Servlet 4.0 を導入する

Java サーブレット・アプリケーション内で HTTP/2 サーバー・プッシュと新しい HttpServletMapping インターフェースを使用する方法

Alex Theedom

2018年 12月 06日

Servlet 4.0 は HTTP/2 のサーバー・プッシュ・テクノロジーを完全に統合しているだけでなく、サーブレットのマッピング URL のランタイム・ディスカバリーにも対応しています。このハンズオン・チュートリアルでは動画のデモを盛り込み、サンプル・コードを記載して、Java サーブレットと JSF アプリケーション内で HTTP/2 サーバー・プッシュと新しい HttpServletMapping インターフェースを使い始められるよう手引きします。

Java Servlet API は、メインフレーム・サーバー・サイド Java の基礎となるビルディング・ブロックであり、Web サービスを作成するための JAX-RS や、JSF (JavaServer Faces) および JSP (JavaServer Pages) などの Java EE テクノロジーの一部でもあります。Java サーブレット単独でも、動的な Web コンテンツをサポートする幅広い機能を提供しています。例えば、フィルター、Web セキュリティー、そして HTTP リクエストおよびレスポンスを処理する機能などです。

Java Servlet API の最新バージョンとしてリリースされた Servlet 4.0 は、Java EE 8 仕様における更新の中核にあたります。このチュートリアルで説明するように、Servlet 4.0 は HTTP/2 に対応し、サーバー・プッシュを完全に統合しているだけでなく、JSF 2.3 などのサーブレットをベースとしたテクノロジーでもサーバー・プッシュを使用できるようにします。また、Servlet 4.0 には、サーブレットのマッピング URL のランタイム・ディスカバリーを可能にする、`HttpServletMapping` というインターフェースが追加されています。このチュートリアルでは、この新しいインターフェースを使用する方法も説明します。

HTTP/2 とは何か

HTTP/2 で最優先されている目標は、Web アプリケーション・ユーザーのエクスペリエンスを改善することです。バイナリー・プロトコルである HTTP/2 は、軽量、セキュア、高速というバイナリーならではのメリットのすべてを兼ね備えています。HTTP/2 は元の HTTP プロトコルのセマンティクスを維持していますが、システム間でのデータ転送方法については変更しています。私が書いた、このリンク先の記事「HTTP/2 の内幕」(developerWorks、2017 年 6 月) で、HTTP/2 の包括的な概要を読んでください。

コードを入手する

サーブレットの要点

「Java サーブレット」は、HTTP プロトコルで動作するサーバー・サイドのテクノロジーです。サーブレットはクライアントからサーバーに送信されるリクエスト・メッセージを待機し、送信されたリクエスト・メッセージに対するレスポンス・メッセージをクライアントに返します。リクエスト・メッセージとレスポンス・メッセージは、以下の 2 つの部分からなります。

- メッセージに関する情報を含むヘッダー。
- メッセージのペイロード (メッセージの内容) を格納する本体。

典型的なメッセージ交換では、クライアントがブラウザーまたは別の HTTP クライアント (`curl` など) から特定の URL をリクエストすることによって、サーブレットを呼び出します。

リスト 1 の例で、サーブレットがアクティブになるのは、このサーブレットのパスがリクエストされた時点です。リクエストが Java サーブレットのどのメソッドに委任されるかは、リクエストで使用されている HTTP メソッドによって決まります。この例のリクエストでは `GET` メソッドが使用されているため、リクエストは Java サーブレットの `doGet()` メソッドによって処理されます。

以下のメッセージ交換では、サーブレットのパスとして http://hostname/applicationroot/showlogoservlet がリクエストされています。

リスト 1. 単純なサーブレット実装

```
@WebServlet("/showlogoservlet")
public class SimpleServlet extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException, IOException {

       getServletContext()
         .getRequestDispatcher("/showlogo.jsp")
         .forward(request, response);

   }
}
```

Servlet 4.0 の注目の新機能

Servlet 4.0 で特に注目すべき機能は、サーバー・プッシュと、サーブレットの URL マッピングを実行時に検出する新しい API です。

サーバー・プッシュは HTTP/2 での最も目立つ機能拡張であり、サーブレット内では `PushBuilder` インターフェースを介して公開されます。サーバー・プッシュは JavaServer Faces API 内にも実装されていて、ライフサイクルの `RenderResponsePhase` フェーズの間に呼び出されます。したがって、JSF ページでこのパフォーマンス強化を利用できます。

サーブレット・マッピング・ディスカバリー・インターフェースとして新しく追加された `HttpServletMapping` は、所定のサーブレット・リクエストをアクティブにした URL 情報をフレー

