

6.1 Java 服务器开发基础

- 网络应用原理
- Tomcat
- Servlet

目录

1

网络应用原理

2

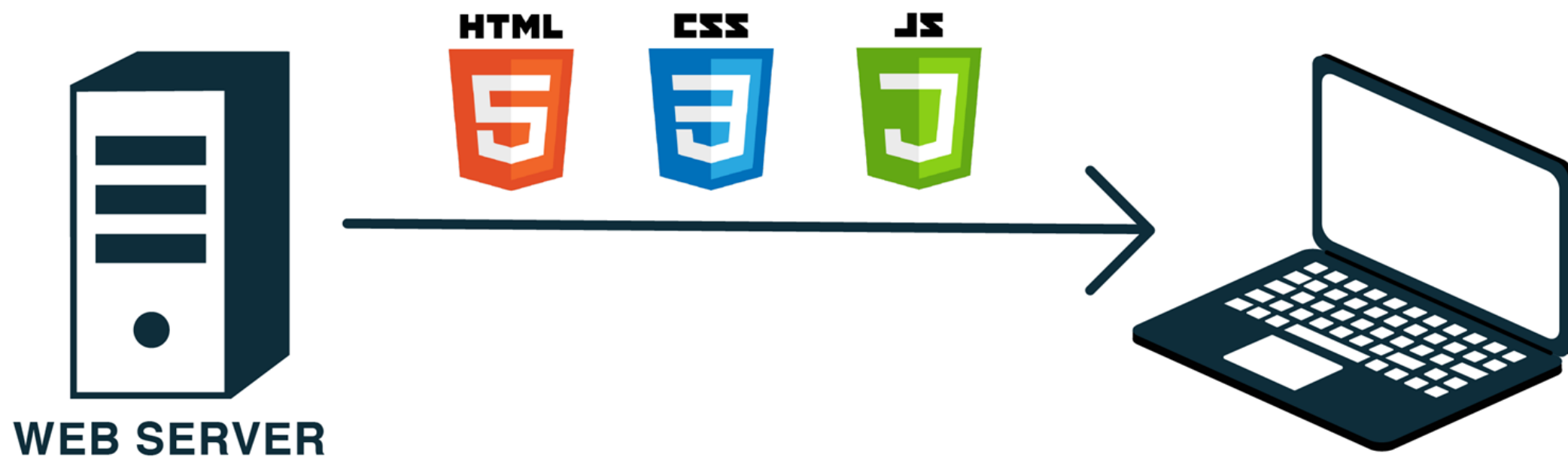
Tomcat

3

Servlet

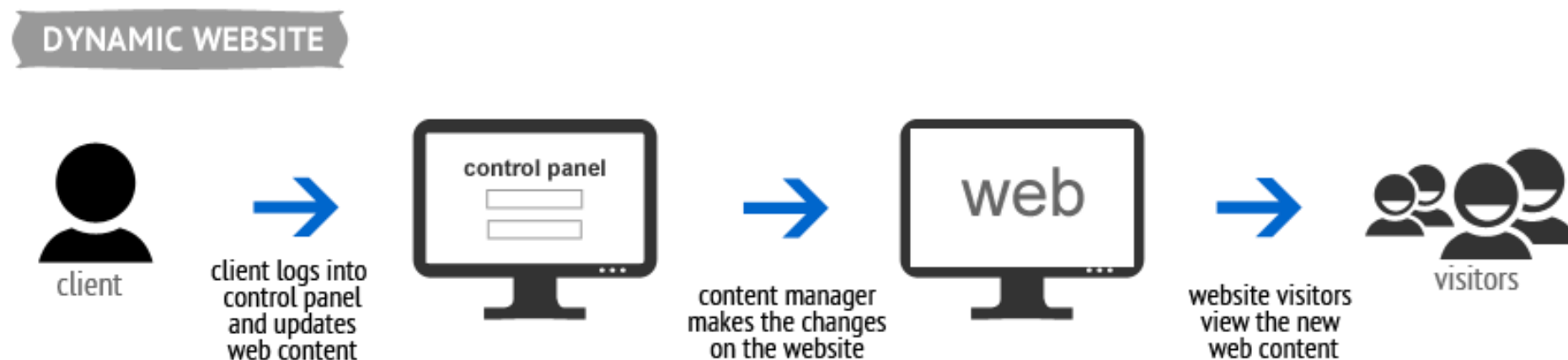
静态网页

- 网络应用程序大体上可以分为两种，即**静态网页**[静的ウェブページ]和**动态网页**[動的ウェブページ]。
- 早期的网站以提供**静态网页**为主。这些网页使用 HTML 语言来编写，被直接存放在服务器上。用户使用浏览器通过 **HTTP** 协议请求服务器上的页面，网站服务器将接收到的请求处理后，将存好的网页直接发送给客户端浏览器，显示给用户。



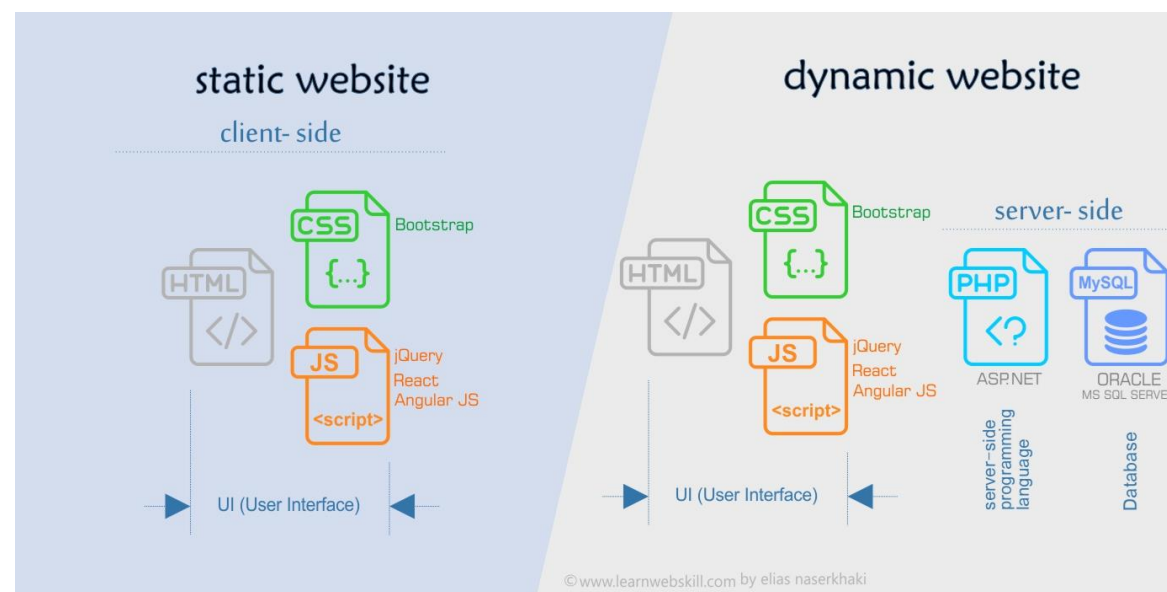
动态网页

- 随着网络的发展，很多线下业务开始向网上发展，基于互联网[インターネット]的网络应用也变得越来越复杂。用户所访问的资源已不再是只局限于服务器上保存的静态网页，更多的内容需要根据用户的请求动态生成，即**动态网页**。
- 这些网站通常结合 HTML 和动态脚本语言（如 JSP、ASP、PHP 等）编写。编写后的程序会被部署到**网络服务器**上。当收到用户请求后，服务器将运行动态脚本代码，产生浏览器可以解析的 HTML 代码，返回给客户端浏览器，显示给用户。



动态网页与静态网页

- 初学者经常会错误地认为带有动画效果的网页就是动态网页，其实不然。动态网页是指具有交互性、内容可以自动更新的网页，并且内容会根据访问的时间和访问者而改变。这里所说的交互性，是指网页可以根据用户的要求动态地改变或响应。
- 由此可见，静态网页类似于十几年前研制的手机，只能使用**出厂时设置**的功能和铃声；而动态网站则类似于现代智能手机，用户不再是只能使用机器中默认的铃声，而是可以根据自己的喜好**任意设置**。



