BEA WebLogic Server

WebLogic File Services ユーザーズ ガイド

BEA WebLogic Server 6.1

マニュアルの日付 : 2002 年 6 月 24 日

著作権

限定的権利条項

商標または登録商標

WebLogic File Services ユーザーズ ガイド

日付	ソフトウェアのバージョン
2002 年 6 月 24 日	BEA WebLogic Server バージョン 6.1

目次

このマニュアルの内容

1. WebLogic File サービスの使い方

このマニュアルの内容

このマニュアルでは、クライアントサイドからサーバ上のネイティブ オペレーティング システム ファイルへのアクセスに利用する WebLogic File サービスの使い方について説明します。

このマニュアルの内容は以下のとおりです。

- 第 1 章「WebLogic File サービスの使い方」では、WebLogic File について紹介し、WebLogic File API、および WebLogic File を使用したプログラミング方法について説明します。

対象読者

このマニュアルは、WebLogic Server 環境で動作する Java アプリケーション内のファイルの読み書きに関心があるアプリケーション開発者を主な対象としています。WebLogic Server プラットフォーム、Java および Java 2, Enterprise Edition (J2EE) プログラミング、ファイル I/O の概念に読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックして、指定してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 ファイルずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[PDF files] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をインストールしていない場合は、Adobe の Web サイト（http://www.adobe.co.jp/）で無料で入手できます。

サポート情報

BEA WebLogic Server のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで **docsupport-jp@bea.com** までお送りください。寄せられた意見については、WebLogic Server のドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェアの名前とバージョン、およびドキュメントのタイトルと日付をお書き添えください。

本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSUPPORT（www.bea.com）を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所

- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
太字	用語集で定義されている用語を示す。
〔Ctrl〕+ 〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例：
	`import java.io.Serializable;`
	`public String getName();`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な箇所を示す。 例： `void **commit** ( )`

表記法	適用
monospace italic text	コード内の変数を示す。 例： `String *expr*`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 例： LPT1 SIGNON OR
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。
[ ]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。 例： `buildobjclient [-v] [-o name ] [-f *file-list*]...[-l *file-list*]...`
|	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	コマンドラインで以下のいずれかを示す。 ♦ 引数を複数回繰り返すことができる ♦ 任意指定の引数が省略されている ♦ パラメータや値などの情報を追加入力できる 実際には、この省略符号は入力しない。 例： `buildobjclient [-v] [-o name ] [-f *file-list*]...[-l *file-list*]...`
. . .	コードサンプルまたは構文で項目が省略されていることを示す。 実際には、この省略符号は入力しない。

1　WebLogic File サービスの使い方

この章では、WebLogic File サービスについて説明します。内容は以下のとおりです。

- WebLogic File サービスの概要
- WebLogic File API リファレンス
- WebLogic File サービスを使用したプログラミング

WebLogic File サービスの概要

WebLogic File は、クライアントサイドからサーバ上のネイティブ オペレーティング システム ファイルへの高速なアクセスを実現します。クライアント API は、Java（`java.io.InputStream` と `java.io.OutputStream`）の最低限の共通機能を拡張したものです。そのため、リモート ファイルの操作に特化した追加サービスと一緒に、既存のコード内でシームレスに使用できます。

WebLogic File は、サービスとしてロギング、インスツルメンテーション、ワークスペースのような他のすべての WebLogic 機能も利用できます。File サービスを含む WebLogic のすべてのコンポーネント ベースのサービスは、WebLogic フレームワークに統合され、アクセスとリソースを共有できます。それらの API は、ネットワーク化された複雑なアプリケーションの構造を簡潔化する共通の側面を多数共有しています。アプリケーションでこれらのサービスを使用する場合、それらはオブジェクトへのアクセスとクライアント リソースを共有できます。

WebLogic File では、他の WebLogic サービスと同じように、クライアントはファクトリ メソッドを使用して `T3FileInputStream` と `T3FileOutputStream` オブジェクトを生成します。これらのクラスでは、既存のクライアント コードにプラグインできるように、標準 Java `InputStream` と `OutputStream` クラスを拡張しています。また、リモート ファイル ストリーム固有の追加メソッドも提供します。

