

6.1 Java でのウェブ開発

- ウェブ開発の基本
- Tomcat
- Servlet



1

ウェブ開発の基
本

2

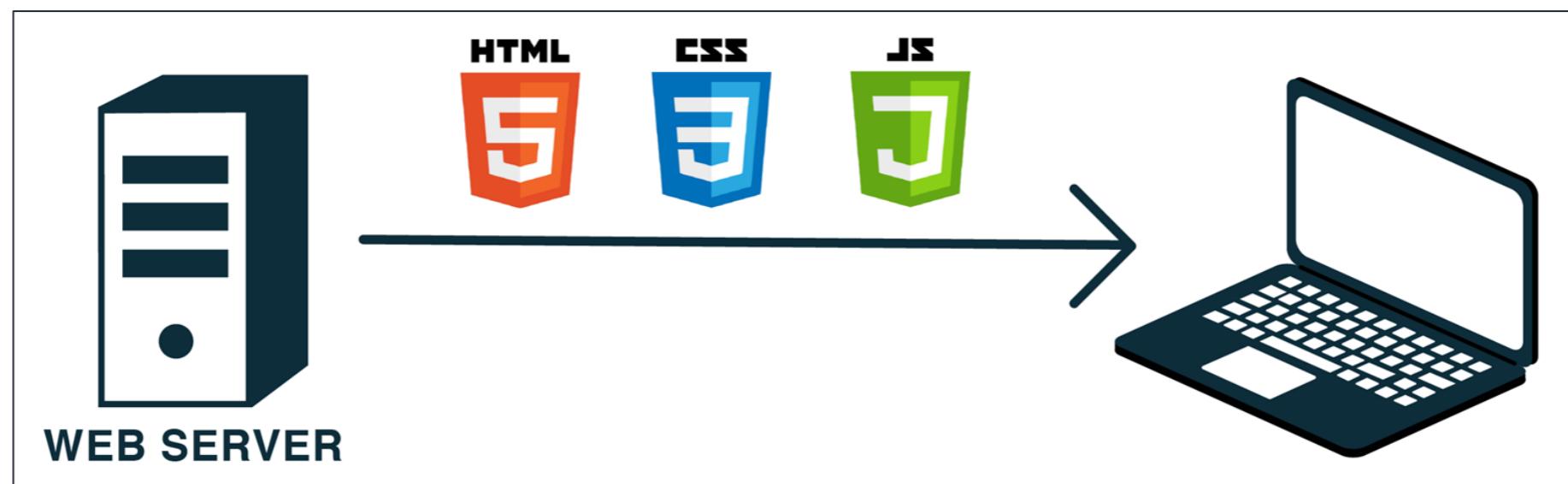
Tomcat

3

Servlet

静的ウェブページ

- ウェブ開発は大まかに、**静的ウェブページ**と**動的ウェブページ**の2種類に分けられます。
- 初期のウェブ開発は、**静的ウェブページ**[Static Web Page]が中心でした。ページは、HTML/CSSで書かれ、サーバに**直接保存**されました。ユーザーはブラウザを使ってHTTPプロトコルでサーバ上のページを要求して、ウェブサーバはリクエストを処理し、保存されたページを直接ユーザーに送信して表示させます。



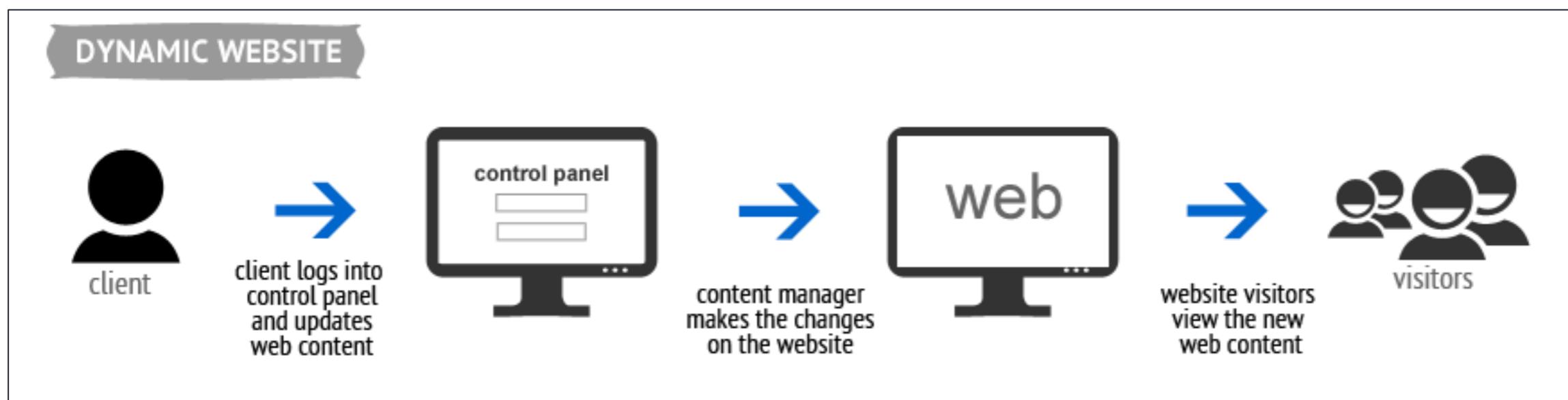
動的ウェブページ

- インターネットの発達に伴い、多くのオフラインビジネスがオンラインに移行し始め、インターネットに基づいたウェブ開発はますます複雑になってきています。ユーザーがアクセスするリソースは、ウェブサーバに保存されている静的ページにとどまらず、リクエストに基づいて**動的に生成されるコンテンツ**、すなわち**動的ウェブページ**[Dynamic Web Page]がより多く必要とされているのです。

次へ 

◀ 前へ

- これらのウェブサイトは通常、HTML/CSS とスクリプト言語（JSP、ASP、PHP など）の組み合わせで書かれています。書き上げたプログラムは、ウェブサーバにデプロイされます。ユーザーのリクエストを受けると、サーバはスクリプトを実行して、ブラウザが解析できる HTML を生成し、ユーザーに表示するためにブラウザに返します。



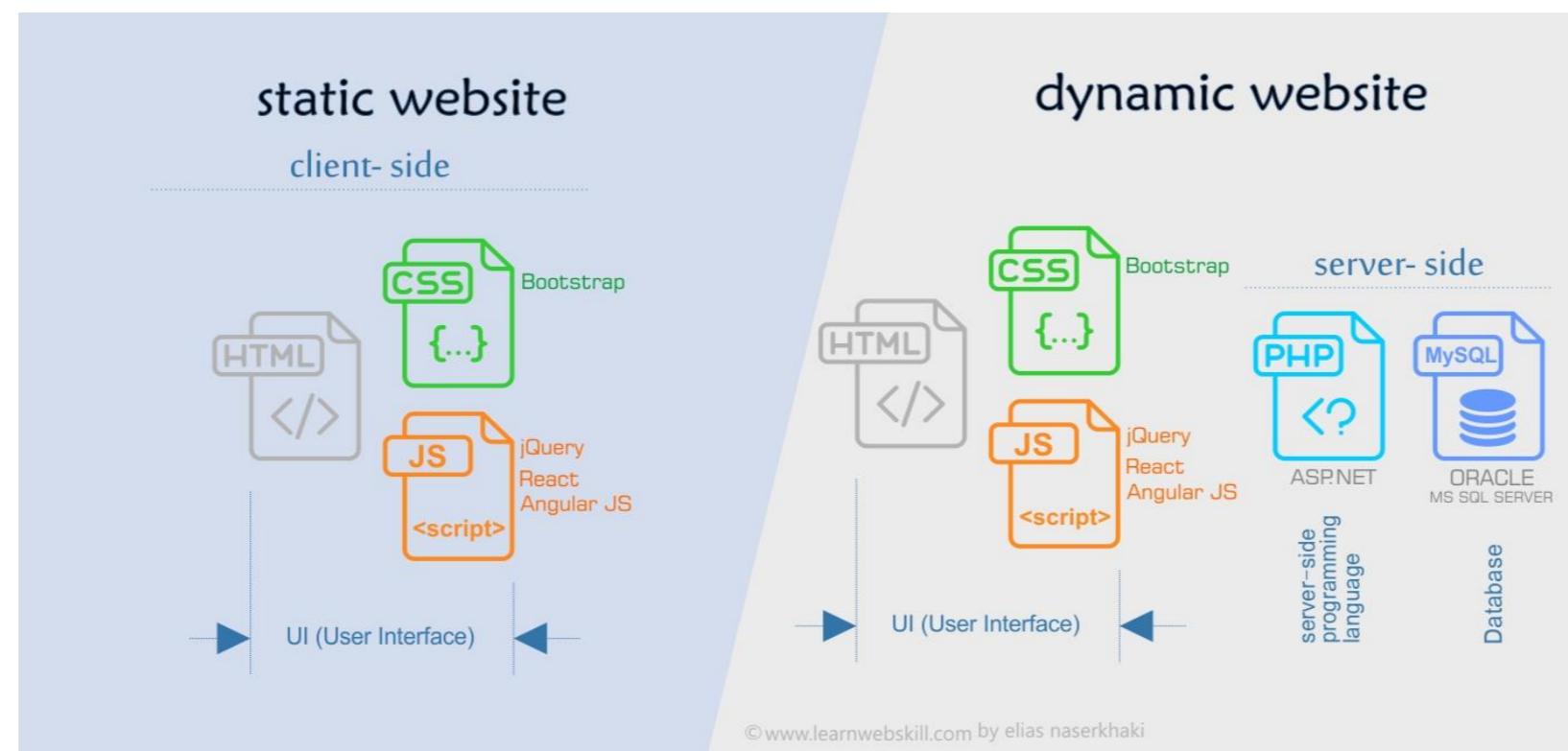
動的ウェブページと静的ウェブページ

- 初心者の方は、アニメーション効果があるウェブページこそ動的ページだと勘違いされるのですが、そうではあります。
- 動的ウェブページとは、インタラクティブで自動的に更新されるコンテンツを持ち、アクセスしたときの状況に応じて異なる内容が表示されるウェブページです。
- ここでの「インタラクティブ」は、ウェブページがユーザーのリクエストに応じて動的にレスポンスすることを意味します。

次へ 

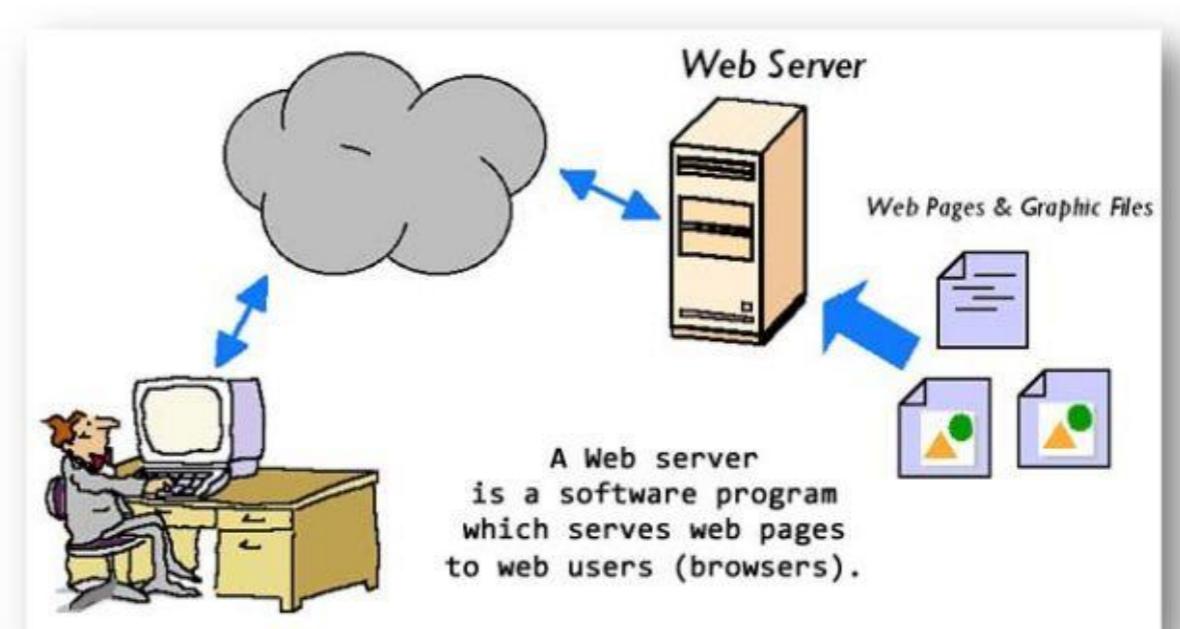
◀ 前へ

- つまり、静的ウェブページは、10年以上前に開発された携帯電話に似て、工場出荷時に設定された機能や着信音しか使えません。一方、動的ウェブページは、現代のスマートフォンに似て、機械にデフォルト着信音に限らず、ユーザーが自分の好みに応じて自由に設定できます。



ウェブサーバ

- ウェブサーバ [Web Server] は、ハードウェアのサーバコンピュータ、ソフトウェアのサーバソフト、あるいはその両方からなる整体を意味します。



ウェブサーバ（ハードウェア）

- ハードウェアのウェブサーバは、ウェブサービスのソフトウェアやウェブサイトの構造ファイル（HTML、CSS、JavaScript のコード、画像、動画、テキストなどのデータファイル）を格納するコンピュータです。インターネットに接続され、インターネットに接続された他の機器との物理データのやりとりを実現します。



マネージドサーバ

- 自分のコンピュータですべてのファイルを管理できますが、**専用サーバ**に格納する方がはるかに便利です。なぜなら：
 - 常に稼働し続けて、インターネットに接続されます；
 - 常に同じ IP アドレスを持ちます（すべてのプロバイダーが家庭用 ネットワークに固定 IP アドレスを提供するわけではないです）；
 - サードパーティプロバイダーによる管理されます。
- このような理由から、優秀なホスティング事業者（レンタル サーバ）を見つけることは、ウェブサイトを立ち上げる上で 重要なステップとなります。各社のサービスを比較し、自分のニーズと予算に合ったものを選ぶ必要があります。サービスの価格は、無料から月額数百万円まで様々です。
- ホスティング事業者を決めたら、そのウェブサーバにファイルをアップロードする必要があります。

