BEA WebLogic Server™

WebLogic Time Services プログラマーズ ガイド

BEA WebLogic Server バージョン 7.0
マニュアルの改訂 : 2002 年 6 月 28 日

WebLogic Time Services プログラマーズ ガイド

パート番号	マニュアルの改訂	ソフトウェアのバージョン
なし	2002 年 6 月 28 日	BEA WebLogic Server バージョン 7.0

目次

このマニュアルの内容

対象読者 v
e-docs Web サイト v
このマニュアルの印刷方法 vi
サポート情報 vi
表記規則 vii

1. WebLogic Time サービス（非推奨）のプログラミング

Time サービスの非推奨について 1-1
概要 1-2
WebLogic Time のアーキテクチャ 1-2
WebLogic Time API 1-3

2. WebLogic Time を使用した実装

クライアント上での反復トリガのスケジューリング 2-1
WebLogic クライアントからの反復サーバサイド トリガのスケジューリング 2-3
手順 1. ScheduleDef および TriggerDef インタフェースの実装 2-3
手順 2. WebLogic クライアントからの ScheduledTrigger の作成 2-5
複雑なスケジュールの設定 2-7
再スケジューリング 2-7
ScheduledTrigger の停止 2-8

このマニュアルの内容

このマニュアルでは、BEA WebLogic Server™ 上で実行される WebLogic Time サービスのアーキテクチャについて説明します。

このマニュアルの構成は次のとおりです。

- 第 1 章「WebLogic Time サービス（非推奨）のプログラミング」では、WebLogic Time サービスのアーキテクチャについて概説します。
- 第 2 章「WebLogic Time を使用した実装」では、WebLogic Time サービスの実装方法について説明します。

対象読者

このマニュアルは、アプリケーションに Time サービスを実装するアプリケーション開発者を対象としています。このマニュアルは、Web テクノロジ、オブジェクト指向プログラミング手法、および Java プログラミング言語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] をクリックします。

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 章ずつ印刷できます。

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、[ドキュメントのダウンロード] 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 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 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@beasys.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
〔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.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

斜体の等幅テキスト	コード内の変数を示す。 例： `String CustomerName;`

表記法	適用
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 BEA_HOME OR
{ }	構文の中で複数の選択肢を示す。
[ ]	構文の中で任意指定の項目を示す。 例： `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
|	構文の中で相互に排他的な選択肢を区切る。 例： `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
...	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。
. . .	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。

1 WebLogic Time サービス（非推奨）のプログラミング

WebLogic Time API は、将来の日付と時刻に発生するか、またはスケジュール通りに定期的に繰り返すアクション（トリガ）をスケジューリングするためのメカニズムを提供します。以下の節では、Time サービスの機能について概説します。

- Time サービスの非推奨について
- 概要
- WebLogic Time のアーキテクチャ
- WebLogic Time API

Time サービスの非推奨について

WebLogic Time サービスは、WebLogic Server バージョン 6.1 より非推奨になりました。サードパーティのスケジューリング サービスについては、『WebLogic Server ツール』の「J2EE スケジューリング ツール」を参照してください。

BEA Systems, Inc. では、WebLogic Server に含まれる、JMX タイマ サービスの実装である javax.management.timer.Timer の使用をお勧めします。詳細とマニュアルについては、『JavaTM Management Extensions（JMX）』を参照してください。

概要

Time API を使用すると、クライアントの JVM 内か、またはクライアントに代わって WebLogic Server 上で、ユーザが記述した任意のトリガをスケジューリングして実行できます。Time API は、自動的に発生するアクションを設定するための信頼できる分散可能な方法を提供します。

注意： Time サービスはクラスタ内の個々の WebLogic Server で使用できますが、サービス自体はクラスタ化できません。WebLogic Time API は、ロード バランシングやフェイル オーバなどのクラスタ機能を利用しません。

