（2xx 以外の最終レスポンスに対する「ACK」も、プロトコルの技術的な理由のみで必要ですが、これは単に捨てられます。）
13. コンテナは(OCS, as3, ss3)コンテキスト内で、OCS アプリケーションに「ACK」を渡します。次いで、処理が返ってきた時、通常の SIP のルーティングルールに従って、「ACK」をアプリケーションサーバの外側にプロキシします。
14. （「ACK」と同様に）「BYE」を受信した時にも、「BYE」に対して同じ処理が行われます。

これは機能のインタラクションの古典的な例を示しています。位置情報サービスに適用されると、呼の番号制限機能が迂回されてしまうため、アプリケーションを実行する順序が重要となります。コンテナは通常、管理者がアプリケーションに優先度付けをする何らかの手段を提供することになります。

論理的には、各アプリケーションは SIP クライアントとサーバのトランザクション・オブジェクトの集合を持ち、SIP 仕様[RFC3261, 章17]で定義されているトランザクション状態マシンに従って動作します。同様に、論理的には、各プロキシアプリケーションは、それぞれにプロキシロジックのインスタンスを実行します。例えば、プロキシロジックは独自のレスポンスコンテキスト[RFC3261, 16 章]を持っています。この仕様書では、RFC3261 で定義された状態マシンを追加のルールにより補強します。例えば、「100」レスポンス、2xx 以外に対する「ACK」、「CANCEL」に対するレスポンス等は無視されることになります。

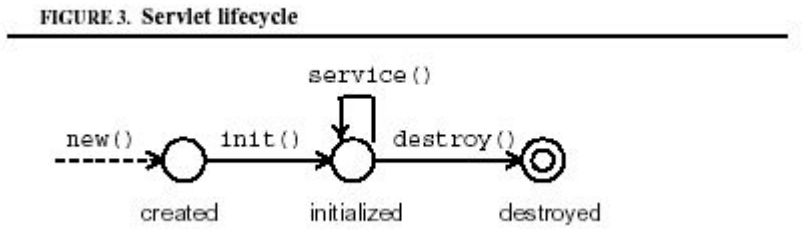
3. サーブレットインタフェース

「Servlet」インタフェースは、Servlet API の抽象化クラスの中心であり、SIP Servlet API の中心の API を構成しています。すべてのサーブレットは、直接的に「Servlet」インタフェースを実装するか、または、より一般的には「Servlet」インタフェースを実装するクラスを拡張しています。一般的な Servlet API では、「Servlet」インタフェースを実装したクラス「GenericServlet」を定義しています。

SIP Servlet API は、「GenericServlet」インタフェースを継承したクラス「SipServlet」を定義し、受信したメッセージのタイプに応じて振分けを実行します。ほとんどの目的のために、開発者は「SipServlet」を継承し、自分のサーブレットを実装することになります。

3.1. SIP Servlet のライフサイクル

SIP サーブレットは、 [Servlet API, 2.3 章]で定義されたサーブレットのライフサイクルに従って動作します。そこで記述されているすべての項目は、SIP サーブレットにも直接適用されます。 ライフサイクルを図 3 に表わします。



サーブレットコンテナはサーブレットクラスをロードし、インスタンスを生成して、そのインスタンスの init() を呼び出し、「ServletConfig」オブジェクトとして設定情報を渡します。正常に初期化されると、サーブレットはサービスを提供可能となります。コンテナは受信するリクエストやレスポンスと共に、service() メソッドを呼び出すことで、繰り返しサービスを起動します。コンテナがサーブレットインスタンスを廃棄することを決めると、destroy() メソッドを呼び出します。これにより、init() で確保されたリソースが開放され、ガーベージコレクションが実行されます。

ルールや他の設定情報を動的に変えたり、アプリケーションのコード自体を更新するような、明示的な設定手段はありません。実装により、独立したアプリケーションを新しい設定やコード込みで配備することや、古いアプリケーションを廃棄するといった作業をサポートしても構いません。

3.2. SIP メッセージの処理

基本「Servlet」インタフェースは、クライアントのリクエストを扱うために service メソッドを定義しています。このメソッドは、コンテナがサーブレットインスタンスにメッセージを送信する毎に呼び出されます。

サーブレットアプリケーション内で並列にメッセージを扱うために、開発者はサーブレットを、ある時点で service メソッドが複数スレッドにより実行されるような、マルチスレッド対応に設計する必要があります。

一般的に、サーブレットコンテナは別スレッドで service メソッドを並列に実行することにより、同じサーブレットへ

の並列のリクエストを処理します。

SIP サーブレットは、受信するリクエストとレスポンスの両方を扱うために呼び出されます。どちらの場合でも、メッセージは「Servlet」インタフェースの「service」メソッドを介して送信されます。

```
void service (ServletRequest req, ServletResponse res)
        throws  ServletException, java.io.IOException;
```

イベントがリクエストであれば、レスポンスの引数は 「null」になります。逆もまた同じで、イベントがレスポンスであれば、リクエストの引数は 「null」になります。SIP イベントを処理するために呼び出す際、引数として「SipServletRequest」と「SipServletResponse」インタフェースを実装する必要があります。「SipServlet」インタフェースの「service」メソッドの実装は、受信するメッセージの中の、リクエストを「doRequest」メソッドに、レスポンスを「doResponse」メソッドに一つ一つ振り分けます。

```
protected void doRequest (SipServletRequest req);
protected void doResponse (SipServeletResponse res);
```

これらのメソッドは、以下の章で示すように、更に振り分けられていきます。

3.3. SIP 固有のリクエストの処理メソッド

「SipServlet」抽象サブクラスには、基本「Servlet」インタフェースで利用できる以上のたくさんのメソッドが定義されています。これらのメソッドは、リクエストによる SIP の処理を実行するために、「SipServlet」クラスの中で「doRequest」メソッドから（また間接的に「service」メソッドから）自動的に呼び出されます。これらのメソッドを以下に示します。

- doInvite　　SIP の「INVITE」リクエストを処理します
- doAck　　SIP の「ACK」リクエストを処理します
- doOptions　　SIP の「OPTIONS」リクエストを処理します
- doBye　　SIP の「BYE」リクエストを処理します
- doCancel　　SIP の「CANCEL」リクエストを処理します
- doRegister　　SIP の「REGISTER」リクエストを処理します
- doPrack　　SIP の「PRACK」リクエストを処理します
- doSubscribe　　SIP の「SUBSCRIBE」リクエストを処理します
- doNotify　　SIP の「NOTIFY」リクエストを処理します
- doMessage　　SIP の「MESSAGE」リクエストを処理します
- doInfo　　SIP の「INFO」リクエストを処理します

最初の 6 個のメソッドは、SIP の基本仕様（RFC3261）で定義されているリクエストメソッドに対応します。以降のメソッドはいろいろな SIP の拡張が定義するリクエストメソッドに対応します。「PRACK」は RFC3262 で定義されているもので、6.7.1 章で議論します。「SUBSCRIBE」と「NOTIFY」メソッドは SIP イベント通知フレームワーク[RFC3265]と SIP プレゼンスフレームワーク[SIMPLE]で定義されています。「MESSAGE」メソッドはインスタントメッセージ[RFC3428]をサポートするものであり、最後の「INFO」メソッドはダイアログ内での汎用的なシグナリング機能を提供するものです[RFC2976]。

これらのメソッドの「SipServlet」における実装は、以下のようになります。「doAck」と「doCancel」メソッドは何もしません。他のすべてのメソッドでは、リクエストが初期リクエストかどうかを調べます（8.2.8 章に説明します）。リクエストが初期リクエストの場合、状態コード「500」で拒否されるか、または、何も実行しません（アプリケーションが初期リクエストをプロキシした際には、コンテナはメソッドコールが返ってきた時に、後続リクエストを送信します）。サーブレットは通常、サービスのために必要なメソッドのみを上書きします。

注意. 受信するリクエストの処理は非同期に実行されます。サーブレットは受信するリクエストの「service」メソッドを同時に実行することを要求されていません。リクエストは通常他のイベントによって起動されるまで、後で復旧してレスポンスを返すために「SipSession」や「SipApplicationSession」オブジェクトに保存することもできます。サーブレットが受信したリクエストに対して応答しない場合でも、コンテナ自体はレスポンスを生成しません。

アプリケーションが、「SipServlet」が知らない SIP メソッドを扱いたい場合、以下のように「doRequest」を上書きして、「super」を呼び出します。

```
protected void doRequest(SipServletRequest request)
        throw ServletException, IOException
{
        if ( “STORE”.equals(request.getMethod() )) {
                doStore(request);
        } else {
                super.doRequest(request);
        }
}
```

3.3.1. リクエストの受信

SIP サーブレットは、リクエストを処理するために、以下のような場合に起動されます。

- 初期リクエストの場合。サーブレットに関連するルールと一致して、コンテナがこのルールの適用を選択します。
- 後続リクエストの場合。UA アプリケーションかプロキシで、初期リクエストがレコードルート設定されたダイアログに関するものとなっています。
- 「INVITE」への 2xx レスポンスに対する「ACK」。アプリケーションが UAS として、あるいはレコードルート設定によりプロキシされます。
- 「INVITE」への「CANCEL」。アプリケーションが受信して、まだ最終レスポンスを返していない場合です。

どの場合でも、サーブレットは「Servlet.service」メソッドを利用し、引数に「SipServletRequest」オブジェクトと「null」のレスポンスを使用して呼び出されます。

「ACK」以外のリクエストの処理中に、サーブレットが例外を発生すると、そのリクエストに対して、サーブレットコンテナは「500」レスポンスを生成する必要があります。レスポンスのヘッダかボディの中に、問題の原因を特定するた

めの追加情報を含めることができます。

3.4. SIP 特有のレスポンスを処理するメソッド

「doResponse」メソッドは、レスポンスの状況コードの種別に従って以下のメソッドに振り分けられます。

- doProvisionalResponse　　SIP の「1xx」付加情報レスポンスを処理します
- doSuccessResponse　　SIP の「2xx」レスポンスを処理します
- doRedirectResponse　　SIP の「3xx」レスポンスを処理します
- doErrorResponse　　SIP の「4xx」、「5xx」、「6xx」レスポンスを処理します

3.5. インスタンスの数

「Servlet API、2.2 章」を参照して下さい。

注意. コンテナで生成されるサーブレットの数は、この仕様書で「アプリケーションインスタンス」と呼ぶ数とは異なります。後者の「アプリケーションインスタンス」という用語は、10 章で説明するプロトコルセッションを含み、それと一緒に使われるアプリケーションセッションを示しています。サーブレットオブジェクトは、アプリケーションインスタンスとも独立で、通常、多くの別のアプリケーションインスタンスに属するリクエストを処理します。

4. サーブレットコンテキスト

「ServletContext」は、その中で動作する SIP アプリケーションをサーブレットの観点で定義したものです。Java Servlet 仕様書の 3 章で「ServletContext」[Servlet API]が説明されており、これが SIP Servlet API にも適用できます。以下の章では、SIP Servlet API に特有の問題について説明します。

4.1. SipFactroy

サーブレットは、様々なインタフェースのインスタンスを生成するために、「SipFactory」インタフェースを使用します。

- **リクエスト**：「createRequest」メソッドは「SipServletRequest」インタフェースのインスタンスを生成し、新しいダイアログでリクエストを生成する際に、UAC アプリケーションによって使われます。既に存在しているダイアログ内で後続リクエストを生成する時には、代わりに「SipSession.createRequest」を使います。
- **アドレスオブジェクト**：「URI」、「SipURI」、「Address」インスタンスを生成するために利用します。
- **アプリケーションセッション**：新しいアプリケーションセッションを生成するために利用します。これは主にアプリケーションが起動された際に、自身を初期化するために使われるメソッドで、慎重に使う必要があります。

サーブレットコンテナは、必ず「javax.servlet.sip.SipFactory」インタフェースのインスタンスを作成する必要があります。このインタフェースでは、サーブレットに対して「javax.servlet.sip.SipFactory」と同じ名前のコンテキスト属性を利用する機能を提供します。

4.2. 拡張サポート

SIP サーブレットコンテナは、「javax.servlet.sip.supported」という名称で「ServletContext」のパラメータに「java.util.List」インタフェースの不変インスタンスを設定する必要があります。この「List」には、コンテナが提供する SIP 拡張機能を「String」名で格納します。アプリケーションはこれを利用することで、コンテナが特定の拡張をサポートしているか否かを判断することができます。使用例は 6.7.1 章を参照して下さい。

4.3. コンテキストパス

Servlet API は、「Context Path」という概念を定義しています。これは Web アプリケーションと関連する URL パスを表わします。Web アプリケーションのコンテキストパスで始まる HTTP URL を持つすべてのリクエストは、対応するサーブレットコンテキストにルーティングされます。SIP URI はパスの概念を持たないため、以下の「ServletContext」メソッドは、SIP だけのサーブレットアプリケーションやコンテナにとっては意味がなく、「null」を返す必要があります。

```
ServletContext getContext(String uripath);
String getRealPath(String path);
RequestDispatcher getRequestDispather(String path);
```

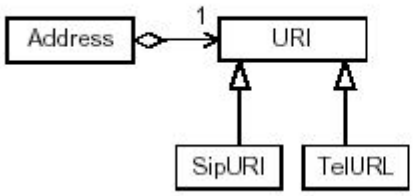
リソースのローディングに関連して、SIP だけのサーブレットのコンテキストパスは常に「/」となります。HTTP と SIP

を組合せたアプリケーションを、HTTP サーブレットに対応したコンテナで実行する場合は、コンテキストパスは HTTP サーブレットにより定義されているため、リソースのローディングは Java Servlet の仕様書[Servlet API]に従って進められます。

5. アドレス指定

アドレス指定(Addressing)は、多くの SIP 機能の中で特定の役割を持ちます。リクエスト URI に加えて、一つ以上のアドレスを伝えるために多くのヘッダ項目が定義されています。例えば、「From」、「To」、「Contact」です。これらのアドレスのフォーマットは、一般的には同じです。アドレスは、オプションとしてディスプレイネームとパラメータの組と一緒の、一つの URI から構成されています。図 4 は、SIP Servlet API のアドレス指定の抽象化とその間の関係を示しています。

FIGURE 4. Addressing abstractions



4.1 章で説明した「SipFactory」インタフェースは、これらのインタフェースのインスタンスを生成するために使われます。

5.1. アドレスインタフェース

以下の「Address」BNF 規則に従って、「Address」インタフェースがヘッダフィールドの値を示すために使われます。

address = (name-addr | addr-spec) *(SEMI generic-param)

構成する非終端子は RFC3261、25 章で定義されています。以下は不完全ですが、その内容を示しています。

name-addr = [display-name] LAQUOT addr-spec RAQUOT
addr-spec = SIP-URI / SIPS-URI / absoluteURI
display-name = *(token LWS) / quoted-string

SIP 定義の基本部分は、この文法に従う以下のヘッダフィールドの組を定義しています。(「From」、「To」、「Contact」、「Route」、「Record-Route」、「Reply-To」、「Alert-Info」、「Call-Info」、「Error-Info」[RFC3261])。「SipServletMessage」インタフェースは、どのアドレスのヘッダフィールドも操作できるメソッドを定義しています(6.4.1 章を参照して下さい)。これは、上に示した RFC3261 準拠のヘッダフィールドだけでなく、[RFC3325]の「P-Asserted-Identity」、「P-Preferred-Identity」や[RFC3327]の「Path」、[refer]の「Refer-To」等の拡張ヘッダも含んでいます。

将来の SIP の拡張では、もっとたくさんのヘッダフィールドを定義することになると思われます。アプリケーションは「Strings」ではなく「Address」オブジェクトとしてヘッダフィールドにアクセス可能な一般的なメソッドを持つ方が望ましいでしょう。

SIP サーブレットコンテナは普通、この様なヘッダの知識を持っていますが、同様に「unknown」(不明の)ヘッダも

扱える必要があります。(6.4.1 章で説明するメソッドを呼び出すことにより) アプリケーションが不明のヘッダをアドレスヘッダとしてアクセスしようとする際には、コンテナは「address」規則に従って「Address」オブジェクトとしてヘッダフィールドのすべての値を解析することが必要となります。

アドレス用の複数のヘッダフィールドの定義は、次のように少しずつ異なっています。

- いくつかのアドレスでは、特別のパラメータを定義しており、取れる値にも制限があります。例えば、「From」と「To」ヘッダは、「tag」パラメータを定義しており、その値はトークンとする必要があります。
- いくつかのアドレスには、全くパラメータを持たないものがあります。例えば、「Reply-To」です。
- いくつかのアドレスでは、ディスプレイネームを持たないものもあります。

アドレス BNF 規則では、多数のヘッダフィールドのために、すべての正しい値のスーパーセットを定義します。SIP サーブレットコンテナが実際の限定的な、特定アドレスベースのヘッダを定義していれば、メッセージをシリアライズする時にそのような制限をチェックすることが必要となります。

5.1.1. From と To ヘッダフィールド

「From」と「To」ヘッダフィールドには、各々UAC と UAS のアドレスが含まれています。7.1.1.1 章において、クローニングを通して From と To ヘッダの完全性と不変性を、コンテナがどのように保証するかを議論することにします。

5.1.2. Contactヘッダフィールド

「Contact」ヘッダフィールドは、特別なコメントを保証する別のアドレスヘッダです。

「Contact」ヘッダフィールドは、ユーザから到達可能な一つ以上の場所 (URI) を示しています。このヘッダフィールドは二つの異なる役割を果たします。一つは、UA が通信相手の UA にダイアログの後続のメッセージを送るための実アドレスを指定するための機能です。例えば、INVITE の中や INVITE に対する「2xx」レスポンスに現れます。この場合、「Contact」ヘッダフィールドは必ず一つの値を持ちます。サーブレットは、この場合「Contact」ヘッダに値を設定してはいけません。コンテナは、どのネットワークインタフェースを使用しているかを知っており、この場合の「Contact」ヘッダを選び、設定する責任を持ちます。コンテナは、アプリケーションが「Contact」ヘッダフィールドに設定しようとすると、「IlleagalArgumentException」を発生させることが必要となります。

「Contact」は、「Register」リクエストやレスポンスの中と、「3xx」と「485」レスポンスの中で利用されます。これらのメッセージの中の「Contact」ヘッダフィールドの値は、ユーザの別のアドレスを示しており、一つ以上あっても構いません。これらは、SIP サーブレットが「Contact」アドレスを設定する際に利用されます。

「Contact」ヘッダフィールドは、二つのパラメータ 「q」と「expires」を定義しています。「Address」インタフェースには、これらのパラメータを持つメソッドが含まれています。

UAC が正確な値を知ることなく、関連付けられた「address-of-record」を削除したい場合、特別な「Contact」の値「*」を「REGISTER」リクエスト中で利用します。「isWildcard」メソッドはアドレスの「Contact」値がワイルドカードの時「true」を返します。また、「SipFactory.createAddress」は (「*」の値を与えられた) ワイルドカード「Address」を返しま

す。ワイルドカード「Address」は、「Contact」ヘッダの中でしか許されないことに注意する必要があります。

5.2. URIs

SIP の各エレメントでは、URI によりアドレスを指定します。リクエストを生成する際やプロキシを行う際に、SIP サーブレットは URI を指定することにより、宛先を識別することができます。SIP は独自の URI のスキームを定義しており、SIP コンテナはそのスキームをサポートする必要があります。特定の実装では、tel URL [RFC2806]のような他の URI スキームの処理方法を定義することも可能です。

SIP サーブレットの実装では、例えば「3xx」レスポンスの「Contact」ヘッダが「mailto」や「http URL」を含んだり、コンテナが「Contact URL」表現する URI を含む「Address」オブジェクトを組み立てることができるように、どのようなスキームの URI も表現可能できることが必要となります。また、「SipFactory.createURI」では、正しい URI 文字列を指定した際には、URI インスタンスを返すことが必要です。コンテナはその URI に基づいて、SIP リクエストをルーティングできない可能性もありますが、アプリケーションにはその情報を提供する必要があります。

5.2.1. Sip URI

「Sip URI」インタフェースは、SIP URI と SIPS URI を表現します。SIP サーブレットの実装では、SIP URI をベースにリクエストをルーティングできることが要求されます。

SIP と SIPS URI は email アドレスと良く似ています。URI は、「user@host」という形式で表現し、「user」がユーザか電話番号を、「host」がホスト名やドメイン名や IP アドレスを示しています。また、SIP/SIPS URI はパラメータとヘッダから構成されます。パラメータが使えるコンテキスト上の制限に関しては、RFC3261(19.1.1 章)を参照して下さい。ヘッダは「Contact」ヘッダの中か、あるいは例えば Web ページへのリンクの様な外部の URI の中で現れる、SIP/SIPS URI の中だけで使用することができます。
例えば、以下の SIP URI の

sip:alice@example.com;transport=tcp?Subject=SIP%20Servlets

では、「tcp」という値を持つトランスポートパラメータと、「SIP Servlet」という値を持つ「Subject」ヘッダを含まれています。

SIP/SIPS URI の文字列はエスケープ文字を含んでいても構いません。SIP サーブレットコンテナは、これらの文字列をサーブレットに渡す前にエスケープを除く処理を行います。同様に、様々な SIP/SIPS URI コンポーネントで「setter」に渡される文字列には、エスケープする必要のある文字を含んでいる可能性があるため、コンテナは必要に応じてこの値をエスケープする必要があります。

文法的には、SIP と SIPS URI は URI スキームの名前以外は同じです。意味的には、SIPS スキームは TLS を使うという点で異なっています。
RFC3261 によれば、

SIPS URI は、リソースが安全に接続されることを指定する。これは特に、TLS が UAC とその URI を持つドメイン間で使われることを意味している。安全性のメカニズムはそのドメインのポリシーに依存し、そのポリシーの

基で、安全な通信がユーザを接続するために使われる。安全な通信環境が必要な際には、SIP URI で書かれたリソースは、スキームを変えるだけで SIPS URI にアップグレードできる。

文法的に同じため、SIP と SIPS URI は両方とも「SipURI」インタフェースで表され、同じように使うことができます。「isSecure」メソッドが、「SipURI」が SIP と SIP URI のどちらをを表しているか判別するために利用できます。また「setSecure」メソッドでスキームを変更することができます。

5.2.2. TelURL

Tel UTL は、[RFC2806]で定義されています。tel URL スキームは、電話網上の端末のアドレス、例えば電話番号を表すのに利用されます。SIP サーブレットコンテナは、tel URL に基づいた SIP リクエストをルーティングできても、できなくても構いません。しかし、tel URL を解釈できる必要があります。「SipFactory.createURI」は、正しい tel URL が指定された時に、「Tel URL」インタフェースのインスタンスを返すことが必要です。

6. リクエストとレスポンス

この章では、SIP サーブレットメッセージオブジェクトの構造について説明します。7 章と 8 章では、メッセージ処理の実行上の側面について説明します。

6.1. SipServletMessage インタフェース

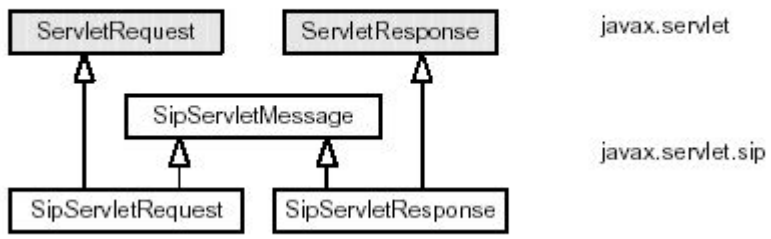
汎用の Servlet API には、サーブレットがクライアントからリクエストを受信した際に、対応する「ServletRequest」オブジェクトの内容を検査し、「ServletResponse」オブジェクトに様々な属性を設定するという、一般的な処理が定義されています。HTTP サーブレットは必ずサーバ側に配置され、受信するリクエストに対してレスポンスを生成します。このモデルは HTTP と良く合っています。

SIP サービスでは、リクエストを開始することや、プロキシすることが実行できなければいけません。そのために SIP のリクエストとレスポンスのクラスは対称である必要があります。つまり、リクエストは読み込み可能なだけでなく、書き込み可能である必要があります。同様に、レスポンスも書き込み可能なだけでなく、読み込み可能であることが必要となります。

「SipServletMessage」インタフェースは、「SipServletRequest」と「SipServletResponse」に対して、共通のメソッドを多数定義しています。例えば、メッセージヘッダやコンテンツへの 「setter」 や 「getter」 等があります。

図 5 は、SIP のリクエストとレスポンスのインタフェースが、HTTP のリクエストとレスポンスと同様に、汎用の「javax.servlet」インタフェースを拡張し、更に「SipServletMessage」インタフェースを追加していることを表しています。

FIGURE 5. Request-response Hierarchy



6.2. 暗黙のトランザクション状態

「SipServletRequest」や「SipServletResponse」オブジェクトは一つの SIP トランザクションに属しています。(SIP 仕様により定義された)トランザクション状態マシンをベースに、処理のポイントにより、送信して良いメッセージには制限が付けられることになります。サーブレットがトランザクション状態マシンに違反するメッセージを送信しようとすると、コンテナは「IlleagalStateException」を発生させます。

以下の条件の一つを満たすような「SipServletMessage」は、「commited」（訳注：コミットされた）と呼ばれます。

- メッセージが、最終レスポンスが生成されるような、受信された“入の”リクエストである。
- メッセージが、送信される“出の”リクエストである。
- メッセージが、UAC として動作するサーブレットが受信した、“入り”のレスポンスである。
- メッセージが、上流側に転送されるレスポンスである。

6.3. メッセージコンテンツへのアクセス

HTTP Servlet API は、ストリームメタファにより、メッセージのコンテンツへのアクセス手段を提供します。アプリケーションは送信されたデータに対して、「InputStream」か「Reader」を使ってアクセスでき、「OutputStream」か「Writer」を使ってコンテンツを作成できます。

SIP は、メッセージのコンテンツについては、HTTP より e-mail に似ています。メッセージは一般的に短く、「Content-Length」ヘッダをすべてのメッセージに含める必要があるという SIP の要求仕様のために、作成されたコンテンツに対して、チャンクエンコーディングを使うのは実用的ではありません。また、SIP アプリケーションは、しばしばメッセージのコンテンツをパージング済みの表現により取り扱うことが必要となります。

このような理由から、SIP メッセージにはストリームベースのコンテンツのアクセス手段はありません。「ServletRequest」のメソッド「getInputStream」と「getReader」は、「null」を返し、同様に、「ServletResponse」のメソッド「getInputStream」と「getReader」は、「null」を返すことが必要です。