网络服务器（硬件）

- **网络服务器**[ウェブサーバー]可以代指硬件或软件，或者是它们协同工作的整体。
- **硬件**上的网络服务器是一台存储了网络服务软件以及网站的**组成文件**（如 HTML、CSS、JavaScript 代码；图片、视频、文本等数据文件）的计算机。它接入到互联网，并支持与其他连接到互联网的设备进行物理数据的交互。

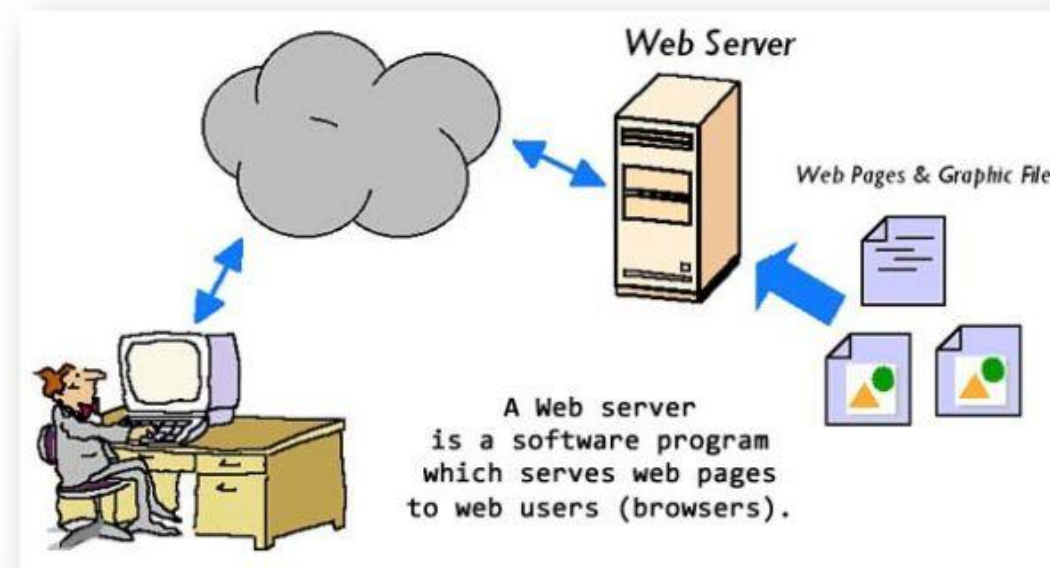


托管服务器

- 严格来说，你可以在你自己的计算机上托管所有的这些文件，但是在一个**专用的网络服务器**上存储它们会方便得多，因为它：
 - 会一直启动和运行；
 - 会一直与互联网连接；
 - 会一直拥有一样的 IP 地址（不是所有的运营商都会为家庭网络提供一个固定的 IP 地址）；
 - 由一个第三方提供者维护。
- 因为以上这些原因，寻找一个优秀的托管提供者是建立你的网站的关键一步。你需要比较不同公司提供的服务，并选择一个适合你的需求和预算的服务（服务的价格从免费到每月上百万日元不等）。
- 设置好一个网络托管解决方案后，你必须上传你的文件到该网络服务器。

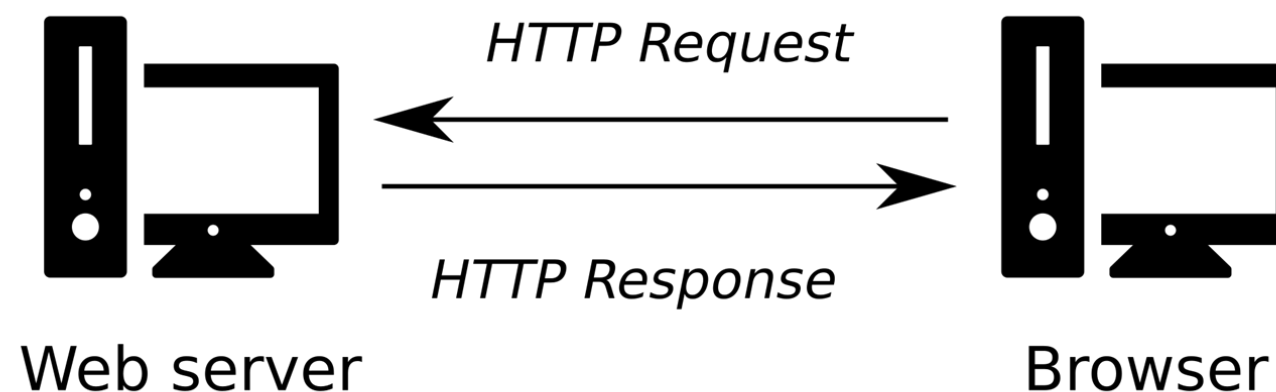
网络服务器（软件）

- 软件上的网络服务器可以控制网络用户如何访问托管文件的几个部分。通常，网络服务器至少要具备 **HTTP 服务器** 的功能。**HTTP 服务器** 是一种能够理解 URL 和 HTTP 协议的软件。它可以接受 **HTTP 请求**[HTTP リクエスト] 并给出 **HTTP 响应**[HTTP レスポンス]。
- 用户通过服务器网站对应的域名（比如“mozilla.org”）可以访问这个服务器，并获得服务器存储或动态生成的文件。



网络服务器的通信方式

- 浏览器需要一个托管在网络服务器上的文件时，浏览器通过发起 **HTTP 请求**，向服务器索取这个文件。
- 如果这个请求到达了正确的网络服务器（硬件），HTTP 服务器（软件）将会通过请求中的数据找到需要的文档，并把它通过 **HTTP 响应** 发送回用户的浏览器。
- 有时，服务器因某些原因无法完成请求，便可能返回一个 **状态码** [ステータスコード]，说明请求失败的原因。比如服务器上不存在对应的文档，便会返回状态码 **404**。



动态网络服务器

- 静态网络服务器，又叫**堆栈**[スタック]，由计算机（硬件）和 HTTP 服务器（软件）组成。我们称它为“静态”是因为这个服务器把它托管文件的“保持原样”地传送到你的浏览器。
- 动态网络服务器由一个静态的网络服务器加上额外的软件组成，最普遍的是一个应用服务器和一个**数据库**[データベース]。我们称它为“动态”是因为这个应用服务器在通过 HTTP 服务器把文件传送到浏览器前会对这些托管文件进行更新。

接次页 

- 举个例子，要生成你在浏览器中看到的最终网页，应用服务器或许会用一个数据库中的内容填充一个 HTML 模板。
- 有些网站，比如维基百科，有成千上万的网页，但它们不是真正的 HTML 文档，而是由少数的 HTML 模板以及一个巨大的数据库生成出来的。这样的设置让它更快、更简单地维护以及分发内容。



HTTP

- **超文本传输协议**（Hypertext Transfer Protocol, **HTTP**）是用于在网络上传输超媒体文件（如 HTML）的底层协议。使用 HTTP 最典型的场景是在浏览器和服务端之间传递数据以供人们浏览。现行的 HTTP 标准的版本是 HTTP/2。
- URL 中，“http://”的部分称为“**schema**”，一般位于网络地址的开头。以 <https://google.com> 为例：该地址说明请求文档时使用 HTTP 协议；这里的 https 代指 HTTP 协议的安全版本，即使采用 TLS 协议加密。
- **协议**[プロトコル]是为了在两台计算机间交流而制定的一套规则。HTTP 是**基于文本的**（所有通信都以纯文本的形式进行）以及**无状态的**（当前通信不会考虑以前的通信状态）。这些特性使我们在互联网上浏览网页变得十分方便。

什么是 HTTP

- HTTP 为客户端和服务端间如何沟通提供清晰的规则。就目前而言，只需要知道以下几点：
 - 只有客户端可以制定 HTTP 请求，然后只会发送到服务器。服务器将响应客户端的 HTTP 请求。
 - 当通过 HTTP 请求一个文件时，客户端必须提供这个文件的 **URL**。
 - 网络服务器必须响应**每一个** HTTP 请求，至少也要回复一个错误信息。