WebLogic Time のアーキテクチャ

WebLogic Time は、他の WebLogic Server API と多くの特性を共有する軽量で効率的な API です。WebLogic Time は、Schedulable オブジェクトから作成される ScheduledTriggerDef オブジェクトに基づいて構築されています。ScheduledTriggerDef オブジェクトは、アクションのスケジュールの開始、停止、または反復を実行します。Triggerable オブジェクトは、スケジュールに従って実行されるアクションを定義します。オブジェクト ファクトリを使用すると、ScheduledTrigger を作成できます。オブジェクト ファクトリは、WebLogic Server 内の少ないリソースを管理するための明確で使いやすい手法を提供します。

スケジュールの情報は、効率的にリンクされた一連のリストに保持されます。このリストは、新しいトリガがスケジューリングされ、実行されるときから最も近い時点でのみソートされます。たとえば、火曜日の 12:15:30 から 1 週間のトリガをスケジューリングした場合、これは最初は単純に次の火曜日のスケジュールに挿入されます。火曜日の正午になると正午 12 時のスケジュールがソートされ、12 時 15 分になるとその時刻（分）のトリガがソートされます。このようにすることで、非常に多くのスケジュールが設定されている環境で、スケジューリングのオーバーヘッドが大幅に削減されます。

また、WebLogic Server は、Time サービスのユーザ間でのタイムゾーン、クロック精度、およびレイテンシの差異を絶えず調整します。WebLogic トリガは、ミリ秒の精度で使用できるリアル タイム トリガではありません。WebLogic トリガは正常に使用すると、およそ 1 秒以内の精度で確実に機能します。

WebLogic Time API

ScheduledTrigger は、そのコンストラクタで以下の 2 つのオブジェクトを使用します。

- `weblogic.time.common.Schedulable` または `weblogic.time.common.ScheduleDef` のいずれかを実装するオブジェクト
- `weblogic.time.common.Triggerable` または `weblogic.time.common.TriggerDef` のいずれかを実装するオブジェクト

ScheduledTrigger オブジェクト ファクトリ メソッドに渡されるオブジェクトも、クライアントサイド オブジェクトの場合があります。この場合、WebLogic クライアントはそれ自身の JVM 内で ScheduledTrigger を作成、スケジューリング、および実行します。クライアントサイド オブジェクトは、Schedulable （または ScheduleDef）および Triggerable （または TriggerDef）を実装しなければなりません。

`TimeServicesDef` インタフェースも、クライアントとサーバの時刻関連情報を取得するためのメソッドを提供します。

- `currentTimeMillis()` は、現在のサーバ時刻を、「ローカル サーバ時刻」フォーマットで返します。ローカル サーバ時刻は、メソッドの呼び出し元とサーバの間の伝達遅延が調整されたサーバの時刻です（メソッドの呼び出し元がサーバの場合はゼロ、クライアントまたは別の WebLogic Server が呼び出し元の場合は正のミリ秒）。
- `getRoundTripDelayMillis()` は、クライアントとサーバ間の往復遅延を示すミリ秒数を返す。このメソッドは、概要で説明されているアルゴリズムに依存しています。

- `getLocalClockOffsetMillis()` は、概要で説明されているアルゴリズムに基づいて、クライアントとサーバのクロック間のオフセットを示すミリ秒数を返します。

`weblogic.time.common.TimeRepeat` クラスは、Schedulable を実装します。このユーティリティ クラスは、反復トリガの設定に使用できる作成済みのスケジューラです。これを使用する場合は、トリガを繰り返す間隔（ミリ秒単位）を示す `int` を渡してから、その `schedule()` メソッドを開始時刻を指定して呼び出します。

警告： トリガが例外を送出した場合、そのトリガは再スケジューリングされません。これは、失敗したトリガが無期限に再実行されないようにするためです。例外の発生後にトリガを再スケジューリングする場合、その例外を取得してそのトリガを再びスケジューリングしなければなりません。