「SipServletMessage」インタフェースには、メッセージコンテンツに対する「setter」と「getter」が定義されています（exceptions は除きます）。

```
int getContentLength();
void setContentLength(int len);
String getContentType();
void setContentType(String type);
Object getContent();
byte[] getRawContent();
void setContent(Object obj, String type);
```

このインタフェースは、JavaMai API が javax.mail.Part インタフェース[JavaMail]で、メッセージコンテンツに対するアクセスを定義している方法と似ています。「getRawContent」メソッドは、JavaMail にはありませんが、バックトゥバックユーザエージェント(B2BUA)を書く時に有効となります。B2BUA では、あるメッセージから別のメッセージに、内部を解析するオーバーヘッドを与えることなく、コンテンツをコピーすることが必要となります。（コンテンツの中身に何が入っているか気にしなくても良いためです。）

どの型のオブジェクトが「getContent」によって返されるかは、メッセージの「Content-Type」に依存しています。これは、MIME タイプ text/plain に対して文字列オブジェクトを返すために、またコンテナが特別の知識を持たない

MIME メディアタイプを表現するために必要となります。「multipart」 MIME コンテンツに対しては、javax.mail.Multipart オブジェクトを返すことが推奨され、text 以外のコンテンツのタイプが不明なものに対しては、コンテナは byte[] 型を返すことが必要です。

同様に、「setContent」は、どんな MIME タイプでも、 byte[] 型を受け付ける必要があり、コンテンツタイプ text と一緒に使う際には String 型を受け付けます。String 型以外のオブジェクトとコンテンツタイプ text が一緒に呼ばれた場合、コンテナは本体の文字列データを作るために、コンテンツの「Object」に対して toString() を呼び出します。繰り返しとなりますが、「multipart」の MIME タイプを扱う際には、SIP サーブレットの実装が javax.mail.Multipart コンテンツを扱えることを推奨しています。

6.3.1. 文字エンコーディング

上で示したいくつかのメッセージコンテンツ accessor は、生の 8 ビット bytes 文字と 16 ビット Unicode 文字の間で変換を行える必要があります。「SipServletMessage」の文字エンコーディング属性は、どのマッピングを使うかを示しています。

```
String getCharacterEncoding();
void setCharacterEncoding(String enc)
        throws UnsupportedEncodingException;
```

文字エンコーディングは文字列で指定され、[RFC 2278]で文書化された規則に従います。

文字エンコーディングは、「getContent」や「setContent」メソッドの動作に影響を与えます。メッセージの文字エンコーディングは、「setCharacterEncoding」、「setContentType」や「setConetntLanguage」を呼び出すことによって変更することができます。

受信するメッセージに対しては、文字エンコーディングは「Content-Type」ヘッダフィールドの charset パラメータで指定されています。送信側が正しく charset を設定していない場合には、受信したメッセージを特定のエンコーディングで扱わせるために「setCharacterEncoding」を使うことができます。

6.4. ヘッダ

サーブレットは、「SipServletMessage」インタフェースの以下のメソッドを使って SIP メッセージのヘッダにアクセスすることができます（[Servlet API、4.3 章と 5.2 章]も参照して下さい）。

```
String getHeader(String name);
ListIterator getHeaders(String name);
Iterator getHeaderNames();
void setHeader(String name, String value);
void addHeader(String name, String value);
```

「getHeader」メソッドは、名前のあるヘッダフィールドの値に対するアクセスを行います。いくつかのヘッダフィール

ド、例えば「Warning」は、SIP メッセージ内に複数の値を持っている可能性があります。その場合、「getHeader」はヘッダフィールドの最初の値を返します。「getHeaders」メソッドでは、これらの値を「String」オブジェクトのイテレータとして返し、指定されたヘッダフィールドのすべての値に対してアクセスすることが可能となります。

「setHeader」メソッドは、名前と値を指定することにより、ヘッダを設定します。既にそのヘッダが存在する場合には、新しいヘッダに置き換えられます。指定した名前に複数の値がある場合には、全部の値がクリアされ、新しい値で置き換えられます。

「addHeader」メソッドは、特定の名前と値を指定することにより、メッセージにヘッダを追加します。既に指定された名前のヘッダが一つ以上ある場合には、新しい値が既にあるリストに追加されます。

6.4.1. アドレスヘッダフィールド

上で示したメソッドは、SIP ヘッダフィールドの値を文字列として扱います。5.1 章で議論したように、多くの SIP ヘッダフィールドにはアドレスが含まれています。パージングされた形式でアドレスにアクセスする機能を有していると、便利であるだけでなく、文字列としてヘッダフィールド値にアクセスするよりも良い性能を期待することができます。

以下のメソッドは、「Address」インタフェースに対して定義されています。

```
Address getAddressHeader(String name);
ListIterator getAddressHeaders(String name);
void setAddressHeader(String name, Address addr);
void addAddressHeader(String name, Address addr, boolean first);
```

ヘッダフィールドがメッセージの中で複数の値を持っていれば、「getAddressHeader」メソッドは最初の値を返します。「getAddressHeaders」メソッドは、「Address」オブジェクトのイテレータを返すことにより、指定されたヘッダフィールドのすべての値に対するアクセスが可能となります。

メッセージから取得した、またはメッセージに設定した「Address」オブジェクトでは、「Address」オブジェクトに対する変更を行うことにより、メッセージの対応するヘッダフィールドの値がそれに従って変更されることになります。

「Address」オブジェクトは、同時に一つ以上の「SipServletMessage」に属することができます。このような場合、「Address」の変更は、両方のメッセージのヘッダフィールドの値を変更します。このような共有機能は、意図しないバグを引き起こすことがあります。アプリケーション作成者は、アドレスのコピーを行い、共有を避けることで、予防的なプログラムを行うことができます。

6.4.2. システムヘッダ

「system header」という用語は、サーブレットコンテナが管理するヘッダにおいて、サーブレットが「setHeader」や「addHeader」を呼ぶことにより、直接的に変更してはいけないヘッダという意味を表しています。これには「Call-ID」、「From」、「To」、「CSeq」、「Via」、「Record-Route」、「Route」ヘッダがあります。また、セッションのシグナリングのアドレスを参照するために使われる時、つまり、「REGISTER」リクエストとレスポンス、更に「3xx」と「485」レスポンス、(以

上の 4 つ) 以外のメッセージの中では、「Contact」も同様にシステムヘッダに含まれています。サービスがこれらのヘッダを直接設定する必要はなく、これを禁止することにより、コンテナが (SIP の) プロトコル上の正しい動作を行うことが容易になります。SIP サーブレットコンテナは、システムヘッダを変更しようとすると「IlleagalArgumentException」を発生させます。

コンテナは、システムヘッダが「Address」オブジェクトや「getAddressHeader」メソッドが返す「ListIterator」を介してアクセスされた場合でも、この不変性の要求を満足させる必要があります。

6.4.3. TLS 属性

受信するリクエストやレスポンスが、安全なプロトコル、例えば TLS によって送信されている場合には、この情報は「SipServletMessage」インタフェースの「isSecure」メソッドを使って判断できることが必要となります。

注意.「isSecure」メソッドは、メッセージが安全な転送手段を使って受信されたか否かを判断します。勿論、SIP は hop-by-hop の形態でしばしば使われるため、メッセージが end-to-end で安全なプロトコルで送られたことを意味してはいません。どこかで安全でないリンクを通して送信されている可能性もあります。

メッセージに TLS の証明があった場合、サーブレット開発者に対しては、「java.security.cert.X509Certificate」型のオブジェクトの配列として export されます。これらは、「SipServletRequest」と「SipServletResponse」からもアクセスすることができます。リクエストには「javax.servlet.request.X509Certificate」属性が、レスポンスには「javax.servlet.response.X509Certificate」属性が使われます。

6.5. トランスポートレベルの情報

以下の「SipServletMessage」のメソッドにより、アプリケーションは受信したメッセージからトランスポートレベルの情報を取得できます。

```
String getLocalAddr();
int getLocalPort();
String getRemoteAddr();
int getRemotePort();
String getTransport();
```

これらのメソッドは、受信するメッセージに対してのみ意味があります。メッセージをローカルに受信した時、(getLocalAddr, getLocalPort)は IP アドレス／ポート番号を返します。(getRemoteAddr, getRemotePort)はメッセージの送信元の IP アドレス／ポート番号を返します。同様に getTransport はメッセージで使われたトランスポートプロトコルを返します (例、“UDP”, “TCP”, “TLS”, “SCTP”)。

6.6. リクエスト

「SipServletRequest」オブジェクトでは、ヘッダとメッセージのボディを含む SIP リクエストの情報をカプセル化しま

す。

6.6.1. パラメータ

リクエストパラメータは、リクエストの一部としてクライアントからサーブレットコンテナに送信される文字列です。パラメータは、アプリケーションには名前－値の組として提供されます。与えられたパラメータ名には、複数のパラメータ値を設定することができます。以下の「ServletRequest」のメソッドがパラメータにアクセスするために使用することができます。

```
String getParameter(String name);
Enumeration getParameterNames();
String[] getParameterValues(String name);
```

「getParameterValues」メソッドは、パラメータ名に関連するパラメータ値のすべてを含む「String」オブジェクトの配列を返します。「getParameter」メソッドの返す値は、「getParameterValues」の返す「String」オブジェクトの配列の第一要素と必ず一致している必要があります。

注意. 複数の値を持つパラメータのサポートは、主に HTTP のために利用されます。HTML フォームでは、複数の値を持つパラメータを含むことが許されています。

受信する「SipServletRequest」をアプリケーションに渡す場合、コンテナはリクエスト自体の性質に依存した方法でパラメータを設定します。

- ルールの一つが一致し、アプリケーションが呼び出される初期リクエストに対して、リクエスト URI が SIP URI か SIPS URI なら、渡されるパラメータはリクエスト URI の中で表現され、その他の URI スキームなら、そのパラメータセットは未定義となります。
- 予めロードされた「Route」ヘッダが、呼び出されるアプリケーションを指定する初期リクエストに対しては、そのパラメータは「Route」ヘッダの SIP URI か SIPS URI になります。
- ダイアログ中の後続リクエストに対しては、アプリケーションに渡されるパラメータは、アプリケーション自身が初期リクエストかレスポンスのために「Record-Route」ヘッダに設定したものとなります（8.4 章を参照して下さい）。これは、通常は Route ヘッダフィールドの先頭になります。しかし、上流の SIP 要素が“strict router”の時、リクエスト URI として返ってくる可能性があります（RFC3261 を参照して下さい）。上流の SIP 要素が strict router かを判断し、それに従って正しくパラメータを設定するのはコンテナの役割となります。

6.6.2. 属性

属性は、リクエストに関連したオブジェクトです。属性は、コンテナが API 経由では表現できないような情報を設定することや、サーブレットが他のサーブレットと（「RequestDispatcher」経由で）通信するための情報を設定しても構いません。属性は「ServletRequest」インタフェースの以下のメソッドを使ってアクセスさせることができます。

```
Object getAttribute(String name);
Enumeration getAttributeNames();
void setAttribute(String name, Object o);
void removeAttribute(String name);
```

一つの属性名には、一つの属性の値だけが関連付けられます。

「javax.servlet.sip.」のプレフィックスで始まる属性の名前は、SIP Servlet API の仕様で定義するために予約済みのものとなっています。属性セットに設定されるすべての属性は、Java 言語のパッケージ命名規則[JLS]に従って命名することが推奨されています。

6.7. レスポンス

レスポンスオブジェクトは、UA サーバからクライアントに送信されるヘッダとコンテンツをカプセル化しています。HTTP の場合とは異なり、一つのリクエストに対して複数のレスポンスが返ることがあります。

「SipServletResponse.getRequest()」メソッドは、レスポンスに関連するリクエストオブジェクトを返します。UAC と UAS のアプリケーションにおいては、これは「SipServletRequest」となります。プロキシアプリケーションでは、分岐に対して送信されたリクエストのオブジェクトになります。この場合、元のリクエストを求めるには「getProxy().getOriginalRequest()」メソッドを使うことができます（8.2.3.2 章と 10.2.3 章を参照して下さい）。

6.7.1. 信頼できる暫定レスポンス

暫定レスポンス（状態コードが 100-199 のレスポンス）は、SIP の基本仕様では確実には送信されません。ただし、「100」以外の「1xx」の送信に対して、信頼性と順番を保証する拡張仕様が定義されています[RFC 3262]。これは、PSTN 網との連携と良く関連付けられますが、暫定レスポンスがサービスを実現するために不可欠な情報を運んでいる場合に有効です。

「100rel」拡張のサポートは、SIP コンテナではオプション扱いとなります。（「100rel」は暫定レスポンスの拡張のために定義されたオプションタグの名前です。）サポートする場合には、コンテナは「ServletContex」経由でアプリケーションに提供されるサポートリストに、「100rel」の文字列を入れることが必要となります（4.2 章を参照して下さい）。アプリケーションは、実行時にサポートリストにこの「String」があるか否かで、（サポートされているか否かを）判断することができます。

「1xx」を信頼性を持って送信したい時には、アプリケーションは「SipServletResponse.sendReliably()」.を呼び出します。このメソッド呼び出しが成功すれば、コンテナはレスポンスを確実に送信します。なお、レスポンスが「100」以外の「1xx」レスポンスでない、リクエストが「INVITE」でない、UAC が「Supported」か「Requested」ヘッダに 100rel のサポートを指定していない、またはコンテナ自体が 100rel をサポートしていない場合には、当該メソッドでは「Rel100Exception」を発生させることになります。

100rel 拡張をサポートするコンテナでは、「RSeq」と「RAck」ヘッダはシステムヘッダになります（6.4.2 章を参照して下さい）。つまり、コンテナにより扱われ、アプリケーション側では追加･変更･削除を行うことはできません。「PRACK」リクエストは他の後続リクエストと同様に扱われます。これは、対応する「INVITE」と同じ「SipSession」と関連付けられ、「SipServlet」の実装が「doPrack()」メソッドにディスパッチする、「Servlet.service()」に渡されることを意味してい

ます。

RFC3262 で定められた時間内に、信頼できる暫定レスポンスのための「PRACK」が受信されない場合、アプリケーションが（そのメソッドを）実装していれば、コンテナは「SipErrorListner」インタフェースの「noPrcakReceived」メソッドを使ってアプリケーションに通知を行うことができます。「INVITE」トランザクションに対して、RFC3262 が推奨している「5xx」レスポンスを返すかどうかはアプリケーションに依存しています。元の「INVITE」リクエストは、レスポンスのなかった信頼できるレスポンスと同様、「SipErrorListner」に渡される「SipErrorEvent」から取得することができます。

100rel をサポートしたコンテナは、信頼できる暫定レスポンスの再送を受信した場合、アプリケーションを再び呼び出すことは行いません。

同様に、100rel をサポートしたコンテナは、UAC アプリケーションが受信した信頼できる暫定レスポンスを「RSeq」ヘッダフィールドで定義した順番で受取ることを保証しています。

6.7.2. バッファリング

Servlet API は、レスポンスとしてサーバにより返されるコンテンツのバッファリングに関連して、いくつかの「ServletResponse」メソッドを定義しています[Servlet API, 5.1 章]。

- **getBufferSize**
- **setBufferSize**
- **reset**
- **flushBuffer**

これらのメソッドは、HTTP サーブレットの性能を大きく向上させます。一方、HTTP とは異なり、SIP はコンテンツの転送プロトコルを意図しておらず、バッファリングには通常あまり関心がありません。したがって、これらのメソッドは有効な結果を生むことがないため、実装では何もしなくても構いません。なお、「getBufferSize」が 0 を返すことを推奨しています。

「isCommitted」メソッドは、メッセージが 6.2 章で示した意味でコミットされていれば、「true」を返します。

6.8. 国際化

言語の識別タグが、SIP の様々な場面で使われます。これには、「Accept-Language」と「Content-Language」ヘッダといくつかの「Content-Type」ヘッダの「charset」パラメータの値があります

SIP Servlet API では、言語は「java.util.Locale」のインスタンスで識別されます。

注意. 「javax.servlet」のインタフェース「ServletRequest」と「ServletResponse」は国際化に関連するメソッドを含んでいますが、これらはサーブレットだけが受信したリクエストにのみ対応しており、SIP Servlet API に対しては不十分です。

6.8.1. 利用できる言語の指定

ユーザエージェントは、プロキシやピアの UA に対して、コンテンツや警告等を、どの自然言語で受信する必要があるかを、オプションで指定することができます。この情報は UA 側から「Accept-Language」ヘッダを使って通知することができます。

以下のメソッドは、「SipServletMessage」インタフェースでメッセージの送信元が望ましいロケールを指定するために利用します。

void setAcceptLanguage(Locale locale);
void addAcceptLanguage(Locale locale);

「setAcceptLanguage」メソッドは、「Accept-Language」ヘッダの望ましい言語を設定し、既に設定されている「Accept-Language」ヘッダをすべて削除します。一方、「addAcceptLanguage」は使用可能なロケールのリストに対して、(優先度が最も低い) 別のロケールを追加します。

以下のメソッドは、「SipServletMessage」インタフェースでメッセージの送信元が望んでいたロケールを決定するために利用します。

Locale getAcceptLanguage();
Iterator getAcceptLanguages();

「getAcceptLanguage」メソッドはクライアントがコンテンツに受入れる望ましいロケールを返します。クライアントの望ましい言語を決定するために、「Accept-Language」ヘッダをどのように解釈しなければならないかについては、RFC2616 (HTTP1.1)の 14.4 章を参照して下さい。

「getLocales」メソッドは、「Locale」オブジェクトの集合を表す「Iterator」を返します。その集合は、望ましい順番で、メッセージの送信元の UA が受入れ可能なロケールを示しています。

クライアントが望ましいロケールを指定しなければ、「getLocale」はサーブレットコンテナのデフォルトのロケールを返すことが必要となります。「getLocale」では、デフォルトのロケールの「Locale」要素の「Iterator」を返します。

6.8.2. メッセージコンテンツの言語の指定

ボディを含むメッセージを送信する時、SIP サーブレットは「SipServletMessage」インタフェースの「setContentLanguage」メソッドを呼ぶことにより、言語を指定することができます。

void setContentLanguage(Locale locale);

このメソッドでは、(SIP 仕様で記述された別の手段により) クライアントにロケールを正確に伝えるために「Content-Language」ヘッダを正しく設定しなければなりません。

特定のコンテンツタイプに対して、charset コンポーネント付で「setContent」や「setContentType」メソッドを呼び出

すことにより、メッセージの文字エンコーディングが設定されることに注意して下さい。

サーブレットの開発者により指定されない場合は、メッセージコンテンツのデフォルトのエンコーディングは“UTF-8”となります。

メッセージを受信する時に、サーブレットは以下の「SipServletMessage」のメソッドを使うことにより、メッセージのロケールの情報を取得することができます。

Locale getContentLanguage();

7. ユーザエージェントとしての動作

この章では、SIP サーブレットがリクエストの開始やリクエストへのレスポンスのような、様々な UA の動作を実行する方法について説明します。SIP 仕様では、クライアントとサーバのトランザクション状態が示され、各トランザクション状態において許される動作を定義しています。6.2 章で説明されるように、アプリケーションが SIP の状態マシンに違反する動作を実行しようとする場合、コンテナにより「IllegalStateException」を発生します。

SIP のクラインとサーバという概念は、特定のトランザクションに関連する時に意味があります。UA として動作するアプリケーションは、しばしばダイアログ中でクライアントとサーバの両方の動作を行うことが必要となります。

7.1. クライアント機能

以下では、アプリケーションがどのようにリクエストを送り、レスポンスを受信するかを示します。

7.1.1. 初期リクエストの生成

リクエストを開始する時、UAC は初めに「SipServletRequest」オブジェクトを生成し、次いでそのリクエストを送信します。リクエスト内容を変更した際も同様です。リクエストの生成は、リクエストが初期リクエストで既に存在する SIP のダイアログに所属していなければ、「SipFactory」のオーバロードされた「createRequest」メソッドを呼び出すことにより行われます。

より正確には、SIP サーブレットは既に存在するダイアログに属さない、新しいリクエストを送信するために、以下のような手順を実行することになります。

- 「SipFactory」を調べる。「SipFactory」は「ServletContext」の属性として提供されます。
- オーバロードされた「SipFactory.createRequest」の中の一つを呼び出します。
- 結果の「SipServletRequest」を必要なら変更します。例えば、コンテンツをセットします。
- リクエストオブジェクトの「send」を呼び出します。

「SipFactory」インタフェースでは、新しいリクエストを生成するために以下のメソッドを定義しています。

```
SipServletRequest createRequest(
        SipApplicationSession appSession,
        String method,
        Address from,
        Address to);
SipServletRequest createRequest(
        SipApplicationSession appSession,
        String method,
        URI from,
        URI to);
SipServletRequest createRequest(
```

SipApplicationSession appSession,
String method,
String from,
String to) throws ServletException;

返される「SipServletRequest」は、指定した「ApplicationSession」に属する新しい「SipSession」の中で存在しています。新たに作成された SipSession のためのハンドラは、アプリケーションの「*default servle*t」です（10.2.6 章を参照して下さい）。これは、アプリケーションが 「SipSession.setHandler」 を呼び出すことによって変えることができます。コンテナには、返ってきたリクエストに新しくグローバルで一意の「Call-ID」を割当てることが、SIP の仕様により要求されています。「From」と「To」ヘッダが引数の from と to によって指定されます。コンテナは「CSeq」ヘッダを加えることに責任に持ちます。リクエストのメソッドは引数 method によって与えられます。リクエスト URI のデフォルト値は「To」ヘッダの URI となります。「REGISTER」リクエストでは、これに加え、SIP リクエスト URI のユーザ部分が空きであることが要求されています。アプリケーションは新たに作成される「SipServletRequest」において、「setRequestURI」を呼び出すことによって、この値を変えることができます。

注意. SIP では「ACK」と「CANCEL」リクエストは、特別な状態を持っています。7.1.6 と 7.1.7 章では、アプリケーションがどのように、何時、これらのリクエストを生成するかについて議論します。上で述べた「SipFactory」メソッドは、「ACK」と「CANCEL」を生成するために呼ばれると、「IlleagalArgumentException」を発生することになります。

一度作成してしまえば、アプリケーションはリクエストオブジェクトを変更しても構いません。例えば、リクエスト URI を設定する、ヘッダを追加する、コンテンツを設定する等があります。その後、リクエストは「SipServletRequest」の「send」メソッドを呼ぶことにより送信されます。リクエストは、SIP 仕様に従って、リクエスト URI とリクエスト内の「Route」ヘッダをベースに、コンテナによりルーティングされます。

「SipFactory.createRequest」の 4 番目のバージョンが、B2BUA を書くために用意されています。これに関しては、7.3 章を参照して下さい。

7.1.1.1. From と To のアドレスのコピー処理

「Address」オブジェクトがメッセージのヘッダフィールド値としてセットされる時、それらの値はコピーされません。しかし、 From と To ヘッダフィールドは、コンテナによって管理され、SIP 仕様により、タグ・パラメータに関する要求を課せられるという点で、特別なものとなっています。

このような理由から、「SipFactory.createRequest」メソッドは、それを新しく作るリクエストの「From」と「To」ヘッダフィールドの値とする前に、「from」と「to」の引数からディープコピーを作成します。コピーされた「to」アドレスが「tag」引数付だった場合、外に出る初期リクエストの「To」ヘッダフィールドは、tag なしでなければならないため、（タグがコピーから）削除されます。コピーされた「from」アドレスは、SIP 仕様に従って、新しい「tag」引数を付与されます。どのような「from」と「to」の URI も、SIP の「From」と「To」ヘッダの文脈で許されなければ、コピーから削除されます。これには、ヘッダや様々な引数も含まれます。

コピーされた from と to の「Address」オブジェクトは新しい「SipSession」と関連付けられます。ダイアログが確立されれば、コンテナは、相手側の SIP 要素によって選ばれた値で、「to」タグの値を更新しなければなりません。同じ「SipSession」に属する後続リクエストは、同じ「From」と「To」ヘッダを持ちます。アプリケーションは、「SipSession」か新しく生成したリクエストからコピーされた「from」と「to」「Address」オブジェクトを（新しいタグを反映して）取出すことができます。しかし、それを変更することは許されていません。

7.1.2. 後続リクエストの生成

後続リクエストでは、リクエストは既に確立したダイアログの中で作られます。アプリケーションでは、そのダイアログを表す「SipSession」の以下のメソッドを使うことにより、リクエストを作成します。

SipServletRequest createRequest(String method);

このメソッドは、コンテナが提供するリクエスト URI と、「Call-ID」/「From」/「To」/「CSeq」/「Route」ヘッダの値から、リクエストを組み立てます。アプリケーションは、リクエストオブジェクトに対して「send」メソッドを呼び出す前に、リクエストを変更しても構いません。

7.1.3. Route ヘッダフィールドの値の追加

UAC は、リクエストの「Route」ヘッダフィールドに、「Route」ヘッダフィールドの値として、SIP URI や SIPS URI を追加しても構いません。これにより、リクエストが相手先に届く前に、「Route」の値で示されたプロキシを経由するという影響がでることになります。これは SIP の仕様において、予めロードされた「Route」として規定されているものです。

「SipServletRequest」インタフェースは、「Route」ヘッダフィールドに要素を追加するために「pushRoute」メソッドを定義しています。

void pushRoute(SipURI uri);

「Route」ヘッダはスタックのように動作します。pushRoute はリストの先頭に要素を追加し、SIP のルーティングのアルゴリズムはそれを頭から取り出します。そのため、「pushRoute」は何度も呼び出され、リクエストは最後に追加されたプロキシを最初に訪れることになります。

同様のメカニズムを、プロキシアプリケーションで利用することが可能です。8.2.1 章を参照して下さい。

7.1.4. UAC としてのリクエストの送信

一旦作成されたリクエストは、「SipServletRequest」オブジェクトの「send()」を呼び出すことにより、送信されます。

void send() throws IOException;

「send」メソッドが例外を発生させた場合、リクエストの送信が失敗し、アプリケーションはこのトランザクションに関するコールバックも受信することはありません。つまり、アプリケーションはこのリクエストに対するレスポンスを受信することがありません。

「send」メソッドが例外を発生しない場合、コンテナはアプリケーションに最終レスポンスを応答することが必要とな

ります。コンテナはリクエストを非同期に送信しても良く、その場合、「send」メソッドが成功して返った後に、送信が失敗する可能性もあります。そのような状況では、コンテナは自分で最終レスポンスを生成します。この場合、「404 (Not Found)」が適当と考えられます。