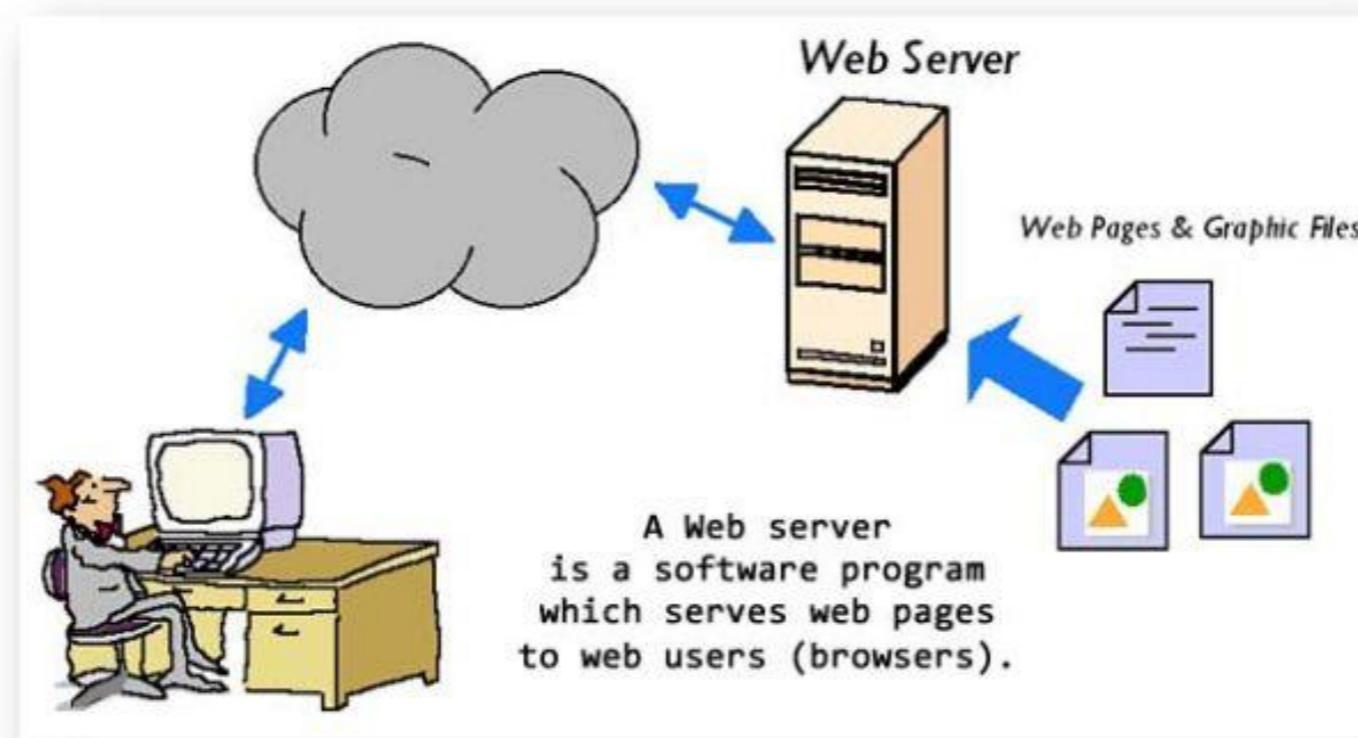
ウェブサーバ（ソフトウェア）

- ソフトウェアのウェブサーバは、ユーザーが管理されたファイルのいくつかの部分にアクセスする方法を制御することができるプログラムです。
- 一般的に、サーバソフトウェアには少なくとも、HTTP サーバの機能を備えている必要があります。HTTP サーバとは、URL (URI) と HTTP プロトコルを理解できるソフトウェアです。HTTP リクエストを受け付け、HTTP レスポンスを返すことができます。

次へ 

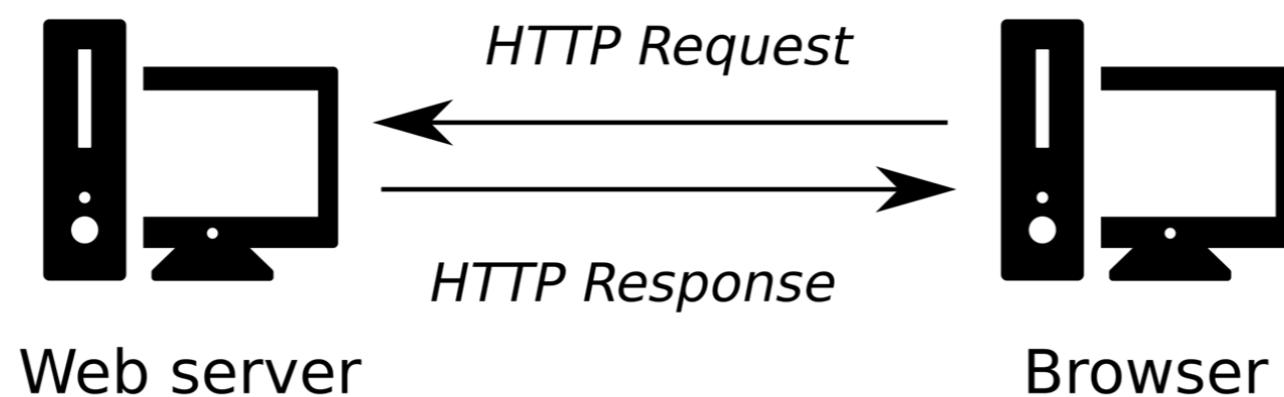
◀ 前へ

- ユーザーは、ウェブサイトに対応するドメイン名（「google.com」「mozilla.org」など）を介してサーバにアクセスし、サーバに保存されているファイルや動的に生成されるファイルを取得することができます。



ウェブサーバの通信方式

- ブラウザがウェブサーバに管理されているファイルを必要とするとき、ブラウザは **HTTPリクエスト** [HTTP Request] を行うこと でサーバにそのファイルを要求します。
- リクエストが問題なくサーバ（ハードウェア）に到達すると、サーバ（ソフトウェア）はリクエストのデータを使って目的の文書ファイルを見つけ、**HTTPレスポンス** [HTTP Response] としてユーザーのブラウザに送り返します。



次へ ➞

◀ 前へ

- サーバが何らかの理由でリクエストを処理できない場合、その失敗した理由を記した**ステータスコード**[Status Code]を返すことがあります。例えば、対応するファイルがサーバ上に存在しない場合、ステータスコード 404 が返されます。



A screenshot of a browser window displaying a Whitelabel Error Page. The address bar shows 'localhost:8080'. The main content area has a dark background with white text. It displays the following information:

← → ⌂ ⓘ localhost:8080

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.
Mon Aug 22 10:31:30 JST 2022
There was an unexpected error (type=Not Found, status=404).
No message available

動的ウェブページのサーバ

- 静的ウェブページのサーバは、**スタック**[Stack]とも呼ばれ、コンピュータ（ハードウェア）とHTTPサーバ（ソフトウェア）で構成されています。このサーバは、管理しているファイルをそのままユーザーのブラウザに配信するため、「静的」と呼ばれています。
- 動的ウェブページのサーバは、静的ウェブページのサーバに加え、**アプリケーションサーバ**（例えば、我々はこれから作るJavaアプリケーションとか）と**データベース**などの追加ソフトウェアで構成されています。HTTPサーバ経由でブラウザに配信される前に、このアプリケーションサーバが管理するファイルを更新するため、「動的」と呼んでいます。

次へ 

◀ 前へ

- 例えば、ブラウザに表示される最終的なウェブページを生成するために、アプリケーションサーバは HTML テンプレートを、データベースの内容で取り換えます。
- 「Wikipedia」のように何万ページもあるサイトもありますが、それらは本当に何万個の HTML ファイルではなく、少数の HTML テンプレートと膨大なデータベースから**生成された**ものです。これにより、コンテンツ配信だけでなく、メンテナンスも迅速かつシンプルに行えるようになりました。

The screenshot shows the English Wikipedia homepage. It features the iconic globe logo and the text "WIKIPEDIA The Free Encyclopedia". Below the logo, there are links for "Main page", "Contents", "Current events", "Random article", "About Wikipedia", "Contact us", "Donate", and "Contribute". The main content area has tabs for "Article" and "Talk". The title "Wikipedia" is displayed, followed by the subtitle "From Wikipedia, the free encyclopedia". A paragraph of text describes Wikipedia as an online encyclopedia. At the bottom, there is a summary of the site's history and its status as the largest and most-read reference work in history.

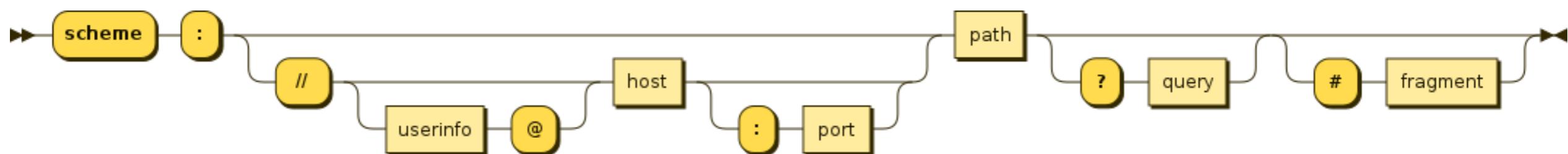
The screenshot shows the Japanese Wikipedia homepage. It features the same globe logo and "WIKIPEDIA" text as the English version. The main content area has tabs for "Article" and "Talk". The title "Wikipedia" is displayed, followed by the subtitle "From Wikipedia, the free encyclopedia". A paragraph of text describes Wikipedia as an online encyclopedia. On the right side, there is a sidebar with links for "メインページ", "コミュニティ・ポータル", "最近の出来事", "新しいページ", "最近の更新", "おまかせ表示", "練習用ページ", "アップロード (Wikimedia Commons)", and "以前の外観に切り替え". Below the sidebar, there is a section for new users with tips and a link to the welcome page. The footer contains copyright information and a link to the English Wikipedia page.

HTTP

- **プロトコル**[Protocol]とは、2台のコンピュータ間で通信のするために設定されたルールです。
- **HTTP** (Hypertext Transfer Protocol) とは、マルチメディアファイル (HTML など) をネットワーク上で転送する基礎となるプロトコルです。HTTP が使われる最も典型的な場面は、人々が閲覧するブラウザとサーバの間でデータを受け渡す時です。現在使われた HTTP 規格のバージョンは HTTP/2 です。
- HTTP は**テキスト型** (すべての通信が**文字列**で行われる) であり、**ステートレス型** (現在の通信が以前の通信の状態を**考慮しない**) です。これらの機能により、インターネットの閲覧が便利になりました。

スキーム

- URLの「http://」の部分は「**スキーム**[Scheme]」と呼ばれ、通常、アドレスの先頭に配置される。
- 「https://google.com」を例にとると、ドキュメントが HTTP プロトコルを使用してリクエストされていることを示します。
- ここでの「https」は、TLS プロトコルを使用して暗号化されていた、安全なバージョンの HTTP プロトコルを表しています。



HTTP によるサーバ通信

- HTTP は、クライアントとサーバの通信方法について明確なルールを定めています。とりあえず知っておくべきのは:
 - HTTP リクエストは、常にクライアントから、サーバに送信されます。サーバは、クライアントからの HTTP リクエストに応答（レスポンス）するだけです。
 - HTTP でファイルを要求する場合、クライアントはこのファイルの URL を提供する必要があります。
 - ウェブサーバは、すべての HTTP リクエストにレスポンスします。リクエストごとに、少なくともステータスコードの 1 つを返さなければなりません。

HTTP リクエスト

- HTML の `<form>` タグに関する以前 (➡ §1.4.1) の勉強から、ユーザーが送信ボタンをクリックすると HTTP リクエストが行われることを思い出してください。例えば、下記のフォームでは、ユーザーが名前を記入して送信した後、「**フォームの情報**と「サーバの『/HelloWorld』ページに移動してください」という要求を含むリクエストが送信されます。

```
<form action="/HelloWorld">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Forms

First name:

John

Last name:

Doe

Submit

次へ ➡

 前へ

- ユーザーが送信をクリックすると、入力された情報はサーバ上のプログラム（コントローラーともいう）に渡されます。プログラムは、ユーザーの情報を使って何かを行い（例えば、正当なユーザーかどうかを判断）、適切な HTTP レスポンスでリクエストに応答します。この例では、レスポンスによって、「/HelloWorld」ページに移動します。

```
<form action="/HelloWorld">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

HTML Forms

First name:

Last name:

ウェブ開発の技術

- ウェブ開発では、通常にクライアント側とサーバ側の両方の技術が適用されます。クライアント側の技術は情報コンテンツを表示するために使われます。サーバ側の技術はビジネスロジックを処理すると、データベースをやり取りするために使われます。
- クライアントの技術は、これまで勉強してきた HTML、CSS、JavaScript に関するものが中心です。

サーバの技術

- サーバの代表的な技術は、以下のようなものがあります：
 - **CGI** (Common Gateway Interface) : ブラウザとサーバ間の対話を可能にする動的ウェブページを作成するために使用される最初の技術です。
 - **PHP**: Personal Home Page (個人ホームページ) という言葉に由来しましたが、今では PHP は名詞の略称ではなく、動的なページを開発するための技術の名称となっています。
 - **JSP** (Java Server Page) : Java の API に従った、Java に基づいての動的ウェブ技術で、静的コンテンツを表す HTML コードの中に、JSP タグで Java コードを埋め込んで動的コンテンツを生成します。Java の様々な API を自由に使えます。



Q & A

Question and answer



1 ウェブ開発の基
本

2 Tomcat

3 Servlet

Tomcatとは

- Tomcat は、ウェブサーバ（ソフトウェア）の一種です。Tomcat を通して、簡単にリクエストを受信してレスポンスすることができます（Tomcat を使わない場合は、リクエストを受信して応答するために自分でソケット[Socket]を書く必要があります）。
- Tomcat は、後に紹介する Servlet や JSP のページを読み込むためのコンテナでもあります。
- Tomcat については、あまり深く学ぶ必要はなく、インストールと起動の方法を学び、各ディレクトリの意味を理解すればよいのでしょう。

Tomcatのインストール

- Tomcat 9.0 を使用しています。ダウンロードリンク:
 <https://tomcat.apache.org/download-90.cgi>。

9.0.45

Please see the [README](#) file for packaging information. It explains what every distribution contains.