パッケージには、1 つの例外クラス、TimeTriggerException が含まれています。

2 WebLogic Time を使用した実装

以下の節では、WebLogic Time サービス（非推奨）の実装方法について説明します。

- クライアント上での反復トリガのスケジューリング
- WebLogic クライアントからの反復サーバサイド トリガのスケジューリング
- 複雑なスケジュールの設定
- 再スケジューリング
- ScheduledTrigger の停止

クライアント上での反復トリガのスケジューリング

反復トリガをスケジューリングする最も単純なケースは、WebLogic クライアント上でスケジューリングおよび実行される ScheduledTrigger を作成することです。このケースでは、Schedulable と Triggerable の両方を実装するクラスを記述し、これらのインタフェースのメソッドを実装します。

次のコード例は、どのようにトリガをスケジュールして実行するかを示したものです。

```
import weblogic.time.common.*;
import weblogic.common.*;
import java.util.*;
import weblogic.jndi.*;
import javax.naming.*;
import java.util.*;


class myTrigger implements Schedulable, Triggerable {
```

```
  ...
}
```

まず、TimeServices ファクトリから ScheduledTrigger オブジェクトを取得する必要があります。TimeServices ファクトリは、`getT3Services()` メソッドを介して WebLogic Server 上の T3Services リモート ファクトリ スタブから取得します。

注意： クライアント アプリケーションではなく、WebLogic Server インスタンスの `TimeServices` インタフェースへのハンドルを取得するには、静的メソッドの `weblogic.common.T3Services.getT3Services()` を使用します。

次に、トリガの `schedule()` メソッドと `cancel()` メソッドを呼び出します。次に例を示します

```
public myTrigger() throws TimeTriggerException {
  // T3Services ファクトリを取得する
  T3ServicesDef t3 = getT3Services("t3://localhost:7001");

  // ファクトリから ScheduledTrigger を要求する
  // このクラスをスケジューリングと実行に使用する
  ScheduledTriggerDef std =
    t3services.time().getScheduledTrigger(this, this);
  // スケジューリングを開始する
  std.schedule();
  // クラスは、トリガのスケジューリング後に他の作業を行う場合もある
 // 終了したら、トリガをキャンセルする
  std.cancel();
}
```

クラスは、以下のインタフェースに定義されているメソッドを実装しなければなりません。

`Schedulable`

`Schedulable` インタフェースは、`schedule()` という 1 つのメソッドを持っています。このメソッドを使用すると、トリガを実行する時刻を設定できます。

```
public long schedule(long time) {
  // 5 秒ごとのトリガをスケジューリングする
  return time + 5000;
}
```

`Triggerable`

`Triggerable` インタフェースは、1 つのメソッド `trigger()` だけを持っています。ここでは、クライアントは時刻が設定されたトリガに反応してアクションを起こします。

```
public void trigger() {
  // trigger メソッドはアクションが起こる場所である
  System.out.println("trigger called");
}
```

このコード例には、スケジューラとトリガの両方を実装する単一のクラスが含まれています。これは、2 つの必須メソッドがスケジューリングまたは実行に必要なクラス変数を共有するので便利です。

WebLogic クライアントからの反復サーバサイド トリガのスケジューリング

WebLogic フレームワーク内の任意の場所で実行できる、より柔軟なスケジューラとトリガを記述できます。これを実装するには、`ScheduleDef` と `TriggerDef` を、より単純なインタフェース `Schedulable` と `Triggerable` の代わりに実装します。この例では、WebLogic Server 上（または WebLogic フレームワーク内の任意の場所）で再スケジューリングおよび実行される反復トリガを作成するという、より柔軟な実装を示します。

このシナリオでスケジューリング済みトリガを作成する手順は次のとおりです。まず、ScheduleDef と TriggerDef を実装するクラスを記述する必要があります。この例では、これらのインタフェースを別々のクラスに実装します。

これらのクラスをコンパイルして、WebLogic Server の `serverclasses` ディレクトリに置きます。最後に、クライアント アプリケーションからこれらのクラスを使って `ScheduledTrigger` を作成します。