WebLogic File は、要求のサイズとは無関係なサイズのバッファでのデータの送信や、readAhead と writeBehind バッファリングを使用することによって、ネットワークでの読み書きのパフォーマンスを高めます。この実装では、いくつかの方法でデータ送信速度を向上させます。

- アプリケーションが要求するサイズとは異なるサイズのバッファでデータを送信します。アプリケーションは、性能に悪影響を与えることなく小規模な要求を多数行うことができます。
- クライアントは、データの先読みを実行します。つまり、アプリケーションよりも先にバッファを自動的に要求します。アプリケーションが 1 つのバッファ データを処理している間に、次のバッファが同時に取り出されます。
- クライアントはデータの後書きを実行します。つまり、サーバ上のディスクにフラッシュされた以上のものを、アプリケーションを使ってバッファに書き込みます。アプリケーションがデータのバッファの準備をしている間に、以前のバッファがディスクに書き込まれます。未処理バッファがすべてフラッシュされたという確認を受け取るまで、フラッシュ処理はクライアント上でブロックします。

アプリケーションは、送信バッファ サイズ、データ先読みバッファ数、データ後書きバッファ数を指定することができ、デフォルト値を使用することもできます。デフォルトのバッファ サイズは 100K で、データ先読みバッファとデータ後書きバッファのデフォルトはどちらも 1 です。

WebLogic File によって設定されるデフォルト値は、一般に最高の速度を得るための最適値です。デフォルトを使用しない場合は、以下のヒントを参考にして他の値を選択してください。

- バッファ サイズの設定。一般に、送信バッファ サイズが大きければ大きいほど、送信の実際のスピードは速くなります。その違いは、きわめて大きくなります。1K バッファを使用する場合、100K バッファに比べてスピードが 1 桁遅くなります。ただし、大きなバッファはクライアント サイドでより多くのメモリが必要となるため、コンフィグレーションに応じて最も効率的な設定を行う必要があります。
- *readAhead* および *writeBehind* バッファの設定。readAhead と writeBehind の最適値は、転送速度に対するアプリケーションのバッファ処理速度によって異なります。常に遅いアプリケーションでは、readAhead と writeBehind を 1 つにすると最大の効果が得られます。常に速いアプリケーションでは、readAhead と writeBehind を増やしても、効果はまったくありません。したがって、ほとんどの場合、デフォルトの 1 が最適値となります。しかし、

アプリケーションがバッファを処理する速度が変動する場合、そのアプリケーションが常にその最高速度で動作できるように `readAhead` と `writeBehind` を増やすことができます。

このマニュアルには、WebLogic File API に固有の情報が記載されています。また、『WebLogic Server アプリケーションの開発』も参照してください。Java で初めて `InputStream` と `OutputStream` を扱う場合は、JavaSoft チュートリアルも参照してください。

WebLogic File API リファレンス

weblogic.io.common パッケージは、以下のクラスとインタフェースで構成されています。

```
パッケージ weblogic.io.common
クラス java.lang.Object
  インタフェース weblogic.io.common.IOServicesDef
    クラス java.io.InputStream
      クラス weblogic.io.common.T3FileInputStream
    クラス java.io.OutputStream
      クラス weblogic.io.common.T3FileOutputStream
  インタフェース weblogic.io.common.T3File
  インタフェース weblogic.io.common.T3FileSystem
  クラス java.lang.Throwable
    (java.io.Serializable を実装)
    クラス java.lang.Exception
    クラス weblogic.common.T3Exception
```

WebLogic File のオブジェクトとクラス

`weblogic.io.common.T3File`
`weblogic.io.common.T3FileSystem`

インタフェース `T3File` および `T3FileSystem` は、T3File と T3FileSystem を定義します。T3File はローカル（通常はクライアントサイド）またはリモート（通常はサーバサイド）のファイルを表し、同じくローカルまたはリモート ファイルを表す T3FileSystem によって作成されます。T3File と T3FileSystem を使用すると、ローカル ファイルとリモート ファイルを均等に扱うコードを簡単に作成できます。これらの