7.1.5. レスポンスの受信

UAC アプリケーションは、「100」レスポンス（と再送）を除く、すべての受信するレスポンスに対して呼び出されます。呼び出されるサーブレットは、そのリクエストが属する「SipSession」の現在のハンドラとなります（10.2.6 章を参照して下さい）。コンテナは、引数に「SipServletResponce」とリクエスト引数に「null」を入れ、「Servlet.service」を呼び出します。サーブレットが「SipServlet」抽象クラスを継承していれば、レスポンスは状態コードの値に基づいて、更に振り分けられます。

7.1.5.1. 複数のダイアログの処理

下流のプロキシに対するフォーキングにおいては、一つの「INVITE」リクエストに対して複数の「2xx」レスポンスを受信することがあります。このような場合、10.2.3 章で示すように、二番目とそれに続くダイアログのために、コンテナは元の「SipSession」を複製します。複製されたセッションオブジェクトは、同じアプリケーションデータを持ちますが、その「createRequest」メソッドでは二番目以降のダイアログに属したリクエストを生成します。そこでは、そのダイアログに特有の「To」タグを持っています。

7.1.6. ACK の送信

SIP では、「INVITE」リクエストへの最終レスポンスにより、UAC から UAS へ、そのレスポンスに対する「ACK」が送信されます。「2xx」以外の最終レスポンスへの「ACK」は、信頼性を高めるために必要であり、hop-by-hop に送信されます。これは、UAC だけでなくプロキシにおいても生成されます。一方、「2xx」最終レスポンスに対する「ACK」は、そのシグナルがセッションの設定を完了させ、ボディの中に意味的に有用な情報として、例えば、IP アドレスやセッションのメディアストリームのコーデック等を運ぶことができます。この「ACK」は、UAC から UAS に向かってエンドツウエンドに送信されます。

一般的に、「2xx」最終レスポンスへの「ACK」だけが、サービスの興味の対象となります。そのため、SIP サーブレットコンテナは「2xx」以外の最終レスポンスへの「ACK」を生成する責任があり、アプリケーションは「2xx」最終レスポンスのための「ACK」の生成に責任を持ちます。「ACK」を表すリクエストオブジェクトの生成は、「SipServletResponse」のメソッドを呼び出すことにより行います。

SipServletRequest createAck()

アプリケーションは、「send」を呼び出す前に、「ACK」オブジェクトを変更しても構いません。「2xx」の再送を受信し、コンテナが UAC アプリケーションに「2xx」の再送を行ってはいけない場合、アプリケーションが生成した、「2xx」「ACK」の再送は、コンテナの責任として実行することが必要となります。

できるだけ早く再送を止めるために、コンテナが「2xx」以外の最終レスポンスへの「ACK」の生成を、アプリケーションの呼び出しよりも優先して行うことを推奨しています。

7.1.7. CANCEL の送信

UAC は、「INVITE」に対する「CANCEL」リクエストを送信することにより、進行中の「INVITE」リクエストを中止することができます。UAC として動作する SIP サーブレットも、元の「INVITE」リクエストオブジェクトに対して、以下の「SipServletRequest」メソッドを呼び出すことができます。

SipServletRequest createCancel()

アプリケーションは、「Send」を呼び出すことにより、「CANCEL」リクエストを送信します。「CANCEL」リクエストに対するレスポンスは、アプリケーションに渡されないことに注意して下さい。

UAC とプロキシは、「1xx」レスポンスを受信するまでは、「INVITE」リクエストを中止することはできませんが[RFC3261、9.1 章]、SIP サーブレットアプリケーションは構いません。暫定レスポンスを受信するまで、「CANCEL」の送信を遅らせるのは、コンテナの責任となります。

7.2. サーバ機能

定義上は、受信するリクエストに対して最終レスポンスとして応答する SIP サーブレットは、対応するトランザクションの UAS となります。

7.2.1. レスポンスの送信

サーブレットは、受信するリクエストに対して、一つの最終レスポンスを生成することや、多くの暫定レスポンスを生成することができます。これはリクエストオブジェクトの「createResponse」を呼び出すことにより実行できます。その後、結果の SipServletResponse は、修正された後に「send」メソッドの呼び出しにより送信されます。

「INVITE」への「2xx」レスポンスは、エンドツウエンドに再送されるという意味で、特別なものです。SIP の仕様は、階層化ソフトウェアモデルの観点で定義され、「INVITE」を受信したトランザクションユーザ（この場合、UAS）は、定期的に「2xx」を再送する責任があります[RFC 3261、13.3.14 章]。SIP サーブレット AIP では、この仕事はコンテナで処理する必要があります。これは、完全に RFC 3261 の中でプレゼンテーションの目的のために使用された論理的なモデルの範囲内となります。

リクエストをプロキシする際、アプリケーションは（普通）それ自体の最終レスポンスを生成しません。一方、アプリケーションは、トランザクションが終了するまでは、暫定レスポンスを生成し続けても構いません。これは、上で説明したようなメカニズムを使って動作します。

7.2.2. ACK の受信

アプリケーションは、上流に送信した「INVITE」に対する「2xx」レスポンスとして、「ACK」を通知されます。

アプリケーションは、「INVITE」に対する「2xx」以外の最終レスポンスとして、「ACK」を通知されません。これらの「ACK」は、最終レスポンスの信頼性のために必要なものであり、通常はアプリケーションにとっては必要ありません。

トランザクションがタイムアウトする前に、UAC が受信した「INVITE」リクエストに対する「ACK」を送信することに失敗することがあります。これは、アプリケーションに対して、13.1 章で説明する「SipErrorListener」メカニズムを使って通知されます。

7.2.3. CANCEL の受信

アプリケーションに渡されたリクエストに対する「CANCEL」を受信し、アプリケーションがまだ元のリクエストに対して応答もプロキシも行っていない場合、コンテナは元のリクエストに対して「487(Request Terminated)」を返し、「CANCEL」リクエストに対して「200 OK」の最終レスポンスを返します。更に、アプリケーションには「CANCEL」リクエストを表す「SipServletRequest」オブジェクトを渡すことで通知します。アプリケーションは「CANCEL」リクエストを受信した後で、元のリクエストに応答しようとしてはいけません。また、「CANCEL」の通知自体に応答する必要もありません。

コンテナが「487」レスポンスを生成することと、SIP サーブレット自体が独自のレスポンスを作成することの間には、明らかな競合状態があります。この競合状態を解決するためには、通常の Java のメカニズムを使って処理します。アプリケーションが勝てば、「CANCEL」リクエストを受信したことは通知されませんが、コンテナが勝ち、サーブレットが「CANCEL」通知を受信する前にレスポンスを返そうとすると、コンテナは「IllegalStateException」を発生させます。

7.3. バックトゥバックユーザエージェント

バックトゥバックユーザエージェント（b2bua）は、二つ以上のダイアログ用のエンドポイントの役割を持ち、何らかの方法により、この二つのダイアログとの間でリクエストとレスポンスを送信します。b2bua は、サービスを壊す可能性があるため、不適当であると考えらる場合があります。この可能性は、b2bua が二つのエンドポイント間に位置し、それらのエンドポイント間のシグナリングを仲介することが理由となっています。b2bua が、二つのエンドポイント間で利用されているエンドエンドのサービスのことを知らない場合、b2bua は不注意にサービスを壊す可能性があります。

しかし、b2bua は SIP アプリケーション開発者のための重要なツールであり、SIP Servlet API により支援されます。ここで紹介する動作は、受信するリクエストから送信されるリクエストに対して、未知のヘッダをすべてコピーすることにより、エンドエンドのサービスを壊す危険の最小化を目指すものです。

b2bua として動作したいアプリケーションが初期リクエストを受信すると、以下の「Sipfactory.createRequest」を呼び出すことができます。

```
SipServletRequest createRequest(
        SipServletRequest origRequest,
        boolean sameCallId);
```

このメソッドは、以下に示す点を除いて第一引数で与えたリクエストと同一のリクエストを生成します、

- 新しいリクエストの「From」ヘッダフィールドは、コンテナにより選ばれた新しいタグを持っています。
- 新しいリクエストの「To」ヘッダフィールドは、タグを持っていません。
- 新しい Call-ID がオプションで設定されます。これは、引数 sameCallID の値に依存します。

- 「Record-Route」と「Via」ヘッダフィールドはコピーされません。通常、コンテナは、リクエストをアプリケーションサーバの外部に送信する時に、リクエストに独自の「Via」ヘッダフィールドを追加します。
- 「REGISTER」リクエスト以外では、「Contact」ヘッダフィールドはコピーされません。

「Route」ヘッダフィールドは、「origRequest」に指定されていた場合には、新しいリクエストにコピーされることに特に注意して下さい。

このメソッドは、便利さと実行効率のために用意されています。他の「SipFactory.createRequest」メソッドに関しては、返されるリクエストは新しい「SipSession」に属しています。

8. プロキシ

SIP の重要な機能の一つに、リクエストをルーティングする能力があります。これは、受信するリクエストに対して、そのリクエストを受信すべき宛先を決定する機能です。リクエストをプロキシする機能は、多くの SIP サービスにおいて最も基本となるものです。いくつかのケースにおいて、サービスはプロキシするか、リダイレクトするかの選択を行うことができます。ただ、サービスがシグナリングのパス上にあり、レスポンスを受ける必要があることから、多くのサービスではプロキシを要求することになります。

HTTP と SIP サーブレット API の間の重要な違いの一つには、HTTP サーブレットが元のサーバ上でだけ実行され、受信するリクエストへのレスポンスだけに関連するのに対して、SIP サーブレットがプロキシサーバ上に置かれ、受信するリクエストに直接応答するだけでなく、プロキシできなければならない点があります。

プロキシは受信する「SipServletRequest」から得られる「Proxy」オブジェクトを介して初期化され、制御されます。状態を保持するプロキシを行う場合には、SIP トランザクション毎に最大一つの「Proxy」オブジェクトがあります。つまり、元のリクエストオブジェクトか、あるいはトランザクションに属する別のメッセージ上で呼び出される時には、「SipServletMessage.getProxy()」は同じ「Proxy」インスタンスを返します。

8.1. パラメータ

いくつかの「Proxy」パラメータが、様々なプロキシの動作を制御します。

- **recurse:** サーブレットエンジンが自動的に再帰するか否かを指定するフラグ。再帰が可能な時、サーブレットエンジンは、「3xx」リダイレクトのレスポンスを受信すると、そのコンタクトアドレスに自動的にプロキシしようとします。デフォルト値は、「true」 です。
- **recordRoute**: アプリケーションがこのダイアログのシグナリングの経路に残るか否かを制御するフラグ。アプリケーションがダイアログの後続リクエストを受けたい時に 「true」を設定しなければなりません。デフォルト値は、「false」 です。
- **parallel:** 複数の宛先にプロキシする時に、並列(「true」)に行うか、逐次(「false」)に行うかを指定するフラグ。パラレルサーチの場合は、サーバは前のリクエストの最終レスポンスを待たずに複数の宛先にリクエストを送信しても構いません。デフォルト値は、「true」 です。
- **stateful:** プロキシの動作がトランザクションステートフルかどうかを決めるフラグ。デフォルト値は「true」であり、 ステートフルという意味です。トランザクションの状態を管理しないで送信することは、より良い性能を潜在的に持つことを意味しています。一方、アプリケーションに対しては、意味的な影響がないため、コンテナはこの引数をヒントと考えても構いません。
- **supervised:** サーブレットがレスポンスを処理するために呼ばれるか否かを示すフラグ。Supervised フラグを「false」に設定すると、その「Proxy」オブジェクトが関連するトランザクションにだけ影響することに注意して下さい。デフォルト値は、「true」 です。
- **sequentialSearchTimeout:** コンテナが、分岐をキャンセルして次の宛先にプロキシする前に、最終レスポンスを待つ時間。デフォルト値は、配備記述子の「sequential-search-timeout」要素の値であるか、またはコンテナが定めた値です。

これらの引数のすべての組合せが許されているわけではありません。特に、もし「recurse」、「parallel」、「supervised」のいずれかが 「true」 であれば、プロキシはステートフルである必要があります。

「Record-Route」や「stateful」フラグは、最初に「Proxy.proxyTo」を呼び出す前にだけ設定することができます。呼び出し後に設定しようとすると、「IllegalStateException」を発生します。プロキシ処理の最中でも、「sequentialSearchTimeout」引数と同様に、「recurse」、「parallele」、「supervised」フラグは、アプリケーションが変更しても構いません。

8.2. 操作

アプリケーションは、「SipServletRequest」から「Proxy」オブジェクトを取得し、プロキシ操作を実行します。受信する「INVITE」リクエストに対して、「Proxy」オブジェクトを生成する時、サーバはアプリケーション自身が既に送信していなければ、「100」暫定レスポンスを上流に送信します。これはクライアントからの再送を中止するために実行します。

リクエストをプロキシする前に、アプリケーションは「Proxy」パラメータを望ましい値に変えても構いません。または、リクエストオブジェクト自体を変更しても構いません。これにはヘッダの追加や削除、本体の変更や置換えも含んでいます。アプリケーションはプロキシ操作を開始するために、「proxyTo()」を呼び出します。このメソッドは下流の宛先を指定するために、一つのURIでもURIのリストでも取れるようにオーバロードされています。(SIP Servlet APIに)準拠したサーブレットエンジンは、SIP URIとSIPS URIを処理できることが要求され、Tel URI等のその他のURIをサポートすることもできます。

「ProxyTo」メソッドの一つに渡される「URI」毎に、コンテナはリクエストがプロキシされる分岐を生成します。プロキシされたリクエストは、指定された宛先 URI で置換えられたリクエスト URI を持ち、変更されたリクエスト URI に基づいてルーティングされるか、指定されていれば、「Route」ヘッダフィールドの先頭に基づいてルーティングされます。

最終レスポンスが上流に転送されるまでは、コンテナが転送すべき宛先の集合にアドレスを追加するために、アプリケーションは「ProxyTo」メソッドを何回でも呼び出すことができます。

シグナリング･パス上に留まりたいが、ルーティング上の決定も行わず、また呼の設定にも影響を及ぼさないアプリケーションは、リクエスト URI を変更せずにレコードルートを可能にしてプロキシしても構いません。これは、以下のように行います。

```
public void doInvite(SipServletRequest req) {
  ...
  Proxy p = req.getProxy();
  p.setRecordRoute(true);
  p.proxyTo(req.getRequestURI());
  ...
```

```
}
```

リクエストは、一致する規則を備えた他のアプリケーションに、あるいはリクエスト URI によって指定された外部宛先へプロキシされることになります。

8.2.1. Route ヘッダの設定

SIP の仕様で規定されているように、プロキシはリクエストが目的地に配達される前に特定の集合のプロキシを経由すると命じる、ローカルポリシーを持っている可能性があります[RFC 3261, 16.6 章]。これは、リクエストの Route ヘッダフィールド上へ、1 セットの SIP/SIPS URI を設定することにより行われます。

この設定は、「SipServletRequest」インタフェースの「pushRoute」メソッドの役割となっており、その引数として、リクエストがその最終目的地に到着する前に訪れられるべきプロキシを指定します。

「Route」ヘッダフィールドに一つの値だけが加えられる場合、コンテナは実際に Route ヘッダフィールドの値を追加することなく、通常の転送ロジックを回避して、代案として単一の「pushRoute」呼び出しで指定されたプロキシのアドレスとポートに対して、単にリクエストを送信する可能性があります。これは[RFC 3261]の中で指定された制約に従います。

8.2.2. レスポンスの送信

アプリケーションは、プロキシ処理の前や最中に、情報提供のためのレスポンスを生成する可能性があります。これは UAS としての動作と同様の方法で行われます。

一旦、リクエストがプロキシされた場合、アプリケーションは通常それ自身で最終レスポンスは生成しません。しかし、最終レスポンスを生成することが有用な場合があるため、いくつかの付加的な制約付きで許可されています。

最終レスポンスが上流に送信されていない限り、アプリケーションはそれ自身の最終レスポンスを作成し、「send()」を実行しても構いません。あたかも最終レスポンスが、プロキシの(仮想の)分岐から受け取られたかのように、以下の制限付きで、コンテナは動作します。

- 「2xx」レスポンスである場合、それ自身のレスポンスを受けるアプリケーションに通知せずに、直ちに上流に送信されます。
- プロキシ・トランザクションに未解決の分岐がある時に生成された、「2xx」以外の最終レスポンスは、レスポンスコンテキストに実際の分岐から受信したレスポンスと同じ影響を与えます。結果として、それが最適なレスポンスとして選択されるならば、コンテナは通常のレスポンスと同様のコールバックを実行します。
- 受信した最適なレスポンスが「2xx」以外で、アプリケーションが「doResponse」コールバックの中で最終レスポンス(「2xx」でも「2xx」以外でも)を生成していれば、生成したレスポンスのために再度アプリケーションを呼び出すことなく、直ちにレスポンスが送信されます。

この最後の項は、アプリケーションが、例えば、コンテナに最適なレスポンスとして選ばれたものとは別のエラーレスポンスを返すことを可能にします。アプリケーションが最適なレスポンスを受信し、独自のレスポンスを生成した場

合、それ以上多くの宛先にプロキシはできないことに注意して下さい。

これらの規則は、SIP サーブレットコンテナが SIP の仕様に違反しないことを保証するために設計されています。

上流に送信された最終レスポンスがアプリケーション自身によって生成された時には、コンテナは「SipSession」の状態を、アプリケーションが UAS であるように更新することが必要となります。

8.2.3. レスポンスの受信

ステートレスモードでプロキシされたリクエストについては、レスポンスはアプリケーションの関与なしに、常に上流に転送されます。

ステートフルモードの場合、サーブレットコンテナは、上流でのプロキシ処理のために、以下のような受信したレスポンスを自動的に転送する責任があります、

- 「100」以外の情報提供のためのすべてのレスポンス
- すべての宛先から最終レスポンスを受けた時に、受信した中の最適なレスポンス
- すべての「2xx」レスポンス

更に、「supervised」フラグが「true」の場合、サーブレットエンジンはそれらを上流に進める前にこれらのレスポンスに対してアプリケーションを起動します。そのアプリケーションでは、通知のコールバックの中で、レスポンス内容を修正して構いません。その場合、修正済のレスポンスが上流に転送されます。これは、例えば、レスポンスのボディ中のアドレスを修正する必要のあるアプリケーションレベルゲートウェイを構成する場合に有用となります。

UAC の場合には、アプリケーションは受信した「100」レスポンスに対しては、起動されません。

「2xx」と「6xx」レスポンスを受信した時には、サーバはすべての未解決の分岐に「CANCEL」を行い、新しい分岐は作成しません。

受信した最適な最終レスポンスが「2xx」でなかった場合、アプリケーションは「Proxy.proxyTo」メソッドを用いて宛先のアドレスを追加することができます。プロキシは、まるでアプリケーションに（仮の）最適なレスポンスを通知する前にそのメソッドが呼ばれたかのように、追加の宛先毎に新しい分岐を作ります。サーブレットが一つ以上の追加の宛先にプロキシしている場合、新しい分岐で“より良い”レスポンスを受信する可能性がある限り、コンテナは直には受信した最適なレスポンスを転送しません。受信した最適なレスポンスに対するコールバックとして「proxyTo」を呼び出す機能は、例えば無応答転送(call-forward-no-answer)のようなサービスを書くために役に立ちます。

これまでに説明した動作の結果として、アプリケーションは、一つのトランザクションに対して一つ以上の最終レスポンスを通知される可能性があります。これは以下の場合に起こります。

1. アプリケーションが、最終レスポンスの通知の中で、更に多くの宛先にプロキシした。
2. 一つ以上の「2xx」の再送が受信された。8.2.3.2 章を参照して下さい。
3. 複数の宛先が「2xx」レスポンスを返した。

最初の二つのケースでは、アプリケーションが、同じ最終レスポンスについて、二度以上の通知を受ける可能性があります。

8.2.3.1. INVITE に対する 2xx レスポンスの処理

7.2.1 章で説明したように、「INVITE」に対する「2xx」レスポンスは特別なものです。このレスポンスは、トランザクション層をバイパスする再送信であり、クライアントとサーバのトランザクションを直ちに終了させることになります[RFC 3261, 13.3.1.4 章 and 17.1.1 章].。一般的には、SIP プロキシは「2xx」の再送を「Via」ヘッダフィールドの値によりステートレスで上流に転送します。アプリケーションサーバの場合には、一つ以上のプロキシアプリケーションが「2xx」レスポンスを上流に転送する前に、変更する可能性があるという複雑な問題があります。このような場合、再送される「2xx」が、「INVITE」に対して最初に受信した「2xx」と全く同じ方法で変更されていることを保証する必要があります。

これはいくつかの異なる方法で実現することができますが、どのような方法を選択するのかは実装に任されています。一つの解決方法は、「INVITE」に対応する最初の「2xx」を受信した際には、「INVITE」のクライアント・サーバトランザクションを終了しないことです。この方法では、アプリケーションサーバは元の「2xx」と類似の方法で「2xx」再送を扱うことができ、必要な時にアプリケーションを起動し、そのメッセージを上流に転送することになります。このような理由のために、「INVITE」をプロキシするアプリケーションは、「2xx」レスポンスの再送信を受信する準備を行い、「2xx」を最初受信したのと同じ方法で再送を修正できる必要があります。

8.2.3.2. プロキシの分岐へのレスポンスの関連付け

レスポンスを受信したことの通知を受けた際、一つ以上の未解決の分岐がある場合に、どの分岐からのレスポンスを受けたのかを判断できることが、アプリケーションにとって役に立ちます。アプリケーションはリクエスト URI によって分枝を識別します。レスポンスが特定する分岐に対するものか否かを判断するためには、分岐のリクエスト URI とアプリケーションにより「ProxyTo」メソッドに渡された URI オブジェクトを比較することよって実行することができます。

プロキシされたリクエストに対するレスポンスについては、「SipServletResponse.getRequest()」の実行により、レスポンスが受信された分岐に対して送信されたリクエストを表わす「SipServletRequest」オブジェクトが返ります。次いで、リクエスト URI はそのリクエストオブジェクトから得ることにでき、以前「proxyTo」メソッドに渡した宛先の URI と比較することができます。

コンテナに対しては、分岐のリクエストオブジェクトのリクエスト URI が、アプリケーションが「proxyTo」メソッドに渡したものと同じ実際の URI オブジェクトであることを保証することが要求されています。つまり、参照の比較がこのチェックに有効であることを意味しています。(勿論、再帰によって、アプリケーションが明示的に要求しておらず、URI も持っていない分岐を作ることがあります。)

8.2.4. CANCEL の送信

「INVITE」をプロキシする操作は、「Proxy.cancel()」を呼び出すことにより中止することができます。これにより、コンテナがプロキシされた「INVITE」に対して「CANCEL」リクエストを発行し、すべての未解決の分岐を終了させること

ができます。通常の場合では、プロキシの動作は、SIP 仕様に従って続行されます。レスポンスの処理に関する限り、分岐をキャンセルすることは、最終レスポンスの生成を早める手段と考えることができます。(supervised フラグが「true」であると仮定すると) アプリケーションはレスポンスを処理するために依然として呼び出されることになります。

「Proxy.cancel()」の呼び出しは、現在進行中のすべての分岐をキャンセルし、プロキシ・トランザクションの現在のターゲット集合をキャンセルする効果があります。アプリケーションが引き続いて追加のターゲットに対して、「proxyTo」を呼び出すと、プロキシ・トランザクションは通常このターゲットに対する分岐を作ります。

UAC の場合では、「CANCEL」レスポンスを受信した時に、アプリケーションは起動されません。

個々の分岐をキャンセルする手段はありません。

暫定レスポンスを受信するまで、コンテナが「CANCEL」を送信することを遅らせなければならないという、7.1.7 章に述べられたコメントは、プロキシ・トランザクションの分岐にも当てはまります。

8.2.5. CANCEL の受信

ある「Proxy」オブジェクトに対応したトランザクションに対して、「CANCEL」が受信される場合、サーバは「200」を使って「CANCEL」に応答します。更に、

- 元のリクエストがまだプロキシされていなければ、コンテナは「487 (Request Terminated)」最終レスポンスにより応答します。
- あるいは、すべての分岐がキャンセルされ、レスポンス処理が通常通りに継続されます。

いずれの場合も、アプリケーションは、「CANCEL」リクエストにより起動されます。サーバが CANCEL に既に応答し、適切に未解決の分岐を取り消している場合は、通知のみとなります。「487」レスポンスを送信するサーバとリクエストをプロキシするアプリケーションの間の競合条件は、7.2.3 章で議論された UAS の場合と同様に扱われます。

8.2.6. ACK の送信

SIP 仕様は、「INVITE」リクエストに対する「2xx」以外の最終レスポンス用の「ACK」を生成することを、ステートフルプロキシに要求します。これらの「ACK」の目的は、下流のサーバでレスポンスの再送を止めるためのものであり、意味的な重要性を持っていません。したがって、SIP Servlet API では、サーブレットコンテナが「2xx」以外の最終レスポンス用の「ACK」の生成の責任を持ち、SIP サーブレットがプロキシされたリクエスト用の「ACK」を生成してはいけません。

8.2.7. ACK の受信

「2xx」以外の最終レスポンス用の「ACK」は、単に捨てられます。「2xx」用の「ACK」は他の後続リクエストとして扱われ、コンテナによってプロキシされます。アプリケーションは、明示的に「ACK」を転送してはいけません。

アプリケーションがレコードルートを設定した「INVITE」をプロキシした場合、「ACK」を修正する(例えば、「ACK」のボディに入れて運ばれるセッション記述内のアドレスを修正する)機会を与えるために、「ACK」がプロキシされる前に

アプリケーションが起動されます。この場合、元のリクエストをプロキシする時と同様に、「ACK」はその修正済のものが下流にプロキシされます。

8.2.8. 後続リクエストの処理

2 章に説明したように、“後続の”リクエストは既に確立されたアプリケーションパスをベースにアプリケーションにディスパッチされます。反対に、初期リクエストは一致したルールをベースにアプリケーションにディスパッチされます。

「recordRoute」フラグを「true」にしてプロキシする場合、サーバは同じダイアログの中で後続リクエストを受信する可能性があります。この場合、SIP 仕様[RFC 3261,16.6 章].の中で指定されているように、サーバが「Route」ヘッダのトップで指定された宛先へリクエストをプロキシすることを除いては、上に述べたように処理が進みます。

アプリケーションがリクエストオブジェクトを渡された際に、サーバが下流にプロキシする前であれば、その呼び出しの中でリクエストオブジェクトを修正することができます。アプリケーションは、明示的に後続リクエストをプロキシしてはいけません。「SipServletRequest」インタフェースの「isInitial」メソッドは、リクエストが初期リクエストであれば、「true」を返します。そこで、アプリケーションが受信するリクエストを明示的にプロキシすべきか否かを判断するための手段として利用することができます。