手順 1. ScheduleDef および TriggerDef インタフェースの実装

この例では、`setServices()` および `scheduleInit()` メソッドが呼び出されるように、スケジューラは Schedulable ではなく ScheduleDef を実装します。同じ理由で、トリガも Triggerable ではなく TriggerDef を実装します。これらのオブ

ジェクトは、ParamSet で初期化できるという点と、T3Services スタブを通して WebLogic サービスにアクセスできるという点が、これら自身が実装するインタフェースとは異なります。これら 2 つの相違点は、以下の理由で重要です。

クライアントサイドのデプロイメントとサーバサイドのデプロイメントで異なるバージョンを作成する必要がありません。これは、T3ServicesDef インタフェースがリモート スタブだからです。

オブジェクトを動的にインスタンス化するときは、デフォルト コンストラクタを呼び出さなければなりません。したがって、Time インタフェースを含むすべてのサービス関連インタフェースでは、オブジェクトの初期化パラメータを渡すことができるように、ParamSet を引数として取る scheduleInit() メソッドを実装する必要があります。

次に、ScheduleDef の単純な実装を示します。

```
package examples.time;

import weblogic.common.*;
import weblogic.time.common.*;
import java.util.*;

class MyScheduler implements ScheduleDef {

  private int interval = 0;
  private T3ServicesDef services;

  public void setServices(T3ServicesDef services) {
    this.services = services;
  }

  public void scheduleInit (ParamSet ps) throws ParamSetException {
    interval = ps.getParam("interval").asInt();
  }

  public long schedule(long currentMillis) {
    return currentMillis + interval;
  }
}
```

次に、TriggerDef を実装する単純なクラスを示します。この場合、Trigger のパラメータを設定または取得する必要はありません。つまり、何もしないメソッドを実装します。

```
package examples.time;

import weblogic.common.*;
import weblogic.time.common.*;
```

```
import java.util.*;

public class MyTrigger implements TriggerDef {

  private T3ServicesDef services;

  public void setServices(T3ServicesDef services) {
    this.services = services;
  }

  public void triggerInit (ParamSet ps) throws ParamSetException {
  // 空のメソッド定義
  }

  public void trigger(Schedulable sched) {
    System.out.println("trigger called");
  }
}
```

手順 2. WebLogic クライアントからの ScheduledTrigger の作成

スケジューラとトリガを設定するこのメソッドでは、getScheduledTrigger() ファクトリ メソッドに渡す Scheduler および Trigger オブジェクトを作成する必要があります。これらは、「手順 1. ScheduleDef および TriggerDef インタフェースの実装」で既に作成してあります。

これらのクラスは、既にコンパイルして WebLogic Server の CLASSPATH にデプロイしてあります。次に、これらのクラスを使ってサーバの JVM で実行するトリガをスケジューリングするクライアントを記述します。

ParamSet を使って、クライアントと WebLogic Server がインスタンス化するオブジェクトとの間で初期化パラメータを渡します。ScheduleDef を実装するために手順 1 で記述したクラスは、呼び出し側が設定するパラメータ「interval」に依存するため、1 つの Param を持つ ParamSet を作成します。TriggerDef を実装するために記述したクラスは、初期化パラメータは必要としません。

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

  // ParamSet を作成して、ScheduleDef オブジェクトの
  // 初期化パラメータを渡す。1 つのパラメータ「interval」を
  // 10 秒に設定する
  ParamSet schedParams = new ParamSet();
  schedParams.setParam("interval", 10000);
```

getT3Services() メソッドをクライアント クラスに追加します。次に、サーバ上で ScheduledTrigger をインスタンス化する、Scheduler および Trigger ラッパー オブジェクトを作成します。Scheduler および Trigger ラッパー オブジェクトは、必要であれば、ターゲット クラスの名前とこれを初期化する ParamSet を保持します。