インタフェースのオブジェクトは、WebLogic フレームワーク内のすべてのサービス関連のオブジェクトと同様に、オブジェクト ファクトリに対する要求によって割り当てられます。これによって、開発者はリソースを細かく管理することができます。

`weblogic.io.common.T3FileOutputStream`
`weblogic.io.common.T3FileInputStream`

`weblogic.io.common` パッケージの 2 つのクラス（`T3FileInputStream` と `T3FileOutputStream`）は、サーバサイドの読み書きアクセスをファイルに提供します。

`weblogic.io.common.IOServicesDef`
`weblogic.common.T3ServicesDef`

クラス変数 `services` を使用して、WebLogic クライアントは、`weblogic.common.T3ServicesDef` のメソッドを通して WebLogic Server のサービスにアクセスします。WebLogic Files と WebLogic File Systems には、メソッド `T3ServicesDef.io()` を通してアクセスし、`weblogic.io.common.IOServicesDef` オブジェクトを返します。`IOServicesDef` インタフェースには、IOServices オブジェクト ファクトリから T3FileSystem を要求するメソッドがあります（「T3File 関連オブジェクトの作成」を参照）。`IOServicesDef.getFileSystem()` に引数としてクライアントから `fileSystem` の名前を入力すると、T3FileSystem オブジェクトが返されます。サーバサイド オブジェクトからは、空の文字列または `null` を引数として `IOServicesDef.getFileSystem()` を呼び出します。これは、サーバの作業ディレクトリに対応するファイル システムへのポインタを返します。

T3FileSystem インタフェースには `IOServicesDef` オブジェクト ファクトリから T3File を要求するためのメソッドがあり、T3File インタフェースにはそのファイルを読み書きするための `T3FileInput/OutputStream` を要求するメソッドがあります。

以下のコードに、T3FileSystem リモート インタフェース、T3File、およびファイルに書き込むための OutputStream をクライアントがどのように取得するかを示します。

```
T3ServicesDef t3services;
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "t3://localhost:7001");
env.put(Context.INITIAL_CONTEXT_FACTORY,
          weblogic.jndi.WLInitialContextFactory.class.getName());
Context ctx = new InitialContext(env);
```

```
t3services = (T3ServicesDef)
ctx.lookup("weblogic.common.T3Services");
ctx.close();
T3FileSystem myFS = t3services.io().getFileSystem("usr");

T3File myFile = myFS.getFile("myDirectory/myFilename");
T3FileOutputStream t3os = myFile.getFileOutputStream();
t3os.write(b);
```

起こり得る例外を扱うために、try/catch ブロックでそのコードを囲む必要があります。

T3File の T3FileInputStream または T3FileOutputStream を取得するには、T3File オブジェクト上で T3File.getFileInputStream() または T3File.getFileOutputStream() を直接呼び出します。T3FileInputStream オブジェクトと T3FileOutputStream オブジェクトは、どちらも標準 java.io.* クラスを拡張します。

ファイルを読み書きするための WebLogic Server の設定

WebLogic File サービスを使用する前に、クライアントが使用する 1 つまたは複数のパスのプレフィックス（*fileSystem*）を設定しておく必要があります。File T3 サービスの名前属性とパス属性は、Administration Console で設定します。たとえば、ファイル システム名 users をサーバ ホスト上のパス \usr\local\tmp にマップするには、名前を users、パスを \usr\local\tmp として指定します。

IOServicesDef ファクトリから T3FileSystem を要求するときは（最終的には T3File の作成と、入出力ストリームを使用した T3File の読み書きに使用される）、getFileSystem() メソッドの引数として登録されている fileSystem 名を使用します。返される T3FileSystem オブジェクトは、指定した fileSystem にマップされます。

セキュリティ上の理由から、T3Client はファイル システム名の一部として登録されている最下位のディレクトリよりも上位のディレクトリのファイルにはアクセスできません。ファイル名には、ドット ドット（..）を組み込むことはできません。これを行うと、例外が送出されます。たとえば、\users\..\filename を読み出したり書き込んだりしようとすると、例外が発生します。