後続リクエストのアプリケーションを通知する呼び出し中の場合を除いて、アプリケーションは後続リクエストを拒絶することができます。呼び出しを実行した後、アプリケーションがそのための最終レスポンスを生成していなければ、コンテナはリクエストをプロキシします。また、例えば、アプリケーションがプロキシの認証処理を実行したい場合等に、プロキシアプリケーションには後続リクエストに応答する能力が必要となります。

8.2.9. Max-Forward のチェック

「Max-Forwards」ヘッダは、SIP リクエストがその宛先へ行く途中で通過することができる SIP エレメントの数を制限する役割を持ちます。このヘッダは、各ホップで一つ減算される整数により構成されています。RFC 3261 では、プロキシされるリクエストについては、「Max-Forwards」が 0 より大きい値を持つ必要があることを規定してします[RFC 3261,16.3 章]。

SIP サーブレットコンテナは、アプリケーションがリクエストをプロキシするか否かを予め知ることができないため、アプリケーションを呼び出す前に「Max-Forwards」のチェックを行うことはできません。その代り、コンテナはアプリケーションが「SipServletRequest」の「getProxy()」を呼び出した時、その値のチェックを行います。「Max-Forwards」が 0 の場合には、「TooManyHopsException」が発生し、その例外がアプリケーションによって処理されない場合には、コンテナが受取り、「483 (Too many hops)」を生成することが必要となります。

8.3. プロキシとセッション

アプリケーションがリクエストをプロキシする時、アプリケーションがレコードルート設定されていれば、そのアプリケーションは同じトランザクションや後続のトランザクションに関連したリクエストやレスポンスを処理するために、引き

続いて呼び出される可能性があります。

既存のトランザクションあるいは後続リクエストに対してイベントが生じた、または、アプリケーションが以前に「SipApplicationSession」あるいは「SipSession」を取得しているという理由により、アプリケーションが起動される場合は、コンテナはそれらの同じセッションオブジェクトを後続リクエストとレスポンスに関連付けることを保証する必要があります。

また、プロキシをフォーキングする結果として、初期リクエストが複数のダイアログの確立を引き起こす可能性があります。そのような状況の下では、コンテナは受信するレスポンスあるいは後続リクエストを、既存のダイアログの「SipSession」のクローンに関連させることが必要となります。これに関しては、10.2.3 章で議論します。

8.3.1. ステートレスなレコードルート

いくつかのアプリケーションは、ダイアログをステートフルにすることなく、ダイアログのシグナリングの経路に留まりたい場合があります。プロキシアプリケーションがレコードルートを設定するが、アプリケーションセッションや「SipSession」を使用しない場合、コンテナは「Record-Route」ヘッダフィールドを挿入してセッションオブジェクトの生成は避けることができます。コンテナは、そのダイアログで受信する後続リクエストに対して、同じアプリケーションを呼び出すことは依然として必要です（アプリケーションの組合せのために、一つ以上になる可能性はあります）。これらのアプリケーションを識別するために「Record-Route」ヘッダフィールドに状態を挿入することができます。

ステートレスでレコードルートを設定したアプリケーションは、例えば、レスポンスや後続リクエストを受信した時に、後でセッションオブジェクトを生成することはできません。プロキシアプリケーションが、初期リクエストのレスポンスに対してセッションオブジェクトを生成する必要がある場合は、プロキシを行う前にセッションオブジェクトを強制的に生成することが必要です。「Record-Route」ヘッダフィールドの中で、エンドポイントに渡される状態を変えるのでは、遅すぎます。

8.4. Record-Route パラメータ

ステートレスでレコードルートを行う場合、SIP プロキシにとってダイアログのエンドポイント（ユーザエージェント）に状態を渡し、ダイアログの後続リクエストの中でそれを受信することが、便利な場合があります。これはプロキシが挿入する「Record-Route」ヘッダフィールドに、パラメータを追加することにより実現できます。URI パラメータを含む「Record-Route」ヘッダフィールド値は、ユーザエージェントのルート集合に影響を与え、後続リクエストでの「Route」ヘッダの中でプロキシに返されます。RFC 3261 のルースルーティングのメカニズムは、SIP プロキシが後続リクエストの「Route」ヘッダフィールドの値からレコードルートさせた URI を再び手に入れることを保証しています。

レコードルートを行うプロキシアプリケーションに、（ステートフル、ステートレスのどちらの場合でも）プロキシされるリクエストの中に、コンテナにより挿入される「Record-Route」ヘッダのパラメータを設定するには、「Proxy.getRecordRouteURI」メソッドを利用します。

```
SipURI getRecordRouteURI();
```

「getRecordRouteURI」メソッドで返される URI は、URI のパラメータの形式で、ユーザエージェントへアプリケー

ション状態を設定するためにだけ利用することができます。アプリケーションは RFC 3261 で定義された SIP URI パラメータを設定することはできません。これには、「transport」、「user」、「method」、「ttl」、「maddr」、「lr」が含まれています。URI の他のコンポーネント、例えば host、port と URI scheme もアプリケーションが変更してはいけません。これらの「Record-Route」 URI コンポーネントは、コンテナによって設定され、アプリケーションがリクエストをプロキシする時点では、有効な値な場合と有効な値でない場合があります。

この方法により、「Record-Route」ヘッダフィールドに追加されたパラメータの値は、6.6.1 章で説明したように、「SipServletRequest」の「getParameter」メソッドを使って後続リクエストから回収することができます。

このメカニズムは、プロキシアプリケーションが「getRecordRouteURI」メソッドによって得られた「URI」にパラメータを加える場合、同じアプリケーションがそのダイアログ中の後続リクエストに対して、「SipServletRequest.getParameter」を利用し、そのパラメータの値を回復できることを保証していることに注目して下さい（ユーザエージェントが SIP 仕様に従っていると仮定した場合）。しかし、実際の「Record-Route」ヘッダに対して、パラメータが変更されない状態として設定される保証はありません。例えば、アプリケーションの組合せのため、パラメータは場合によっては「Contact」ヘッダの中に入る可能性があります。また、アプリケーションパスに複数のレコードルートを実行するプロキシがある場合に、コンテナは一つの「Record-Route」ヘッダだけを設定することを選択する可能性もあります。この場合、コンテナはアプリケーション間の名前の衝突を回避するために、パラメータをエンコードすることが必要です。実装では、ダイアログに関連する他のデータに加えて、パラメータ集合をローカルに格納し、次いで後続リクエストに対して設定されるダイアログ ID をベースにそのデータを検索するような方法を選択することができます。

「Record-Route」ヘッダフィールドでエンドポイントに渡されるデータのサイズには、ハード的な制限はありませんが、アプリケーション開発者は実装によっては大きなサイズのデータでは正常に機能しない可能性があることに留意して下さい。

この版の SIP Servlet API では、プロキシアプリケーションがダイアログの二つのエンドポイントに別の状態を渡すことを許していません。UAS は、リクエストの「Record-Route」ヘッダフィールドを変更せずに、「2xx」レスポンスの中にコピーします。また、プロキシアプリケーションは、「2xx」レスポンスを処理する時に、自身で前に挿入した「Record-Route」パラメータを変更する機会を与えられていません。これは「Record-Route」ヘッダフィールドの値を有効なものとして書き直し、発信者からの後続リクエストの状態と着信者からの後続リクエストの状態を異なるものにしようとします。このような機能は潜在的には有用ですが、性能やセキュリティ（プロキシが書き換えると、レスポンスの「Record-Route」ヘッダがエンドエンドで保護されない）に対して悪影響が出ることになります。

9. タイマサービス

タイマサービスは、アプリケーションがタイマをスケジュールし、タイマが終了する時にアプリケーションが通知を受けることを可能にする、コンテナが提供するサービスです。コンテナは他のリソースと同様にタイマを管理します。タイマは、セッションデータと一緒に保存されるような永続データとして実現することもできます。タイマは指定した時間の後に一回だけ終了するようにスケジュールすることや、指定した間隔毎に繰り返すようにスケジュールすることもできます。

タイマでは、三つのインタフェースをサポートしています。「TimerService」はタイマを生成する時に使うインタフェースです。「ServletTimer」はタイマを表し、コールバック時に渡されますインタフェースです。「TimerListener」は、アプリケーションによって実装され、タイマが終了した時にコンテナにより呼び出されるコールバック用インタフェースです。

9.1. TimerService

「TimerService」インタフェースはコンテナによって実装され、「javax.servlet.sip.TimerService」という名前を持つ「ServletContext」パラメータとして、アプリケーション側で利用することができます。「TimerService」では、以下のメソッドを提供します。

```
ServletTimer createTimer(SipApplicationSession appSession,
        long delay,
        boolean isPersistent,
        java.io.Serializable info);
ServletTimer createTimer(SipApplicationSession appSession,
        long delay,
        long period,
        boolean fixedDelay,
        boolean isPersistent,
        java.io.Serializable info);
```

「ServletTimer」オブジェクトは、アプリケーションセッションに関連付けられます。アプリケーションは、データをアプリケーションセッションに保存し、タイマが満了した時にそれを検索することもできます。「SipApplicationSession」インタフェースの「getTimers」メソッドは、このセッションに関連して現在スケジュールされているすべての「ServletTimer」オブジェクトを含んだ「java.util.Collection」を返します。

「createTimer」メソッドの引数は以下のものから構成されています。

- appSession ： 新しい「ServletTimer」が関連付けられるアプリケーションセッション
- delay：「ServletTimer」が最初に満了するまでの時間（遅れを ミリ秒で表現）
- period ： タイマ満了の期間（時間をミリ秒で表現）
- fixedDelay ： タイマの繰り返しを、fixed-delay か fixed-rate かを指定するパラメータ。意味的には、

「java.util.Timer」と似ています：両方のケース共に、タイマの繰り返しは、ほぼ等間隔で満了します。ただし、fixed-delay の実行は、繰り返しの再スケジュール時に、前の満了中の遅延を無視します。一方、fixed-rate ではタイマは絶対時間に基づいて再スケジュールされます。

- isPersistent ： この値が「true」の時には、サーバがシャットダウンされ、その後にリスタートされた場合に、「ServletTimer」が再作成されます。
- info ： タイマの満了と一緒に届けられるアプリケーションの情報。これはアプリケーションセッションで複数のタイマを設定している時に、アプリケーションがタイマの満了の有効性を判断するのに利用できます。

9.2. ServletTimer

「ServletTimer」インタフェースは、スケジュールされたタイマを表します。アプリケーションセッションとシリアライズされたオブジェクトが、満了のコールバックの中で「ServletTimer」から取得できます。「ServletTimer」インタフェースは、アプリケーションにタイマのキャンセルさせることや、最後の満了時間や次にスケジュールされている満了時間の情報を得るために利用することができます。

「ServletTimer」インタフェースは以下のメソッドを提供しています。

```
SipApplicationSession getApplicationSession();
java.io.Serializable getInfo();
long scheduledExecutionTime();
long getTimeRemaining();
void cancel();
```

「SipApplicationSession」が終了した場合は、そのアプリケーションセッションに属するすべてのタイマがキャンセルされることに注意して下さい。

9.3. TimerListener

アプリケーションは、タイマの満了を「TimerListener」インタフェースを通して通知されます。インタフェースは、次の一つのメソッドを持ちます。

```
void timeout(ServletTimer timer);
```

タイマを使用するアプリケーションは、「TimerListener」インタフェースを実装し、SIP 配備記述子の「listener」要素に実装クラスを宣言する必要があります（13 章を参照して下さい）。例えば、アプリケーションは「javax.servlet.sip.TimerListener」を実装した「com.example.MyTimerListener」クラスを定義します。SIP 配備記述子の中では、このリスナを以下のように宣言します。

```
<listener>
    <listener-class>com.example.MyTimerListener</listener-class>
</listener>
```

配備記述子の中には、一つの「TimerListener」の実装が宣言されることになります。

10. セッション

SIP アプリケーションは、目的としたサービスを提供するために、複数のメッセージを処理することが必要となります。サーブレット自体は、ステートレス（正確には、ダイアログ毎のデータやトランザクション毎のデータを保持しない）であり、API がメッセージを関連付ける機構を提供し、コンテナが“アプリケーションインスタンス”の中で処理される後続メッセージにアプリケーションデータを関連させる方法を規定しています。

HTTP servlet API は、HTTP セッションとしてこのような機構を提供します。「HttpSession」インタフェースは、特定のクライアントから送信される一連の HTTP リクエストをサーブレットと関連させることを可能にすると共に、アプリケーションデータの保管場所の役割を持っています。

「HttpSession」と同等の役割は、SIP Servlet API の「SipSession」インタフェースが持ちます。このインタフェースは、二つのユーザエージェント間のポインツーウポイント関係を表しており、大凡 SIP のダイアログ[RFC 3261]に対応するものとなります。

しかし、SIP アプリケーションは一般的に Web アプリケーションより複雑です。

- 多くのサービスは複数のダイアログから構成されます。例えば、会議アプリケーション、B2BUA アプリケーションや 3PCC があります。
- 複合したアプリケーションは他のネットワーク要素と複数のプロトコルを使って通信します。例えば、SIP、HTTP、 e-mail 等があります。

これは、一つのアプリケーションインスタンスが複数のポイントツウポイントの関係から構成されても良いことを表わしています。また、これらの関係は別のプロトコルを使って実現しても構いません。これは、プロトコルセッションおよびアプリケーションセッションという概念によって、SIP Servlet API に反映されています。プロトコルセッションは、主にポイントツーポイントの関係を表わすプロトコルに特有のセッションオブジェクトです。「SipSession」と「HttpSession」インタフェースは、両方ともプロトコルセッションの例となっています。

アプリケーションセッションは、アプリケーションインスタンスを表わしています。アプリケーションセッションは多くのプロトコルセッションを含み、アプリケーションデータの保管場所として使用されます。同じアプリケーションインスタンスに属するプロトコルセッションはすべて、同じ「SipApplicationSession」に属しています。例えば、B2BUA の役割を持つ SIP サーブレットは、各アプリケーションインスタンスに対して、二つの「SipSessions」と一つの「SipApplicationSession」から構成されます。

コンテナは最適化のために、例えば、アプリケーションに要求されるまで、アプリケーションセッションと SIP セッションオブジェクトの生成を遅延させることができます。ただし、セッションオブジェクトを常に作成する実装と比べて、違いが判別できないような実装を行うことが必要です。

10.1. SipApplicationSession

アプリケーションセッションは、アプリケーションデータに対するストレージの提供と、多くのプロトコルセッションの関連付けという二つの機能を提供します。

注意. アプリケーションセッションは、SIP 特有のものではありません。論理的には「javax.servlet.ApplicationSession」として実装することも可能ですが、実用的な理由から、SIP Servlet API のこの版では、アプリケーションセッションを「javax.servlet.sip.SipApplicationSession」として定義します。我々の希望は、Servlet API の将来の版でアプリケーションセッションの概念が採用され、「SipApplicationSession」が削除され、「ApplicationSession」を継承するように再構成されることです。

10.1.1. プロトコルセッションイテレータ

「Iterator」オブジェクトにより実現される「SipApplicationSession」の二つメソッドにより、含まれるプロトコルセッションを求めることができます。

- **getSessions()** は、すべての有効な子供のプロトコルセッションオブジェクトを繰り返し取り出します。
- **getSessions(String protocol)** は、指定したプロトコルのすべての有効な子供のプロトコルセッションオブジェクトを繰り返し取り出します。例えば、“SIP”ですべての「SipSession」が得られ、“HTTP”ですべての「HttpSession」が得られます。

10.1.2. SipApplicationSession のライフタイム

コンテナはセッションデータを管理し、セッションオブジェクトの回収を可能にする機構を提供します。アプリケーションセッションが終了すると、そのセッションに含まれるプロトコルセッションはすべて破棄されます。アプリケーションセッションはタイムアウトするか、サーブレットが明示的に「invalidate」メソッドを呼び出すことで、破棄されます。性能の観点からは、アプリケーションができるだけ早く明示的にセッションオブジェクトを無効にすることが推奨されています。

注意. アプリケーションが、「invalidate」できるという理由で、アプリケーションセッションの生成が強制させられるのは不適切です。「getApplicationSession(boolean create)」メソッドの引数に「false」を指定することで、セッションオブジェクトの強制的な生成を避けることができます。

タイマは、アプリケーションセッションが明示的に無効にするか否かに関わらず、アプリケーションセッションが破棄されることを保証します。サーブレットは、アプリケーションセッションのタイムアウト通知(「SipApplicationSessionListener」インタフェース)に関する登録を行うことができます。

「sessionExpired」のコールバックメソッドの中では、アプリケーションが、「setExpires」の呼び出しによりアプリケーションセッションのライフタイムの拡張を要求することができます。「setExpires」では、タイムアウトした「SipApplicationSession」上で、引数としてセッションが再び終了する時間を指定することになります。コンテナは、セッションのライフタイムの拡張を許可するか、あるいは修正済のタイムアウト値でそれを許すか、または拒絶することになります。タイムアウト値を受け付ける際には、コンテナがアプリケーションセッションのライフタイム及びタイムアウトに関して、自身のポリシーを適用することができます。例えば、アプリケーションセッションに対して、ライフタイムの合計の最大値を強制するようなポリシーを選択することができます。

アプリケーションのライフタイムが、何らか「外部の」イベント(サーブレットコンテナに未知のイベント)に依存するような場合に、アプリケーションが非現実的な長いタイムアウト値を設定させないようにするため等に、セッションのライフタイムを拡張する機能が必要となります。

10.2. SipSession

「SipSession」オブジェクトは、確立しているダイアログとして、あるいはダイアログが実際に確立される前の段階で、ポイントツーポイントの SIP の関係を表わします。「SipSession」は「getSession」メソッドを呼び出すことにより「SipServletMessage」から得ることができます。

10.2.1. SIP ダイアログとの関係

「SipSession」は、実際の SIP のダイアログ(RFC 3261 に定義)か、あるいは仮のダイアログ(「Pseudo dialog」)を表わします。SIP のダイアログは、初期(early)状態か、確認状態(confirmed)か、終了(terminated)状態のいずれかの状態にあります。仮のダイアログの概念は、ダイアログが RFC 3261 の意味で確立される前、更にはそれが「2xx」以外の最終レスポンスを受信し初期の状態から移行した後に、間違いのない明確な意味を表現するためにダイアログの定義を拡張したものです。「SipSession」インタフェースは、仮のダイアログに対して、この概念を具体化したものとなっています。なお、このような拡張のために、いくつかの「SipSession」インスタンスは、SIP のダイアログとは一致していません。

SIP のダイアログの状態マシンを図 6 に示します。

FIGURE 6. The SIP dialog state machine.

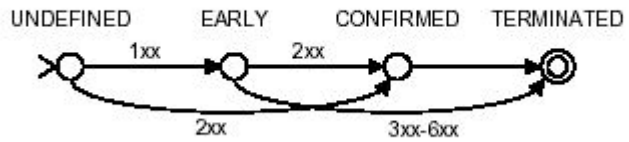
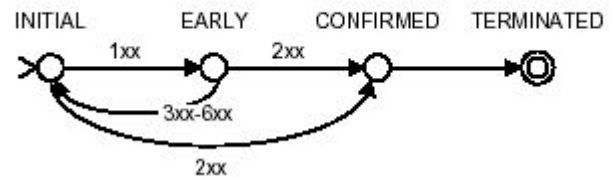


図 6 の「未定義(undefined)」の状態は、実際の状態ではありません。「INVITE」の結果として生成されるダイアログに対して、「To」タグを含む「1xx」や「2xx」の受信の際に、このような状態が現れます。

FIGURE 7. The SipSession state machine.



「SipSession」の状態マシンを図7に示します。SIP ダイアログの状態マシンとは以下の点で異なっています。

- 初期(「*initial*」)状態も有効な状態です。初期状態では、複数のリクエストを発行することができます。また 10.2.2.2 章で定義する規則により、ローカルな「CSeq」のようなダイアログ状態の更新の方法を指定します。

- 初期状態で受信した「2xx」以外の最終レスポンスは、「SipSession」を終了させず、トランザクションが初期状態に遷移します。初期状態では、追加のリクエストを生成することができます。

第二あるいはそれ以降のレスポンスが、一つのリクエストに対する複数のダイアログを確立させる場合、標準の SIP ルールが「SipSessions」にも適用されます。例えば、もし最初の「INVITE」リクエストに対して、二つの「200」レスポンスを受信した場合、コンテナは第二の「200」の受信に対して、第二の「SipSessions」を生成します。この第二の「SipSessions」は、10.2.3.2 章に記述されるように、「INVITE」が生成されたことに関連しています。その後、両方の「SipSessions」は、確認状態にあるダイアログを表現することになります。

初期状態は、同じ「Call-ID」、「From」(タグを含む)、「To」(タグを含まない)を持ち、同じ「CSeq」空間で生成される複数のリクエストを、UAC が生成することを可能にするために導入されています。例えば、以下のような状況で役に立ちます。

- UAC がダイアログの外部で開始されたリクエスト用の「3xx」を受信し、「3xx」内の「Contact」アドレスを使ってリトライを実行する場合、新しいリクエストのために同じ「To」/「From」/「Call-ID」を再利用することが推奨されます[RFC 3261,8.1.3.4 章]。
- UAC が“失敗でない”「4xx」レスポンスを受信する場合、リクエストをリトライすることができます。例えば「401」、「407」、「413」、「415」、「416」、「420」[RFC 3261,8.1.3.5 章]があります。
- UAC から同じ登録サーバに送信される「REGISTER」リクエストはすべて、同じ「Call-ID」ヘッダフィールドの値を使用する必要があります。また、各リクエストを送信する毎に、1 だけ「CSeq」をインクリメントすることも必要となります[RFC 3261,章 10.2]。
- セッションタイマの拡張を使用する UAC が、初期の「INVITE」に対する「422」レスポンスを受信する場合、同じ「Call-ID」で、しかもより大きい「Min-SE」値を用いて、リトライします[タイマ]。

これらの例はすべて、ダイアログを確立することなく類似のリクエストを作成する必要があることを示しています。また、他にもダイアログ外のリクエストを「Call-ID」により関連づけて、「CSeq」ヘッダフィールドで順序づけることが必要となるケースが存在するでしょう。

ここで示された「仮のダイアログ」は、UAC アプリケーションのために定義されているのに注意して下さい。コンテナは、受信するリクエストを後続リクエストとして処理し、これらのリクエストが実際に作成された SIP のダイアログに属す時にだけ、既にあるセッションにルーティングします。例えば、コンテナが同じ「Call-ID」/「CSeq」/「From」(タグを含む)を持った別の「INVITE」に「3xx」レスポンスを送信した後、コンテナが受信する「INVITE」を後続リクエストとして扱うことは、期待されていません。

10.2.2. SipSession 内のダイアログ状態のメンテナンス

UA は、ダイアログの中で後続リクエストを作成するために「SipSession」のメソッド「createRequest」を使用します。これは、アプリケーションが UA の役割を行う場合、コンテナが「SipSession」に SIP ダイアログの状態を関連させる必要があることを意味しています。ダイアログの状態は RFC 3261 に定義されており、次のデータから構成されています：ローカル/遠隔の URI、ローカル/遠隔のタグ、ローカル/遠隔のシーケンス番号、ルートセット、遠隔のターゲット URI、安全性のフラグ。

10.2.2.1. ダイアログの中の場合

UA の役割を持つアプリケーションのために、「SipSession」は RFC 3261 に従ってダイアログの状態を追跡できる必要があります。

プロキシは、自分で新しいリクエストを生成することができません。そこで、「SipSession.createRequest」は「IllegalStateException」を発生することになります。しかし、メソッドの「getCallId」、「getLocalParty」及び「getRemoteParty」は実装する必要があります。「getLocalParty」は、発信者のアドレス(そのダイアログの初期リクエストの「From」ヘッダの値)を返します。また、「getRemoteParty」は、着信者のアドレスを返します。これは「Call-ID」/「From」/「To」が、任意のアプリケーション状態を持つ「プロキシ」「SipSession」と関係付けされる必要があることを意味しています。

10.2.2.2. INITIAL SipSession 状態の場合

UAC あるいは UAS が初期(early)状態から図 7 の初期（initial）状態に遷移する場合、「SipSession」で保持しているダイアログ状態も以下のように更新されます（これらのダイアログ状態コンポーネントの定義については、RFC3261 を参照して下さい）。

- リモートターゲットがリモート URI にリセットされます。
- リモートタグのコンポーネントが消去されます。
- リモートシーケンス番号が消去されます（例えば、-1 を設定します）。
- ルートセットが消去されます。
- 「安全性」のフラグが 「false」 に設定されます。

これらの規則の結果として、同じ「SipSession」の中で生成されたリクエストは、以下の特徴を持つことになります。

- すべて同じ「Call-ID」を持ちます。
- すべてが同じ「From」ヘッダフィールドの（同じ空でないタグも含めた）値を持ちます。
- 同じ「SipSession」の中で生成されたリクエストの「CSeq」は、「SipSession」の状態に関わらず、各リクエストが生成される毎に単調に 1 だけ増加します。
- 初期状態の同じ「SipSession」で生成されたリクエストはすべて、同じ「To」ヘッダフィールドの値（タグパラメータを持たない）を持っています。
- 初期状態の「SipSession」オブジェクトはルートセットを持たず、リモートシーケンス番号は未定義です。また、「安全性」フラグは 「false」 であり、リモートターゲット URI は「To」ヘッダフィールドの値の URI となります。

10.2.3. SipSession の生成

ダイアログと「SipSession」の間の大きな違いは、SIP リクエストがダイアログ(例えば、「OPTIONS」及び「REGISTER」)の外部で存在する可能性があり、SIP Servlet API ではメッセージがすべて「SipSession」に属すという点にあります。

SIP ダイアログは、ダイアログをセットアップできる方法を備えたリクエストに対して、失敗以外のレスポンスが受信した結果として作成されます。基本の SIP 仕様では、一つの方法(すなわち「INVITE」)を定義します。ダイアログの処

理は、プロキシがフォークすることにより、複雑になります。これは、複数の UAS で単一のリクエスト(例えば INVITE)を受け付けることができることを意味し、これにより複数のダイアログを作成させることになります。このような場合、UAC の SIP サーブレットは、複数の「SipSession」を扱うことになります。