HTTP 请求

- 回忆一下之前学过的 HTML `<form>` 表单标签 (← § 1.4.1)，用户点击 submit 按钮便发出了一个 HTTP 请求。比如，下图的表单用户填完姓名提交之后，一个包含表单信息，并“使页面跳转到 /HelloWorld”的请求将会被发送。

```
<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:

- 用户点击 submit 后，填写的信息将会被传到后端的**控制器****[コントローラー]**（如 Java 程序）。控制器将对用户的信息进行某些处理（比如判断是否为合法用户），然后针对用户的请求作出相应的 HTTP 响应。这里的响应则是跳转到 /HelloWorld 页面。

网络应用技术

- 在开发网络应用程序时，通常需要应用客户端和服务端两方面的技术。其中，客户端应用的技术主要用于展现信息内容；而服务器端应用的技术则主要用于进行业务逻辑的处理和与数据库的交互等。
- 客户端应用的技术主要就是与我们之前学习的 HTML、CSS 及 JavaScript 相关的技术。
- 服务器端常用技术包括：
 - **通用网关接口**（Common Gateway Interface, CGI）：最早用来创建动态网页的一种技术，它可以使浏览器与服务器之间产生互动关系。
 - **PHP**：来自于 Personal Home Page 一词，但现在的 PHP 已经不再表示名词的缩写，而是一种开发动态网页技术的名称。
 - **JSP**（Java Server Page）：以 Java 为基础开发的动态网页技术，沿用了 Java 强大的 API。JSP 页面中不仅包含表示静态内容的 HTML 代码，还能嵌入 Java 代码与 JSP 标记以生成动态的内容。

Q & A

Question and answer

目录

1

网络应用原理

2

Tomcat

3

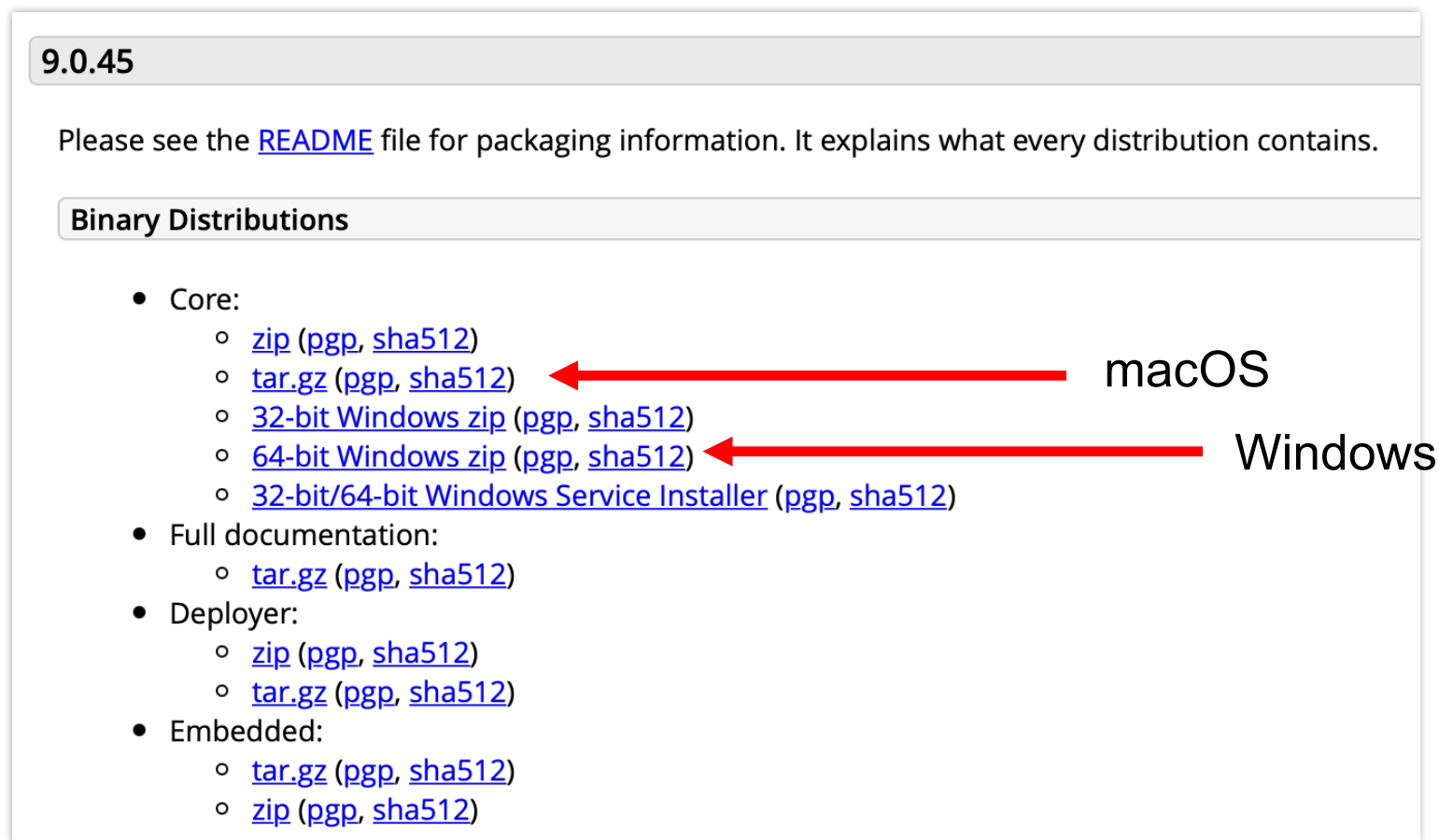
Servlet

Tomcat 简介

- **Tomcat** 是一个网络服务器（软件）。通过它我们可以很方便地接收和响应请求（如果不使用 Tomcat，我们需要自己写 *Socket* 来接收和响应请求）。
- Tomcat 也是一个可以用来装载之后要介绍的 *Servlet* 以及 *JSP* 页面的容器。
- Tomcat 其实我们并不需要学太多的知识，只要学会安装和启动以及了解一下各个目录的含义就差不多了。

Tomcat 的安装

- 我们统一使用 Tomcat 9.0，已下载的同学不必重复下载。
- 下载链接： <https://tomcat.apache.org/download-90.cgi>。



9.0.45

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

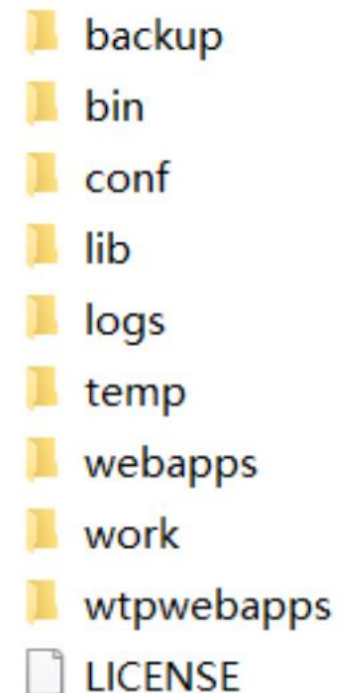
Binary Distributions

- 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](#))

- 下载之后解压，请务必记住解压包的地址，之后会用到。

Tomcat 项目目录结构

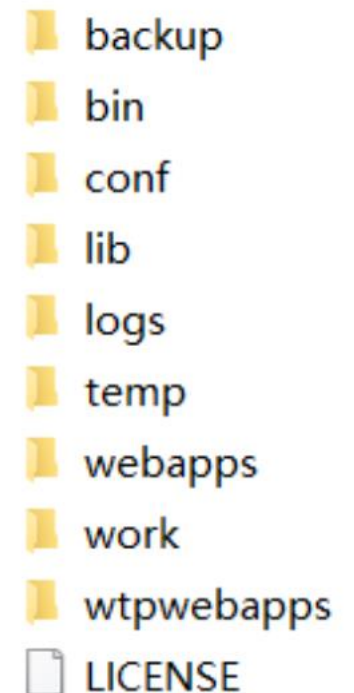
- bin: 该目录下存放的是二进制可执行文件。
- conf: 这个目录下有 4 个最为重要的文件:
 - server.xml: 配置整个**服务器信息**。例如修改端口号, 设置编码, 添加虚拟主机等。
 - tomcat-users.xml: 存储 Tomcat **用户**的文件, 包括用户名、密码、用户的角色信息等。可以按着该文件中的注释信息添加 Tomcat 用户, 然后就可以在 Tomcat 主页中进入 Tomcat Manager 页面。
 - web.xml: 部署描述符文件。这个文件中注册了很多 **MIME** 类型, 即文档类型。客户端与服务器之间用这些类型说明文档的种类。比如, 用户请求一个 HTML 网页, 服务器就会告诉客户端响应的文档是 text/html 类型。浏览器通过这个 MIME 类型就知道如何处理它了。
 - context.xml: 对所有**应用**的统一配置, 通常我们不会去配置它。



接次页 ➤

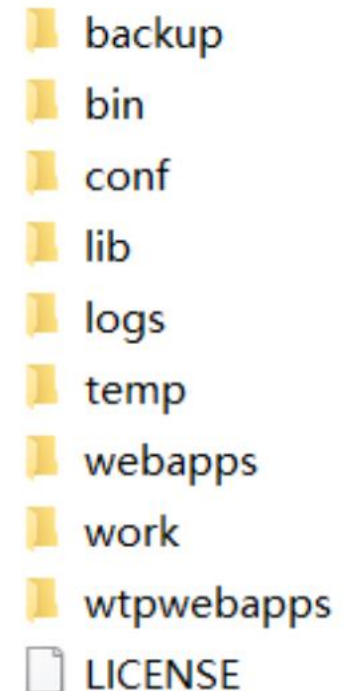
 接前页

- lib: Tomcat 的类库里有很多 Jar 包。如果需要添加 Tomcat 依赖的 Jar 包，可以把它放到这个目录中。这个目录中的 Jar 包可以被所有项目共享，但这样你的应用放到其他 Tomcat 下时就不能再共享这个目录下的 Jar 包了，所以建议只把 Tomcat 需要的 Jar 包放到这个目录下。
- logs: 这个目录中都是日志文件，记录了 Tomcat 启动和关闭的信息，如果启动 Tomcat 时有错误，那么异常也会被记录在日志文件中。
- temp: 存放 Tomcat 的临时文件，这个目录下的文件会在停止 Tomcat 后被删除。

接次页 



- **webapps**: 存放 Web 项目的目录。其中，每个文件夹都是一个项目。其中 **ROOT** 是一个特殊的项目，在地址栏中没有给出项目目录时，对应的就是 **ROOT** 项目。
<http://localhost:8080/examples> 进入示例项目。其中 **examples** 就是项目名，对应了文件夹的名字。
- **work**: 运行时生成的文件。最终运行的文件都在这里。它是通过 **webapps** 中的项目生成的。把这个目录删除后再次运行时会重新生成 **work** 目录。当客户端用户访问一个 **JSP** 文件时，Tomcat 会通过 **JSP** 生成 **.java** 文件，然后再编译 **.java** 文件生成 **.class** 文件，这些文件都会存放到这个目录下。



Q & A

Question and answer



目录

1

网络应用原理

2

Tomcat

3

Servlet

Java Web 应用程序

- 一个 Java Web 应用程序是由一组 Servlet、JSP 页面、Java 类以及其它的资源组成的运行在网络服务器（Tomcat）上的完整的应用程序，以一种结构化的有层次的目录形式存在。

什么是 Servlet

- Servlet 是 SUN 推出的一套规范，规定了如何用 Java 来开发动态网站。也就是说，Java 可以用来开发网站后台，但是要遵循一定的标准。
- Servlet 可以使用所有的 Java API，类库丰富，功能强大。
- 通过Servlet，你可以：
 - 接收用户通过 <form> 表单提交的信息；
 - 查询数据库，获取用户信息、文章内容、页面点击次数等；
 - 生成验证码，防止机器恶意注册。
 -

Servlet 的例子

- 例如，要在网页上显示一个 IP 地址，它的 HTML 源码是：