注意： Windows NT システム上のプロパティ ファイルでプロパティを設定するとき、シングル バックスラッシュ（\）はエスケープ文字として解釈されるため使用できません。プロパティを設定するときにシングル バックスラッシュを使用すると、次のようなエラー メッセージが表示されます。

```
java.io.FileNotFoundException:Remote file name <filename>
malformed
```

この場合、次の例のように、ダブル バックスラッシュを使用します。

```
weblogic.io.volume.vol=c:\\remote\\temp
```

または、代わりにパーサによって Window スタイルの構文に正しくマップされるフォワード スラッシュを使用してください。

```
weblogic.io.volume.vol=c:/remote/temp
```

T3File 関連オブジェクトの作成

以下の例では、リモートの T3File を読み書きするために必要な入出力ストリーム要求を取得する方法を示します。T3FileSystem インタフェースからリモート T3File を取得します。ここで、users は Administration Console を使って指定される fileSystem の名前で、WebLogic Server ホスト上の絶対パス \usr\local\users にマップされます。

```
T3ServicesDef t3services = getT3Services("t3://localhost:7001");
// IOServicesDef ファクトリから T3FileSystem オブジェクトを取得する
// 登録済み fileSystem を引数として指定する
T3FileSystem myFS = t3services.io().getFileSystem("users");
// T3FileSystem から T3File を取得する
T3File myFile = myFS.getFile("ben/notes");
// ファイルに書き出すための OutputStream を取得する
T3FileOutputStream t3os = myFile.getFileOutputStream();
// バイト 「b」を OutputStream に書き出す
t3os.write(b);
```

このコードでは、1 バイトが作成され、WebLogic Server ホストのパス \usr\local\users\ben\notes にマップされるファイルに書き出されます。

メソッド getT3Services() は、クラス weblogic.common.T3Client に存在します。このメソッドは、クライアントに追加できます。

この簡単な例は、最も一般的な使い方を示したものです。ほかにも、最初に T3FileSystem または T3File オブジェクトを作成せずに、直接 T3FileInputStream または T3FileOutputStream を要求できる一連のコンビニエンス メソッドを使用して IOServicesDef ファクトリから特定の T3File 関連オブジェクトを要求する方法があります。

以下に、IOServicesDef ファクトリが提供するコンビニエンス メソッドの使用例を示します。

pathname 引数と一緒に `getFileInput/OutputStream()` メソッドを呼び出すことによって、IOServicesDef ファクトリから直接 T3FileInputStream または T3FileOutputStream オブジェクトを要求できます。次のようにします。

`/registeredFileSystem/fileName`

`registeredFileSystem` は Administration Console でパス属性として登録したマウントポイントで、*`fileName`* は宛先ファイルの名前です。

T3FileSystem 上で呼び出されたメソッドから T3FileInputStream または T3FileOutputStream オブジェクトを取得せず、直接そのオブジェクトを要求するときは、fileSystem の名前の先頭にスラッシュを挿入する必要があります。これを行わないと、サーバが次のようなエラーを生成します。

`java.io.FileNotFoundException:Remote file name` *`filename`* `is relative`

この T3FileInputStream オブジェクトは、デフォルトのバッファ サイズと readAhead を使用します。デフォルト設定のバッファ サイズと readAhead/writeBehind を使用しない場合は、別のファクトリ メソッドを使用してこれらの値を設定することができます。次の例では、InputStream オブジェクトは 1024 バイトのバッファ サイズと 3 つの readAhead バッファで作成されます。

```
int bufferSize = 1024;
int readAhead = 3;

T3ServicesDef t3services = getT3Services("t3://localhost:7001");
InputStream is =
    t3services.io().getFileInputStream("/users/myfile",
                                       bufferSize,
                                       readAhead);
```

次の例では、OutputStream オブジェクトは 1024 バイトのバッファと 2 つの writeBehind バッファで作成されます。`getT3Services()` の詳細については、T3Services クラスの javadoc を参照してください。

```
int bufferSize = 1024;
int writeBehind = 2;

T3ServicesDef t3services = getT3Services("t3://localhost:7001");
Outputstream os =
  t3services.io().getFileOutputStream("/users/myfile",
                                      bufferSize,
                                      writeBehind);
```