「SipSession」と SIP ダイアログの間の関係は、下のようにまとめることができます。

初期リクエストから一つのダイアログがセットアップされた場合、初期の「SipServletRequest」の「SipSession」はそのダイアログに対応します。一つ以上のダイアログが確立される時、第一のダイアログは既存の「SipSession」に対応し、10.2.3.2 章で定義された方法に従って、後に続くダイアログ毎に新しい「SipSession」が作成されます。

「SipSession」が生成される場合の規則は、以下のようになります。

- 「SipFactory.createRequest」により作成された、ローカルに生成される初期リクエストは新しい「SipSession」に属します。
- 受信する初期リクエストは新しい「SipSession」に属します。
- ダイアログを確立しない初期リクエストに対するレスポンスは、リクエストの元の「SipSession」に属します。
- SIP ダイアログ(例えば、初期の「INVITE リクエスト」への「2xx」)を確立する最初のメッセージは、他のメッセージがダイアログに属するように、オリジナルの「SipSession」と関係付けられます。
- ダイアログを確立する後続メッセージは、オリジナルの「SipSession」を基に生成された新しい「SipSession」に関連付けられます。これに関しては、10.2.3.2 章を参照して下さい。

プロキシするアプリケーションが、新しい「SipSession」に関係するレスポンス(「INVITE」リクエストに対する二番目の「2xx」レスポンス)を受信する場合、そのレスポンスの「getSession」メソッドは新しく派生した「SipSession」を返します。オリジナルの「SipSession」は、オリジナルのリクエストオブジェクトにより引き続き利用することができます(「Proxy」インタフェースの「getOriginalRequest」メソッドにより、利用できます)。

10.2.3.1. ダイアログを生成する拡張

基本の SIP 仕様に対する拡張として、ダイアログを確立することができる追加の方法を定義します。SIP のイベントフレームワークは、そのような拡張の一つとなります[RFC 3265]。イベントフレームワークでは、後続の「NOTIFY」リクエストが追加のダイアログを生成するのと一緒に、初期の「SUBSCRIBE」に対する「2xx」レスポンスがダイアログを確立する可能性があります。実際、「SUBSCRIBE」に対する「2xx」レスポンスの前に、一つ以上の「NOTIFY」リクエストが発信者とプロキシに到着します。その「NOTIFY」では、ダイアログを確立します。

ダイアログを確立することができる拡張機能を“理解する”SIP サーブレットコンテナは、第二のダイアログ、及びダイアログを確立させることができる単一のリクエストを送った結果として作成される後続のダイアログに対応して、新しい派生した「SipSession」オブジェクトを作成できることが必要となります。

10.2.3.2. 派生した SipSession

派生した「SipSession」は、オリジナルのリクエストに関連した「SipSession」のコピーとなっています。このコピーは、新しいダイアログを作成するメッセージがアプリケーションに渡される時に作成されます。新しい「SipSession」は着信

者(後続リクエストでの「To」ヘッダの値)のアドレスのタグパラメータの値、及びルートセットのアドレスに対する値だけが異なります。これらの値は、SIP 仕様によって定義されるダイアログを確立するメッセージから取り出すことができます。クローンを作られた「SipSession」の属性の集合は、オリジナルのものと同じです。特に、値はクローンとはなりません。

第二および後続の「2xx」レスポンス(あるいは「To」タグ付の「1xx」レスポンス)に対応する、新しい「SipSessions」は、「SipServletResponse」の「getSession」メソッドにより利用することができます。リクエストの「オリジナルのSipSession」は、オリジナルのリクエストオブジェクトによって引き続き利用することもできます。

10.2.4. SipSession のライフタイム

コンテナには、ダイアログが終了する際に、ダイアログ状態を追跡することや、「SipSession」を無効にすることが要求されていません。親のアプリケーションセッションがタイムアウトするか、あるいはアプリケーションにより明示的に「invalidate」メソッドを利用することで、「SipSession」が無効にされる場合に、「SipSession」インスタンスは破棄され、コンテナによってコンテナメモリから除去されます。

「SipSession」は、親の「SipApplicationSession」と独立のタイムアウト値を持ってはいません。上で述べたように、親のセッションがタイムアウトする時に、子供のプロトコルセッションはすべてタイムアウトすることになります。

無効になった「SipSession」に対して、データの検索や保存を行おうとすると、コンテナによって「Illegal-StateException」が発生することになります。「createRequest」を呼び出す場合も同様です。

親のアプリケーションセッションがタイムアウトした、または「SipSession」が明示的に無効にされたという理由で、「SipSession」が終了する場合、コンテナはその「SipSession」についての情報をメモリからすべて除去することができます。この場合、対応するダイアログに属する後続リクエストが受信されると、コンテナは次のいずれかの方法で処理を行います。

1. リクエストを拒否します。
2. アプリケーションの呼び出しを行うことなく、リクエストをルーティングします。
3. アプリケーションパスが空きの場合、初期リクエストとして扱います。つまり、既存の規則でのマッチングを行い、新しいアプリケーションパスを確立するプロセスに従って、アプリケーションの呼び出しを行います。

ダイアログ状態をすべて除去した後続リクエストが、既にロードされた「Route」を持った新しい初期リクエストではなく、本当の後続リクエストかどうかを判断できないような実装となる可能性もあります。この場合、コンテナは新しい初期リクエストとして扱い、ルールのマッチングを実行するしか選択肢がありません。

同じアプリケーションパス上の異なるアプリケーションインスタンスが、異なるライフタイムを持つ可能性があります。コンテナは、部分的に無効にされたアプリケーションパスに対応するダイアログ上で受けた後続リクエストに対して、上記のオプション 1 あるいは 2 を適用することができます。ここで、部分的に無効にされたアプリケーションパスとは、すべてではなく少なくとも一つのアプリケーションインスタンスが無効にされているものを表わします。

部分的に無効にされたアプリケーションパスに属すリクエストを受信する場合、コンテナは最初に第一の無効にされたインスタンスまでのすべてのアプリケーションを起動します。その後、オプション 1 の場合には、リクエストは拒否され、オプション 2 の場合には、次の有効なアプリケーションインスタンスが起動されます。あるいは、アプリケーションパスが使い尽くされた場合、リクエストは 8.2.8 章のルールに従ってプロキシされます。

10.2.5. RequestDispatcher インタフェース

アプリケーションの様々な部分を処理する複数のサーブレットからアプリケーションが構成されることを可能にする、構造化メカニズムが必要となります。HTTP の場合には、サーブレットが、別のサーブレットへリクエストを転送するか、あるいはそれ自身のレスポンスに別のサーブレットの出力を含めるために、「RequestDispatcher」を使用することができます。

SIP サーブレットは同じ方法で「RequestDispatcher」インタフェースを使用することができます。同じアプリケーションで、別のサーブレットへのリクエストあるいはレスポンスを渡すためのメカニズムとして「forward」メソッドを使用します。これは“間接的なメソッドコール”と考えることができます。（SIP の場合には、レスポンスとリクエストに同じように適用されるため、「RequestDispatcher」という名前は適切ではありません。）なお、「include」メソッドは、SIP サーブレットでは意味を持っていません。

10.2.6. SipSession ハンドラ

「RequestDispatcher」インタフェースが、SIP サーブレットに対する一つの構造化メカニズムを提供しています。加えて、「SipSession」インタフェースが、ハンドラの概念を持っています。ハンドラはサーブレットへのリファレンスであり、コールバックインタフェースとなっています。コンテナは「SipSession」に関連するすべての受信イベントを、このコールバックインタフェースを通してアプリケーションにディスパッチします。

サーブレットアプリケーションが新しい初期リクエストを生成すると、「SipSession」ができます。この時点で、コンテナが、新しく作成された「SipSession」のためのハンドラとなるデフォルトサーブレットを、アプリケーションに割り当てます。「SipSession」ハンドラは、新しく受信するリクエストと同様に、すべてのリクエストのためのレスポンスを処理するために起動されます。アプリケーションのデフォルトサーブレットは、SIP 配備記述子の中にリストされる最初のサーブレットとして定義されます。

「SipSession」が受信する初期リクエストの処理の一部として生成される場合、呼び出されたサーブレットが「SipSession」の初期のハンドラとなります。

「SipSession」の生成のされ方に関わらず、アプリケーションは、同じアプリケーション内の異なるサーブレットを特定の「SipSession」に対するハンドラとして設定することができます。これは「SipSession」の以下のメソッドを呼び出すことにより行われます。

void setHandler(String name);

引数は、「ServletContext.getNamedDispatcher」の呼び出しのものと同じサーブレット名です。サーブレット名は、一般的に配備記述子を使ってサーブレットと関連付けられます。

10.2.7. SipSession への属性のバインド

サーブレットは、名前によって「SipSession」にオブジェクトの属性をバインドできます。セッションにバインドされたオブジェクトは、同じ「ServletContext」に属し、同じセッションの一部として識別されるリクエストを処理するサーブレットのどれからでも利用できます。

セッションに入れられる、あるいはセッションから除かれる時に、一部のオブジェクトでは通知を要求する可能性があります。この情報は、オブジェクトに「SipSessionBindingListener」インタフェースを実装することにより得ることができます。このインタフェースは、オブジェクトがセッションにバインドされた時、あるいはバインドを解かれた時に、通知を行う、以下のメソッドを定義しています。

- **valueBound**
- **valueUnbound**

オブジェクトが、「SipSession」インタフェースの「getAttribute」メソッドによって利用可能になる前に、「valueBound」メソッドが呼び出される必要があります。一方、オブジェクトが、「SipSession」インタフェースの「getAttribute」メソッドによって利用できなくなった後には、「valueUnbound」メソッドが呼び出される必要があります。

10.3. 最後にアクセスされた時間

「SipApplicationSession」インタフェースと「SipSession」インタフェースの「getLastAccessedTime」メソッドは、現在のリクエストの前にセッションがアクセスされた時間を、サーブレットが認識するために利用することができます。セッションは、セッションの一部であるメッセージがサーブレットコンテナによって処理された時に、アクセスされたと考えられます。「SipApplicationSession」の最後のアクセスされた時間は、常にそれが含んでいるプロトコルセッションの中のどれから最後にアクセスされた時間になります。

10.4. 重要なセッションのセマンティクス

10.4.1. スレッドの問題点

リクエストをスレッドとして実行する複数のサーブレットは、同時に一つのセッションオブジェクトにアクセスする可能性があります。開発者は、セッションリソースへのアクセスを適切に同期させることに対する責任を持つ必要があります。

10.4.2. 分散環境

分散実行可能なアプリケーション内では、セッションの一部であるすべてのリクエストは、ある時点においても一つの VM 上でのみ処理することができます。更に、コンテナは、「setAttribute」または「putValue」のメソッドを適切に使用することにより、「SipSession」クラスのインスタンスに含まれるオブジェクトを、すべて扱うことができる必要があります。

- コンテナは、「Serializable」インタフェースを実装したオブジェクトを受入れられる必要があります。

- コンテナは、Enterprise JavaBeans やトランザクションへの参照を利用して、「SipSession」の中で他のオブジェクトに対するストレージ機能をサポートすることができます。
- セッションのマイグレーションは、コンテナに特有の機能により処理されることになります。

オブジェクトが「Serializable」でないセッションに入れられる場合、あるいは、そのための特定のサポートが利用可能でない場合、サーブレットコンテナは「IllegalArgumentException」を発生させても構いません。コンテナがオブジェクトを格納するセッションのマイグレーションに対して、必要なメカニズムがサポートできない場合には、オブジェクトに対して「IllegalArgumentException」を発生させることが必要となります。

これらの制限は、分散されていないコンテナ内ではこれ以上、並列処理による問題が発生しないことを開発者が保証されていることを、意味します。

コンテナ供給者は、分散システム内のアクティブなノードから、システム内の異なるノードに対して、セッションオブジェクトとその内容を移動させる能力を持つことにより、ロードバランシングとフェイルオーバのような、スケーラビリティとサービス品質 (QoS) を保証することができます。

分散コンテナが、サービス品質を保証するために、セッションを永続化するか移動する場合、「SipSession」とそれらの属性をシリアライズするために、JVM 純正のシリアライズ機構を使う必要はありません。開発者は、セッションの属性に対して、コンテナが「readObject()」及び「writeObject()」メソッドを呼び出すことは保証されてはいませんが、「Serializable」な属性が保存されることは保証されています。

コンテナは、セッションのマイグレーションの間に、「SipSessionActivationListener」を実装するセッションの属性を通知することが必要です。また、セッションのシリアライズの前に不活性化するリスナと、セッションのデシリアライズの後に活性化するリスナを、通知することも必要です。

分散アプリケーションの開発者は、コンテナが一つ以上の Java VM 上で動作する可能性があることを意識する必要があります。開発者はアプリケーションの状態を保存する際に、静的な変数やインスタンス変数に依存することはできません。アプリケーションの状態は、EJB あるいはデータベースを利用して保存することが必要です。

11. リクエストからサーブレットへのマッピング

Servlet エンジンは、特定のサーブレットが起動される条件を指定するマッピング集合を管理します。2 章では、初期リクエストに対するルールのマッチングのされ方とサーブレットの起動され方を説明しています。本章では、ルール言語を定義します。

ルールは、条件集合から構成されます。各々の条件では受信するリクエストの特性をテストします。ここで指定されるルール言語は、SIP リクエストに対するオブジェクトモデルとして定義されています。

11.1. ルールのトリガ

2 章で議論したように、確立された SIP ダイアログに属さないリクエストは、すべての配備済みアプリケーションのルール集合をベースに、コンテナによりルーティングされます。これは、コンテナがアプリケーションの組合せを実行している場合には、アプリケーションから受信したリクエストと同様に、ネットワークから受信されたリクエストについても同じように扱われます。

SIP Servlet API の現在の版では、ルールは SIP リクエストに対して単純に適用が判断されます。SIP リクエストを与えると、ルールが「true」か「false」かの評価を行います。一致したルールは、コンテナによってトリガを与えられ、対応するサーブレットがリクエストを処理するために起動されます。

与えられた初期リクエストに対して一致したルールの中で、どのルールを起動するのかの選択を行うのは、コンテナのポリシーの問題として任されています。ただし、以下の二つの条件に関しては、すべてのコンテナよって監視されることが必要となります。

1. 対応する SIP サーブレットが呼び出される時には、起動されるルールは条件に適合していることが必要です。
2. 同じアプリケーションに属す二つ以上のルールが適合した場合には、配備記述子にリストされた順番で、起動されることが必要です。

最初の条件は、アプリケーションの組合せとは関係なく、初期リクエストを処理するためにサーブレットが呼び出される時には、そのリクエストに適合して「true」となるルールが存在していることを、アプリケーション開発者に保証します。このような意味で、ルールはサーブレットに対する保護者となり、ルールはアプリケーションとコンテナとの間の契約の一部となります。

二番目の条件で与えられる、ルールのアプリケーション内での優先度付けは、コンテナ間でのアプリケーションのポータビリティを保証するために必要となります。

11.2. SIP リクエストのオブジェクトモデル

ルール言語の目的のために、SIP リクエストに対するオブジェクトモデルが定義されます。このモデルはその属性に従ってオブジェクトタイプを定義します。オブジェクトの属性は、未定義オブジェクトであるか、あるいは string や

integer のようなプリミティブのいずれかとなります。ここで使われるモデルは、API 自体のインタフェース定義に密接に関連しています。

SIP servlet API ルール言語のモデルは、以下のオブジェクトタイプと、関連する属性から構成されます。

リクエストオブジェクト (request object): リクエスト自体を表します。ルール適合のためのエントリーポイントです。これは以下の属性を持ちます、

- **method:** リクエストのメソッド、文字列(string)
- **URI:** リクエスト URI、例えば、SIP URI か TelURL
- **from:** 「From」ヘッダの値を示すアドレス
- **to:** 「To」ヘッダの値を示すアドレス

URI:

- **scheme:** URI スキーム

SipURI (extends URI):

- **scheme:** 文字列リテラル (a literal string) – “sip” や “sips”
- **user:** SIP/SIPS URI のユーザ部分
- **host:** SIP/SIPS URI のホスト部分。 これはドメインネームやIPアドレスとなります。
- **port:** URI のポート番号 (10進)。省略された場合、デフォルト値が使用されます。
  (5060 for UDP and TCP, 5061 for TLS)
- **tel:** “user” パラメータが “phone”でなければ、この変数は未定義となります。 それ以外の場合は、この値は電話番号となります。セパレータは省略されて SIP/SIPS URI のユーザ部分に電話番号が含まれます。電話番号は必ず大文字小文字が区別されて、一致処理が行われます (電話番号が ‘A’, ‘B’, ‘C’, ‘D’の文字を含んでいても構いません)。
- **param.*name*:** SIP/SIPS URI の名前の付いたパラメータの値。「*Name*」はSIP/SIPS URI のパラメータ名
  として許されるものである必要があります。

TelURL (extends URI):

- **scheme:** 文字列リテラルの “tel”
- **tel:** tel URL セパレータを除いた加入者名
- **param.*name*:** tel URL の名前の付いたパラメータの値。「*name*」はtel URLのパラメータ名として許される
  ものである必要があります。

Address:

- **uri :** URI オブジェクト。上の URI, SipURI, TelURL タイプ
- **display-name:** 「From」か「To」ヘッダのディスプレイネーム

特に断りのない限り、変数の一致処理を行う場合に、以下にリストされるオペレータはデフォルトで考慮され、変数の値は大文字小文字が区別されることになります。

11.3. 条件

条件はオペレータか論理的コネクタです。以下のオペレータが定義されています。

equal: 変数の値をリテラル値と比較し、変数が定義されており、その値がリテラルと一致すれば、「true」と評価されます。それ以外、結果は「false」となります。

exists: 変数の名前を引数とし、変数が定義されていれば、「true」と評価されます。それ以外は「false」となります。

contains: 第一引数で指定された変数の値が第二引数で指定した文字列リテラルを含んでいれば、「true」と評価されます。

subdomain-of: 与えられた変数がドメインネーム(SIP/SIPS URI *host*)か、電話加入者(SIP か Tel URLs の電話の属性)、リテラル値のいずれかであるなら、変数がリテラルの値によって与えられたドメインのサブドメインを表している場合には、このオペレータは「true」を返します。ドメインネームはサブドメインを構成する DNS の定義に従って一致されます。例えば、ドメインネームの"example.com"と“research.example.com“は、両方とも"example.com"のサブドメインです。IP アドレスもこのオペレータの引数として与えることができますが、正確に同じものだけしか一致しません。「tel」変数の場合には、第一引数が電話番号を示しており、第二引数で与えたリテラルの値と一致した場合に、「true」と評価されます、例えば、電話番号"1 212 555 1212"は"1212555".のサブドメインと考えることができます。

更に、以下の論理コネクタがサポートされています。

and: 複数の条件が含まれ、すべての条件が「true」と評価された場合にのみ「true」と評価されます。

or: 複数の条件が含まれ、少なくとも一つの条件が「true」と評価された場合に「true」と評価されます。

not: 引数の条件の値を反対にします。

「*equal*」と「*contains*」オペレータは、比較を行う時に、大文字と小文字を区別しないこともオプションとして可能です。ただし、デフォルトは大文字小文字を区別します。

11.4. XML のエンコード

変数はリクエストを表現するオブジェクト構造のパスとして表示されます。パス上の個々のオブジェクトは“request.uri.user”のように、“.”によって分離されます。

以下の XML 定義は、上で定義されたルール言語がどのように配備記述子の「servlet-mapping」要素としてエンコードされるかを示しています（15 章に配備記述子のＤＴＤを示します）。

```
<!ENTITY % condition "and | or | not | equal | contains | exists | subdomain-of">
<!ELEMENT servlet-mapping (servlet-name, pattern)>
<!ELEMENT servlet-name (#PCDATA)>
<!ELEMENT pattern (%condition;)>
<!ELEMENT and (%condition;)+>
```

```
<!ELEMENT or (%condition;)+>
<!ELEMENT not (%condition;)>
<!ELEMENT equal (var, value)>
<!ELEMENT contains (var, value)>
<!ELEMENT exists (var)>
<!ELEMENT subdomain-of (var, value)>
<!ELEMENT var (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ATTLIST equal ignore-case (true|false) "false">
<!ATTLIST contains ignore-case (true|false) "false">
```

11.5. 例

表1は、以下の SIP リクエストに対して定義されているすべての変数の値を示しています。

INVITE sip:watson@boston.bell-tel.com SIP/2.0
From: A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc
To: T. Watson <sip:watson@bell-tel.com>
...

TABLE 1. Variables and Corresponding Values

variable	value
request.method	"INVITE"
request.uri	"sip:watson@boston.bell-tel.com"
request.uri.scheme	"sip"
request.uri.user	"watson"
request.uri.host	"boston.bell-tel.com"
request.uri.port	5060
request.uri.tel	*undefined*
request.from	"A. Bell <sip:a.g.bell@bell-tel.com>;tag=3pcc"
request.from.uri	"sip:a.g.bell@bell-tel.com"
request.from.uri.scheme	"sip"
request.from.uri.user	"a.g.bell"
request.from.uri.host	"bell-tel.com"
request.from.uri.port	5060
request.from.display-name	"A. Bell"
request.to	"T. Watson <sip:watson@bell-tel.com>"
request.to.uri	"sip:watson@bell-tel.com"
request.to.uri.scheme	"sip"
request.to.uri.user	"watson"
request.to.uri.host	"bell-tel.com"
request.to.uri.port	5060
request.to.display-name	"T. Watson"

以下のルールの例は、「false」と評価されます。

```
<pattern>
    <and>
        <equal>
          <var>request.method</var>
          <value>INVITE</value>
        </equal>
        <not>
          <contains ignore-case="true">
            <var>request.from.display-name</var>
            <value>bell</value>
          </contains>
        </not>
        <subdomain-of>
          <var>request.from.uri.host</var>
          <value>bell-tel.com</value>
        </subdomain-of>
    </and>
</pattern>
```

この例では、第一と第三の節と“and条件”は満足されていますが、第二の節が 「false」のためルールの適用は失敗しています（「From」のディスプレイネームが文字列 “bell” を含んでいます）。

12. SIP サーブレット アプリケーション

SIP サーブレットアプリケーションは、サーブレット、クラスファイル、及び SIP サーバ上の完全なアプリケーションを構成するその他のリソースを集めたものです。SIP アプリケーションはバンドル販売され、複数ベンダから提供される複数のコンテナ上で稼動することができます。

デフォルトでは、SIP アプリケーションのインスタンスは、どの時点においても、一つの VM 上で動作することが必要です。アプリケーションがその配備記述子で、“distributable”（分散可能）と定義していれば、この動作を無効にすることができます。分散可能と定義されるアプリケーションは、通常のサーブレットアプリケーションが要求されるより、もっと限定的な規定に従うことが必要となります。この規定は、この仕様書を通じて提示されます。

12.1. HTTP サーブレットアプリケーションとの関係

SIP サーブレットアプリケーションが使用するディレクトリ構成は、HTTP サーブレットで定義したもの良く似ています、具体的には、SIP サーブレットアプリケーションの配備記述子、クラスファイル、ライブラリ等は、「WEB-INF/」ディレクトリの下に存在します。融合した HTTP と SIP サーブレットアプリケーションを一つのアーカイブファイルにパッケージ化し、SIP アプリケーションの他の部分を除いて、Web サーバで公開されるユニットとして配布することができます。

12.2. ServletContext との関係

サーブレットコンテナは、サーブレットアプリケーションと「ServletContext」との間の一対一の対応を管理することが必要です。「ServletContext」オブジェクトは、サーブレットにアプリケーションとして扱わせる機能を提供します。これは、SIP と HTTP の融合アプリケーションについても同じで、アプリケーション中のすべてのサーブレットは、同じ「ServletContext」インスタンスを扱うことができます。

12.3. SIP アプリケーションの要素

SIP アプリケーションは以下の項目から構成されます。

- サーブレット
- ユーティリティクラス
- 静的なリソースとコンテンツ（テキスト、スピーチ・アナウンスメント、コンフィグレーションファイル、等）
- すべての上記の要素と関係した説明的なメタ情報

12.4. 配備階層

この仕様では、オープンなファイルシステム、アーカイブファイル、あるいはその他のファイル形式で存在可能な、配備とパッケージング目的で使われる階層構造を定義します。サーブレットコンテナが、この構造をランタイムの表現としてサポートすることを推奨しています。ただし、要求される機能ではありません。

12.5. ディレクトリ構造

SIP アプリケーションは、ディレクトリの構造化された階層の中に存在しています。上で述べたように、HTTP サーブレットのアーカイブファイルのフォーマットとの互換性のため、この階層のルートは、Web サーバから公開されることを意図したファイル用のドキュメントのルートとして提供されます。この機能は、実際には HTTP と SIP サーブレットを組合せたアプリケーションの場合に使われます。

アプリケーションの階層に「WEB-INF」と名付けられた特殊なディレクトリがあります。このディレクトリは、アプリケーションのドキュメントルート内にないアプリケーションと関係するものをすべて含んでいます。SIP だけのアプリケーションでは、通常すべてが「WEB-INF」の下に存在します。「WEB-INF」ノードはアプリケーションの公開されたドキュメントの階層の一部ではありません。「WEB-INF」ディレクトリに含まれないファイルは、コンテナによってクライアントに直接に提供されません。一方、「ServletContext」に対して「getResource()」と「getResourceAsStream()」メソッドを使用すると、サーブレットのコードから「WEB-INF」ディレクトリの内容を見ることができます。アプリケーション特有のコンフィグレーション情報は、開発者がサーブレットのコードからアクセスできる必要がありますが、web クライアントに公開したくない場合は、このディレクトリの下に置くことができます。リクエストはリソースのマッピングに大文字小文字を区別して一致処理を行うため、例えば「/WEB-INF/foo」と「/WEb-iNf/foo」に対するクライアントのリクエストは、「WEB-INF」の下に位置する web アプリケーションのコンテンツが返されてはいけません。

「WEB-INF」ディレクトリの内容は、以下のものから構成されます。

- /WEB-INF/sip.xml　配備記述子。
- /WEB-INF/classes/*　サーブレットとユーティリティクラスのためのディレクトリ。このディレクトリの中のクラスはアプリケーションのクラスローダから使用可能です。
- /WEB-INF/lib/*.jar　Java ARchive ファイル(jar)の領域。このファイルは、サーブレット、beans、web アプリケーションで利用する他のユーティリティクラスを含みます。Web アプリケーションのクラスローダは、これらのどのアーカイブファイルからでもクラスをロードすることができます。

アプリケーションのクラスローダは、最初に「WEB-INF/」の classe ディレクトリからクラスをロードし、次に「WEB-INF/」の lib ディレクトリのライブラリの JAR ファイルをロードする必要があります。