```
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>
```

接次页 ➞



- 那么服务器上的 Java 代码就应该这样写:

```
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         // 初期化处理
10    }
11
12    public void doGet(
13        HttpServletRequest request,
14        HttpServletResponse response
15    ) throws ServletException, IOException {
16        // リスponse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();
```

```
18
19        // println() メソッドで HTML を出力
20        PrintWriter out = response.getWriter();
21        out.println("<!DOCTYPE html>");
22        out.println("<html>");
23        out.println("<head>");
24        out.println("<meta charset=\"UTF-8\">");
25        out.println("<title>IP Address</title>");
26        out.println("</head>");
27        out.println("<body>");
28        out.println("<p>");
29        out.println(request.getRemoteAddr());
30        out.println("</p>");
31        out.println("</body>");
32        out.println("</html>");
33    }
34
35    public void destroy() {
36        // 終了時の処理
37    }
38 }
```

- 用户接收到的 HTML 代码都是通过 **println()** 语句输出的。

JSP

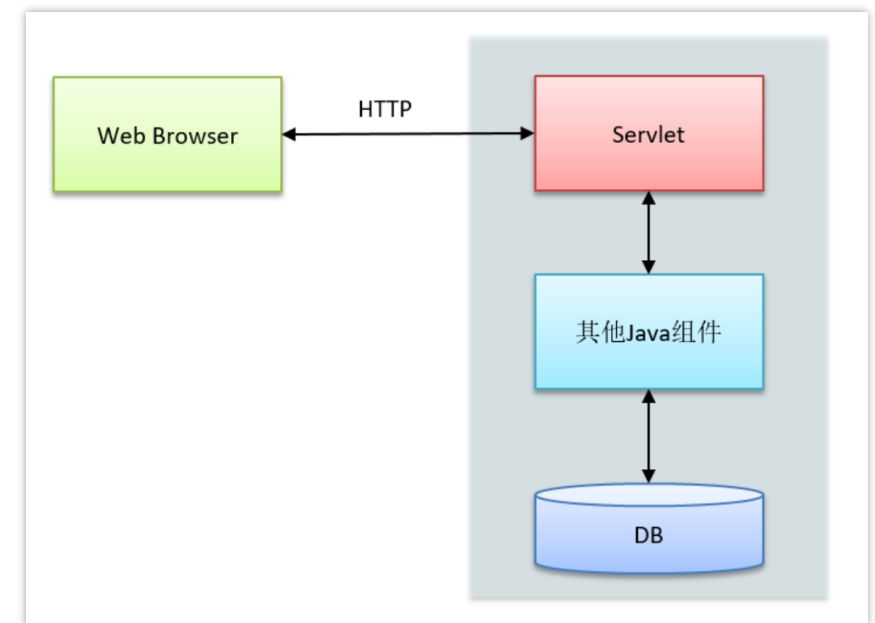
- 以上就是古老的 CGI 程序，需要把 HTML 代码当做字符串，通过输出语句一条一条的输出。互联网初期，CGI 程序大行其道，为互联网的发展做出了不可磨灭的贡献。
- JSP (Java Server Pages, Java 服务器页面) 是为了简化 Servlet 的工作出现的替代品。Servlet 输出 HTML 非常困难，JSP 就是替代 Servlet 输出 HTML 的。
- JSP 是一种基于文本的程序，其特点就是 HTML 和 Java 代码共同存在。

为什么还要学习 Servlet?

- Servlet 放在现在算是一个古老的技术了，现在的项目一般来说还是以 **Spring MVC / Spring Boot** 居多。
- 但是，它是 Java Web 编程技术的根本。现在 Java 的网络框架底层都离不开 Servlet。比如，Spring MVC 的核心用的就是 Servlet。
- 理解了 Servlet，可以让我们学习各类流行框架时更轻松。

Servlet 的功能

- Servlet 用于处理客户端请求的动态资源。当我们在浏览器中键入一个地址回车跳转后，一个 HTTP 请求就会被发送到地址对应的 Servlet 上进行处理。



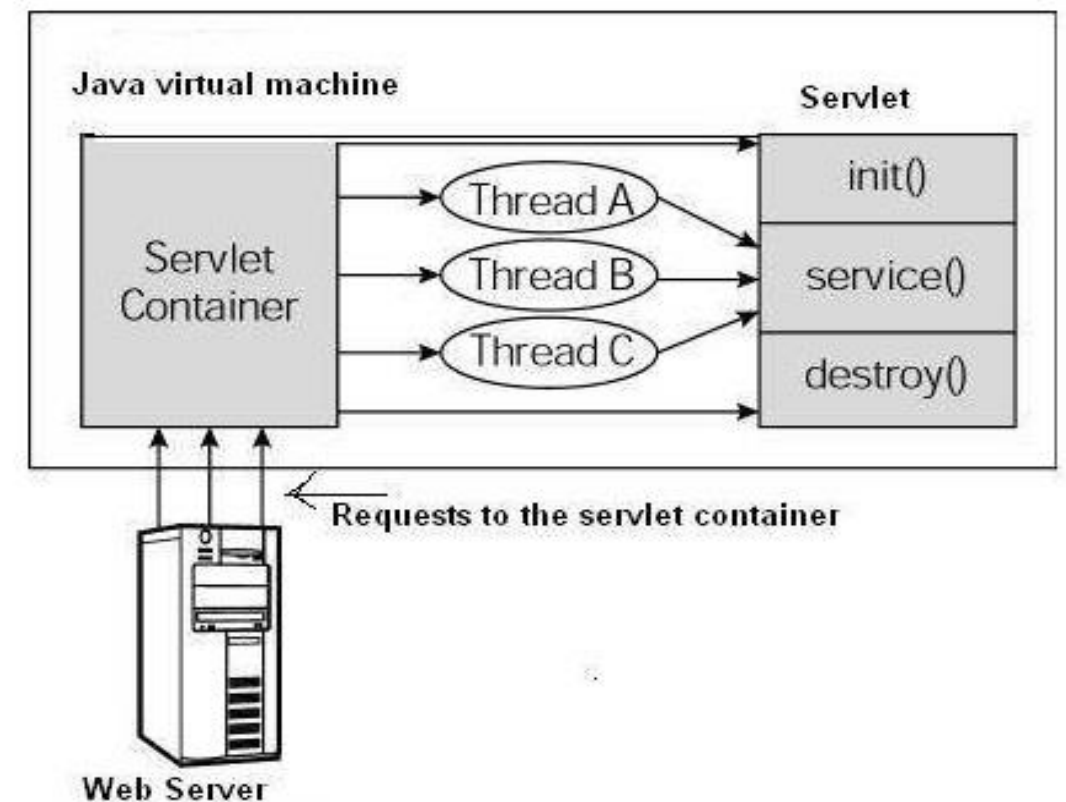
- Servlet 的功能：
 - 接收请求数据：客户端请求会被封装成 `HttpServletRequest` 对象，里面包含了请求头、参数等各种信息。
 - 处理请求：通常我们会使用 `service`、`doPost` 或者 `doGet` 方法接收参数，并且调用业务层（Service 层）的方法来处理请求。
 - 完成响应：处理完请求后，我们一般会转发[フォワード]（Forward）或者重定向[リダイレクト]（Redirect）到某个页面。

Servlet 包

- Servlet 是运行在带有支持 Java Servlet 规范的解释器的 Web 服务器上的 Java 类。
- Servlet 可以使用 `javax.servlet` 和 `javax.servlet.http` 包创建，它是 Java 企业版的标准组成部分，Java 企业版（Java EE）是支持大型开发项目的 Java 类库的扩展版本。
- Java Servlet 就像任何其他的 Java 类一样已经被创建和编译。在你安装 Servlet 包并把它们添加到 Classpath 类路径中之后，就可以通过 JDK 的 Java 编译器或任何其他编译器来编译运行 Servlet。

Servlet 的生命周期

- Servlet 通过调用 **init()** 方法进行初始化。
- Servlet 调用 **service()** 方法来处理客户端的请求。
- Servlet 通过调用 **destroy()** 方法终止（结束）。
- 最后，Servlet 会被 JVM 的垃圾回收器回收。



生命周期详述

● **init():**

- 在 Servlet 的生命周期中，仅执行 1 次 `init()` 方法。它是在服务器装入 Servlet 时执行的，负责初始化 Servlet 对象。可以配置服务器，以在启动服务器或客户机首次访问 Servlet 时装入 Servlet。无论有多少客户机访问 Servlet，都不会重复执行 `init()`。

● **service():**

- 它是 Servlet 的核心，负责响应客户的请求。每当一个客户请求一个 `HttpServlet` 对象，该对象的 `service()` 方法就被调用，并传给这个方法一个“请求”（`ServletRequest`）对象和一个“响应”（`ServletResponse`）对象作为参数。在 `HttpServlet` 中已存在 `service()` 方法。默认的服务功能是调用与 HTTP 请求的方法相应的 `do` 功能。

接次页 



- **destroy():**

- 仅执行 1 次，在服务器端停止且卸载 Servlet 时执行该方法。当 Servlet 对象退出生命周期时，destroy() 负责释放占用的资源。一个 Servlet 在运行 service() 方法时可能会产生其他的线程，因此需要确认在调用 destroy() 方法时，这些线程已经终止或完成。

Servlet 处理请求的步骤

1. 客户端向 Servlet 容器 (Tomcat) 发出 HTTP 请求。
2. Servlet 容器接收客户端的请求。
3. Servlet 容器创建一个 `HttpRequest` 对象，将请求信息封装到这个对象中。
4. Servlet 容器创建一个 `HttpResponse` 对象。
5. Servlet 容器调用 `HttpServlet` 对象的 `service` 方法，把 `HttpRequest` 对象与 `HttpResponse` 对象作为参数传给 `HttpServlet` 对象。
6. `HttpServlet` 调用 `HttpRequest` 对象的有关方法，获取 HTTP 请求信息。
7. `HttpServlet` 调用 `HttpResponse` 对象的有关方法，生成响应数据。
8. Servlet 容器把 `HttpServlet` 的响应结果传回客户端。

Q & A

Question and answer

Servlet 实例

- **HttpServlet** 是服务 HTTP 请求并实现 **Servlet** 接口的 Java 类。网站开发者通常通过继承 **HttpServlet** 来编写自己的 **Servlet**，处理 HTTP 请求。

重写 doGet() 方法

处理 HTTP 请求

```
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 継承 HttpServlet 类
8 public class HelloWorld extends HttpServlet {
9
10     private String message;
11
12     public void init() throws ServletException {
13         // 初期化处理
14         message = "Hello World"; 重写 init() 方法
15     }                                进行网页初始化
16
```