エラーが発生すると、そのファクトリ メソッドは例外 `weblogic.common.T3Exception` を送出します。これは、ネストされた例外の要因となります。

T3FileSystem と T3File

`weblogic.io.common.T3FileSystem`

T3FileSystem は、T3File から構成されています。T3File を作成および管理するには、ファイルの読み書きに使用する T3FileInput/OutputStream を作成します。T3FileSystem は、クライアント上のローカル ファイル システムか、または WebLogic Server 上のリモート ファイル システムを表します。これらを使用すると、ローカル ファイル システムとリモート ファイル システムを均等に扱うコードを簡単に作成できます。

IOServicesDef ファクトリから T3FileSystem を要求するには、`getFileSystem()` メソッドを使用します。`IOServicesDef.getFileSystem()` に引数としてクライアントから fileSystem の名前を入力すると、T3FileSystem オブジェクトが返されます。サーバサイド オブジェクトからは、空の文字列または `null` を引数として `IOServicesDef.getFileSystem()` を呼び出します。これは、サーバの作業ディレクトリに対応するファイル システムへのポインタを返します。`T3FileSystem` インタフェースには、ファイル システム依存型のファイル区切り文字列と、ファイル システム依存型のパス区切り文字列を返す他のメソッドが用意されています。このインタフェースには、中間の T3File オブジェクトを作成することなく、ファイル Input/OutputStreams に直接アクセスできるさらに便利なメソッドも含まれています。

`weblogic.io.common.T3File`

`T3FileSystem.getFile()` メソッドの中の 1 つを呼び出すことによって、T3File を要求します。T3FileSystem 同様、T3File はローカル ファイルまたはリモート ファイルのいずれかを表すことができます。

Input/OutputStreams を使用してファイルを読み書きするメソッドのほかに、このインタフェースにはアクセサリ メソッドも存在します。アクセサリ メソッドで行うのは、T3File オブジェクトに関連付けられているファイル名とパスの取得、親ディレクトリの取得、ファイルが存在しかつ正常な T3file であることの確認、ファイルへの読み書きの検証、長さと最終更新日の確認、名前の変更、ディレクトリの作成、その他のファイルに関連するタスクです。

T3FileInputStream クラス

`weblogic.io.common.T3FileInputStream`

通常、`T3File.getFileInputStream()` メソッドを呼び出して T3FileInputStream を作成し、クラス `T3FileInputStream` のオブジェクトを返します。このクラスは、標準 `java.io.InputStream` クラスを拡張し、新たなメソッドを 2 つ提供します。

```
public int bufferSize();
```

現在のバッファ サイズを返します。

```
public int readAhead();
```

現在のデータ先読みバッファの数を返します。

`T3FileInputStream` にある他の 2 つのメソッドの実装は重要で、`java.io.InputStream` 内でメソッドをオーバーライドします。

- メソッド `available()` は、クライアント上でバッファされた未読データのバイト数を返します。この数は、バッファ サイズ x （1 + データ先読みバッファ数）より大きくなることはありません。
- メソッド `skip()` は、データ先読みで要求されたデータを廃棄することによって開始され、最終的にはサーバに要求を発行して残りのデータがあればそれをスキップします。

  現在、T3FileInputStream は、`java.io.InputStream.mark()` および `java.io.InputStream.reset()` メソッドをサポートしていません。

T3FileOutputStream クラス

`weblogic.io.common.T3FileOutputStream`

通常、`T3File.getFileOutputStream()` メソッドを呼び出して `T3FileOutputStream` を作成し、クラス T3FileOutputStream のオブジェクトを返します。このクラスは、標準 `java.io.OutputStream` クラスを拡張し、新たなメソッドを 2 つ提供します。

```
public int bufferSize();
```

現在のバッファ サイズを返します。

```
public int writeBehind();
```

現在のデータ後書きバッファの数を返します。`T3FileOutputStream` にある他の 2 つのメソッドの実装は重要で、`java.io.OutputStream` 内でメソッドをオーバーライドします。

- メソッド `flush()` は、未処理バッファがすべてサーバにフラッシュされたという確認を受け取るまで、クライアント上でブロックします。
- `close()` メソッドは、自動 `flush()` を実行します。