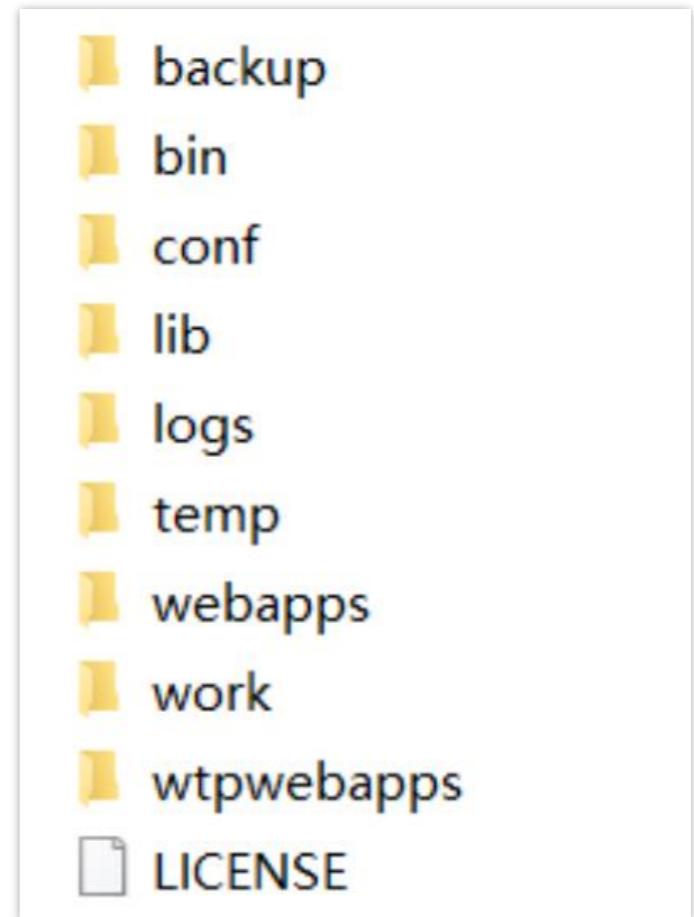
Binary Distributions

- Core:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#) ← macOS
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#) ← Windows
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)
- Full documentation:
 - [tar.gz \(pgp, sha512\)](#)
- Deployer:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
- Embedded:
 - [tar.gz \(pgp, sha512\)](#)
 - [zip \(pgp, sha512\)](#)

- パッケージをダウンロードし、解凍してください。解凍したパッケージのアドレスは、後で必要になりますので、必ず覚えておいてください。

Tomcat プロジェクトのディレクトリ構造

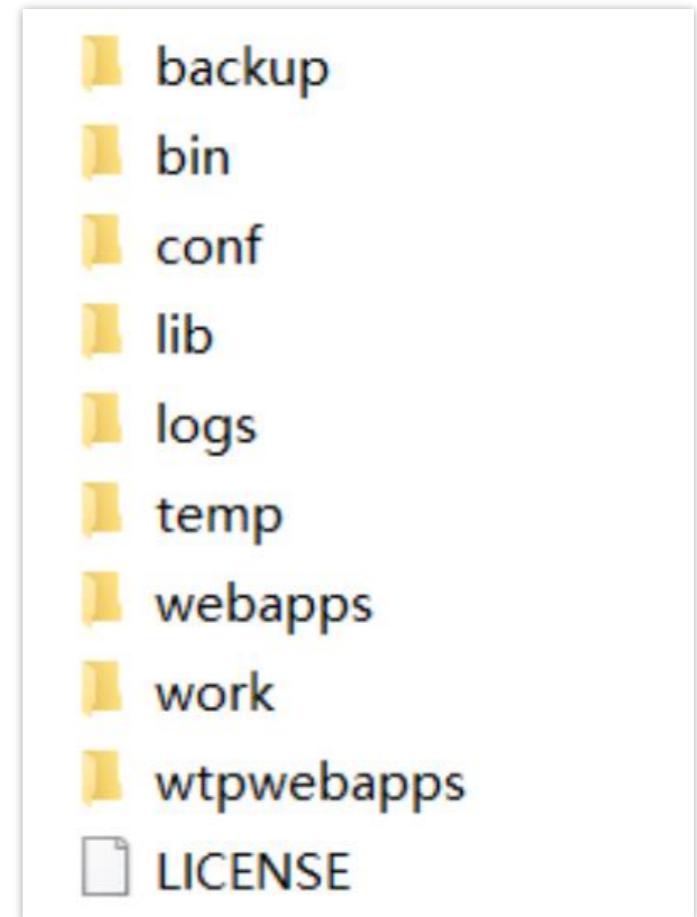
- bin: このディレクトリには、バイナリ実行ファイルを保存します。



次へ ➞

◀ 前へ

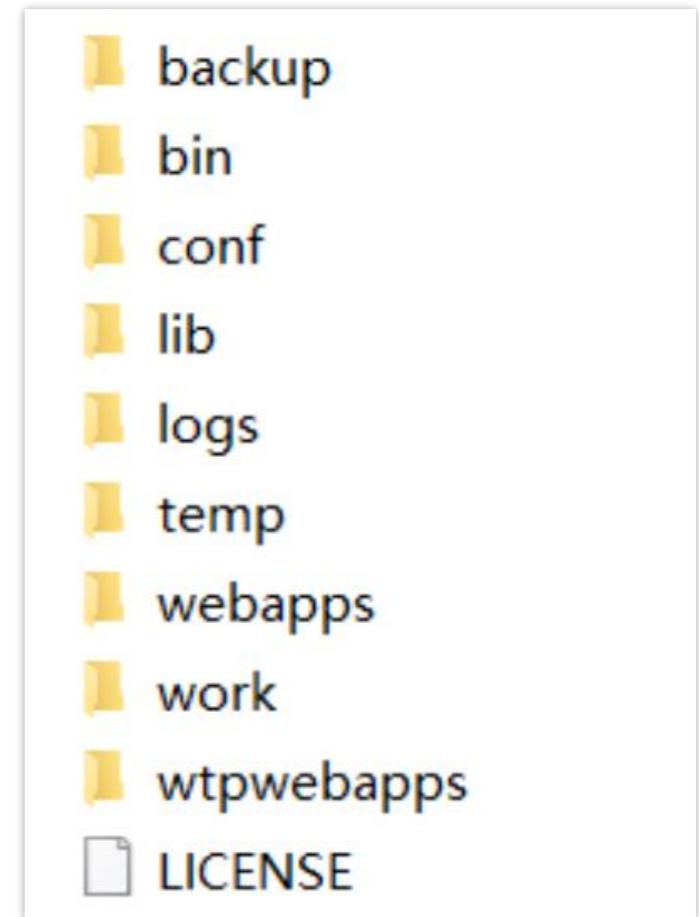
- conf: このディレクトリには、4つの最も重要なファイルがあります:
 - server.xml: サーバの全体の情報を設定。
 - tomcat-users.xml: ユーザー名、パスワード、役割情報などを含む Tomcat のユーザーを保存。
 - web.xml: デプロイメントディスクリプタファイル。いくつかの MIME タイプ（文書タイプ）が記録されています。クライアントとサーバの間で、文書の種類を指定するために使用されます。例えば、ユーザーが HTML ページを要求した場合、サーバは応答が text/html タイプのドキュメントであることを伝えます。
 - context.xml: すべてのアプリケーションを統一で設定します。通常は変更しません。



次へ ▶

◀ 前へ

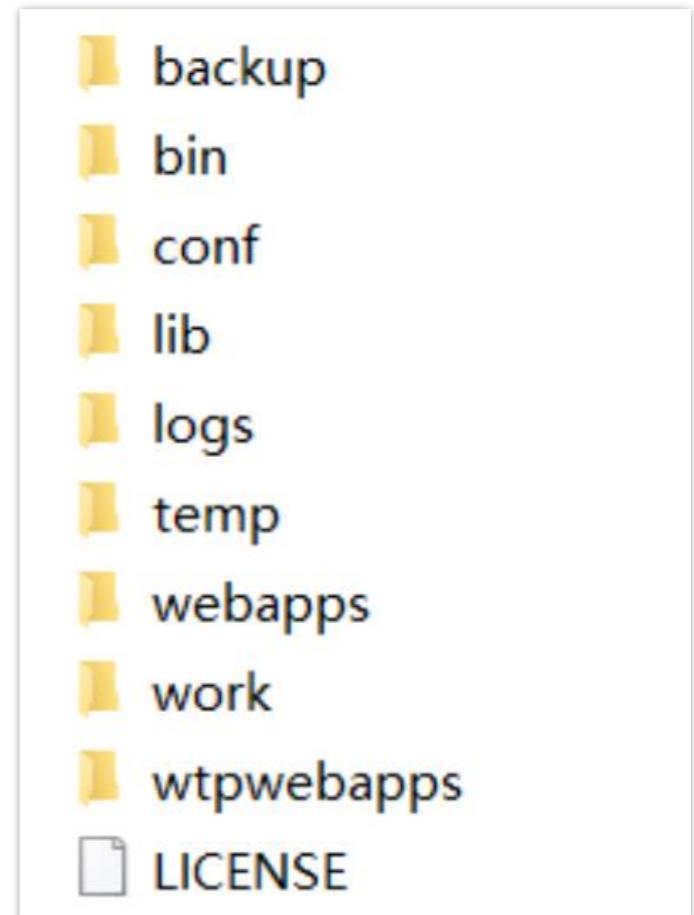
- lib: Tomcat は、クラスライブラリに多くの Jar パッケージを持っています。Tomcat が依存する Jar パッケージを追加する場合、このディレクトリに置くことができます。このディレクトリの Jar パッケージは全プロジェクトで共有できますが、その場合、単独のプロジェクトを別の Tomcat の下に移転したときに、それらの Jar パッケージが共有できなくなるので、このディレクトリには Tomcat が必要とする Jar パッケージのみを置くことをお勧めします。



次へ ▶

◀ 前へ

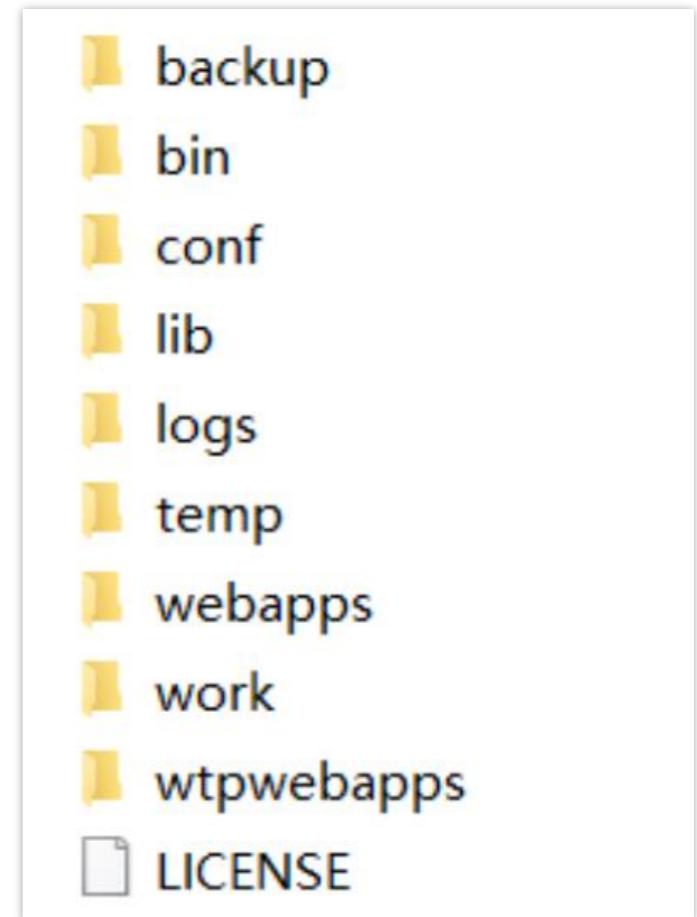
- logs: このディレクトリには、Tomcat の起動と終了に関する情報を記録するログファイルがあり、Tomcat の起動時にエラーが発生した場合は、その例外もログファイルに記録されます。
- temp: このディレクトリは Tomcat の一時ファイルを保持し、Tomcat を停止したときに削除されます。



次へ ➞

◀ 前へ

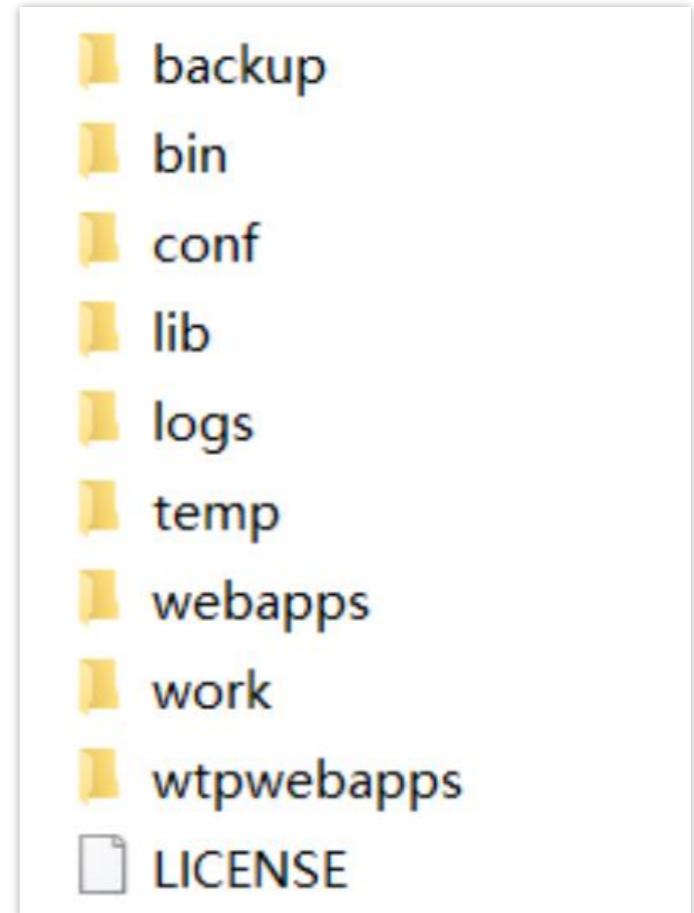
- `webapps`: ウェブプロジェクトが格納されるディレクトリ。これらのフォルダーは、それぞれプロジェクトです。その中、「ROOT」は特別なプロジェクトで、アドレスバーにプロジェクトディレクトリが与えられていない場合は、ROOT プロジェクトに対応します。例えば、「`http://localhost:8080/examples`」は `examples` プロジェクトにアクセスします。ここで、`examples` はプロジェクト名で、フォルダ名に対応します。



次へ ▶

 前へ

- work: 実行時に生成されるファイル。最終的なランタイムファイルはここにあります。webapps のプロジェクトから生成されます。このディレクトリを削除して、再度コンパイルすると、ワークディレクトリが再生成されます。クライアントユーザが JSP ファイルにアクセスすると、Tomcat は JSP から「.java」ファイルを生成し、「.java」ファイルをコンパイルして「.class」ファイルを生成し、このディレクトリに格納します。





Question and answer



1 ウェブ開発の基
本

2 Tomcat

3 Servlet

Java ウェブアプリケーション

- Java ウェブアプリケーションは、ウェブサーバ（Tomcat）上で動作するアプリケーションで、Servlet、JSP ページ、Java クラス、その他のリソースをセットで構成され、階層化されたディレクトリに格納されています。