```
17     public void doGet(
18         HttpServletRequest request,
19         HttpServletResponse response
20     ) throws ServletException, IOException {
21         // リスponseのタイプを設定
22         response.setContentType("text/html");
23
24         // println() メソッドで HTML を出力
25         PrintWriter out = response.getWriter();
26         out.println("<h1>" + message + "</h1>");
27     }
28
29     public void destroy() {
30         // 終了時の処理
31     }
32 }
```

重写 destroy() 方法

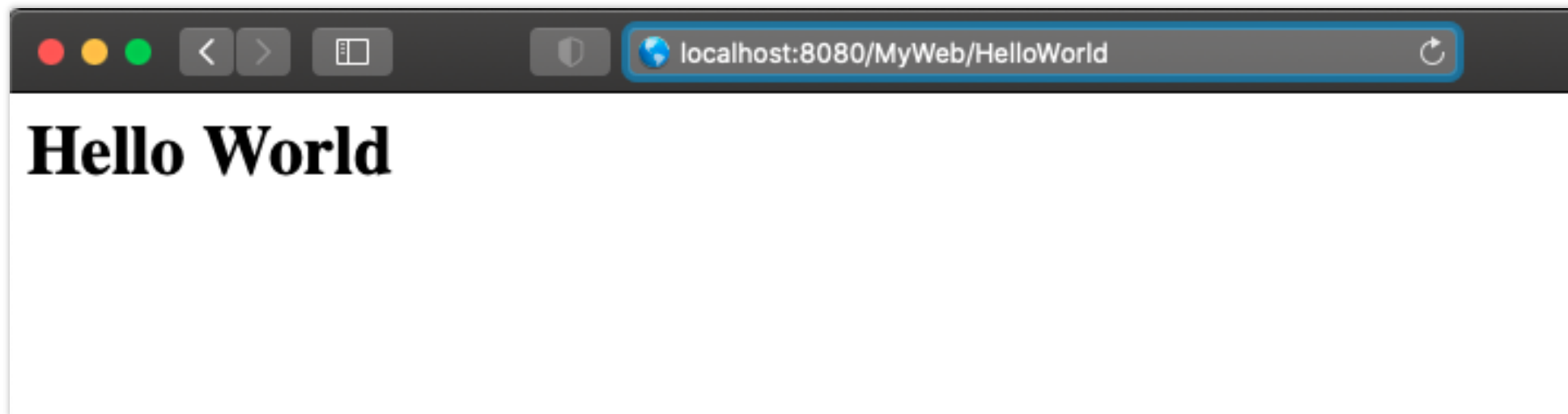
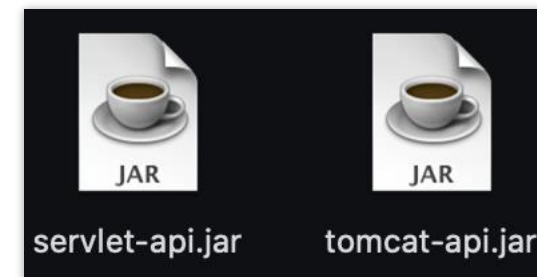
Servlet 开发基本步骤

1. 创建 Servlet 类