ムワークが取得できるようにします。内部動作に URL 情報が必要となるフレームワークには、このインターフェースが特に役立つはずです。

以降のセクションで、サーバー・プッシュの概要と、JSF 2.3 でのサーバー・プッシュを含め、Java サーブレット内でのこの機能の動作を説明します。また、新しいサーブレット・マッピング・ディスカバリー機能を明らかにするメッセージ交換の例も紹介します。

Servlet 4.0 でのサーバー・プッシュ

サーバー・プッシュ機能は、サーバーがクライアント・リクエストのリソース要件を予測することを可能にします。サーバーがリソース要件を予測できれば、リソース処理を完了する前に、その予測に基づいて該当するリソースをクライアントに送信できます。

サーバー・プッシュのメリットを理解するために、画像とその他の依存関係 (CSS ファイルや JavaScript など) で構成される Web ページを考えてみてください。クライアントがその Web ページをリクエストするだけで、サーバーはリクエストされた Web ページを分析し、ページをレンダリングするために必要なリソースを判断し、それらのリソースをクライアントのキャッシュに先行送信します。

サーバーはこのすべての処理を、元の Web ページに対するリクエストの処理と並行して行います。したがって、クライアントがレスポンスを受信するころには、必要なリソースがすでにキャッシュ内にあるというわけです。

PushBuilder

Servlet 4.0 では、サーバー・プッシュは `PushBuilder` インターフェースを介して公開されます。したがって、サーバー・プッシュを使用するには、`newPushBuilder()` メソッドを呼び出して、`HttpServletRequest` から `PushBuilder` のインスタンスを取得する必要があります。リスト 2 に、`PushBuilder` インスタンスを取得する方法を示します。

リスト 2. 新しい PushBuilder の取得方法

```
@Override
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {

   PushBuilder pushBuilder = request.newPushBuilder();

}
```

`newPushBuilder()` メソッドを呼び出すたびに、`PushBuilder` の新しいインスタンスが返されてきます。サーバー・プッシュが使用可能でなければ、`newPushBuilder()` メソッドから `null` が返されます。場合によっては、トランザクションにサーバー・プッシュを使用しないよう、クライアントがこの機能を拒否することもできます。また、クライアントがセキュアな接続を使用していない場合も、サーバー・プッシュは機能しません。このことから、`PushBuilder` インスタンスに対してメソッドを呼び出す前に、必ず `null` 戻り値をテストしてください。

名前が示唆するように、`PushBuilder` はビルダー・パターンを実装します。Builder パターン実装では、複数のミューテーター・メソッドを鎖状につなげてプッシュ・リクエストを作成します。

これらのメソッドによって、HTTP ヘッダー、メソッドのタイプ (許容される値は `GET` のみ)、クエリー文字列、セッション ID、リソースのパス (つまり、プッシュされるリソースのパス) を設定して `PushBuilder` インスタンスを構成します。

元の `HttpServletRequest` インスタンスからのリクエスト・ヘッダーのほとんどは、そのまま `PushBuilder` インスタンスに追加されますが、以下のヘッダーはサーバー・プッシュが正常に機能するために必須であるというわけではないため、除外されます。

- 条件ヘッダー
- Range ヘッダー
- Expect ヘッダー
- Authorization ヘッダー
- Referrer ヘッダー

ここからは、サーバー・プッシュの動作を組み立てて起動する方法を見ていきましょう。

1. プッシュするリソースを設定する

リソースをクライアントにプッシュする前に設定しなければならない構成は、「パス」だけです。パスを設定するには、`path()` メソッドを呼び出します。このメソッドは `PushBuilder` オブジェクトのパスの値を変化させるため、呼び出すのは一度だけでなければなりません。リソースのパスは、それが絶対パスであることを示すスラッシュ (「/」) で始めることができます。スラッシュで始まっていないパスは、関連するリクエストのコンテキスト・パスを基準とした相対パスであると見なされます。パスにはクエリー文字列を含めることができます。その場合、クエリー文字列は `queryString()` メソッドによって設定されるあらゆる文字列とマージされます。

2. リソースをプッシュする

次に、リソースをクライアントにプッシュする `push()` メソッドを呼び出します。呼び出された `push()` メソッドは、クライアントとの間でプッシュ「カンバセーション」を開始します。舞台裏では、クライアントに `PUSH_PROMISE` フレームが送信されます。このフレームは、リソースを送信する目的を伝える通知のような働きをします。クライアントは `RST_STREAM` を返すことで、リソースを拒否することもできます。このメカニズムにより、クライアントはどのリソースを受信するかについての制御を維持できるようになっています。つまり、不要なリソースやキャッシュ済みのリソースによって過負荷状態にならないよう、クライアントが保護されているということです。