Servlet とは

- **Servlet** とは、SUN による、Java で動的ウェブページを開発する方法を定義した仕様である。つまり、Java をウェブサイトのバックエンドの開発に使うために、Servlet の基準に従わなければなりません。
- Servlet は、Java のすべての API を使用でき、豊富で強力なライブラリを備えています。
- Servlet を通して：
 - <form> でユーザーから送信された**情報を受信できます**；
 - **ユーザー情報、記事内容、ページヒット数**などのデータベースが照会できます；
 - 悪意のあるアクセスを防ぐための**認証機能**も備えています。

なぜ Servlet を学ぶのか？

- Servlet は現在ではもう古い技術であり、実際の開発は Spring MVC・Spring Boot のようなフレームワークで行うのは一般になりました。
- しかし、Servlet は Java のウェブプログラミング技術の基幹で、例えば Spring MVC のコアも実際に Servlet が使われています。
- Servlet の概念をちゃんと理解できれば、様々な流行っているフレームワークの習得も容易になるでしょう。

Servlet の例

- 例えば、ウェブページに IP アドレスを表示する場合、その HTML コードは次のようになります：

```
1 <!DOCTYPE html>
2
3 <html>
4 <head>
5   <meta charset="UTF-8">
6   <title>IP Adress</title>
7 </head>
8
9 <body>
10  <p>Your IP Address: 172.0.0.1 .</p>
11 </body>
12 </html>
```

次へ ➞

◀ 前へ

- それに対応するサーバ上の Java コードは次のように:

```
1 // 必要なライブラリを導入
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5
6 // HttpServlet クラスを継承
7 public class HelloWorld extends HttpServlet {
8     public void init() throws ServletException {
9         // 初期化処理
10    }
11
12    public void doGet(
13        HttpServletRequest request,
14        HttpServletResponse response
15    ) throws ServletException, IOException {
16        // リスポンスのタイプを設定
17        response.setContentType("text/html");
18
19        // println() メソッドで HTML を出力
20        PrintWriter out = response.getWriter();
21
22        out.println("<!DOCTYPE html>");
23        out.println("<html>");
24        out.println("<head>");
25        out.println("  <meta charset=\"UTF-8\">");
26        out.println("  <title>IP Adress</title>");
27        out.println("</head>");
28        out.println("<body>");
29        out.println("    <p>");
30        out.println(request.getRemoteAddr());
31        out.println("    </p>");
32        out.println("  </body>");
33        out.println("</html>");
34    }
35
36    public void destroy() {
37        // 終了時の処理
38    }
39}
```

- ユーザーが受け取った HTML コードは、`println()` 文で出力されます。

JSP

- これは基本的な CGI プログラムでした。HTML コードを文字列として扱い、print 文で一つ一つ出力する必要がありました。インターネットの発展初期では、CGI が大流行し、インターネットの発展に欠かせない貢献をしました。
- JSP (Java Server Pages) は、Servlet の作業を簡略化するために登場した代替ツールである。Servlet は HTML の出力が非常に難しく、JSP は HTML の Servlet 出力の代わりとなるものです。
- JSP は、HTML と Java のコードを併記した文法をベースにするプログラムです。

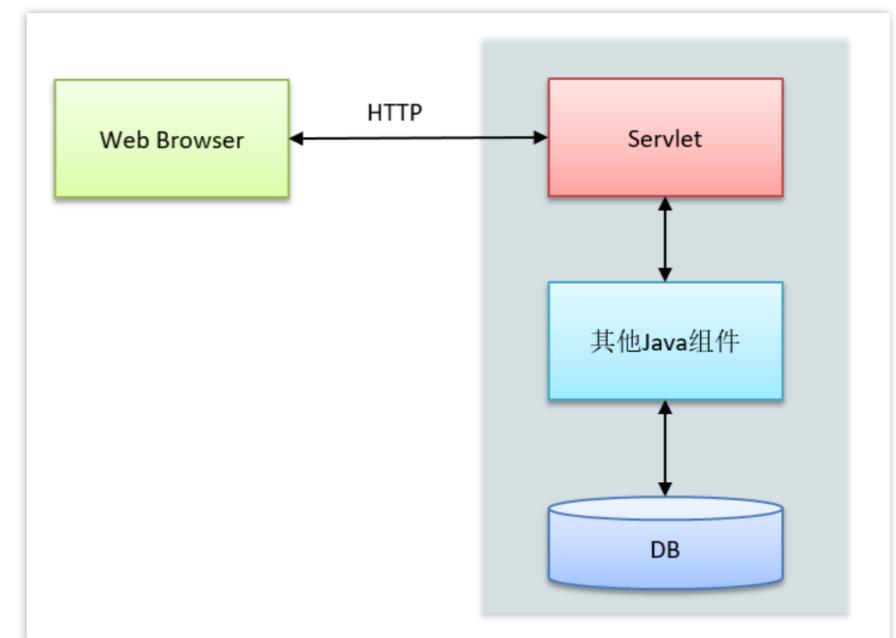
Servlet パッケージ

- Servlet は、Java Servlet の仕様をサポートするインタプリタを持つ Web サーバ上で動作する Java クラスです。
- Servlet は、`javax.servlet` と `javax.servlet.http` パッケージを使用して作ることができます。大規模な開発プロジェクトをサポートする Java ライブラリの拡張版である Java Enterprise Edition (Java EE) バージョンは、スタンダードでそれらが搭載されています。
- Servlet は、他の Java クラスと同様に、すでに作成され、コンパイルされています。Servlet パッケージをダウンロードし、クラスパスをビルドパスに追加すると、JDK の Java コンパイラやその他のコンパイラで Servlet アプリケーションをコンパイルして実行することができるようになります。

Servlet の機能

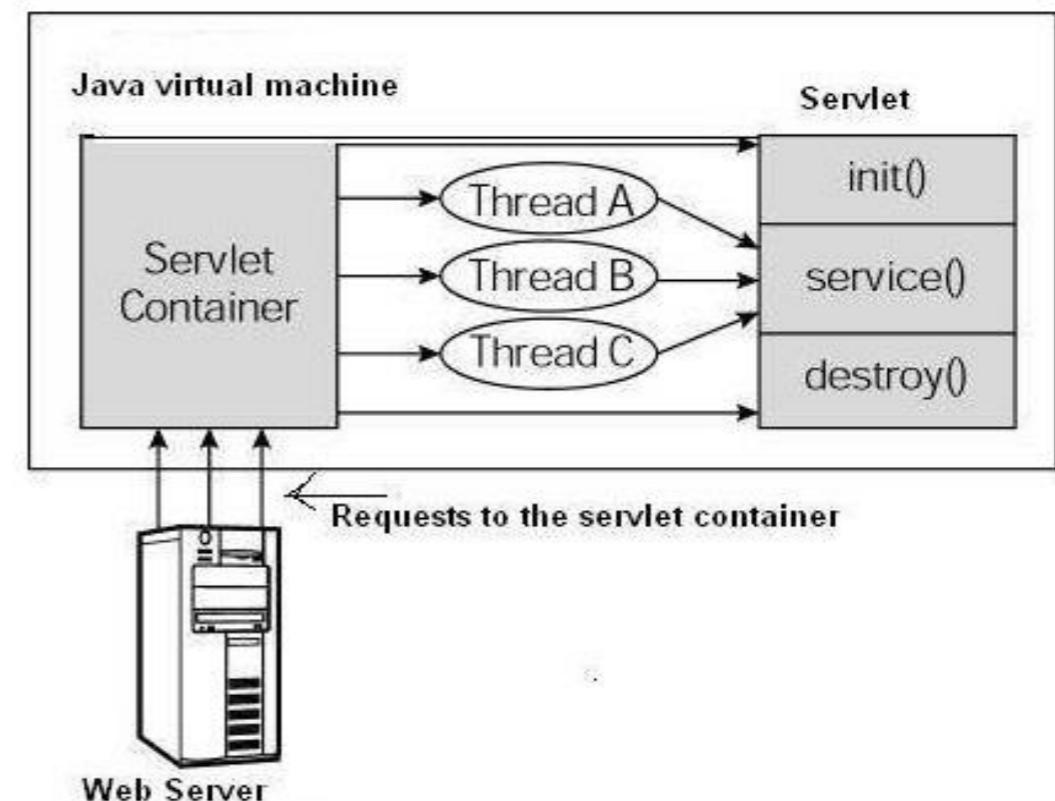
- **Servlet の機能:**

- **リクエストデータの受信:** クライアントのリクエストは HttpServletRequest オブジェクトにされ、リクエストヘッダやパラメータなど様々な情報を含んでいます。
- **リクエストの処理:** service()、doPost() または doGet() などのメソッドを使ってパラメータを受け取り、ビジネス層のメソッドを呼び出してリクエストを処理します。
- **レスポンスの終了:** リクエストを処理した後、ページにフォワード (Forward) またはリダイレクト (Redirect) します。



Servlet のライフサイクル

1. `init()` メソッドを呼び出すことで初期化されます。
2. `service()` メソッドを呼び出すことでクライアントのリクエストを処理します。
3. `destroy()` メソッドを呼び出すことで終了します。
4. 最後に、Servlet は JVM のガベージコレクターによって回収される。



ライフサイクルの詳細

● init():

- `init()` メソッドは、`Servlet` のライフサイクル中に 1 回だけ実行されます。`Servlet` がサーバにロードされたときに実行され、`Servlet` オブジェクトの**初期化**を担当します。
- サーバは、サーバの起動時またはクライアントが `Servlet` に最初にアクセスしたときに `Servlet` をロードします。いくらクライアントが `Servlet` にアクセスしても、`init()` が繰り返し実行されることはありません。

次へ ➔

◀ 前へ

● service():

- Servlet の中心となるもので、クライアントからのリクエストにレスポンスする役割を担っています。
- クライアントが HttpServlet オブジェクトをリクエストすると、そのオブジェクトの service() メソッドが呼び出され、ServletRequest オブジェクトと、ServletResponse オブジェクトがパラメータとして渡されます。
- service() メソッドは HttpServlet クラスに既に存在しています。デフォルトでは、HTTP リクエストのメソッドに対応した do 機能 (doGet(), doPost()) を呼び出します。

次へ ▶

 前へ

- `destroy()`:

- `destroy()` メソッドは、`Servlet` がサーバ側で**停止**し、アンロードされたときに一度だけ実行されます。
- `Servlet` オブジェクトがライフサイクルを終了するとき、`destroy()` は占有していたリソースを解放する役割を果たします。`Servlet` は `service()` メソッドの実行中に他のスレッドを生成することがあるので、`destroy()` メソッドが呼ばれるまでに、これらのスレッドの終了または破棄を確保する必要があります。

Servlet がリクエストを処理する手順

1. クライアントは Servlet コンテナ (Tomcat) に HTTP リクエストを送信します。
2. Servlet コンテナはクライアントからのリクエストを受け取ります。
3. Servlet コンテナは HttpServletRequest オブジェクトを作成し、このオブジェクトにリクエスト情報を格納します。
4. Servlet コンテナは、HttpServletResponse オブジェクトを作成します。

次へ 

前へ

5. Servlet コンテナは HttpServlet オブジェクトの service() メソッドを呼び出し、HttpRequest オブジェクトと HttpServletResponse オブジェクトを HttpServlet オブジェクトにパラメータとして渡します。
6. HttpServlet は HttpRequest オブジェクトの関連するメソッドを呼び出し、HTTP リクエストに関する情報を取得します。
7. HttpServlet は HttpServletResponse オブジェクトの関連するメソッドを呼び出して、レスポンスデータを生成します。
8. Servlet コンテナは HttpServlet のレスポンス結果をクライアントに返します。



Question and answer

Servlet の実例

- **HttpServlet** は、HTTP リクエストを処理する Java クラスで、Servlet インターフェイスを実装しています。ウェブ開発者は、その HttpServlet を継承して独自の処理を書くことが多いです。

リクエストを処理するため doGet() メソッドをオーバーライド。

```

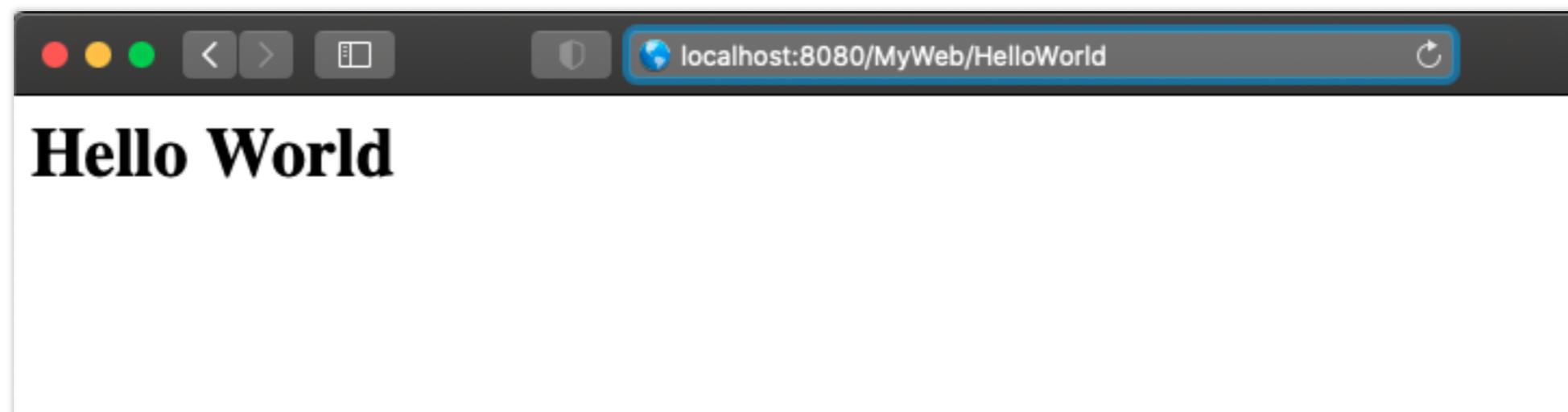
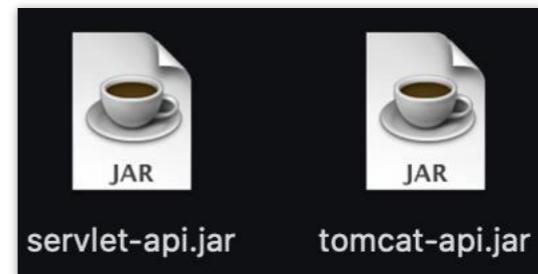
1 // 必要なライブラリを導入
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.annotation.WebServlet;
5 import javax.servlet.http.*;
6
7 // HttpServlet クラスを継承          HttpServlet
8 public class HelloWorld extends HttpServlet {
9
10    private String message;           クラスを継承
11
12    public void init() throws ServletException {
13        // 初期化処理
14        message = "Hello World";      init() メソッドを
15    }                                オーバーライドし、
16                                         ウェブページを初期化
17
18    public void doGet(
19        HttpServletRequest request,
20        HttpServletResponse response
21    ) throws ServletException, IOException {
22        // リスポンスのタイプを設定
23        response.setContentType("text/html");
24
25        // println() メソッドで HTML を出力
26        PrintWriter out = response.getWriter();
27        out.println("<h1>" + message + "</h1>");
28    }
29
30    public void destroy() {
31        // 終了時の処理
32    }