2. 标明 URL 路径：配置 web.xml 或在类文件里加注解“@WebServlet("/xxx")”。
3. 发布到 Tomcat 服务器。
4. 启动 Tomcat。
5. 打开浏览器，输入网址访问 Servlet：
 - http://服务器 IP:端口号/appName/Servlet 提供的 URL

创建 Servlet HelloWorld

- 需要：

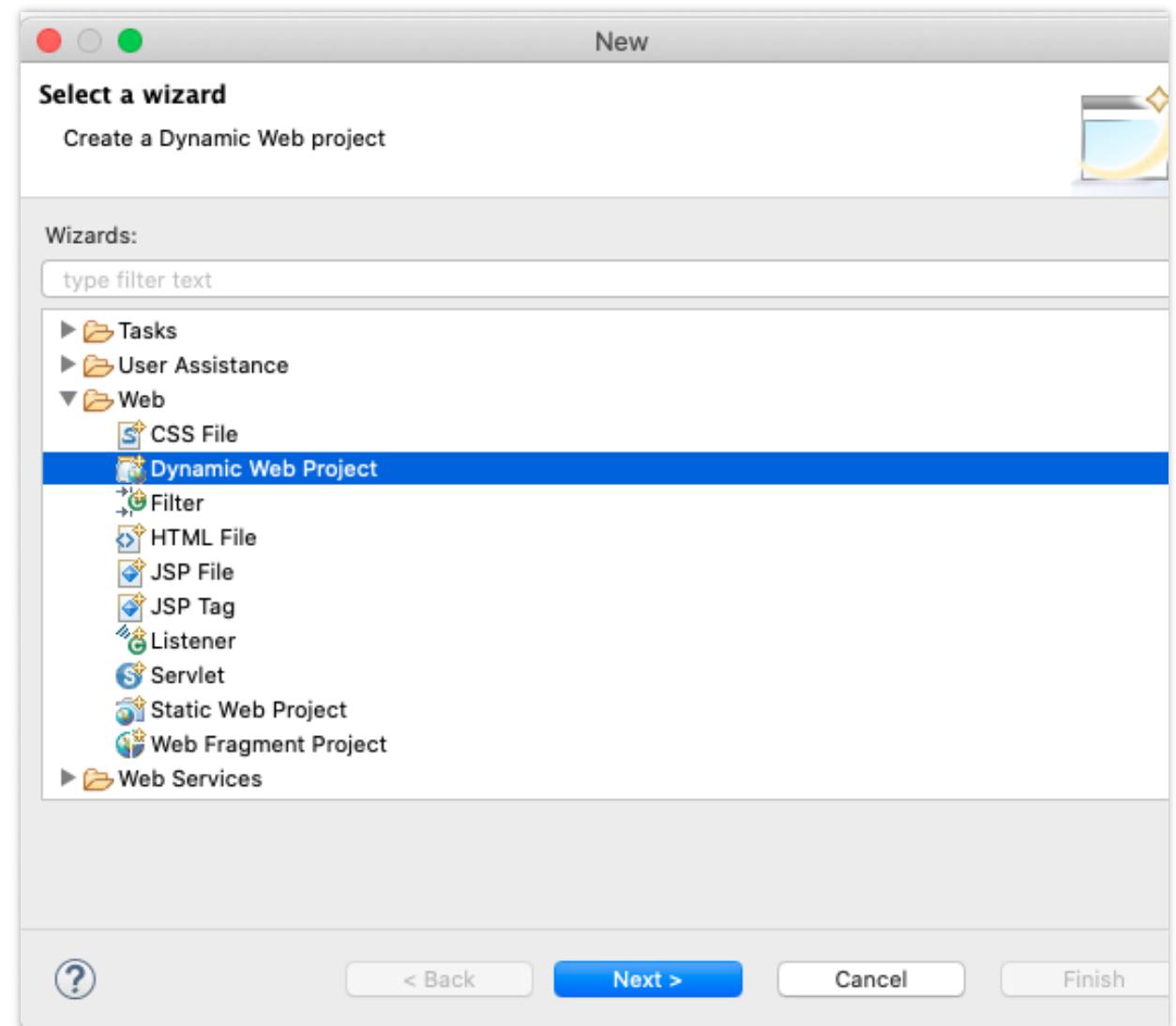
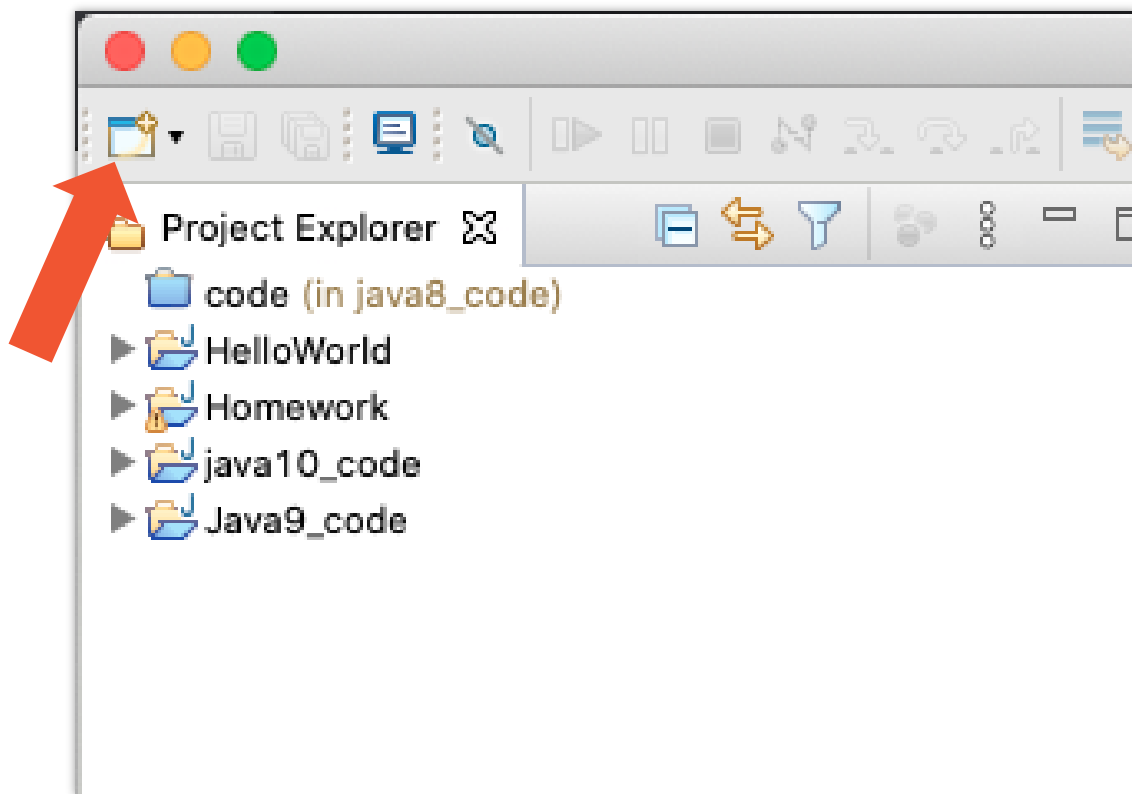
- Eclipse EE、
- 刚才下好的 Tomcat、
- 课件中的 `servlet-api.jar` 包和 `tomcat-api.jar` 包。



效果图

步骤 1: 创建项目

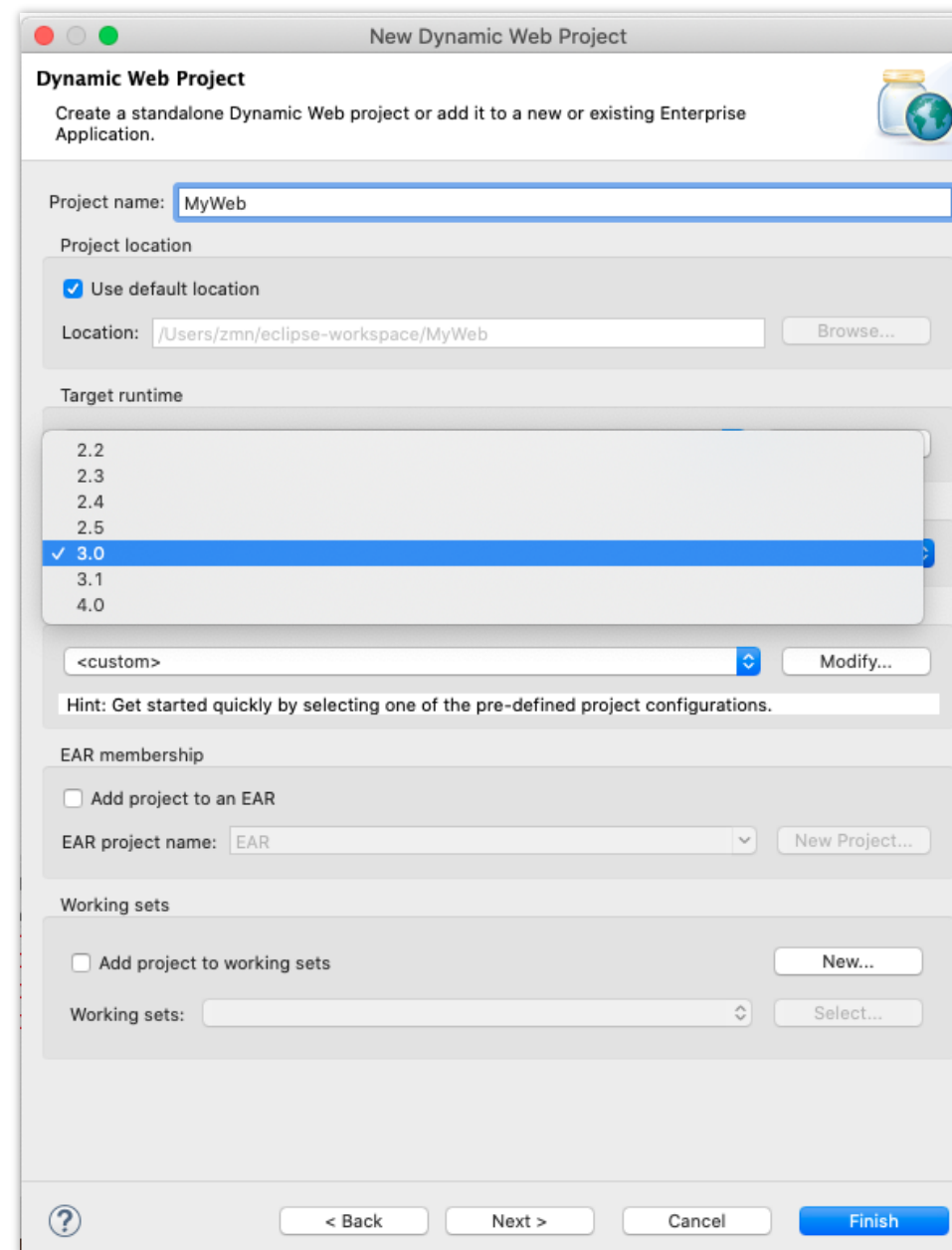
- 打开 Eclipse, 找到 Dynamic Web Project, 点击 Next。



接次页 ➞

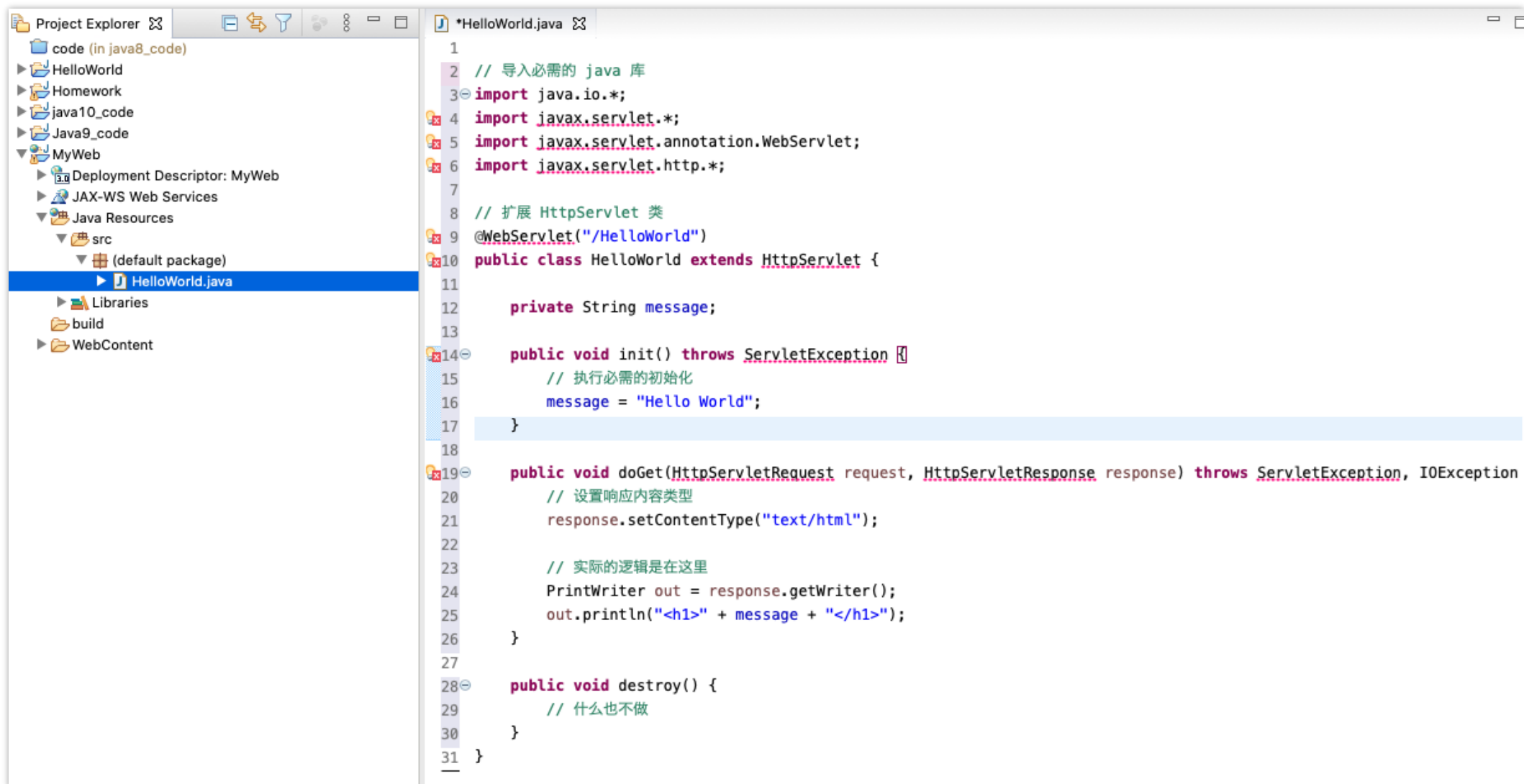


- 项目命名为 MyWeb。要手动设置 **Dynamic Web Module** 为 **version 3.0**，Finish。



步骤 2：定义 Servlet 类

- 找到 MyWeb/Java Resources/src。