12.5.1. 例、アプリケーションのディレクトリ構成

以下は、SIP サーブレットアプリケーションのサンプルで利用するすべてのファイルの一覧です。

/WEB-INF/sip.xml
/WEB-INF/wakeup.wav
/WEB-INF/lib/foo.jar
/WEB-INF/classes/WakeupServlet.class

以下は同じサンプルですが、HTTP サーブレットのコンポーネントが含まれています、

/wakeup.html
/register.html

/WEB-INF/sip.xml
/WEB-INF/web.xml
/WEB-INF/wakeup.wav
/WEB-INF/lib/foo.jar
/WEB-INF/classes/RegisterWakeupCall.class
/WEB-INF/classes/WakeupServlet.class

後者の場合、SIP と HTTP とは別々に配備記述子 (sip.xml と web.xml) があります。「WEB-INF/」の下にないリソースは、融合したアプリケーションの HTTP 部分で提供されるコンテンツの一部分となっています。

12.6. サーブレットアプリケーションのアーカイブファイル

サーブレットアプリケーションはパッケージ化され、標準の Java アーカイブツールを使用して、Servlet ARchive フォーマット(sar)ファイルに入ることができます。例えば、click-to-dial アプリケーションは、click2dial.sar と呼ばれるアーカイブファイルで配布されます。

このような形式でパッケージにされた時、Java アーカイブツールにとって有用な情報を含む、META-INF ディレクトリがあります。その内容は「ServletContext」上の「getResource()」と「getResourceAsStream()」の呼び出しを行うことにより、サーブレットがアクセスできますが、コンテナは直接 Web クライアントのリクエストに対するレスポンスの内容として、このディレクトリを提供することはできません。

12.7. SIP アプリケーションの構成記述子

以下は、SIP アプリケーション配備記述子のコンフィグレーションのタイプと配備の情報を表わします。

- 「ServletContext」 初期化パラメータ (init parameters)
- セッションのコンフィグレーション (Session configuration)
- サーブレット定義 (Servlet definitions)
- セキュリティ (Security)

12.8. 拡張の依存関係

多くのアプリケーションがコードやリソースを利用する場合、サーブレットコンテナの現在の実装では、代表的にはコンテナ内のライブラリファイルとしてインストールされることになります。これらのファイルは、共通あるいは標準の API により、ポータビリティを犠牲にしないで使用することができます。一つあるいは数個のアプリケーションからだけ呼び出されるファイルは、アクセスを可能にするために、サーブレットアプリケーションの一部にすることがあります。

アプリケーション開発者は、サーブレットコンテナにどの拡張がインストールされているのかを、知る必要があります。また、コンテナはポータビリティを保証するために、SAR 形式内のサーブレットライブラリの依存関係を知ることが必要となります。

サーブレットコンテナは、サーブレットアプリケーションに対して、JAR ファイル中の有効なリソースやコードを示すことや、アプリケーションがそれを有効にするためのメカニズムを持つことが推奨されます。コンテナは、ライブラリファイルあるいは拡張仕様の編集･変更のための便利な機能を提供することが必要です。

アプリケーション開発者は、SAR ファイルのリスト拡張の「META-INF/MANI-FEST.MF」エントリを提供することが推奨されています。manifesto エントリのフォーマットは、準の JAR manifesto フォーマットに従う必要があります。サーブレットコンテナにインストールされた拡張の依存関係を表現するために、manifesto エントリは「http://java.sun.com/j2se/1.3/docs/guide/extensions/version-ing.html」に定義された標準拡張の仕様に従う必要があります。

サーブレットコンテナは、オプションである SAR ファイルの manifesto エントリ、または SAR ファイルの「WEB-INF/lib」エントリ下の JAR ファイルライブラリの manifesto エントリに宣言された依存関係を認識できる必要があります。サーブレットコンテナがこのような方法で宣言された依存関係を扱うことができない場合は、情報付きのエラーメッセージを返すことにより、アプリケーションを拒否することが必要です。

12.9. サーブレットアプリケーションのクラスローダ

コンテナが SAR 中のサーブレットをロードするために使うクラスローダは、その SAR が J2SE や Java Servlet API のクラスを書き換えることを許してはいけません。更に、SAR 中のサーブレットがサーブレットコンテナの実装クラスにアクセスすることを許さないようにすることも必要です。

一方、アプリケーションのクラスローダは SAR のクラスやリソースのパッケージが、コンテナ中で使う JAR ライブラリにあるクラスやリソースよりも、優先してロードされるように実装することを推奨しています。

12.10. サーブレットアプリケーションの置き換え

サーバは、コンテナを再起動せずに、アプリケーションを新しい版で置き換えることができる必要があります。アプリケーションが置き換えられる場合、コンテナはアプリケーションの中でセッションデータを保証することに対する、ロバストなメソッドを提供することが必要となります。

12.11. サーブレットアプリケーション環境

Java 2 Platform Enterprise Edition (J2EE) は、アプリケーションが外部情報の名前付けや構成に関する明確な知識を持たずに、簡単にリソースと外部情報にアクセスすることを可能にする、ネーミング環境を定義しています。

サーブレットは J2EE の不可欠なコンポーネントタイプであるため、サーブレットがリソース及び Enterprise Beans への参照を得ることを可能にするための情報を指定する、SIP サーブレットアプリケーション配備記述子を用意します。この情報を含む配備の要素には、以下のものがあります。

- env-entry
- ejb-ref
- resource-ref

「env-entry」要素は、「java:comp/env」コンテキストに関連した基本環境のエントリ名、予想される環境のエントリ値の Java タイプ（JNDI lookup メソッドが返すオブジェクトの型）、オプションの環境のエントリ値を設定するための情報を含んでいます。「ejb-ref」要素は、サーブレットが Enterprise Bean の home インタフェースを発見することを許すために必要な情報を含んできます。「resource-ref」要素は、resource factory を設定するために必要な情報を含んでいます。

環境設定に関連する J2EE 環境の要求は、「Java 2 Platform Enterprise Edition v1.3 specification 2」の 5 章で説明されています。J2EE 仕様に記述されたアプリケーション環境の機能を実装するために、J2EE の一部でないサーブレットコンテナに対しては、J2EE に準拠した実装が勧められていますが、それが要求されているわけではありません。なお、コンテナが、この環境をサポートするのに必要な機能を実装していない場合は、それに依存するアプリケーションを配備する際に、警告を出すことが必要です。

13. アプリケーションリスナとイベント

Java Servlet Specification 2.3 版では、アプリケーションリスナの概念が導入されています[Servlet API,10 章]。SIP Servlet API は、「javax.serv-let.http」パッケージで定義された HTTP 特有のイベントは除いて、Java Servlet Specification 2.3 版で定義されたアプリケーションリスナとイベントの完全なサポートを要求しています。そこで、「javax.servlet」で定義されたサーブレットコンテキストイベントをサポートすることが必要となります。加えて、本仕様では、サポートが必要となる SIP 特有のイベントが定義されています。以下の章では、サーブレットに対するリスナを実装する方法について説明します。

アプリケーションリスナとイベントの詳細な内容については、[Servlet API,10 章]を参照して下さい。

13.1. SIP サーブレットのイベントタイプとリスナインタフェース

SIP 特有のイベントタイプと、それをモニタするために使うリスナインタフェースを以下に示します。

- **SipApplicationSession 用のリスナ**:
  - **javax.servlet.sip.SipApplicationSessionListener**
    「SipApplicationSession」の生成/削除/タイムアウト用リスナ
- **SipSession 用のリスナ**:
  - **javax.servlet.sip.SipSessionListener**
    「SipSession」の生成/削除用リスナ
  - **javax.servlet.sip.SipSessionAttributeListener**
    サーブレットコンテキスト属性の追加/削除/更新用リスナ
- **error リスナ:**
  - **javax.servlet.sip.SipErrorListener**
    期待された「ACK」や「PRACK」が受信されないことの通知用リスナ
- **timer リスナ:**
  - **javax.servlet.sip.TimerListener**
    「Timer」が起動されたことの通知用リスナ

なお、セッションオブジェクトの属性として保存されるオブジェクトは、属性のバインディングやセッション活性化イベントの通知を受けとるために、「SipSessionBindingListener」と「SipSessionActivationListener」インタフェースを実装することができます。ただし、これらのリスナインタフェースの実装は、配備記述子内には記述されません。

13.2. リスナとして動作するサーブレット

初期化の時点でサーブレットに渡される「ServletConfig」と「ServletContext」オブジェクトは、コンフィグレーション情報、リソースへのアクセス、及びロギングを実行する機能を提供します。リスナは、「ServletConfig」と「ServletContext」オブジェクトに対する簡易なアクセス機能を持っていないため、利用するためにはプログラマが特別な準備を行うことが必要となります。これは可能ではありますが、少々手間がかかります。

このような理由のために、SIP サーブレットは一つ以上のリスナインタフェースを直接実装することができます。このような実装が行われる時、コンテナはリスナ機能に対応するクラスを別々のインスタンスとしては生成しません。最も一般的な「non-SingleThreadModel」のサーブレットの場合には、一つのサーブレットインスタンスをリスナとして用意します。

例えば、以下のクラスが「SipServlet」と「TimerListener」の両機能として動作します。

```
public class MyServlet extends SipServlet
        implements TimerListener
{
        private TimerService timerService;
        public void init() {
                timerService = (TimerService)
                getServletContext().getAttribute(TIMER_SERVICE);
        }
        protected void doInvite(SipServletRequest req) {
                ...
                timerService.createTimer(...);
        }
        public void timeout(ServletTimer timer) {
                log("OK to log here...");
        }
}
```

この場合、コンテナは「MyServlet」のインスタンスを一つだけ生成し、このインスタンスがアプリケーションに対して、「SipServlet」と「TimerListener」の両機能を提供します。

コンテナは通常、「SingleThreadModel」インタフェースを実装して、サーブレット用のインスタンスプールを生成します。このようなサーブレットクラスがリスナとして二役を演じる場合、対応するリスナを呼び出したい時には、コンテナはこれらのサーブレットのどれか一つを選択することになります。

一貫性を維持するため、一つのクラスが両方を実装している場合でも、配備記述子には分かれた「servlet」と「listener」の宣言を行うことが必要です。勿論、「listener」エレメントとして、独立したインスタンスは生成しません。

14. セキュリティ

サーブレットアプリケーションは、アプリケーション開発者によって開発されます。サーブレットアプリケーションは、実行環境にインストールするために、デプロイヤに渡されます。アプリケーション開発者は、配備するアプリケーションのセキュリティの設定の方法をデプロイヤに伝える必要があります。これは、配備記述子のメカニズムを使用することにより、宣言的に行うことができます。

この章では、セキュリティ要件に対応するための配備表現について説明します。サーブレットアプリケーションのディレクトリ構成及び配備記述子と同様に、本章ではランタイム表現のための要求については説明しません。しかし、コンテナがそのランタイム表現の一部として、ここに述べられるエレメントをインプリメントすることが推奨されます。

14.1. はじめに

サーブレットアプリケーションは、多くのユーザによってアクセス可能なリソースを表現します。これらのリソースは、インターネットのような無防備でオープンなネットワーク上でアクセスされます。このような環境では、サーブレットアプリケーションがセキュリティに関して考慮することが必要となります。

品質保証や実装の詳細は異なっても構いませんが、サーブレットコンテナは以下に示すような特性を共有できるようなメカニズムや基盤を持つことが必要です。

認証: 通信する実体同士が、アクセスのために認証された特定のアイデンティティを代表していることを、相互に証明するための手段です。

リソースのアクセスコントロール: 完全性、機密性あるいは制約の有効性を強化する目的で、リソースにアクセス可能なユーザやプログラムの集まりを制限するための手段です。

データの保全性:情報が第三者によって途中で修正されていないことを証明するための手段です。

機密保持またはデータのプライバシー: 情報がアクセスの権限を持ったユーザにだけ利用可能であることを保証するための手段です。

14.2. 宣言的セキュリティ

宣言的セキュリティとは、アプリケーションの外部構造として、役割/アクセスコントロール/認証の要件を含む、アプリケーションのセキュリティ構造を表現するための方法を示しています。配備記述子は、サーブレットアプリケーションにおける宣言的セキュリティの主要な手段となっています。

デプロイヤは、アプリケーションの論理的なセキュリティ要件を、実行環境に特有のセキュリティポリシーの表現にマッピングします。サーブレットコンテナは、実行時に認証と承認を行うために、そのセキュリティポリシー表現を利用します。

サーブレットに対するセキュリティモデルは、発信者か着信者のどちらかの立場に立って、リクエストを処理する際に適用されます。そのセキュリティモデルは、サーブレットが静的なリソースをアクセスするために

「RequestDispatcher」を使用する場合や、サーブレットが「forward」を使用する場合には適用されません。

14.3. プログラム的セキュリティ

プログラム的セキュリティは、宣言的セキュリティだけではアプリケーションのセキュリティモデルを表現することが十分でない場合に、使用されます。プログラム的セキュリティは、「SipServletMessage」インタフェースの以下のメソッドから構成されます。

- getRemoteUser
- isUserInRole
- getUserPrincipal

「getRemoteUser」メソッドは、認証されたクライアントのユーザ名を返します。「isUserInRole」メソッドは、リモートのユーザがセキュリティ上の特別な役割を持つか否かを判断します。「getUserPrincipal」メソッドは、ユーザのプリンシパル名を求め、「java.security.Principal」オブジェクトを返します。サーブレットは、この API の利用により得られた情報を基にして、ビジネスロジックの判断を行うようなことができます。

認証されたユーザがいない場合は、「getRemoteUser」メソッドは「null」を返し、「isUserInRole」メソッドは常に「false」を返します。また、「getUserPrincipal」メソッドは「null」を返します。

注意: レスポンスは、UAS の信任を含むことができます。このような理由のために、プログラム的セキュリティのメソッドは、リクエストと同様にレスポンスに対しても適用します。しかし、プロキシから、認証に失敗したレスポンスの送信者の UAS にチャレンジ（注：認証を再度行う）するメカニズムがないため、SIP の配備記述子では、レスポンスの認証に関する要求を表現することができません。

「isUserInRole」メソッドには、「String」型のユーザの役割名の引数を与えることになります。
「security-role-ref」エレメントは、メソッドに渡される役割名を含んでいる「role-name」サブエレメントを持つ配備記述子の中で宣言されることが必要です。「security-role」エレメントは、ユーザがマッピングされるセキュリティの役割名の値を持つ「role-link」サブエレメントを含んでいる必要があります。呼び出しの返り値を決定する場合、コンテナはセキュリティ上の役割に対して「security-role-ref」のマッピングを使います。

例えば、セキュリティ上の役割のリファレンス“FOO”を、役割名”manager“を持ったセキュリティ上の役割にマップする場合には、次のように表現します。

```
<security-role-ref>
  <role-name>FOO</role-name>
  <role-link>manager</manager>
</security-role-ref>
```

この例の場合、セキュリティ上の役割の“manager”に属するユーザによって呼び出されたサーブレットが API「isUserInRole（”FOO”）」を実行した時には、結果は「true」となります。

「security-role」エレメントと一致する「security-role-ref」エレメントが宣言されていない場合、コンテナはサーブレットアプリケーションのために「security-role」エレメントのリストに対する「role-name」エレメントの引数をチェックすることが必要です。「isUserInRole」メソッドは、発信者がセキュリティ上の役割にマッピングされているか否か決めるために、そのリストを参照します。開発者は、呼び出すサーブレットを再コンパイルすることなく、アプリケーションでの役割名を変更する際に、このデフォルトのメカニズムの使用が柔軟性を制限する可能性があることを気を付ける必要があります。

14.4. 役割

セキュリティ上の役割は、アプリケーション開発者によって定義された、ユーザの論理的なグルーピングを表しています。アプリケーションが配備される時、役割はデプロイヤによって実行環境上のプリンシパルやグループにマッピングされます。

サーブレットコンテナは、プリンシパルのセキュリティ属性をベースに、受信するリクエストと関連したプリンシパルに対して、宣言的あるいはプログラム的セキュリティが適用されます。これは、次のいずれかの場合に起こります。

1. デプロイヤは、実行環境中のユーザグループに対して、セキュリティ上の役割をマッピングします。呼び出しているプリンシプルが属するユーザグループは、そのセキュリティ属性から検索されます。プリンシプルのユーザグループと、デプロイヤによってマッピングされたセキュリティ上の役割を持つユーザグループとが一致する場合にのみ、プリンシプルがそのセキュリティ上の役割の中に含まれています。
2. デプロイヤは、セキュリティポリシードメインの中で、プリンシプル名にセキュリティ上の役割をマッピングします。この場合、呼び出しているプリンシパルのプリンシパル名は、そのセキュリティ属性から検索されます。プリンシパル名と、セキュリティ上の役割がマッピングされたプリンシパル名とが同じ場合にのみ、プリンシパルはセキュリティ上の役割の中に含まれています

14.5. 認証

SIP ユーザエージェントは、例えば以下のメカニズムの一つを使用して、SIP サーバを利用するユーザを認証することができます。

- SIP ベーシック認証
- SIP ダイジェスト認証
- 「P-Asserted-Identity」ヘッダ。認証された発信者/受信者を、上流/下流のプロキシが信用するためのもので、[RFC 3325]内に定義されています。

HTTP サーブレット仕様は、SSL をベースとしたユーザ認証を利用することができます。SSL、TLS 及び IPSec では、認証されているのは実際には前のホップであり、そのホップから「user-data」の制約が強化されています。SIP ではプロキシが一般的なものであり、UAC と直接関連する必要がないため、受信する TLS 接続上で認証するものは、一般に UAC 自身ではありません。

このような理由により、SIP サーブレットコンテナでは、TLS ハンドシェークの一部として受信した信任状をベースとした認証は、一般的には行いません。しかし、いくつかの環境では、TLS 接続が実際に UAC を識別します。そのような場合には、ここで議論された宣言的・プログラム的なセキュリティ機能の実装に対して、この情報を反映することはコンテナにとって合理的なものとなります。これらのセキュリティの特徴は、プロキシではなくエンドユーザに関係しますが、認証されたメッセージの構成を決めるのは個々のコンテナに任されています。

14.6. 認証情報のサーバでの追跡

実行環境で(ユーザやグループなどの)セキュリティ ID がどの役割にマッピングされるか、はアプリケーション特有というよりも環境に特有なものであるため、以下のことが望まれています。

1. ログイン機構とポリシーを、サーブレットアプリケーションが配備される環境のプロパティとすること。
2. 同じコンテナに配備されるアプリケーションはすべて、プリンシパルの表現に同じ認証情報を使うことができ、
3. セキュリティポリシードメインを超える時にだけ、ユーザの再認証が要求されること。

したがって、サーブレットコンテナは、(サーブレットアプリケーションレベルではなく)コンテナレベルでの認証情報を追跡することが要求されます。これは、一つのサーブレットアプリケーションに対して認証されたユーザが、同じセキュリティ ID に許可されたコンテナによって管理される他のリソースにアクセスすることを可能にしています。

14.7. EJB™ 呼び出しにおけるセキュリティ ID の伝搬

セキュリティ ID、あるいはプリンシパルは、Enterprises Bean の呼び出しの際に利用するために、提供される必要があります。サーブレットアプリケーションから Enterprise Bean を呼び出す際のデフォルトモードでは、ユーザのセキュリティ ID が EJB ™ コンテナに伝搬されます。

他のシナリオでは、コンテナがサーブレットコンテナに、あるいは EJB ™ コンテナに認証されていないユーザに対して、次のような呼び出しを許可することが要求されています。

- SIP サーブレットコンテナは、コンテナに認証されていないクライアントによるアプリケーションへのアクセスを提供することが要求されています。
- アプリケーションコードは、サインオン、あるいは発信者の ID をベースとしたデータのカスタマイゼーションだけの処理を実行するものであっても構いません。

これらのシナリオでは、サーブレットアプリケーションの配備記述子は、「run-as」エレメントを指定します。このエレメントが指定された場合には、「run-as」エレメントに定義されたセキュリティ上の役割名を基に、コンテナは発信者のセキュリティ・アイデンティティを EJB ™ レイヤに伝搬させる必要があります。そのセキュリティ上の役割名は、サーブレットアプリケーションに対して定義されたセキュリティ上の役割名の一つとなっている必要があります。

J2EE プラットフォームの一部として動作するサーブレットコンテナでは、「run-as」エレメントを指定することにより、同じ J2EE アプリケーションの中の EJB コンポーネントへの呼び出しと、他の J2EE アプリケーションに配備された EJB コンポーネントへの呼び出しの両機能が提供される必要があります。

14.8. セキュリティ制約の指定

セキュリティ制約は、アプリケーションの意図的な保護を宣言的な方法により指定します。SIP サーブレットのセキュリティ制約は、次のエレメントから構成されます。

- リソースの集り（サーブレットの集合）
- 認証のタイプ
- 承認の制約
- ユーザデータの制約

リソースの集りは、サーブレットと SIP メソッドの集合です。サーブレットは、関連する一つ以上のセキュリティ制約を持つことができます。受信するリクエストを処理するためにサーブレットを呼び出す前に、コンテナはサーブレットと関連するセキュリティ制約のすべて満たすことを保証することが必要となります。セキュリティ制約が満たされない場合には、リクエストは「401」あるいは「407」のステータスコードにより拒絶することが必要です。

注意. SIP サーブレットのリソースの集りは、HTTP サーブレット API の場合と同じように、URL パターンの代りに起動されているサーブレットの名前によって識別されます。

認証のタイプは、コンテナが受信したリクエストを認証する時に、「401 (認証失敗)」か「407 (プロキシでの認証が必要)」のどちらのレスポンスのステータスコードを返す必要があるかの指標を示しています。

承認の制約は、リソース集合によって記述されるリソースへのアクセスが可能なユーザが属している、セキュリティ上の役割の集合となっています。ユーザが許可された役割に属していない場合は、そのユーザはリソースへのアクセスが拒否されます。承認の制約が役割を何も定義していない場合は、そのセキュリティ制約により定義されたサーブレットアプリケーションには、どのユーザもアクセスすることができません。

ユーザデータの制約は、クライアント・サーバのトランスポートレイヤに対する要求事項を記述しています。その要求は、コンテンツの完全性（通信プロセスのデータの改竄を防ぐ）や機密性（途中で読まれることを防ぐ）に対するものになります。コンテナは、「integral」か「confidential」とマークされたリソースへのリクエストに対して応答するためには、TLS を使うことが必要となります。元のリクエストが TCP 上の場合、コンテナはクライアントを TLS ポートにリダイレクトすることが必要となります。

14.9. デフォルトポリシー

デフォルトでは、リソース（サーブレット）にアクセスするためには、認証は必要ありません。配備記述子によって指定された場合にだけ、リソースの集りへのリクエストに対しては、認証が必要となります。

15. 配備記述子

本章では、配備記述子をサポートする SIP サーブレットコンテナに対する SIP サーブレット仕様 v1.0 の要求事項について説明します。配備記述子は、アプリケーション開発者、アプリケーション構成者及びデプロイヤの間で、サーブレットアプリケーションのエレメントとコンフィグレーション情報を共有するために利用します。

15.1. HTTP サーブレット配備記述子との違い

SIP サーブレットの配備記述子は、HTTP サーブレットの配備記述子と非常に良く似ています。主要な違いを以下に示します。

- ルートエレメントが、「web-app」ではなく「sip-app」になります。
- 受信されるリクエストは、HTTP に特有の URL のマッピングではなく、11 章で定義したルール言語に基づいて、サーブレットにマッピングされます。
- 次のエレメント（子供のエレメントと共に）は SIP では意味がないため、配備記述子に存在しません。
  - 「form-login-config」，「mime-mapping」，「welcome-file-list」，「error-page」，「taglib」，「jsp-file」
- リソースの集りは、HTTP での URL パターンの集合ではなく、名前が付けられたサーブレットの集合として定義されます。
- Servlet API v2.3 で導入されたフィルタの概念はサポートされていません。SIP Servlet API のアプリケーション組合せのモデルでは、一つのリクエストが複数のアプリケーションにより実行されることを許しています。これは、フィルタと全く同じものではありませんが、フィルタに対する要求事項の多くを扱うことができます。

SIP サーブレットアプリケーションは、以下に示される XML DTD に適合した配備記述子を持つことが必要です。12 章で述べられたように、この配備記述子は、サーブレットアプリケーションの「/WEB-INF」ディレクトリの中のファイル「sip.xml」として配置されます。

15.2. SIP と HTTP アプリケーションの融合

アプリケーションは SIP と HTTP のコンポーネントの両方（更には、潜在的には、まだ表現されていませんが、他のプロトコル用の servlet API に関連するコンポーネント）を持つことができます。このようなアプリケーションの場合、HTTP Servlet API により定義された DTD に適合した HTTP 配備記述子の「web.xml」と、SIP 配備記述子の「sip.xml」の両方が必要となります。

多くの配備記述子エレメントは、アプリケーション全体に適用されます。次の規則は、SIP と HTTP の両方の記述子に現われる場合の扱われ方を示しています(この規則は、SIP あるいは HTTP と並存する、他の servlet API に対する配備記述子にも適用されます)。

- **distributable:** この“tagging”エレメントが一方の配備記述子の中に現れる場合、他方にも現れる必要があります。
- **context-param:** 同じアプリケーションに属する SIP および HTTP サーブレットが、一つの「ServletContext」

に渡されます。「ServletContext」から利用可能な初期化パラメータの集合は、SIP および HTTP 配備記述子の中の「context-param」エレメントで指定されたパラメータを結合したものです。パラメータは、両方のディスクリプタの中で設定されることができますが、その場合、両方の宣言に同じ値を持つことが必要となります。

- **listener:** アプリケーションのリスナの集合は、SIP および HTTP 配備記述子に定義されたリスナの結合されたものとなります。

更に、アプリケーションの「display-name」と「icon」は同一で、しかも、そのエレメントが一方に現れた場合、他方の中にも現れることが必要です。

15.3. 配備記述子のエレメント

構成/配備情報の次のタイプが、SIP サーブレットアプリケーション配備記述子の中に存在し、セキュリティ文法の例外を含めて、すべてのサーブレットコンテナに対して提供することが要求されています。

- 「ServletContext」初期化パラメータ
- セッションのコンフィグレーション
- サーブレット定義
- サーブレットのマッピング
- アプリケーションのライフサイクルのリスナクラス
- エラーハンドラ
- フィルタ定義とフィルタマッピング
- セキュリティ