  ファイルが書き込まれている間にサーバ上でエラーが発生した場合、クライアントには非同期で通知され、後続のすべての操作（`write()`、`flush()`、または `close()`）は `java.io.IOException` を生成します。

WebLogic File サービスを使用したプログラミング

次に、アプリケーション内で T3File 関連オブジェクトを要求して使用する方法について順を追って説明します。

- 手順 1. パッケージのインポート
- 手順 2. リモート T3Services インタフェースの取得
- 手順 3. T3FileSystem と T3File の作成

- 手順 4. OutputStream オブジェクトの作成と使用
- 手順 5. InputStream オブジェクトの作成と使用

これらの手順では、コード例が示してあります。

手順 1. パッケージのインポート

プログラムにインポートするパッケージ以外にも、WebLogic File アプリケーションは以下のパッケージをインポートします。

```
import java.io.*;
import weblogic.common.*;
import weblogic.io.common.*;
```

手順 2. リモート T3Services インタフェースの取得

WebLogic クライアント アプリケーションから、WebLogic Server 上にある T3ServicesDef リモート ファクトリ インタフェース経由で T3File サービスにアクセスします。クライアントは、JNDI ルックアップを通して T3Services オブジェクトへのリモート スタブを取得します。`getT3Services()` というメソッドを定義してリストに登録します。このメソッドをクライアントに追加すると、T3Services スタブにアクセスできます。`getT3Services()` の詳細については、T3Services クラスの javadoc を参照してください。

次のように、WebLogic Server の URL を引数とするメソッドを簡単に呼び出すことができます。

```
T3ServicesDef t3services = getT3Services("t3://weblogicurl:7001")
```

手順 3. T3FileSystem と T3File の作成

一般に、ファイルの読み書きを開始するには、以下の手順を実行します。

- T3FileSystem システムを取得します。
- T3FileSystem オブジェクトに T3File を要求します。このファイルに対して読み書きを行うことができます。

IOServices ファクトリにアクセスするには、T3ServicesDef リモート インタフェースを使用します。IOServices ファクトリ メソッド getFileSystem() を呼び出して T3FileSystem オブジェクトを取得します。WebLogic Server 上に引数として登録されているファイル システム名を指定します。ファイル システムの登録は、Administration Console を使って行います。

この例では、ファイル システム プロパティとして名前が myFS、パスが \usr\local に設定されているものとします。

myFS にマップされる T3FileSystem に作成された T3File は、物理的には WebLogic Server のホストのディレクトリ \usr\local にデプロイされます。次に、T3FileSystem と test という名前の T3File を取得するコードを示します。

```
T3FileSystem t3fs =
  t3services.io().getFileSystem("myFS");
  T3File myFile = t3fs.getFile("test");
```

次に示すように、ファイルに読み書きする前にこのファイルが存在しているかどうかを調べることもできます。

```
if (myFile.exists()) {
  System.out.println("The file already exists");
}
else {
  // バイト配列を含んだファイルを作成する。次の手順で
  // それを出力ストリームに書き出す
  byte b[] = new byte[11];
  b[0]='H'; b[1]='e'; b[2]='l'; b[3]='l'; b[4]='o'; b[5]=' ';
  b[6]='W'; b[7]='o'; b[8]='r'; b[9]='l'; b[10]='d';
}
```

手順 4. OutputStream オブジェクトの作成と使用

前の手順では、WebLogic Server 上の T3File に書き込むバイトの配列を作成しました。通常、T3File を作成し、T3File.getOutputStream() メソッドで OutputStream を要求してその T3File に書き込みます。

次の例では、前の手順で作成した T3File myFile を使ってこのプロセスを示します。

```
Outputstream os =
   myFile.getFileOutputStream();
   os.write(b);
   os.close();
```

OutputStream オブジェクトは、使い終わったら必ず閉じてください。

手順 5. InputStream オブジェクトの作成と使用

これで、読み出してその内容を確認する T3File を取得しました。OutputStream オブジェクトと同じパターンで、InputStream オブジェクトを要求して使用します。