```

Servlet HelloWorld を作成します

- 準備:

- Eclipse EE と、
- Tomcat をインストールし、
- `servlet-api.jar` パッケージと `tomcat-api.jar` パッケージをダウンロード (codes フォルダにもある)。



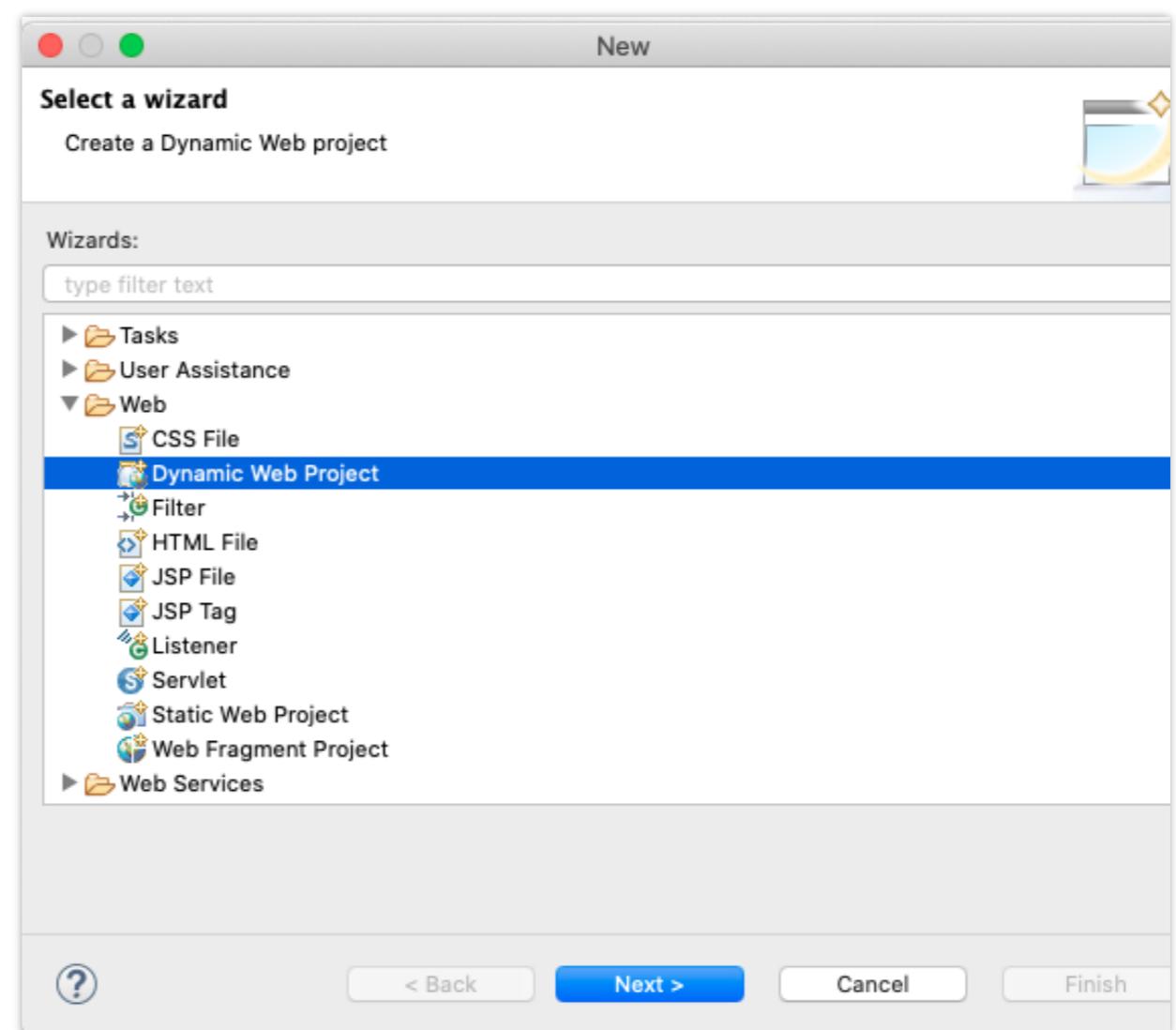
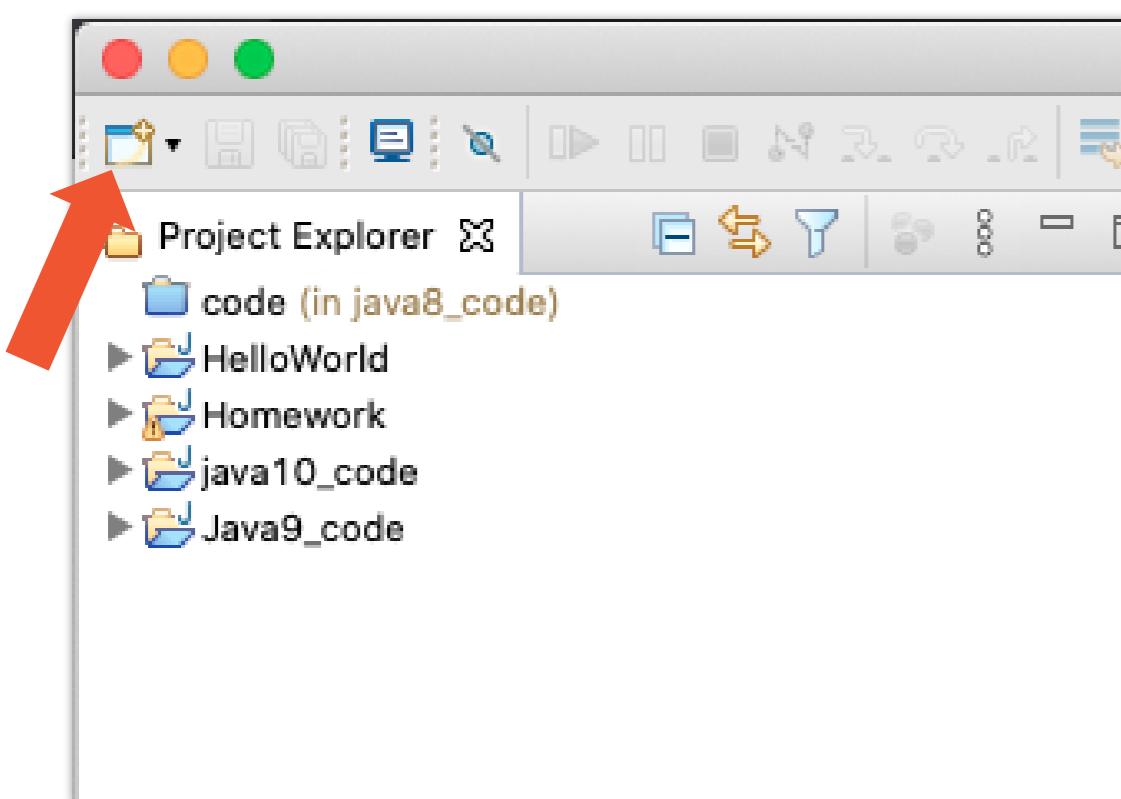
イメージ図

Servlet 開発の手順

1. Servlet クラスを作成。
2. URL のパスを設定します: web.xml で設定するか、クラスファイルに「@WebServlet("/xxx")」というアノテーションを追加。
3. Tomcat サーバにデプロイ。
4. Tomcat を起動。
5. ブラウザを開き、Servlet にアクセスする URL を入力:
 - `http://サーバIP:ポート番号/アプリ名/設定したパス`

ステップ1: プロジェクトの作成

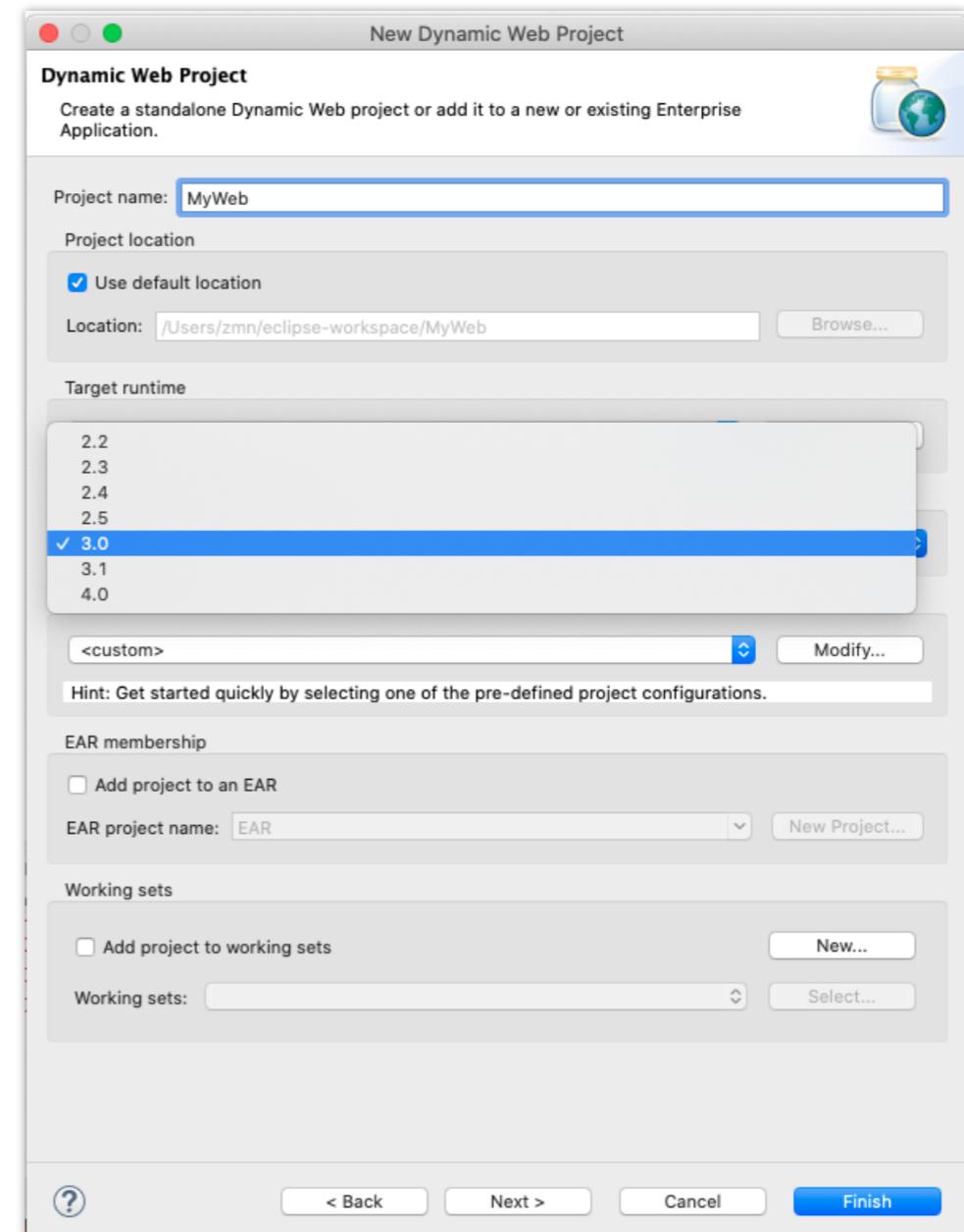
- Eclipse を開き、Dynamic Web Project を探し、Next をクリック。