`PushBuilder` のインスタンスを取得した後は、そのインスタンスを何度でも再利用できます (リスト 3 を参照)。パスと条件ヘッダーは null に設定されますが、その他すべてのフィールドはそのままの状態に維持されます。これらのフィールドは、別のサーバー・プッシュで再利用できます。

リスト 3. PushBuilder インスタンスを再利用する

```
PushBuilder pushBuilder = request.newPushBuilder();

if (pushBuilder != null) {
   pushBuilder.path("images/hero-banner.jpg").push();
   pushBuilder.path("css/menu.css").push();
   pushBuilder.path("js/marquee.js").push();
}
```

リスト 3 では、`path()` メソッドによって `PushBuilder` インスタンスに `hero-banner.jpg` のパスが設定されて、このパスが呼び出し側 `push()` メソッドによってクライアントにプッシュされます。`push()` はノンブロッキングのメソッドであり、即座に戻ります。したがって、他のリソース (この例では、`menu.css` と `marquee.js`) を続けてプッシュできます。

このビデオ, サーバー・プッシュの動作 , をご覧になるには、オンライン記事を参照ください。この記事がdeveloperWorksのアーカイブにある場合、ビデオの視聴はできなくなっています。

JSF でサーバー・プッシュを使用する

JavaServer Faces はそのライフライクルの一環として、あらかじめ各ページのリソース要件を特定するため、サーバー・プッシュを使用するのに当然適しています。さらに、開発者の観点からすると、サーバー・プッシュをアクティブ化するために何かを変える必要はまったくありません。JSF 2.3 にアップグレードすれば、何もせずに、この機能を利用できるようになります。

リスト 4 に、JSF とサーバー・プッシュの統合を示します。

リスト 4. JSF ページ内でサーバー・プッシュを使用する

```
<h:head>
   <h:outputStylesheet library="css" name="logo.css"/>
   <h:outputScript library="js" name="logo.js"/>
   <title>JSF 2.3 ServerPush Example</title>
</h:head>
<h:body>
   <h:form>
       <h:graphicImage library="images" name="logo.jpg"/>
   </h:form>
</h:body>
</html>
```

リスト 4 に示されている JSF ページには、以下の 3 つのリソースが必要です。

- logo.css という名前の CSS ファイル
- logo.js という名前の JavaScript ファイル
- logo.jpg という名前の画像

JSF エンジンがページを処理してレンダリングする間、上記のリソースが次々とクライアントにプッシュされます。この動作が開始されるのは、JSF のレンダー・レスポンス・フェーズです。すると、各リソースに対する `ExternalContextImpl.encodeResourceURL()` メソッドが呼び出されて、そのリソースの新しい URL が渡されます。新しい `PushBuilder` オブジェクトは、`ExternalContext` に関連付けられた `HttpServletRequest` インスタンスから取得されます。サーバー・プッシュがサポートされていれば、ページがクライアントにレンダリングされる前に、必要なリソースがクライアントにプッシュされます。

このビデオ, JSF 2.3 でのサーバー・プッシュ , をご覧になるには、オンライン記事を参照ください。この記事がdeveloperWorksのアーカイブにある場合、ビデオの視聴はできなくなっています。

JSP でのサーバー・プッシュ

JavaServer Pages はレガシー・テクノロジーであるため、まだサーバー・プッシュは統合されていません。それでも、サーブレット・フィルターを使用してリソースをキャッシュしてからプッシュするという方法で、JSP 内でサーバー・プッシュを利用することはできます。この方法の使用例については、Jetty 9 の `PushCacheFilter` を参照してください。

HttpServletMapping インターフェース

Servlet 4.0 で新しく導入されたサーブレット・マッピング・ディスカバリー API を使用すると、サーブレットをアクティブにする原因となった URL を、サーバーが実行時にチェックすることができます。例えば、`file.ext` をリクエストする場合、`/path` と `/path/file.ext` により、URL パターンが `/path/*`、`*.ext` となっているサーブレットがアクティブになります。

この新しい `HttpServletMapping` インターフェースは、サーブレットのマッピング URL のランタイム・ディスカバリーをサポートします。このインターフェースのインスタンスを取得するには、`HttpServletRequest` のインスタンスに対して `getHttpServletMapping()` を呼び出します。サーブレットのマッピング URL に関する情報を取得する場合、以下のメソッドを使用できます。