ここでは、T3File の myFile からの読み出しを行うための InputStream オブジェクトを要求します。これにより、T3File への InputStream が開かれます。次の例では、バイトを読み込みます。まず、読み込むバイトの配列を割り当てます。この配列は、表示できる String を作成するために後で使用します。次に、以下に示すように java.io.InputStream クラスの標準メソッドを使用して、T3File から読み込みます。

```
byte b[] = new byte[11];
InputStream is = myFile.getFileInputStream();
is.read(b);
is.close();
```

ここで、表示する String を作成して結果を確認します。

```
String result = new String(b);
System.out.println("Read from file " + T3File.getName()
                   " on the WebLogic Server:");
System.out.println(result);
is.close();
```

InputStream オブジェクトは、使い終わったら必ず閉じてください。

コード例

この完全なコード例は、配布キットの examples\io ディレクトリに収められている実行可能なサンプルです。このコード例は、同じディレクトリに収められている指示に従ってコンパイルして実行できます。この例は、コマンド行から実行できるように main() メソッドを使用しています。

```
public class HelloWorld {

  public static void main(String[] argv) {
```

```
// WebLogic Server URL、T3FileSystem 名
// および T3File 名の各文字列（String）
String url;
String fileSystemName;
String fileName;

// ユーザの入力をチェックし、正しければそれを使用する
if (argv.length == 2) {
  url = argv[0];
  // クライアント上のローカル ファイル システムを使用する
  fileSystemName = "";
  fileName = argv[1];
}
else if (argv.length == 3) {
  url = argv[0];
  fileSystemName = argv[1];
  fileName = argv[2];
}
else {
  System.out.println("Usage:java example.io.HelloWorld " +
                     "WebLogicURL fileSystemName fileName");
  System.out.println("Example:java example.io.HelloWorld " +
                     "t3://localhost:7001 users test");
  return;
}

// WebLogic Server からリモートの T3Services ファクトリを取得する
try {
  T3Services t3services = getT3Services(url);

  // ファイル システムとファイルを取得する
  System.out.println("Getting the file system " + fileSystemName);
  T3FileSystem fileSystem =
    t3services.io().getFileSystem(fileSystemName);
  System.out.println("Getting the file " + fileName);
  T3File file = fileSystem.getFile(fileName);

  if (file.exists()) {
    // ファイルが存在しているので、何もしない
    System.out.println("The file already exists");
   }
   else {
    // ファイルが存在しないので、新たに作成する
    byte b[] = new byte[11];
```

```
      b[0]='H'; b[1]='e'; b[2]='l'; b[3]='l'; b[4]='o'; b[5]=' ';
      b[6]='W'; b[7]='o'; b[8]='r'; b[9]='l'; b[10]='d';

      // OutputStream を取得してファイルに書き出す
      System.out.println("Writing to the file");
      OutputStream os = file.getFileOutputStream();
      os.write(b);
      os.close();
    }

    // InputStream を取得してファイルから読み込む
    byte b[] = new byte[11];
    System.out.println("Reading from the file");
    InputStream is = file.getFileInputStream();
    is.read(b);
    is.close();

    // 結果をレポートする
    String result = new String(b);
    System.out.println("File contents is:" + result);
  }
  catch (Exception e) {
    System.out.println("The following exception occurred " +
                       "while running the HelloWorld example.");


    e.printStackTrace();
    if (!fileSystemName.equals("")) {
      System.out.println("Make sure the WebLogic server at " +
                 url + " was started with " +
                 "the property weblogic.io.fileSystem." +
                 fileSystemName + " set.");
    }
  }
}

private static T3ServicesDef getT3Services(String wlUrl)
     throws javax.naming.NamingException
{
  T3ServicesDef t3s;
  Hashtable env = new Hashtable();
  env.put(Context.PROVIDER_URL, wlUrl);
  env.put(Context.INITIAL_CONTEXT_FACTORY,
          weblogic.jndi.WLInitialContextFactory.class.getName());
  Context ctx = new InitialContext(env);
```

```
    t3s = (T3ServicesDef) ctx.lookup("weblogic.common.T3Services");
    ctx.close();
    return(t3s);
  }
}
```