```
Scheduler scheduler =
  new Scheduler("examples.time.MyScheduler", schedParams);
Trigger    trigger =
  new Trigger("examples.time.MyTrigger");
```

最後に、Time サービス オブジェクト ファクトリを使用して ScheduledTrigger を作成します。これは、先ほど作成した Scheduler と Trigger という 2 つの引数を取ります。

```
ScheduledTriggerDef std =
  t3.services.time().getScheduledTrigger(scheduler, trigger);
```

getScheduledTrigger() メソッドは、ScheduledTriggerDef object を返します。実行を開始するために、クライアントは ScheduledTriggerDef の schedule() および cancel() メソッドを呼び出します。

反復スケジュールを設定する場合、このパッケージの一部であるユーティリティ クラス TimeRepeat も使用できます。次に、TimeRepeat クラスを使用して、10 秒ごとに繰り返す ScheduledTrigger の定期スケジュールを設定する単純なコード例を示します。ここでも getT3Services() メソッドを使用して WebLogic サーバサイド サービスにアクセスしています。

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

Scheduler scheduler = new Scheduler(new TimeRepeat(1000 * 10));
Trigger   trigger   = new Trigger("examples.time.MyTrigger");

ScheduledTriggerDef std =
  t3services.time().getScheduledTrigger(scheduler, trigger);

std.schedule();
```

複雑なスケジュールの設定

Schedulable オブジェクトの schedule() メソッドを使用すると、複雑なスケジュールを自由に設定できます。次に、スケジューリングの例とヒントをいくつか示します。

schedule() メソッドの引数で実行時刻を指定する方法は、以下の 2 通りです。

- 現在時刻。エポックからの経過時間をミリ秒で指定します。
- 将来の特定日付と時刻。標準 Java クラス（java.util.Date など）を使った日付計算の実行結果をミリ秒で指定します。

schedule() メソッドは long 値を返します。この値を使用すると、反復トリガを設定できます。単純に、schedule() メソッドが最後に呼び出された時刻と、スケジュールを繰り返す間隔（ミリ秒）を返します。

再スケジューリング

この例では、schedule() メソッドを記述して、trigger() メソッドに対する各呼び出しの間をインクリメントする間隔だけ遅延させます。この例では、schedule() メソッドと trigger() メソッドは、同じクラスに実装されています。

trigger() メソッドでは、private int の *delay* を使用して、インクリメントする遅延を設定します。delay は、クラス コンストラクタで 0 に初期化されます。trigger を呼び出すたびに、それは自己のスケジュールをインクリメンタルに調整します。

```
public void trigger() {
  System.out.println("Trigger called");
  // 任意のタスクを実行する . . .
  System.out.println("Trigger completed");
  // delay に 1000 ミリ秒を追加する
  delay += 1000;
}
```

schedule() メソッドでは、最後にスケジューリングされた実行時刻に、最後にスケジューリングされた実行によってインクリメントされた遅延（ミリ秒）を足した時刻を、トリガの次の実行として返します。また、スケジューリングを終了するために、遅延に上限を指定します。

```
public long schedule(long t) {
  System.out.println("-------------------------------------");
  if (delay > 10000) {
    System.out.println("Cancelling Timer");
    return 0;
  }
  else {
    System.out.println("Scheduling next trigger for " +
                       delay/1000 + " seconds");
    return t + delay;
  }
}
```

ScheduledTrigger の停止

ScheduledTrigger は、以下の 2 つの方法で停止できます。

- ScheduledTrigger の cancel() メソッドを呼び出します。
- schedule() メソッドが呼び出されるときにゼロ（0）を返すことで、スケジューリングを終了させます。

これら 2 つのメソッドの間には、若干の違いがあります。schedule() メソッドからゼロを返すと、スケジュールはすぐに終了します。cancel() メソッドを呼び出すと、trigger() の次にスケジューリングされたインスタンスまでクロックは動作し続け、そこでキャンセルされます。