次へ ➞

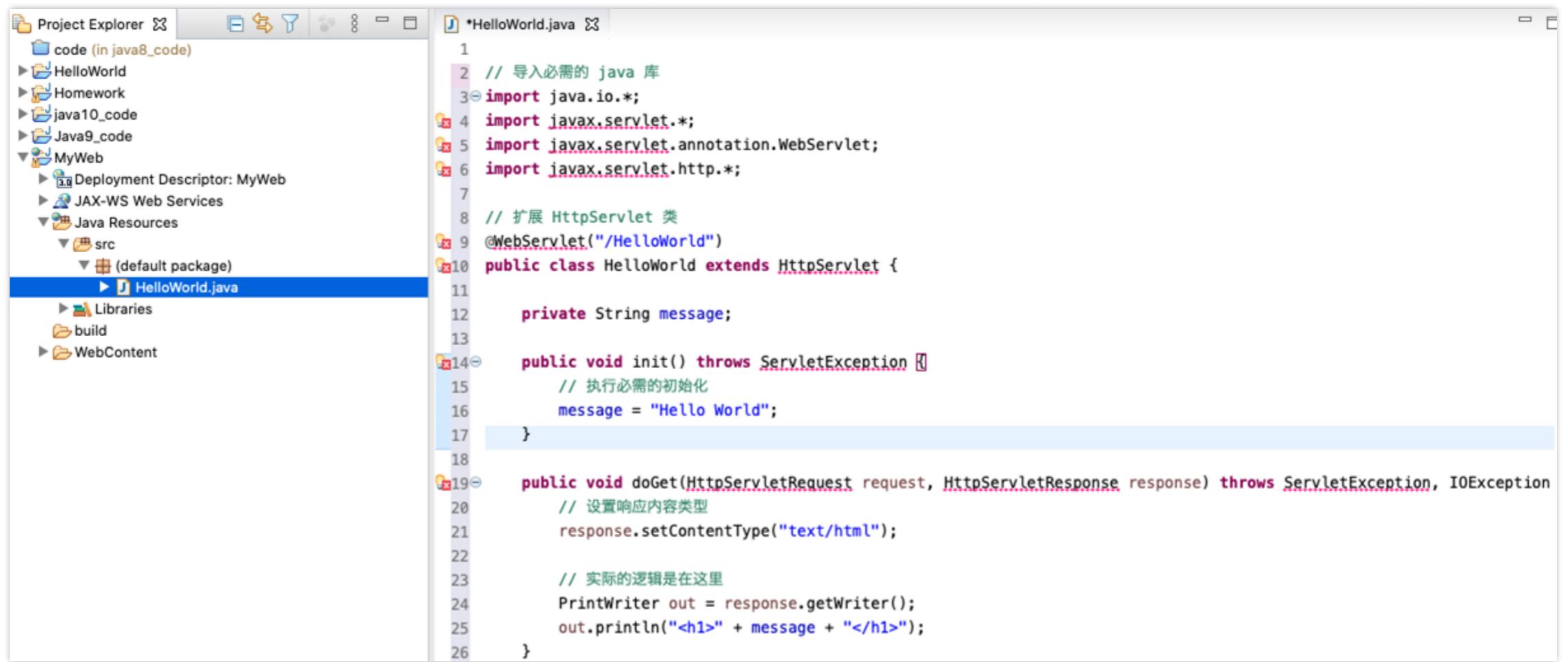
◀ 前へ

- プロジェクト名は「MyWeb」をしてください。Dynamic Web Moduleを手動でバージョン3.0に設定し、Finishをクリックします。



ステップ 2: Servlet クラスの定義

- 「MyWeb/Java Resources/src」を探して、その中に「HelloWorld.java」を作成し、配布されたコードをコピー。

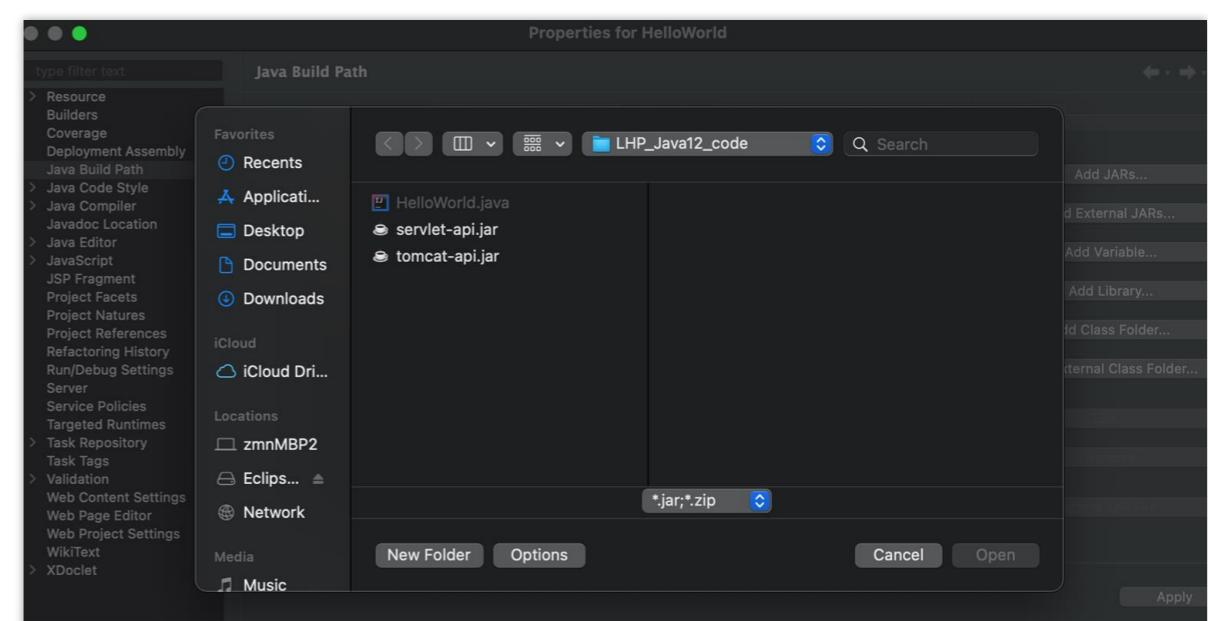
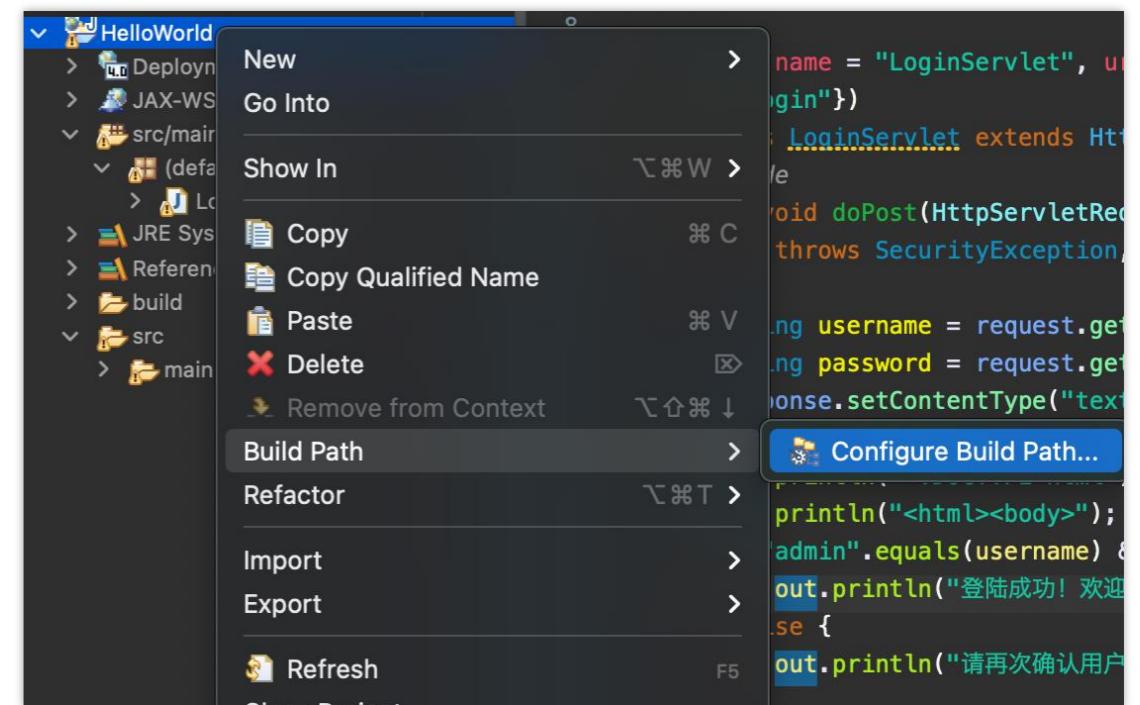


The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a project named 'MyWeb' containing several Java files and folders like 'code (in java8_code)', 'HelloWorld', 'Homework', 'java10_code', 'Java9_code', and 'JAX-WS Web Services'. Inside 'MyWeb', there is a 'src' folder which contains a file named 'HelloWorld.java'. This file is selected and shown in the code editor on the right. The code editor displays the following Java code:

```
1 // 导入必需的 java 库
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.annotation.WebServlet;
5 import javax.servlet.http.*;
6
7 // 扩展 HttpServlet 类
8 @WebServlet("/HelloWorld")
9 public class HelloWorld extends HttpServlet {
10
11     private String message;
12
13     public void init() throws ServletException {
14         // 执行必需的初始化
15         message = "Hello World";
16     }
17
18     public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
19         // 设置响应内容类型
20         response.setContentType("text/html");
21
22         // 实际的逻辑是在这里
23         PrintWriter out = response.getWriter();
24         out.println("<h1>" + message + "</h1>");
25     }
26 }
```

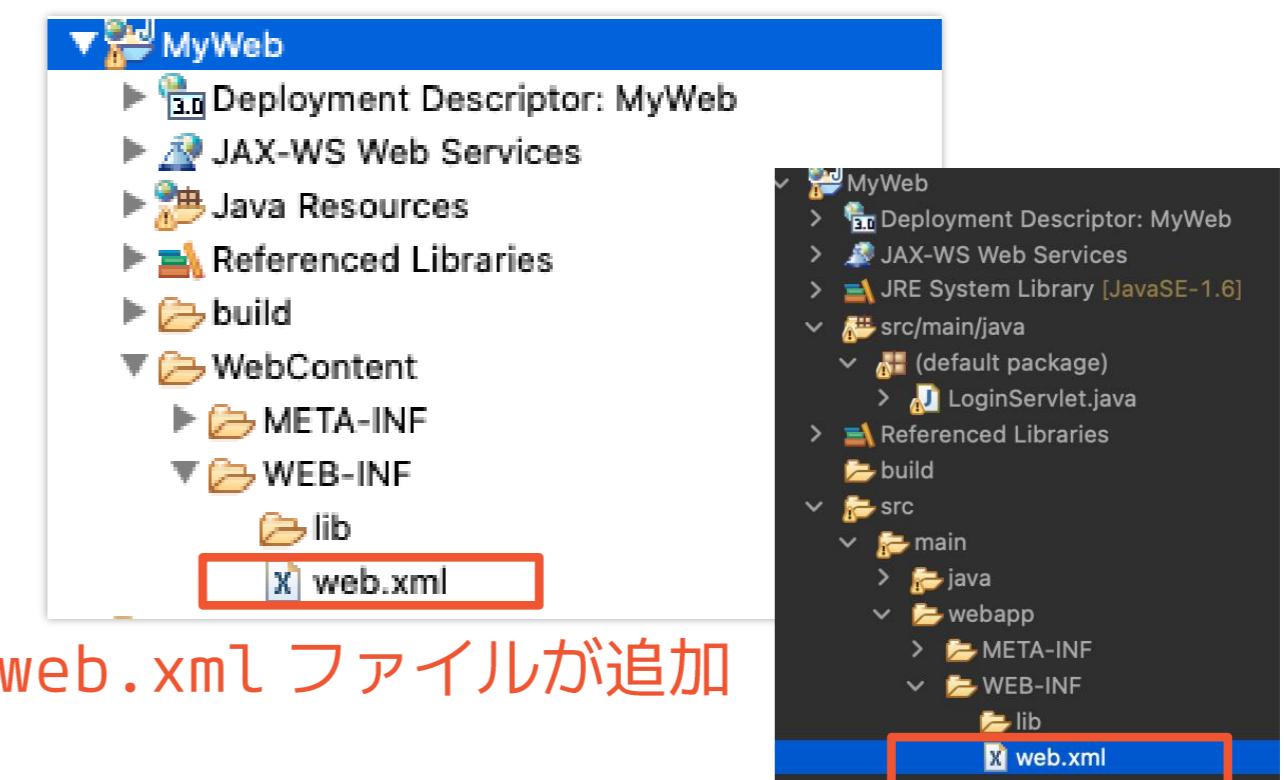
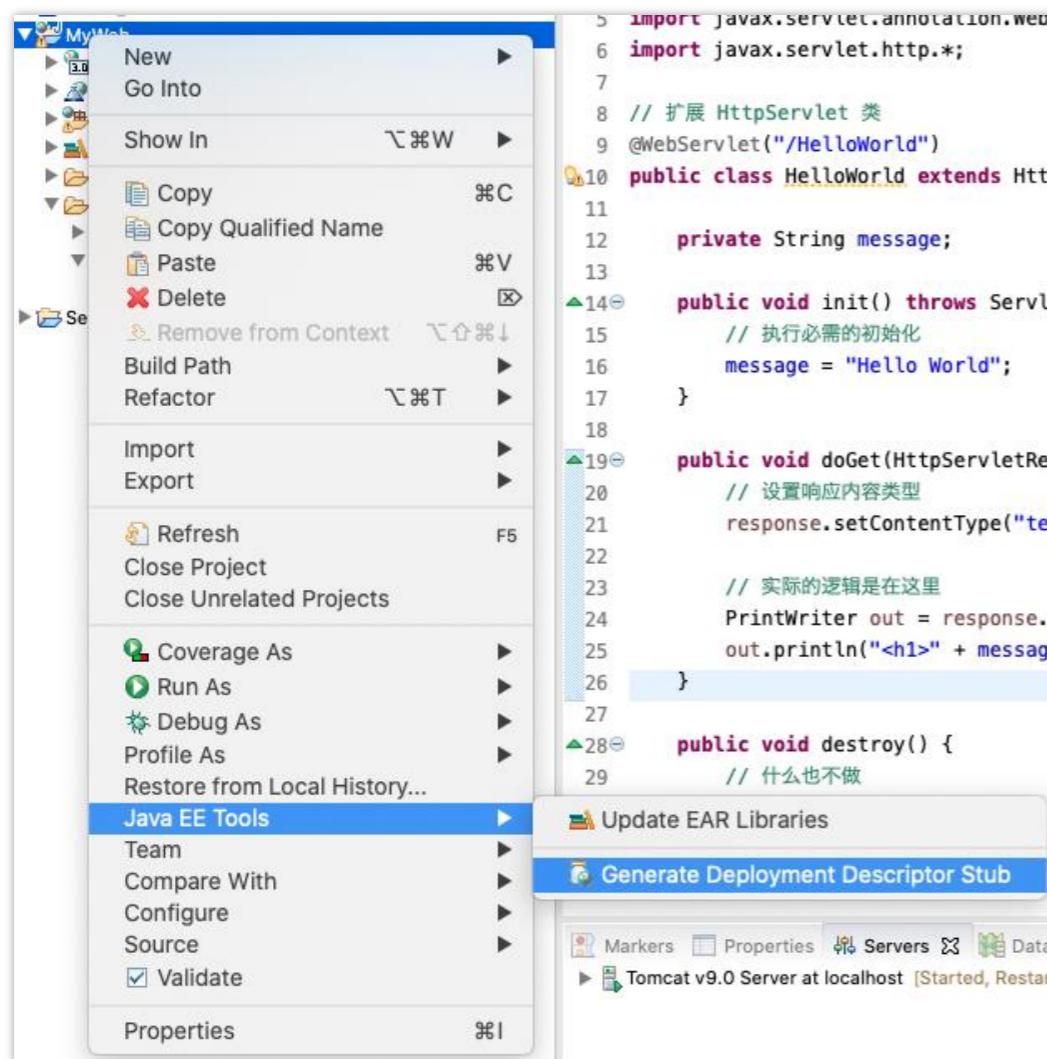
ステップ 3: パッケージのインポート

- コンパイルエラーが出たのは、パッケージがまだインポートされていないため。プロジェクトを右クリック → Build Path → Configure Build Path。
- Libraries を選択し、Add External JARs をクリック。
- 配布されたコードからすべての Jar パッケージを追加。
- Open → Apply and Close。



ステップ 4: web.xml ファイルの追加

- プロジェクトの設定ファイルであり、どの Servlet がどの URL に対応するかを宣言するために使用します。プロジェクトを右クリック → Java EE Tools → Generate。