また、J2EE アプリケーションサーバの一部であるサーブレットコンテナの追加要求をサポートするために、SIP サーブレットアプリケーション配備記述子に、いくつかのエレメントが存在します。これらのエレメントは、JNDI オブジェクト（「env-entry」，「ejb-ref」，「ejb-local-ref」，「resource-ref」，「resource-env-ref」）の検索を許すもので、サーブレット仕様だけをサポートするコンテナでは提供する必要がありません。これらのエレメントの詳しい説明に関しては、DTD コメントを参照して下さい。

15.4. 配備記述子を処理するための規則

本章では、サーブレットアプリケーション用の配備記述子の処理に関して、サーブレットコンテナ及び開発者が注意する必要のある一般的な規則を挙げることにします。

- サーブレットコンテナは、配備記述子のテキスト中の PCDATA に対して、最初の非-空白文字に先立つ空白文字をすべて無視します。更に、最後の非-空白文字より後の空白文字もすべて無視します。
- サーブレットコンテナとサーブレットアプリケーションを操作するツールには、SAR の妥当性をチェックするための広範囲の選択肢があります。これには、配備記述子ドキュメントに記述されている内容の妥当性のチェックが含まれています。また、サーブレットコンテナやツールが、配備記述子が DTD ドキュメントに対して構造的に正しいことを確認することが、推奨されています。ただし、要求仕様ではありません。

更に、(サーブレットコンテナやツールが) 意味的なチェック機能を提供することも推奨されています。例えば、セキュリティの制約で参照された役割が、配備記述子の中で定義されたセキュリティの役割の一つと同じ名前であることをチェックすることが必要です。

規則に適合しない場合には、サーブレットアプリケーション/ツール/コンテナは、開発者に説明用のエラーメッセージを通知することが必要となります。高機能なアプリケーションサーバを提供するベンダに対しては、コンテナとは分離したツールとして、このような妥当性チェックを実行できる環境を用意することが推奨されます。

15.4.1. 配備記述子 DOCTYPE

すべての正しい SIP アプリケーション配備記述子には、以下の「DOCTYPE」宣言が含まれている必要があります。

```
<!DOCTYPE sip-app
        PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
        "http://www.jcp.org/dtd/sip-app_1_0.dtd">
```

15.5. DTD

以下の DTD は、SIP サーブレットアプリケーションの配備記述子に対する XML 文法を定義しています。

```
<!-- The sip-app element is the root of the deployment descriptorfor a SIP servlet application -->
<!ELEMENT sip-app (icon?, display-name?, description?,
distributable?, context-param*, listener*, servlet*,
servlet-mapping*, proxy-config?, session-config?,
resource-env-ref*, resource-ref*, security-constraint*,
login-config?, security-role*, env-entry*, ejb-ref*,
ejb-local-ref*)>

<!-- The icon element contains small-icon and large-icon elements that specify the file names for small and a
large GIF or JPEG icon images used to represent the parent element in a GUI tool. -->
<!ELEMENT icon (small-icon?, large-icon?)>

<!-- The small-icon element contains the name of a file containing a small (16 x 16) icon image. The file name
is a path relative to the root of the SAR. The image may be either in the JPEG or GIF format. The icon can be
used by tools.
Example:
        <small-icon>/employee-service-icon16x16.jpg</small-icon>
-->
<!ELEMENT small-icon (#PCDATA)>

<!-- The large-icon element contains the name of a file containing a large (32 x 32) icon image. The file name
```

is a path relative to the root of the SAR. The image may be either in the JPEG or GIF format. The icon can be used by tools.
Example:
<large-icon>/employee-service-icon32x32.jpg</large-icon>
-->
<!ELEMENT large-icon (#PCDATA)>

<!-- The display-name element contains a short name that is intended to be displayed by tools. The display name need not be unique.
Example:
<display-name>Employee Self Service</display-name>
-->
<!ELEMENT display-name (#PCDATA)>

<!-- The description element is used to provide text describing the parent element. The description element should include any infor-mation that SAR producer wants to provide to the consumer of SAR (i.e., to the Deployer). Typically, the tools used by SAR consumer will display the description when processing the parent element
that contains the description. -->
<!ELEMENT description (#PCDATA)>

<!-- The distributable element, by its presence in a servlet application deployment descriptor, indicates that this servlet applica-tion is programmed appropriately to be deployed into a distributed servlet container -->
<!ELEMENT distributable EMPTY>

<!-- The context-param element contains the declaration of a servlet application's servlet context initialization parameters. -->
<!ELEMENT context-param (param-name, param-value, description?)>

<!-- The param-name element contains the name of a parameter. Each parameter name must be unique in the servlet application. -->
<!ELEMENT param-name (#PCDATA)>

<!-- The param-value element contains the value of a parameter. -->
<!ELEMENT param-value (#PCDATA)>

<!-- The listener element indicates the deployment properties for a servlet application listener bean. -->
<!ELEMENT listener (listener-class)>

<!-- The listener-class element declares a class in the application must be registered as a servlet application listener bean. The value is the fully qualified classname of the listener class.
-->
<!ELEMENT listener-class (#PCDATA)>

<!-- The servlet element contains the declarative data of a servlet. -->
<!ELEMENT servlet (icon?, servlet-name, display-name?,
description?, servlet-class, init-param*, load-on-startup?,
run-as?, security-role-ref*)>

<!-- The servlet-name element contains the canonical name of the servlet. Each servlet name is unique within the servlet applica-tion.
-->
<!ELEMENT servlet-name (#PCDATA)>

<!-- The servlet-class element contains the fully qualified class name of the servlet. -->
<!ELEMENT servlet-class (#PCDATA)>

<!-- The init-param element contains a name/value pair as an initialization param of the servlet -->
<!ELEMENT init-param (param-name, param-value, description?)>

<!-- The load-on-startup element indicates that this servlet should be loaded (instantiated and have its init() called) on the startup of the servlet application. The optional contents of these element must be an integer indicating the order in which the servlet should be loaded. If the value is a negative integer, or the element is not present, the container is free to load the servlet whenever it chooses. If the value is a positive integer or 0, the container must load and initialize the servlet as the application is deployed. The container must guarantee that servlets marked with lower integers are loaded before servlets marked with higher integers. The container may choose the order of loading of serv-lets with the same load-on-start-up value. -->
<!ELEMENT load-on-startup (#PCDATA)>

<!-- The servlet-mapping element defines a mapping between a servlet and a pattern -->
<!ELEMENT servlet-mapping (servlet-name, pattern)>

<!-- The different types of conditions supported. -->
<!ENTITY % condition "and | or | not | equal | contains | exists |
subdomain-of">

<!-- A pattern is a condition: a predicate over the set of SIP requests. -->
<!ELEMENT pattern (%condition;)>

<!-- An "and" condition is true if and only if all its constituent conditions are true. -->
<!ELEMENT and (%condition;)+>

<!-- An "or" condition is true if at least one of its constituent conditions is true. -->
<!ELEMENT or (%condition;)+>

<!-- Negates the value of the contained condition. -->
<!ELEMENT not (%condition;)>

<!-- True if the value of the variable equals the specified literal value. -->
<!ELEMENT equal (var, value)>

<!-- True if the value of the variable contains the specified literal value. -->
<!ELEMENT contains (var, value)>

<!-- True if the specified variable exists. -->
<!ELEMENT exists (var)>

<!-- -->
<!ELEMENT subdomain-of (var, value)>

<!-- Specifies a variable.
Example: <var>request.uri.user</var>
-->
<!ELEMENT var (#PCDATA)>

<!-- Specifies a literal string value that is used to specify rules. -->
<!ELEMENT value (#PCDATA)>

<!-- Specifies whether the "equal" test is case sensitive or not. -->
<!ATTLIST equal ignore-case (true| false) "false">

<!-- Specifies whether the "contains" test is case sensitive or not. -->
<!ATTLIST contains ignore-case (true| false) "false">

<!-- The proxy-config element configures proxy-related parameters. -->
<!ELEMENT proxy-config (sequential-search-timeout?)>

<!-- The sequential-search-timeout element defines the default timeout for sequential searches for all proxy operations performed by this application. The specified timeout must be expressed in a whole number of seconds. The container may override this value as a result of its own local policy. -->
<!ELEMENT sequential-search-timeout (#PCDATA)>

<!-- The session-config element defines the session parameters for this servlet application. -->
<!ELEMENT session-config (session-timeout?)>

<!-- The session-timeout element defines the default session timeout interval for all application sessions created in this servlet application. SipSessions have no timeout independent of that of the containing application session. The lifetime of a SipSession is tied to that of the parent application session. The specified timeout must be expressed in a whole number of minutes. If the timeout is 0 or less, the container ensures the default behavior of sessions is never to time out. -->
<!ELEMENT session-timeout (#PCDATA)>

<!-- The resource-env-ref element contains a declaration of a servlet's reference to an administered object associated with a resource in servlet's environment. It consists of an optional description, the resource environment reference name, and an indication of the resource environment reference type expected by servlet code.
Example:

```
<resource-env-ref>
  <resource-env-ref-name>jms/StockQueue</resource-env-ref-name>
  <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
```

-->
<!ELEMENT resource-env-ref (description?, resource-env-ref-name, resource-env-ref-type)>

<!-- The resource-env-ref-name element specifies the name of a resource environment reference; its value is the environment entry name used in servlet code. The name is a JNDI name relative to the java:comp/env context and must be unique within a servlet applica-tion.
-->
<!ELEMENT resource-env-ref-name (#PCDATA)>

<!-- The resource-env-ref-type element specifies the type of a resource environment reference. It is the fully qualified name of a Java language class or interface. -->
<!ELEMENT resource-env-ref-type (#PCDATA)>

<!-- The resource-ref element contains a declaration of a serv-let's reference to an external resource. It

consists of an optional description, the resource manager connection factory reference name, the indication of the resource manager connection factory type expected by servlet code, the type of authentication (Application or Container), and an optional specification of the share-ability of connections obtained from the resource (Shareable or Unshareable).

Example:

```
        <resource-ref>
          <res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
          <res-type>javax.sql.DataSource</res-type>
          <res-auth>Container</res-auth>
          <res-sharing-scope>Shareable</res-sharing-scope>
        </resource-ref>
-->
```

<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-auth, res-sharing-scope?)>

<!-- The res-ref-name element specifies the name of the resource factory reference name. -->

<!ELEMENT res-ref-name (#PCDATA)>

<!-- The res-ref-name element specifies the name of a resource manager connection factory reference. The name is a JNDI name relative to the java:comp/env context. The name must be unique within servlet application. -->

<!ELEMENT res-type (#PCDATA)>

<!-- The res-auth element specifies whether the servlet code signs on programmatically to the resource manager, or whether the Container will sign on to the resource manager on behalf of the servlet. In the latter case, the Container uses information that is supplied by the Deployer. The value of this element must be one of the two following:

```
        <res-auth>Application</res-auth>
        <res-auth>Container</res-auth>
-->
```

<!ELEMENT res-auth (#PCDATA)>

<!-- The res-sharing-scope element specifies whether connections obtained through the given resource manager connection factory reference can be shared. The value of this element, if specified, must be one of the two following:

```
        <res-sharing-scope>Shareable</res-sharing-scope>
        <res-sharing-scope>Unshareable</res-sharing-scope>
```

The default value is Shareable. -->

<!ELEMENT res-sharing-scope (#PCDATA)>

<!-- The security-constraint element is used to associate security constraints with one or more servlet resource collections -->
<!ELEMENT security-constraint (display-name?,
resource-collection+, proxy-authentication?, auth-constraint?,
user-data-constraint?)>

<!-- The resource-collection element is used to identify a subset of the resources and SIP methods on those resources within a servlet application to which a security constraint applies. If no SIP methods are specified, then the security constraint applies to all SIP methods. -->
<!ELEMENT resource-collection (resource-name, description?,
servlet-name*, sip-method*)>

<!-- The resource-name contains the name of this resource collection
-->
<!ELEMENT resource-name (#PCDATA)>

<!-- The sip-method contains an SIP method (INVITE | BYE |...) -->
<!ELEMENT sip-method (#PCDATA)>

<!-- The presence of the proxy-authentication element indicates to the container that it must challenge the user agent with a 407 (Proxy Authentication Required) response status code when authenticating an incoming request. If not present a 401 (Unauthorized) status code is used. -->
<!ELEMENT proxy-authentication EMPTY>

<!-- The user-data-constraint element is used to indicate how data communicated between the client and container should be protected
-->
<!ELEMENT user-data-constraint (description?,
transport-guarantee)>

<!-- The transport-guarantee element specifies that the communication between client and server should be NONE, INTEGRAL, or CONFIDENTIAL. NONE means that the application does not require any transport guarantees. A value of INTEGRAL means that the application requires that the data sent between the client and server be sent in such a way that it can't be changed in transit. CONFIDENTIAL means that the application requires that the data be transmitted in a fashion that prevents other entities from observing the contents of the transmission. In most cases, the presence of the INTEGRAL or CONFIDENTIAL flag will indicate that the use of TLS is required. -->
<!ELEMENT transport-guarantee (#PCDATA)>

<!-- The auth-constraint element indicates the user roles that should be permitted access to this resource collection. The role-name used here must either correspond to the role-name of one of the security-role elements defined for this servlet application, or be the specially reserved role-name "*" that is a compact syntax for indicating all roles in the servlet application. If both "*" and rolenames appear, the container interprets this as all roles. If no roles are defined, no user is allowed access to the portion of the servlet application described by the containing security-constraint. The container matches role names case sensitively when determining access. -->

<!ELEMENT auth-constraint (description?, role-name*)>

<!-- The role-name element contains the name of a security role.-->

<!ELEMENT role-name (#PCDATA)>

<!-- The login-config element is used to configure the authentication method that should be used, the realm name that should be used for this application, and the attributes that are needed by the form login mechanism. -->

<!ELEMENT login-config (auth-method?, realm-name?)>

<!-- The realm name element specifies the realm name to use in HTTP Basic authorization -->

<!ELEMENT realm-name (#PCDATA)>

<!-- The auth-method element is used to configure the authentication mechanism for the SIP servlet application. As a prerequisite to gaining service from any SIP servlet which is protected by an authorization constraint, a user must have authenticated using the configured mechanism. Legal values for this element are "BASIC", "DIGEST", or "CLIENT-CERT". -->

<!ELEMENT auth-method (#PCDATA)>

<!-- The security-role element contains the declaration of a security role which is used in the security-constraints placed on the servlet application. -->

<!ELEMENT security-role (description?, role-name)>

<!-- The security-role-ref element defines a mapping between the name of role called from a Servlet using isUserInRole(String name) and the name of a security role defined for the servlet application. For example, to map the security role reference "FOO" to the security role with role-name "manager" the sytax would be:

```
<security-role-ref>
  <role-name>FOO</role-name>
  <role-link>manager</manager>
</security-role-ref>
```

In this case if the servlet called by a user belonging to the "manager" security role made the API call

isUserInRole("FOO") the result would be true. Since the role-name "*" has a special meaning for authorization constraints, its value is not permitted here.
-->
<!ELEMENT security-role-ref (description?, role-name, role-link)>

<!-- The role-link element is used to link a security role reference to a defined security role. The role-link element must contain the name of one of the security roles defined in the security-role elements. -->
<!ELEMENT role-link (#PCDATA)>

<!-- The env-entry element contains the declaration of a servlet application's environment entry. The declaration consists of an optional description, the name of the environment entry, and an optional value. If a value is not specified, one must be supplied during deployment. -->
<!ELEMENT env-entry (description?, env-entry-name,
env-entry-value?, env-entry-type)>

<!-- The env-entry-name element contains the name of a servlet applications's environment entry. The name is a JNDI name relative to the java:comp/env context. The name must be unique within a servlet application.
Example:
<env-entry-name>minAmount</env-entry-name>
-->
<!ELEMENT env-entry-name (#PCDATA)>

<!-- The env-entry-value element contains the value of a components environment entry. The value must be a String that is valid for the constructor of the specified type that takes a single String parameter, or for java.lang.Character, a single character.
Example:
<env-entry-value>100.00</env-entry-value>
-->
<!ELEMENT env-entry-value (#PCDATA)>

<!-- The env-entry-type element contains the fully-qualified Java type of the environment entry value that is expected by the components code.
The following are the legal values of env-entry-type:
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.String
java.lang.Short
java.lang.Integer

java.lang.Long
java.lang.Float
java.lang.Double
-->
<!ELEMENT env-entry-type (#PCDATA)>

<!-- The ejb-ref element is used for the declaration of a reference to an enterprise bean's home. The declaration consists of: - an optional description - the EJB reference name used in the code of the servlet that's referencing the enterprise bean - the expected type of the referenced enterprise bean - the expected home and remote interfaces of the referenced enterprise bean – optional ejb-link information, used to specify the referenced enterprise bean -->
<!ELEMENT ejb-ref (description?, ejb-ref-name, ejb-ref-type, home,
remote, ejb-link?)>

<!-- The ejb-ref-name element contains the name of an EJB reference. The EJB reference is an entry in the servlet's environment and is relative to the java:comp/env context. The name must be unique within the servlet application. It is recommended that name is prefixed with "ejb/".
Example:
<ejb-ref-name>ejb/Payroll</ejb-ref-name>
-->
<!ELEMENT ejb-ref-name (#PCDATA)>

<!-- The ejb-ref-type element contains the expected type of the referenced enterprise bean. The ejb-ref-type element must be one of the following:
<ejb-ref-type>Entity</ejb-ref-type>
<ejb-ref-type>Session</ejb-ref-type>
-->
<!ELEMENT ejb-ref-type (#PCDATA)>

<!-- The home element contains the fully-qualified name of the enterprise bean's home interface.
Example:
<home>com.aardvark.payroll.PayrollHome</home>
-->
<!ELEMENT home (#PCDATA)>

<!-- The remote element contains the fully-qualified name of the enterprise bean's remote interface.
Example:
<remote>com.wombat.empl.EmployeeService</remote>
-->

<!ELEMENT remote (#PCDATA)>

<!-- The ejb-link element is used in the ejb-ref or ejb-local-ref elements to specify that an EJB reference is linked to another enterprise bean. The value of the ejb-link element must be the ejb-name of an enterprise bean in the same J2EE application unit. The name in the ejb-link element may be composed of a path name specifying the ejb-jar containing the referenced enterprise bean with the ejb-name of the target bean appended and separated from the path name by "#". The path name is relative to the SAR containing the servlet application that is referencing the enterprise bean. This allows multiple enterprise beans with the same ejb-name to be uniquely identified.
Examples:
<ejb-link>EmployeeRecord</ejb-link>
<ejb-link>../products/product.jar#ProductEJB</ejb-link>
-->

<!ELEMENT ejb-link (#PCDATA)>

<!-- The ejb-local-ref element is used for the declaration of a reference to an enterprise bean's local home. The declaration consists of: - an optional description - the EJB reference name used in the code of THE_COMPONENT that's referencing the enterprise bean - the expected type of the referenced enterprise bean – the expected local home and local interfaces of the referenced enterprise bean - optional ejb-link information, used to specify the referenced enterprise bean -->

<!ELEMENT ejb-local-ref (description?, ejb-ref-name, ejb-ref-type,
local-home, local, ejb-link?)>

<!-- The local element contains the fully-qualified name of the enterprise bean's local interface. Used by ejb-local-ref -->

<!ELEMENT local (#PCDATA)>

<!-- The local-home element contains the fully-qualified name of the enterprise bean's local home interface. Used by ejb-local-ref
-->

<!ELEMENT local-home (#PCDATA)>

<!-- The run-as element, if defined for a servlet, overrides the security identity used to call an EJB by that servlet in this servlet application. The role-name is one of the security roles already defined for this servlet application. Used by: <servlet> -->

<!ELEMENT run-as (description?, role-name)>

<!-- The ID mechanism is to allow tools to easily make tool-specific references to the elements of the deployment descriptor. This allows tools that produce additional deployment information (i.e information

beyond the standard deployment descriptor information) to store the non-standard information in a separate file, and easily refer from these tools-specific files to the information in the standard sip-app deployment descriptor. -->

<!ATTLIST sip-app id ID #IMPLIED>
<!ATTLIST icon id ID #IMPLIED>
<!ATTLIST small-icon id ID #IMPLIED>
<!ATTLIST large-icon id ID #IMPLIED>
<!ATTLIST display-name id ID #IMPLIED>
<!ATTLIST description id ID #IMPLIED>
<!ATTLIST distributable id ID #IMPLIED>
<!ATTLIST context-param id ID #IMPLIED>
<!ATTLIST param-name id ID #IMPLIED>
<!ATTLIST param-value id ID #IMPLIED>
<!ATTLIST listener id ID #IMPLIED>
<!ATTLIST listener-class id ID #IMPLIED>
<!ATTLIST servlet id ID #IMPLIED>
<!ATTLIST servlet-name id ID #IMPLIED>
<!ATTLIST servlet-class id ID #IMPLIED>
<!ATTLIST init-param id ID #IMPLIED>
<!ATTLIST load-on-startup id ID #IMPLIED>
<!ATTLIST servlet-mapping id ID #IMPLIED>
<!ATTLIST proxy-config id ID #IMPLIED>
<!ATTLIST sequential-search-timeout id ID #IMPLIED>
<!ATTLIST session-config id ID #IMPLIED>
<!ATTLIST session-timeout id ID #IMPLIED>
<!ATTLIST resource-env-ref id ID #IMPLIED>
<!ATTLIST resource-env-ref-name id ID #IMPLIED>
<!ATTLIST resource-env-ref-type id ID #IMPLIED>
<!ATTLIST resource-ref id ID #IMPLIED>
<!ATTLIST res-ref-name id ID #IMPLIED>
<!ATTLIST res-type id ID #IMPLIED>
<!ATTLIST res-auth id ID #IMPLIED>
<!ATTLIST res-sharing-scope id ID #IMPLIED>
<!ATTLIST security-constraint id ID #IMPLIED>
<!ATTLIST resource-collection id ID #IMPLIED>
<!ATTLIST resource-name id ID #IMPLIED>
<!ATTLIST sip-method id ID #IMPLIED>
<!ATTLIST proxy-authentication id ID #IMPLIED>
<!ATTLIST user-data-constraint id ID #IMPLIED>

<!ATTLIST transport-guarantee id ID #IMPLIED>
<!ATTLIST auth-constraint id ID #IMPLIED>
<!ATTLIST role-name id ID #IMPLIED>
<!ATTLIST login-config id ID #IMPLIED>
<!ATTLIST realm-name id ID #IMPLIED>
<!ATTLIST auth-method id ID #IMPLIED>
<!ATTLIST security-role id ID #IMPLIED>
<!ATTLIST security-role-ref id ID #IMPLIED>
<!ATTLIST role-link id ID #IMPLIED>
<!ATTLIST env-entry id ID #IMPLIED>
<!ATTLIST env-entry-name id ID #IMPLIED>
<!ATTLIST env-entry-value id ID #IMPLIED>
<!ATTLIST env-entry-type id ID #IMPLIED>
<!ATTLIST ejb-ref id ID #IMPLIED>
<!ATTLIST ejb-ref-name id ID #IMPLIED>
<!ATTLIST ejb-ref-type id ID #IMPLIED>
<!ATTLIST home id ID #IMPLIED>
<!ATTLIST remote id ID #IMPLIED>
<!ATTLIST ejb-link id ID #IMPLIED>
<!ATTLIST ejb-local-ref id ID #IMPLIED>
<!ATTLIST local-home id ID #IMPLIED>
<!ATTLIST local id ID #IMPLIED>
<!ATTLIST run-as id ID #IMPLIED>
<!ATTLIST pattern id ID #IMPLIED>
<!ATTLIST and id ID #IMPLIED>
<!ATTLIST or id ID #IMPLIED>
<!ATTLIST not id ID #IMPLIED>
<!ATTLIST equal id ID #IMPLIED>
<!ATTLIST contains id ID #IMPLIED>
<!ATTLIST exists id ID #IMPLIED>
<!ATTLIST subdomain-of id ID #IMPLIED>
<!ATTLIST var id ID #IMPLIED>
<!ATTLIST value id ID #IMPLIED>

15.6. 例

15.6.1. 基本例

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sip-app
  PUBLIC "-//Java Community Process//DTD SIP Application 1.0//EN"
  "http://www.jcp.org/dtd/sip-app_1_0.dtd">

<sip-app>
  <display-name>A Simple SIP Application</display-name>

  <context-param>
    <param-name>vxml-server</param-name>
    <param-value>sip:server17@vxml.example.com</param-value>
  </context-param>

  <servlet>
    <servlet-name>WeatherService</servlet-name>
    <servlet-class>com.example.Weather</servlet-class>
    <init-param>
      <param-name>weather-server</param-name>
      <param-value>http://example.com/weather</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>WeatherService</servlet-name>
    <pattern>
      <and>
        <equal>
          <var>request.method</var>
          <value>INVITE</value>
        </equal>
        <equal>
          <var>request.to.uri.user</var>
          <value>weather</value>
        </equal>
      </and>
    </pattern>
```