- `getMatchValue()`。リクエストが一致する原因となった、URI パスの該当する部分を返します。
- `getPattern()`。URL パターンの `String` 表現を返します。
- `getServletName()`。サーブレット名の `String` 表現を返します。
- `getMappingMatch()`。`MappingMatch` 列挙型の値として表現された一致のタイプを返します。値は、`CONTEXT_ROOT`、`DEFAULT`、`EXACT`、`EXTENSION`、`PATH` のいずれかになります。

リスト 5 に、上記の 4 つの API メソッドを使用する方法を示します。

リスト 5. Servlet 4.0 でのランタイム・サーブレット・マッピング・ディスカバリー

```
@WebServlet({"/path/*", "*.ext"})
public class ServletMapping extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                         throws IOException {

        HttpServletMapping mapping = request.getHttpServletMapping();
        String mapping = mapping.getMappingMatch().name();
        String value = mapping.getMatchValue();
        String pattern = mapping.getPattern();
        String servletName = mapping.getServletName();
   }

}
```

このビデオ, サーブレット・マッピング URL のランタイム・ディスカバリー , をご覧になるには、オンライン記事を参照ください。この記事がdeveloperWorksのアーカイブにある場合、ビデオの視聴はできなくなっています。

Servlet 4.0 で行われた細かな更新

Servlet 4.0 ではサーバー・プッシュと新しい `HttpServletMapping` インターフェースの他に、小規模ながらも注目すべき機能の追加と更新が行われています。

1. `Trailer` レスポンス・ヘッダーを使用して、送信側がチャンク化したメッセージの末尾に追加フィールドを組み込めるようになりました。メッセージ整合性チェック、デジタル署名、事後処理ステータスなどのメタデータがメッセージ本文の送信中に動的に生成される場合は、このヘッダーを使用して、それらのメタデータを送信します。
2. Servlet 4.0 では、`GenericFilter` および `HttpFilter` という抽象化クラスが追加されています。これらの抽象化クラスは、フィルターの作成を単純化するために、ライフサイクル・メソッド `init()` および `destroy()` の最小限の実装を提供します。
3. Servlet 4.0 で統合された新しい `HTTP Trailer` でも、チャンク化したメッセージの末尾に送信側が追加フィールドを組み込むことができます。
4. `ServletContext` インターフェースに以下の新しいメソッドが追加されました。
    - `addJspFile()`。指定された JSP ファイルを使用するサーブレットを、サーバー・コンテキストに追加します。
    - `getSessionTimeout()` および `setSessionTimeout()`。セッション・タイムアウトへのアクセスを提供します。
    - `getRequestCharacterEncoding()` および `setRequestCharacterEncoding()`。これらのメソッドを使用して、現在のサーブレット・コンテキストでのリクエスト文字エンコーディングにアクセスし、デフォルトのエンコーディングを変更できます。
5. `HttpServletRequest` インターフェース上の `isRequestedSessionIdFromUrl()` メソッドは非推奨になりました。
6. Java SE 8 が改善されたことにより、リスナー・インターフェースにデフォルトのメソッドが追加されました。

まとめ

Servlet 4.0 のリリースで主に目的とされているのは、新しい HTTP/2 プロトコルとその多数のパフォーマンス拡張機能を統合することです。`PushBuilder` インターフェースにより、クライアントにプッシュされるリソースをきめ細かく制御できるようになったことで、最先端の興味深い実装がすでに形になっています。その一例として、Jetty 9 の `PushCacheFilter` Web フィルター内には、`PushBuilder` API を使用したサーバー・プッシュが実装されています。このフィルターは、最初のリクエストが実行された時点でリソースをキャッシュします。そのため、サーバー・サイドでまだリクエストが処理されている間でも、以降のリクエストをクライアントにプッシュすることができます。

JSF 2.3 にはサーバー・プッシュが組み込まれていますが、JavaServer Pages はそうではありません。JSF にサーバー・プッシュが統合されたことは、開発者に大きな恩恵をもたらしています。それは、パフォーマンスを向上させるためにプログラムを改造する作業よりも、動的な Web ページを設計する作業に集中できるからです。JSP にも同じような機能を求める開発者には、Web フィルターなどのカスタム・ソリューションが必要となります。その一例は、Jetty 9 の `PushCacheFilter` Web フィルターです。

関連トピック

- GitHub での Servlet 4.0 の仕様
- HTTP/2 の内幕
- Java EE 8 の新機能
- JSON Binding API 入門
- Get started with the Java EE Security API
- Alex's book: Java EE 8: Only What's New