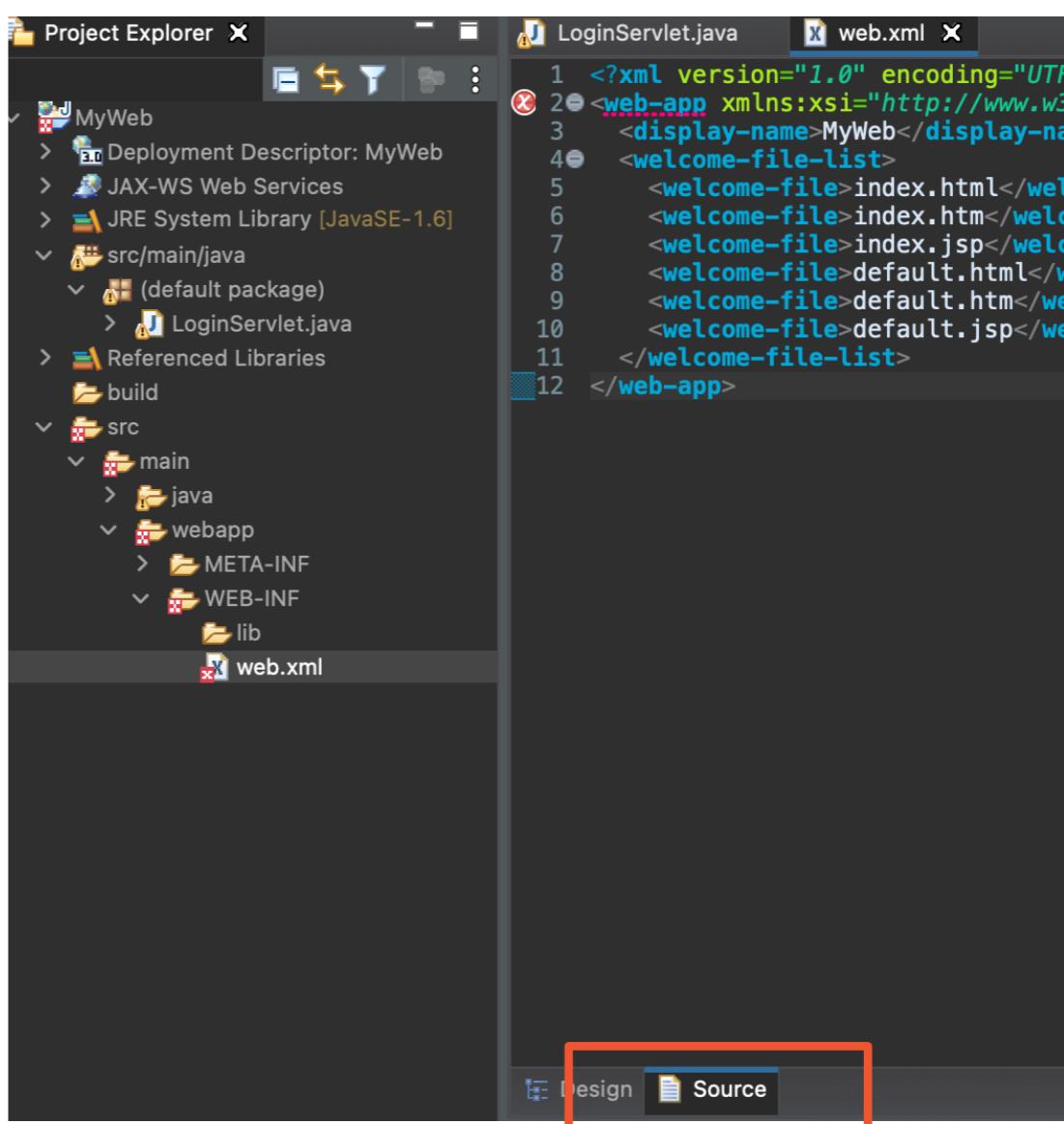
web.xml ファイルが追加

またはこちら

次へ ➞

◀ 前へ

- web.xml ファイルは、Design と Source の二つのモードで表示されていますので、Source モードを選択します。



次へ ▶

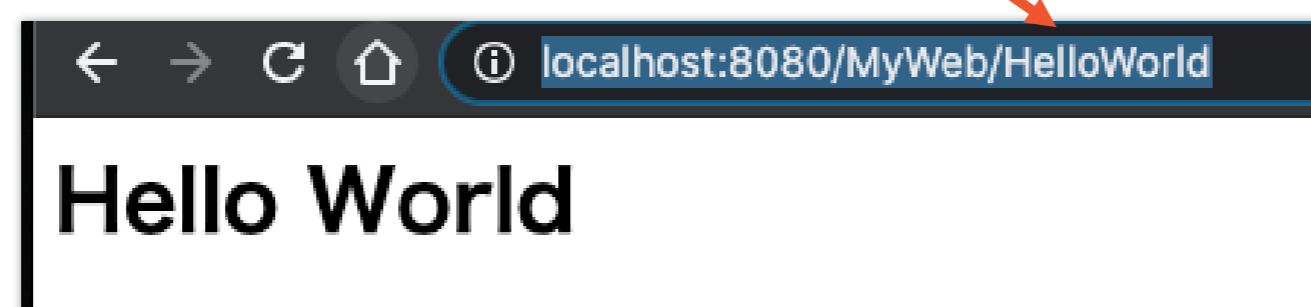
◀ 前へ

- <web-app> の中に以下のようにタグを追加し、保存します：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <display-name>MyWeb</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet クラス名