```
    </servlet-mapping>

    <session-config>
        <session-timeout>15</session-timeout>
    </session-config>
</sip-app>
```

付録 A リファレンス

[3pcc] J. Rosenberg, J. Peterson, H. Schulzrinne and G. Camarillo, “Best Current Practices for Third Party Call Control in the Session Initiation Protocol”, Internet Engineering Task Force, June 2002. Work in progress.

[CPL] J. Lennox and H. Schulzrinne, “CPL: A Language for User Control of Internet Telephony Services”, Internet Engineering Task Force, January 2002. Work in progress.

[JAF] B. Calder and B. Shannon, “JavaBeans Activation Framework Specification”, May 27, 1999.

[JLS] “The Java Programming Language Specification”, http://java.sun.com/docs/books/jls.

[JavaMail] Sun Microsystems, “JavaMail API Design Specification v1.2”, September 2000.

[JAXP] R. Mordani, J. D. Davidson, and S. Boag, “Java API for XML Processing”, February 6, 2001.

[refer] R. Sparks, “The SIP Refer Method”, Internet Engineering Task Force, November 25, 2002. Work in progress.

[RFC 1738] T. Berners-Lee, L. Masinter, and M. McCahill, “Uniform Resource Locators(URL)”, RFC 1738, December 1994.

[RFC 2278] N. Freed and J. Postel, “IANA Charset Registration Procedures”, RFC 2278, January 1998.

[RFC 2327] M. Handley and V. Jacobson, “SDP: Session Description Protocol”, RFC 2327, April 1998.

[RFC 2806] A. Vaha-Sipila, “URLs for Telephone Calls”, RFC 2806, April 2000.

[RFC 2976] S. Donovan, “The SIP INFO Method”, RFC 2976, October 2000.

[RFC 3261] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP: Session Initiation Protocol“, RFC 3261, June 2002.

[RFC 3262] J. Rosenberg and H. Schulzrinne, “Reliability of Provisional Responses in the Session Initiation Protocol (SIP)”, RFC 3262, June 2002.

[RFC 3265] A. B.Roach, “Session Initiation Protocol (SIP)-Specific Event Notification“, RFC 3265, June 2002.

[RFC 3325] C. Jennings, J. Peterson, and M. Watson, “Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks”, RFC 3325, November 2002.

[RFC 3327] D. Willis and B. Hoeneisen, “Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts“, RFC 3327, December 2002.

[RFC 3428] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle, “Session Initiation Protocol (SIP) Extension for Instant Messaging“, RFC 3428, December 2002.

[SERL] R. W. Steenfeldt and H. Smith, “SIP Service Execution Rule Language Framework and Requirements“, Internet Engineering Task Force, May 2001. Work in progress.

[Servlet API] D. Coward, “Java Servlet Specification, Version 2.3”, September, 2001.

[simple] Rosenberg et al., “SIP Extensions for Presence”, Internet Engineering Task Force, March 2002. Work in progress.

[timer] S. Donovan and J. Rosenberg, “Session Initiation Protocol Extension for Session Timer”, Internet Engineering Task Force, July 2002. Work in progress.

付録 B 用語集

Ø 日本語名称 (英語名称):
説明 (訳注) 以降は「日本語名称」 (「英語名称」): 「説明」の順で記述します。

Ø 「address-of-record」 (address-of-record):
address-of-record (AOR) とは、ある URI をユーザがいるかもしれない別の URI に変換できるロケーションサービスのあるドメインを示す SIP URI または SIPS URI です。通常、ロケーションサービスは登録によってそこに居ると見なします。AOR は、しばしばユーザの「公開アドレス」と考えられます。

Ø アプリケーション開発者 (Application developer):
SIP アプリケーションの生産者です。アプリケーション開発者のアウトプットは、サーブレットアプリケーション用のサーブレットクラスのセットとサポート用のライブラリやファイル (例えば、イメージ、圧縮されたアーカイブファイルなど) です。アプリケーション開発者は、一般的にアプリケーションの領域のエキスパートです。開発者は、サーブレットの環境と並列処理の考慮も含めてその影響について熟知しており、また、それに従って SIP アプリケーションを作ることを要求されます。

Ø アプリケーション構成者 (Application assembler):
アプリケーション開発者のアウトプットを入手し、それが配備可能なユニットであることを保証します。そのため、アプリケーション構成者のインプットは、サーブレットアプリケーション用のサーブレットクラス、JSP ページ、HTML ページ、また他のサポート用のライブラリやファイルです。アプリケーション構成者のアウトプットは、サーブレットアプリケーションのアーカイブ、または、開かれたディレクトリ上に配置されたサーブレットアプリケーションです。

Ø アプリケーションパス (Application path):
ある特定のダイアログに属するメッセージ (の受信) のために起動されるアプリケーションインスタンスのリストです。このリストは初期リクエストの処理の副産物として作られます。もし、初期リクエストがコンテナ内で作られていれば、UAC として振る舞うアプリケーションから成ります。もし、アプリケーションが初期リクエストに応答するなら、アプリケーションが record-routing を有効にしてプロキシする場合と同様に、UAS として振る舞うアプリケーションから成ります。アプリケーションパスは論理的な概念であり, 実装上は明示的に現れても現れなくても構いません。

Ø アプリケーションセッション (Application session):
アプリケーションインスタンスを表します。アプリケーションセッションは、アプリケーションデータの保管所として動作し、それが含んでいるプロトコルセッションにアクセスできます。

Ø バックトゥバックユーザエージェント (Back-to-Back User Agent):
バックトゥバックユーザエージェント(B2BUA)は、リクエストを受信して、それをユーザエージェントサーバ(UAS)として扱えるような論理的な要素です。リクエストにどのように応答するべきか決定する為に、ユーザエージェントクライアント (UAC) として振る舞い、そしてリクエストを生成します。プロキシサーバと違い、ダイアログの状態を管理します、

そして自分で生成したダイアログ上で送られる全てのリクエストを扱わなければなりません。それが UAC と UAS の連結であるため、その振る舞いのために明示的な定義は必要ありません。

Ø 呼 (Call):
呼は、通常、マルチメディア会話をするためにセットアップされるピア間のコミュニケーションを指す、非公式な用語です。

Ø コールレッグ (Call leg):
ダイアログの別名です[RFC 2543]。改訂された SIP 仕様の中では使われていません[RFC 3261]。

Ø コールステートフル (Call Statefull):
プロキシは、ダイアログが開始される INVITE リクエストから終了する BYE リクエストまで状態を管理していれば、コールステートフルです。コールステートフルプロキシは、必ずトランザクションステートフルですが、逆は必ずしも真ではありません。

Ø クライアント (Client):
クライアントは、SIP リクエストを送り、SIP レスポンスを受信するような任意のネットワーク要素です。クライアントは、ユーザ（人間）と直接に対話することもしないこともあります。ユーザエージェントクライアントとプロキシはクライアントです。

Ø コミットされたメッセージ (Committed message):
SIP メッセージオブジェクトはこの仕様に基づいて完全に処理され、そしてそれ以上の変更はするべきではありません。セクション 6.2 を参照して下さい。

Ø 会議 (Conference):
複数の参加者を含んだマルチメディアセッションです。

Ø デフォルトサーブレット (Default servlet):
配備記述子の中で、1 番最初に記述されているサーブレットです。

Ø デプロイヤ (Deployer):
デプロイヤは、アプリケーション開発者から提供されたサーブレットアプリケーションのアーカイブファイルやディレクトリ構造を 1 個以上取得し, 特定の実行環境にアプリケーションを配備します。実行環境は特定のサーブレットコンテナと SIP サーバを含んでいます。デプロイヤは、開発者によって宣言された外部環境への依存部分を全て解決しなければなりません。その役割を果たす為に、デプロイヤはサーブレットコンテナ提供者により与えられたツールを使用します。
デプロイヤは、特定の実行環境の専門家です。例えば、デプロイヤはアプリケーション開発者によって定義されたセキュリティ上の役割をサーブレットアプリケーションが配備される実行環境でのユーザグループやアカウントにマッピングする責任があります。

Ø ダイアログ (Dialog):
ダイアログは、しばらくの間持続する2つの UA 間のピアツーピアの SIP の関係です。ダイアログは、INVITE リクエストに対する 2xx レスポンスのような SIP メッセージによって確立されます。ダイアログは、呼識別子(call identifier)、ローカル tag、およびリモート tag により識別されます。ダイアログは、以前は[RFC 2543]のコールレッグとして知られていました。
基本の SIP 仕様では、INVITE-BYE だけがダイアログを生成/終了させるメカニズムとして定義されていました。しかし、ダイアログを開始可能な他のメソッドを定義するために拡張を許しています。イベントフレームワーク[RFC 3265] で定義された SUBSCRIBE/NOTIFY メソッドがこの例です。

Ø 下流 (Downstream):
ユーザエージェントクライアントからユーザエージェントサーバへのリクエストの流れの向きについて言及するときの、トランザクション内を進むメッセージの向きです。

Ø 最終レスポンス (Final response):
SIP トランザクションを終了しない暫定レスポンス(Provisional Response)とは逆に、SIP トランザクションを終了するレスポンスです。2xx, 3xx, 4xx, 5xx, および 6xx は全て最終レスポンスです。

Ø ヘッダ (Header):
ヘッダは、SIP メッセージについての情報を伝える SIP メッセージの構成要素です。ヘッダはヘッダフィールドのつながりとして構成されます。

Ø ヘッダフィールド (Header field):
ヘッダフィールドは、SIP メッセージヘッダの構成要素です。1 個以上のコンマで区切られたヘッダフィールド値か、または同じヘッダフィールド名を持つ(ヘッダフィールド値)から構成されます。

Ø ヘッダフィールド値 (Header field value):
ヘッダフィールド値は、一つの値です。それはヘッダフィールドの多くの中の一つをあらわします。

Ø ホームドメイン (Home Domain):
SIP ユーザにサービスを提供するドメインです。通常これは、登録の address-of-record 内の URI に存在するドメインです。

Ø 通知レスポンス (Informational Response):
暫定レスポンスと同じです。

Ø 初期リクエスト (Initial Request):
既にあるアプリケーションパスではなく、ルールのマッチングによりアプリケーションに送り出されるリクエストです。後続リクエストと比較してください。

Ø イニシエータ(Initiator)、発呼側(Calling Party)、発信者(Caller):
INVITE リクエストでセッション(やダイアログ)を開始する側です。発信者は、ダイアログを確立した最初の INVITE を送ってからそのダイアログの終了までこの役割を維持します。

Ø 招待 (Invitation):
INVITE リクエストのことです。

Ø 被招待者(Invitee)、被招待ユーザ(Invited User)、着呼側(Called Party)、着信者(Callee):
新しいセッションを開始する為に INVITE リクエストを受け取る側です。着信者は、INVITE を受け取ってからその INVITE によって確立されたダイアログの終了までこの役割を維持します。

Ø ロケーションサーバ (Location server):
ロケーションサービスを参照。

Ø ロケーションサービス (Location service):
ロケーションサービスは、着信者のいる可能性のある場所(一つまたは複数)を取得するために、SIP リダイレクションサーバまたはプロキシサーバによって使用されます。ロケーションサービスは 0個以上のコンタクトアドレスへの address-of-record キーのバインディングのリストを含みます。バインディングは様々な方法によって生成, 削除できます。この仕様では、バインディングを更新する REGISTER メソッドを定義する。(訳注: 結構、SIP 仕様書の Glossary からそのままの引用がある。「この仕様書では」は、SIP 仕様書の意味と思われる。)

Ø ルースルーティング (Loose Routing):
プロキシが Route ヘッダフィールドの処理の為に、この仕様で定義されている手順に従う場合、そのプロキシはルースルーティングであると言われます。この手順は、あて先に向かう途中で訪れる必要のあるプロキシのセット(Route ヘッダフィールド中に存在する)から、リクエストのあて先(Request-URI の中に存在する)を分離します。このメカニズムに準拠するプロキシはルースルータ(Loose Router)としても知られています。
(訳注:ここでは「あて先」としたが、本文中は「目的地」を使っている。)

Ø メッセージ (Message):
プロトコルの一部として SIP 要素の間で送られるデータです。SIP メッセージはリクエストかレスポンスのいずれかです。

Ø メソッド (Method):
メソッドは、サーバ上でリクエストが呼び出されることを意味する、主要な機能です。メソッドはリクエストメッセージ自身によって伝えられます。メソッドの例として、INVITE や BYE があります。

Ø アウトバウンドプロキシ (Outbound proxy):
クライアントからのリクエストを受け取るプロキシです。それは Request-URI で解決されたサーバでは無いかもしれま

せん。通常、UA にはマニュアルでアウトバウンドプロキシが設定されます。あるいは自動設定プロトコルによってアウトバウンドプロキシを知ることが出来ます。

Ø パラレルサーチ (Parallel Search)：
パラレルサーチでは、送られてくるリクエストを受け取ると、プロキシはユーザのいると思われる場所にいくつかのリクエストを発行します。シーケンシャルサーチのように一つのリクエストを発行して、最終レスポンスを待ってから次のリクエストを発行するのではなく、パラレルサーチでは、既に発行したリクエストの結果を待たずに複数のリクエストを発行します。

Ø プリンシパル (Principal)：
プリンシパルとは、認証プロトコルによって認証が可能な要素です。プリンシパルは、プリンシパル名(principal name)によって識別され、認証データ(authentication data)を使用して認証されます。プリンシパル名の内容と形式は認証プロトコルに依存します。

Ø プロトコルセッション (Protocol Session)：
プロトコル特有のセッションオブジェクトの通称です。プロトコルセッションは、ポイントツーポイントのシグナリング関係を表します。そして、その関係に関連したアプリケーションデータのためのリポジトリを提供します。例として、それぞれ SIP と HTTP Servlet 仕様で定義された SipSession と HttpSession インタフェースがあります。複数のプロトコルセッションが 1 個のアプリケーションセッションに所属するかもしれません。

Ø 暫定レスポンス (Provisional Response)：
進行状況を示す為にサーバが使用するレスポンスです。そして、暫定レスポンスは SIP トランザクションを終了しません。1xx レスポンスが暫定レスポンスであり、その他のレスポンスは最終レスポンスと見なされます。暫定レスポンスは、信頼性を持って送ることは出来ません。

Ø プロキシ(Proxy)、プロキシサーバ(Proxy Server)：
他のクライアントの代わりにリクエストを生成する為に、サーバやクライアントの両方として動作する中間のエンティティです。プロキシサーバは、主にルーチングの役割を行ないます。これは、リクエストが目的のユーザに「より近い」他のエンティティに送られることを保証することが、プロキシサーバの役割であることを意味します。プロキシはポリシーを強制するにも有用です（例えば、ユーザが電話をかける事を許可されているか明確にする）。プロキシは、リクエストメッセージを転送(forward)する前に、それを解釈し、必要であればその特定部分を書き換えます。

Ø 再帰 (Recursion)：
クライアントが 3xx レスポンスの Contact ヘッダフィールドに含まれる一つ以上の URI に対して新しいリクエストを生成するとき、クライアントは 3xx レスポンスで再帰すると言います。

Ø リダイレクトサーバ (Redirect Server)：
リダイレクトサーバは、受け取ったリクエストに対して 3xx レスポンスを生成するユーザエージェントサーバです。それにより、別の URI セットにコンタクトすることをユーザに指示します。

Ø 登録サーバ (Registrar):
登録サーバは、REGISTER リクエストを受け付け、それが処理するドメインのロケーションサービスに、
そのリクエストで受け取った情報を設定するサーバです。

Ø レギュラートランザクション (Regular Transaction):
レギュラートランザクションとは、INVITE, ACK, CANCEL 以外のメソッドを含む全てのトランザクションです。

Ø リクエスト (Request):
特定の操作を実行する為に、クライアントからサーバに送られる SIP メッセージのことです。

Ø レスポンス (Response):
クライアントからサーバに送られたリクエストのステータスを示す為に、サーバからクライアントに送られる SIP メッセージのことです。

Ø リングバック (RingBack):
リングバックとは、発呼側のアプリケーションによって作られる、着呼側が呼出中 (ringing) であることを示すシグナル音です。

Ø 役割 (開発)、Role(development):
サーブレットアプリケーションの開発や配備、そして実行の期間を通じて、様々な関係者によって行われる行動や責任のことです。いくつかのシナリオでは、一つの関係者が複数の役割を実行しても良いです。また他のシナリオでは、各役割は別の関係者により実行されます。

Ø 役割 (セキュリティ)、Role(Security):
アプリケーション開発者はアプリケーション内で抽象概念(abstruct notion)を利用します。そして抽象概念はデプロイヤによりセキュリティポリシードメイン内のユーザ、ユーザのグループにマッピングすることが出来ます。

Ø ルートセット (Route set):
ルートセットとは、特定のリクエストを送るときに横断しなければならないプロキシのリストです。ルートセットは Record-Route のようなヘッダで知ることも出来るし、設定することも出来ます。

Ø セキュリティポリシードメイン (Security Policy Domain):
セキュリティサービスにおけるセキュリティ管理者によりセキュリティポリシーが定義され、そして強制される範囲です。セキュリティポリシードメインはまた、時々 レルム(realm) と呼ばれます。(訳注: realm 領域、範囲)

Ø セキュリティテクノロジドメイン (Security technology domain):
セキュリティポリシーを強制する為に同じセキュリティ機能 (例えば Kerberos) を使った範囲です。一つのセキュリティテクノロジドメインの中に複数のセキュリティポリシードメインを置くことも出来ます。

Ø サーバ (Server):
サーバとは、サービスを提供する為にリクエストを受け取り、そしてそれらのリクエストにレスポンスを送り返すようなネットワークの構成要素です。サーバの例として、プロキシ、ユーザエージェントサーバ、リダイレクトサーバ、および登録サーバがあります。

Ø シーケンシャルサーチ (Sequential search):
シーケンシャルサーチでは、プロキシサーバは各コンタクトアドレスを順番に試みます。既に試したコンタクトアドレスが最終レスポンスを返した後にのみ次に進みます。2xx や 6xx クラスの最終レスポンスは、常にシーケンシャルサーチを終了させます。

Ø サーブレットアプリケーションアーカイブ (Servlet Application Archive):
サーブレットアプリケーションの全てのコンポーネントを含んだ一つのファイルです。このアーカイブファイルは、指定されたアプリケーションコンポーネント(いくつかまたは全て)を使って、通常の JAR ツールを使って作成されます。サーブレットアプリケーションアーカイブファイルは、".sar"拡張子によって識別できます。SAR ファイルの設計は Web アプリケーションアーカイブ(.war)ファイルフォーマットに由来しています。しかし、異なったプロトコル、例としては SIP と HTTP、に関するサーブレットや配備記述子を含むことが出来ます。

Ø サーブレットコンテナプロバイダ (Servlet Container Provider):
サーブレットコンテナと呼ばれる実行環境や可能であれば SIP サーバを提供するベンダです。実行環境では、SIP アプリケーションが実行できると共に、SIP アプリケーションを配備するために必要なツールも動作します。

コンテナプロバイダは HTTP レベルのプログラミングの専門知識があります。この仕様では SIP サーバとサーブレットコンテナの間のインタフェースを規定していない為、コンテナと SIP サーバの間で要求される機能の実装をどう分割するかはコンテナプロバイダに任されています。

Ø サーブレット定義 (Servlet definition):
Servlet インタフェースのクラスによる実装には、完全に限定された(fully qualified)クラス名として固有の名前を与えます。サーブレット定義には、複数の初期化パラメータを与えることができます。

Ø サーブレットマッピング (Servlet Mapping):
サーブレット定義は、サーブレットコンテナによって URL パスパターンと関連付けられます。そのパスパターンに来たリクエストは全てサーブレット定義で関連付けられたサーブレットによって処理されます。

Ø セッション (Session):
SDP 仕様によれば「マルチメディアセッションとは、マルチメディアを送る側と受ける側および送り手から受け手に流れるデータストリームのセットのことです。マルチメディア会議はマルチメディアセッションの一例です。」([RFC2327])(SDP で定義されたセッションは、一つ以上の RTP セッションから成ります。)この定義のように、着信者を(異なる呼によって)同じセッションに何度も招待することが出来ます。SDP が使われる場合、セッションは、ユーザ名、セッショ

ン ID、ネットワークタイプ、アドレスタイプ、および origin フィールドのアドレス要素を結合したもので定義します。

Ø SIP アプリケーション (SIP application):
SIP サーブレットと静的リソースの集まりです。静的リソースは、音声による注意、文法、VoiceXML スクリプト、またその他のデータを含みます。SIP アプリケーションは、アーカイブにパッケージ化することもできるし、開かれたディレクトリ上に置くことも出来ます。

互換性のあるサーブレットコンテナは全て、SIP アプリケーションを受け付けて、その実行環境にそのコンテンツを配備しなければ成りません。これは、コンテナがサーブレットアプリケーションアーカイブファイルから直接にアプリケーションを実行できることを意味しても良いし、サーブレットアプリケーションのコンテンツをコンテナの適当な場所に移動することを意味しても良い。

SIP アプリケーションと SIP Servlet API の関係は Web アプリケーションと HTTP Servlet API の関係に等しい。

Ø SIP トランザクション (SIP transaction):
SIP トランザクションは、クライアントとサーバの間で発生し、クライアントからサーバに送られた最初のリクエストから、サーバからクライアントに送られた最終レスポンス (非 1xx) までの全てのメッセージから成ります。リクエストが INVITE で最終レスポンスが非 2xx の場合、トランザクションはそのレスポンスに対する ACK も含みます。INVITE リクエストに対する 2xx レスポンスへの ACK は、別のトランザクションです。

Ø SIP/Web アプリケーション、分散可能 (SIP/web application, distributable):
同じホストかまたは別のホストの上で走る複数の Java virtual machine をまたがって、分散したサーブレットコンテナに配備できるように書かれた SIP または Web のアプリケーションです。そのようなアプリケーションでは、配備記述子に distributable 要素を記述します。

Ø ステートフルプロキシ (Statefull Proxy):
リクエストを処理する間、この仕様で定義されているクライアントとサーバのトランザクションステートマシンを維持する、論理的な実体です。また、トランザクションステートフルプロキとも呼びます。ステートフルプロキシの動作はセクション 16 で詳しく定義されています。そして、(トランザクション) ステートフルプロキシは、コールステートフルプロキシと同じではありません。

Ø ステートレスプロキシ (Stateless Proxy):
リクエストを処理する時、この仕様で定義されているクライアントとサーバのトランザクションステートマシンを維持しない、論理的な実体です。ステートレスプロキシは、受け取る全てのリクエストを下流へ、および受け取る全てのレスポンスを上流へと転送します。

Ø ストリクトルーティング (Strict routing):
プロキシが RFC2543 および RFC3261 以前の多くのインターネットドラフトのルート処理ルールに従う場合、そのプロキシはストリクトルーティングと呼ばれます。そのルールは、Route ヘッダフィールドが存在する時に、プロキシに

Request-URI のコンテンツを破壊させます。ルースルーティングの動作を選んで、ストリクトルーティングの動作は RFC3261 では使用しません。ストリクトルーティングを行うプロキシは、ストリクトルータとも呼ばれます。

Ø 後続リクエスト (Subsequent request)：
アプリケーションに引き渡される時に、ルールの一致によってではなく、前にダイアログを生成した初期リクエストを処理した時に作られたアプリケーションパスに基づいて行われるような、リクエストです。

Ø システム管理者 (System Administrator)：
サーブレットコンテナの構成や管理に責任を持つ人です。システム管理者は同様に、運用中に、配備されたアプリケーションが正常に動作しているか監視する責任があります。

この仕様はシステムの運用と管理のための規約については定義していません。管理者は、一般的にはこれらのタスクを遂行するためにコンテナ供給者およびサーバベンダによって提供されるランタイムモニタリングおよび管理ツールを使用します。

Ø システムヘッダ (System headers)：
SIP サーブレットコンテナによって管理され、サーブレットが setHeader や addHeader を呼び出すことで直接変更してはいけないヘッダです。システムヘッダには Call-ID、From、To、CSeq、Via、Record-Route、Route ヘッダが含まれます。また、セッションシグナリングアドレスを指定するために使われる時、例えば、INVITE の中や INVITE に対する 200 番のレスポンスの場合、「Contact」も同様にシステムヘッダに含まれます。システムヘッダについては、6.4.2 で議論します。

Ø TLS (TLS)：
Transport Layer Security の略です。TCP のコネクションを保護し、完全性、機密性、リプレイへの防御、そして認証を提供する、Layer4の手段です。

Ø Uniform resource locator(URL)：
インターネットを使ったリソースの位置およびアクセスのための情報のコンパクトなストリング表現[RFC 1738]。

SIP と SIPS URI は文法的にはメールのあて先の URL に似ています、すなわち、一般的に **sip:user@host** の形式です。ユーザ部は、ユーザ名か電話番号です。ホスト部は、(完全に修飾されたドメイン名) FQDN かまたは、数字の IP アドレスです。SIP URI は、SIP メッセージの中で SIP リクエストの発信者(From)、現在のあて先 (Request-URI)、そして最終の受信者 (To) を示す為に、またリダイレクションアドレス (Contact) を指定する為に使われます。ハイパーリンク (例えば Web ページ) として使う場合、SIP URI は特定のユーザかまたは SIP を使ってコンタクトできるサービスを示します。

Ø 上流 (Upstream)：
ユーザエージェントサーバからユーザエージェントクライアントに戻るレスポンスの流れの向きについて言及するときの、トランザクション内を進むメッセージの向きです。

Ø URL エンコードされた (URL-encoded):
RFC2396 のセクション 2.4 ([RFC 1738, section 2.2]) に則ってエンコードされた文字列です。

Ø ユーザエージェントクライアント (User agent client , UAC):
ユーザエージェントクライアントは、新しいリクエストを生成し、それを送る為にクライアントトランザクションのステートマシンを使用する、論理的な要素です。UAC の役割は、そのトランザクションが存続する期間だけで終了します。言い換えれば、あるソフトウェアがリクエストを開始した場合、それはそのトランザクションが存続するあいだ UAC として動作します。その後それがリクエストを受け取った場合、そのトランザクションの処理の為のユーザエージェントサーバの役割を果たします。

Ø ユーザエージェントサーバ (User agent server, UAS):
ユーザエージェントサーバは、SIP リクエストに対してレスポンスを生成する、論理的な要素です。レスポンスは、リクエストを受け入れるか、リクエストを拒否するか、またはリクエストをリダイレクトします。この役割は、そのトランザクションが存続する間だけで終了します。言い換えれば、あるソフトウェアがリクエストに応答する場合、それはそのトランザクションが存続するあいだ UAS として動作します。その後それがリクエストを生成した場合、そのトランザクションの処理の為にユーザエージェントクライアントの役割を果たします。

Ø ユーザエージェント (User agent, UA):
ユーザエージェントクライアントとしてもユーザエージェントサーバとしても動作できる、論理的な要素です。

Ø Web アプリケーション (Web application):
サーブレット、JSP ページ、HTML ドキュメントと他の Web リソースの集まりです。Web リソースは、イメージファイルや圧縮されたアーカイブやその他のデータを含むかもしれません。Web アプリケーションは、アーカイブにパッケージ化することもできるし、開かれたディレクトリ上に置くことも出来ます。

互換性のあるサーブレットコンテナは全て、Web アプリケーションを受け付けて、その実行環境にそのコンテンツを配備しなければ成りません。これは、コンテナが Web アプリケーションアーカイブファイルから直接にアプリケーションを実行できることを意味しても良いし、Web アプリケーションのコンテンツをコンテナの適当な場所に移動することを意味しても良い。

Ø Web アプリケーションアーカイブ (Web application archive):
Web アプリケーションの全てのコンポーネントを含んだ一つのファイルです。このアーカイブファイルは、指定されたアプリケーションコンポーネント (いくつかまたは全て) を使って、通常の JAR ツールを使って作成されます。Web アプリケーションアーカイブファイルは、".war"拡張子によって識別できます。拡張子".jar"の代わりに新しい拡張子が使われます、なぜなら、".jar"拡張子は、クラスのセットを含み、クラスパスに置くことができ、アプリケーションを起動する為に GUI を使ってダブルクリックする、ようなファイル用に予約されている為です。Web アプリケーションアーカイブのコンテンツは、その様な仕様には向かないため、新しい拡張子が望ましい。