名前を一致させる



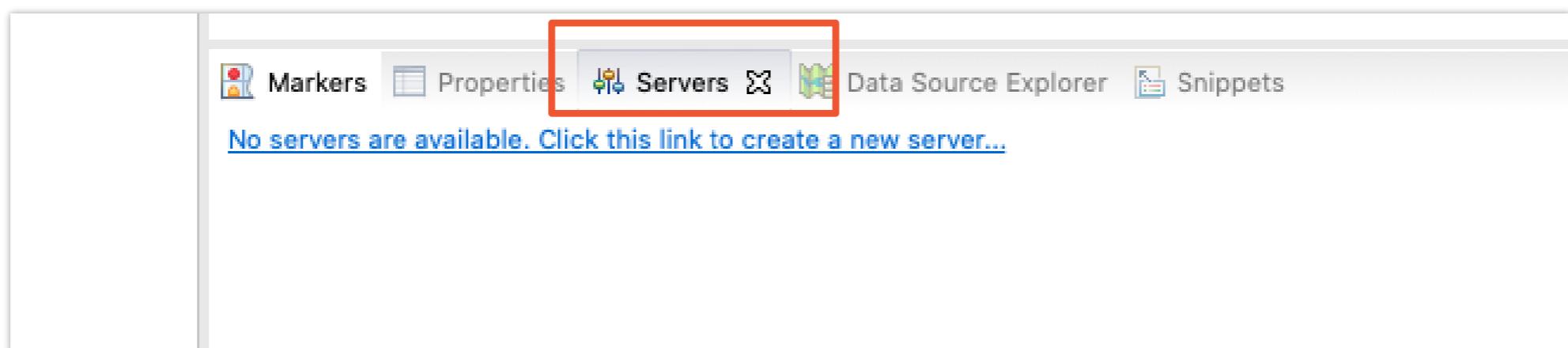
◀ 前へ

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://JAVA.sun.com/xml/ns
```

設定ファイルのコードにコンパイルエラー
が表示する場合、ここで「java」を大文字
に変更して保存してみてください。

ステップ 5: サーバのセットアップ

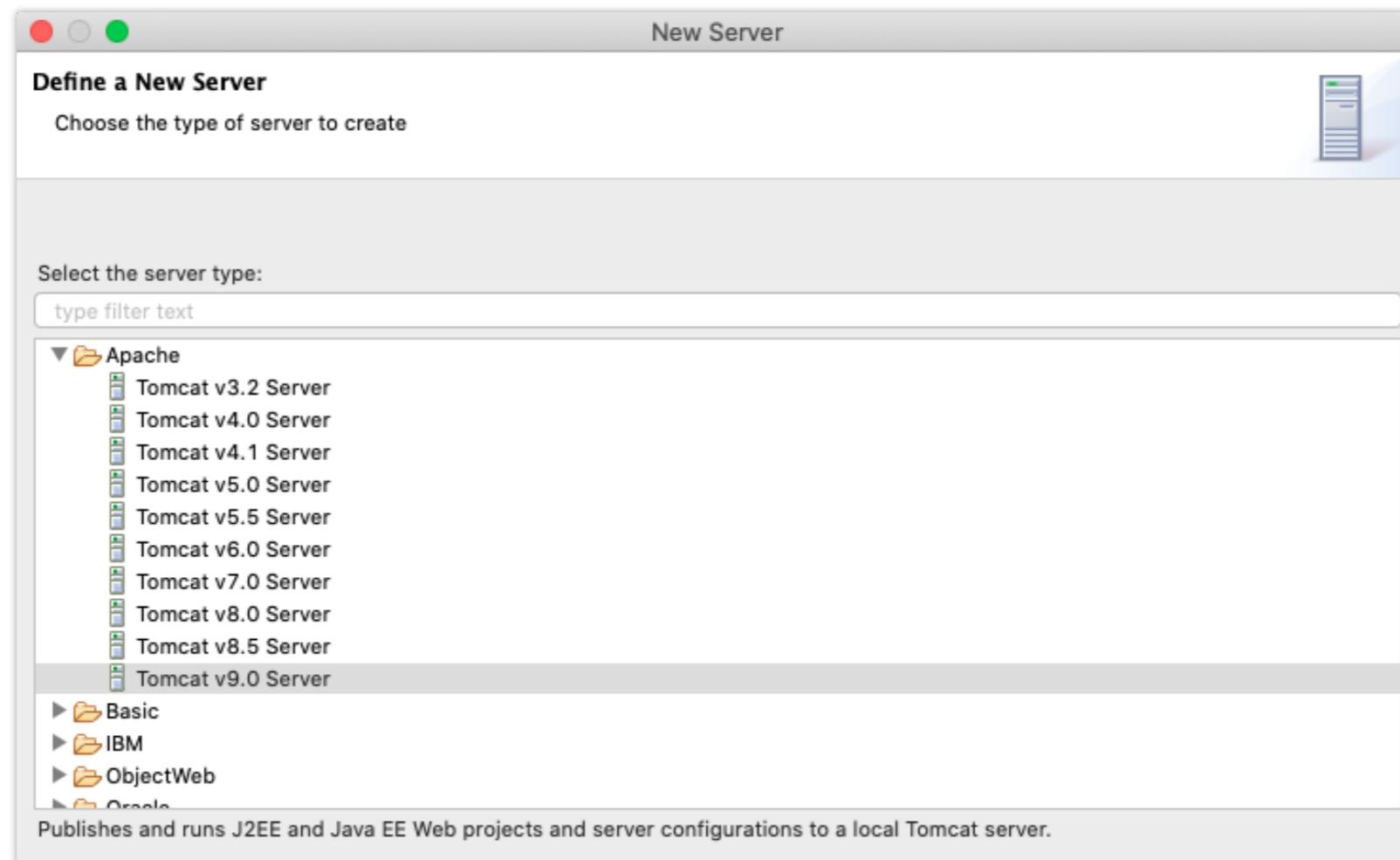
- Eclipse の下にある Servers を選択し、新しいサーバを作成します。



次へ ➞

◀ 前へ

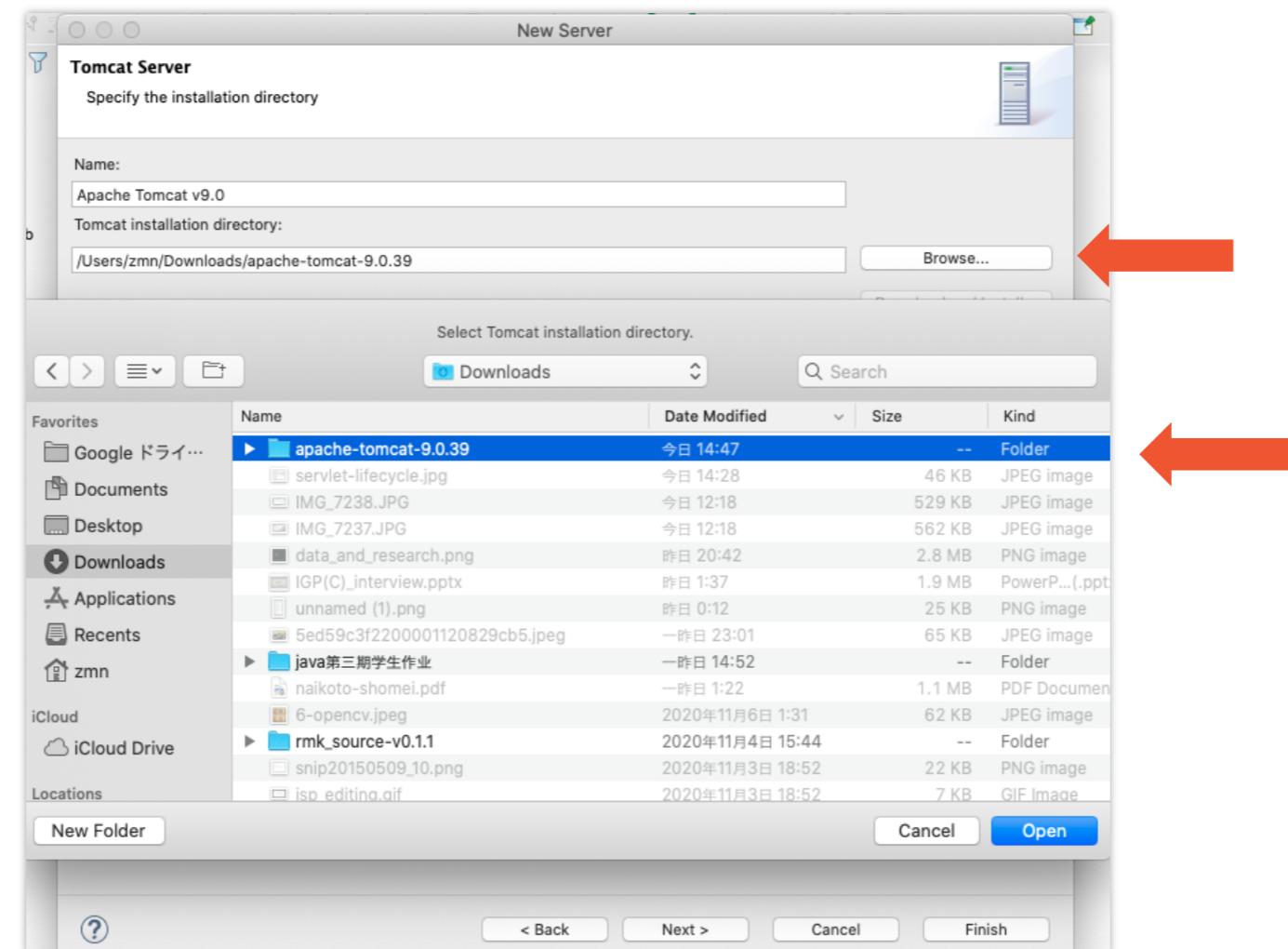
- Apache Tomcat v9.0 を選択し、「Next」をクリックします。



次へ ➡

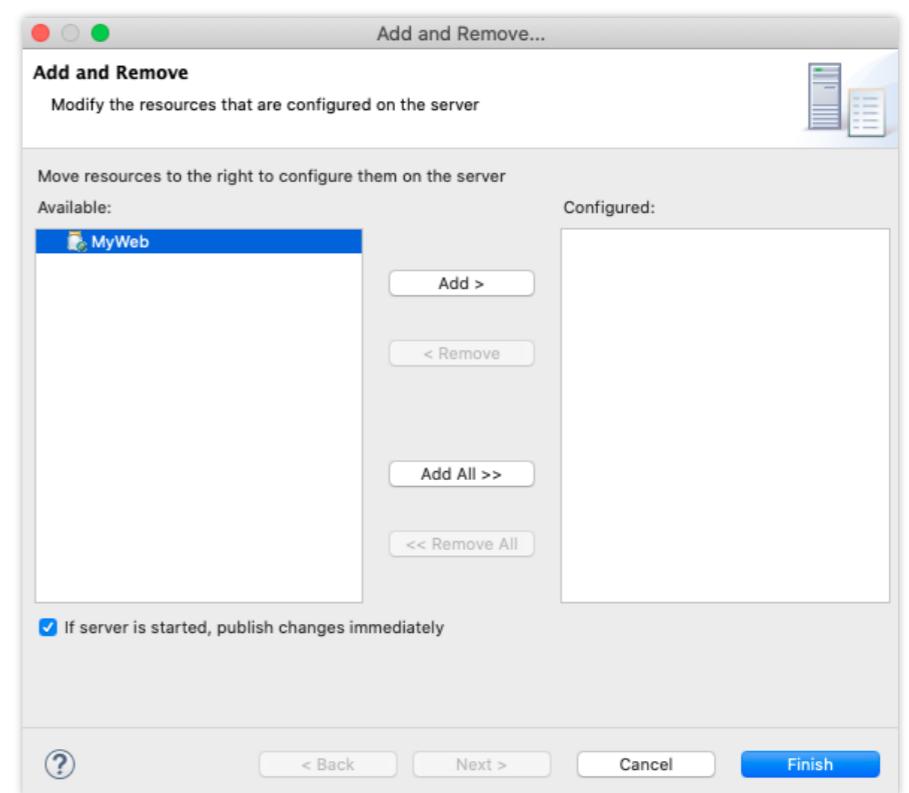
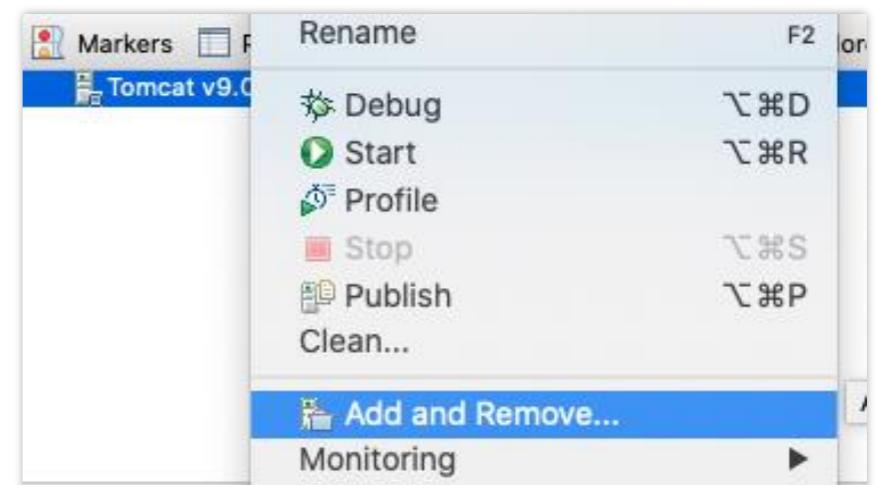
◀ 前へ

- Eclipse でサーバを作るのは初めてなので、Tomcat のパッケージを Eclipse に渡す必要があります。「Browse」をクリックし、先ほどダウンロードした Tomcat を探し、Open → Finish をクリック。



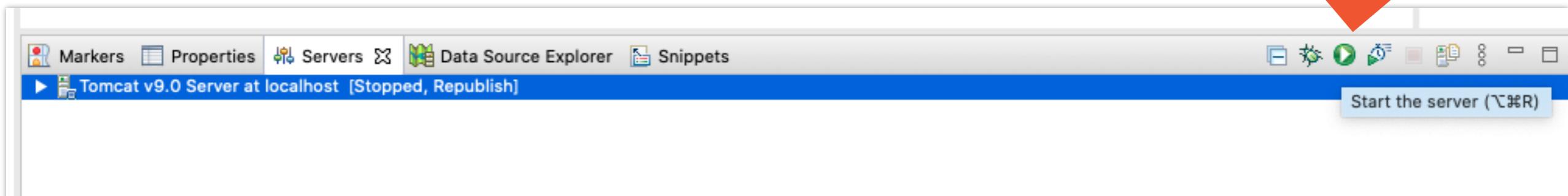
ステップ 6: サーバにプロジェクトの追加

- 作成された Server を右クリックし、Add and Remove を選択します。
- プロジェクト名を選択し、Add → Finishをクリックします。

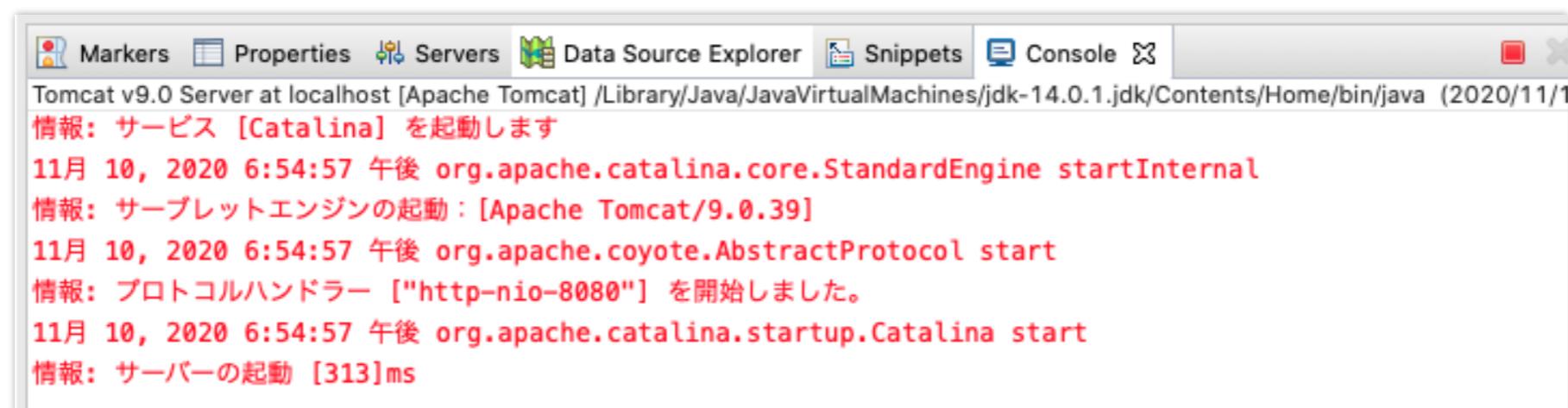


ステップ 7: サーバの起動

- 下の Servers を選択し、緑色のボタンをクリックします。

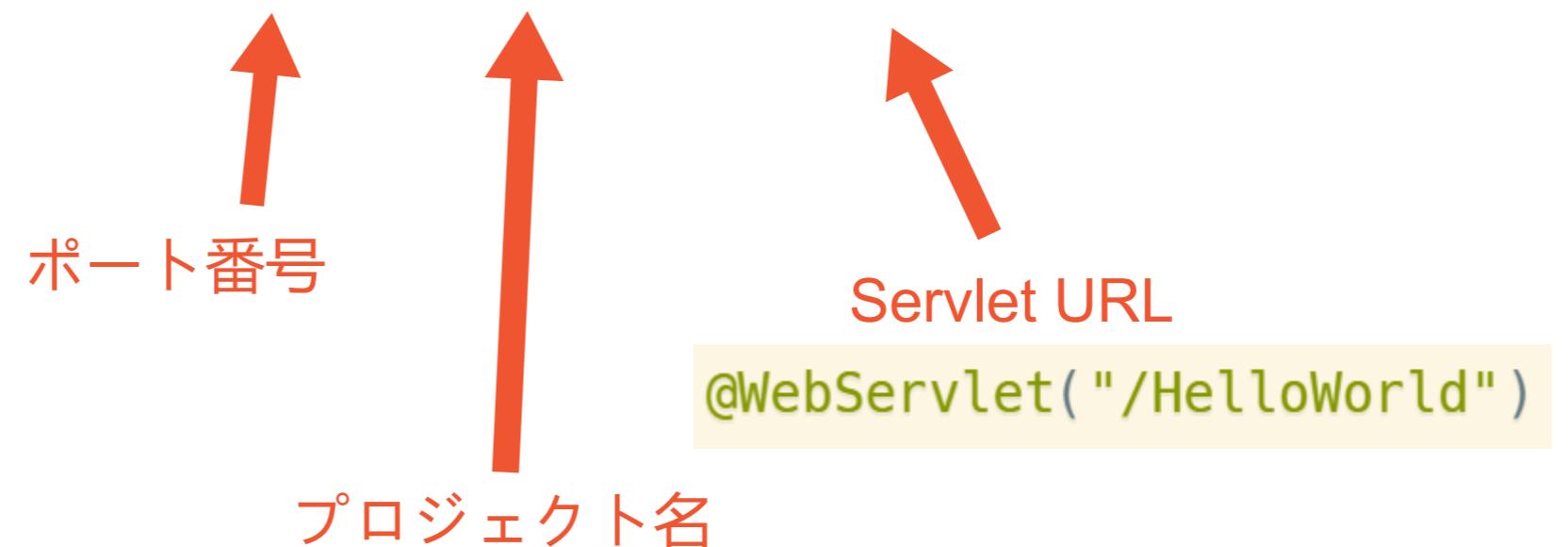


- その後、コンソールにサーバの状態が表示されます。



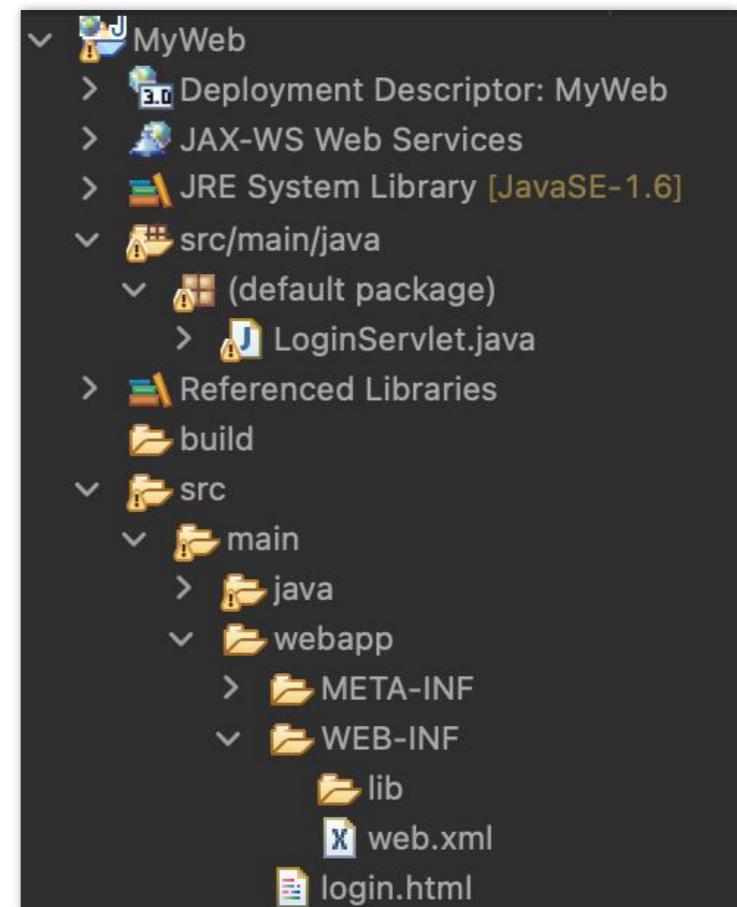
ブラウザでご覧ください

- `http://localhost:8080/MyWeb/HelloWorld`

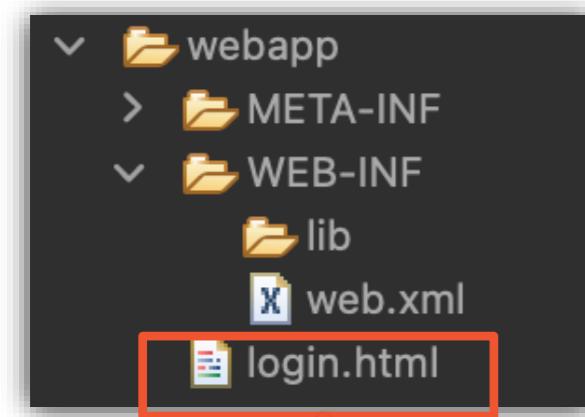


Servlet を用いた動的ログインページ

- MyWeb プロジェクトで書きます。
- webapp または webContent の下に login.html を作成し、配布資料にある login.html のコードをコピーして入れます。
- src の下に LoginServlet.java を作成し、そこに配布資料の LoginServlet.java のコードをコピーして入れます。



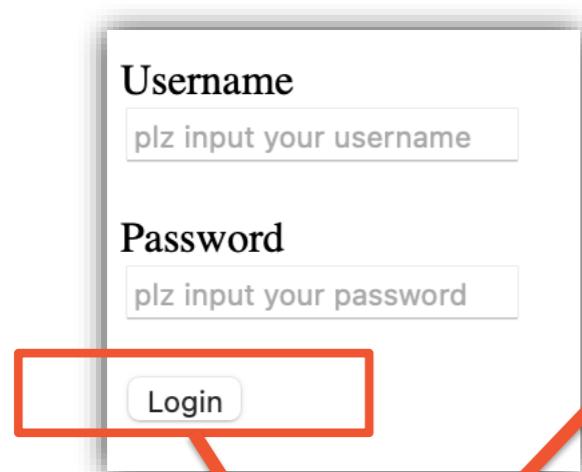
ページのロードと転移の流れ



```

<servlet>
  <servlet-name>login</servlet-name>
  <servlet-class>LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>login</servlet-name>
  <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
  
```

<http://localhost:8080/MyWeb/login.html>



Username
plz input your username

Password
plz input your password

Login

```

<body>
  <form action="/MyWeb/LoginServlet" method="post">
    
```

```

public class LoginServlet extends HttpServlet {
  @Override
  public void doPost(HttpServletRequest request, HttpServletResponse response) {
    
```

<http://localhost:8080/MyWeb/LoginServlet>

ユーザ名またはパスワードが正しくありません。



Question and answer

まとめ

Sum Up

1. ウェブ開発の基本:

- ① 動的ウェブページと静的ウェブページ。
- ② ウェブサーバ（ハードウェアとソフトウェア）。
- ③ HTTP リクエストとレスポンス。

2. Tomcat の基本概念と使用方法。

3. Servlet の基本とプロジェクト作成の手順。

THANK YOU!