- 创建 HelloWorld.java，将课件的 HelloWorld.java 代码复制过来。



The screenshot shows an IDE with the Project Explorer on the left and the HelloWorld.java file open in the editor. The Project Explorer shows the following structure:

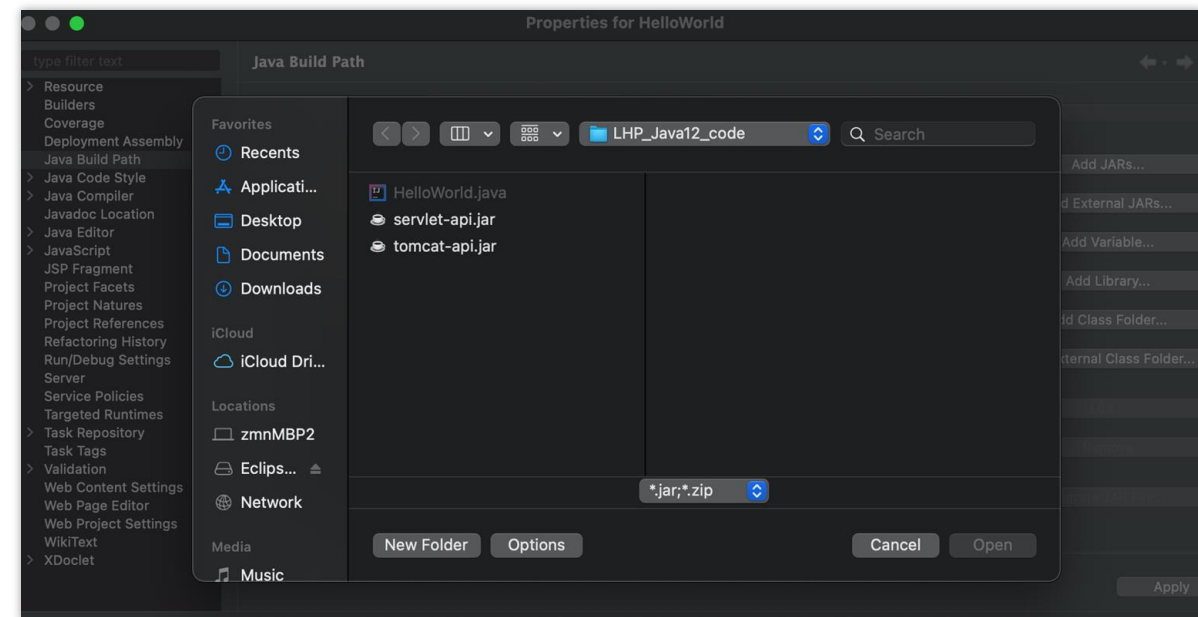
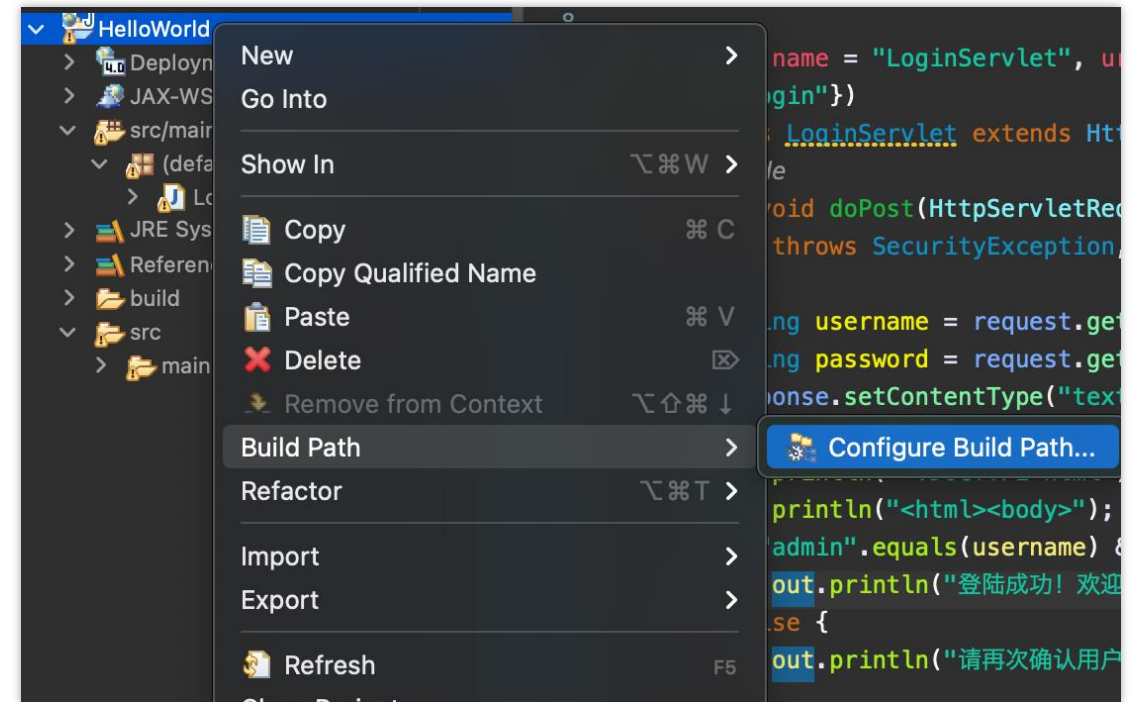
- code (in java8_code)
 - HelloWorld
 - Homework
 - java10_code
 - Java9_code
 - MyWeb
 - Deployment Descriptor: MyWeb
 - JAX-WS Web Services
 - Java Resources
 - src
 - (default package)
 - HelloWorld.java
 - Libraries
 - build
 - WebContent

The HelloWorld.java file contains the following code:

```
1 // 导入必需的 java 库
2
3 import java.io.*;
4 import javax.servlet.*;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.*;
7
8 // 扩展 HttpServlet 类
9 @WebServlet("/HelloWorld")
10 public class HelloWorld extends HttpServlet {
11
12     private String message;
13
14     public void init() throws ServletException {
15         // 执行必需的初始化
16         message = "Hello World";
17     }
18
19     public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
20         // 设置响应内容类型
21         response.setContentType("text/html");
22
23         // 实际的逻辑是在这里
24         PrintWriter out = response.getWriter();
25         out.println("<h1>" + message + "</h1>");
26     }
27
28     public void destroy() {
29         // 什么也不做
30     }
31 }
```

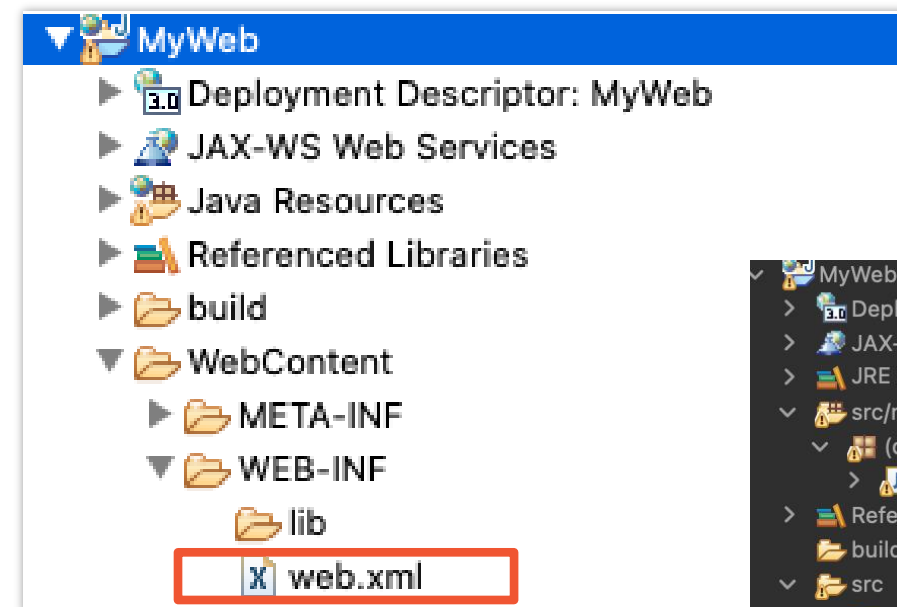
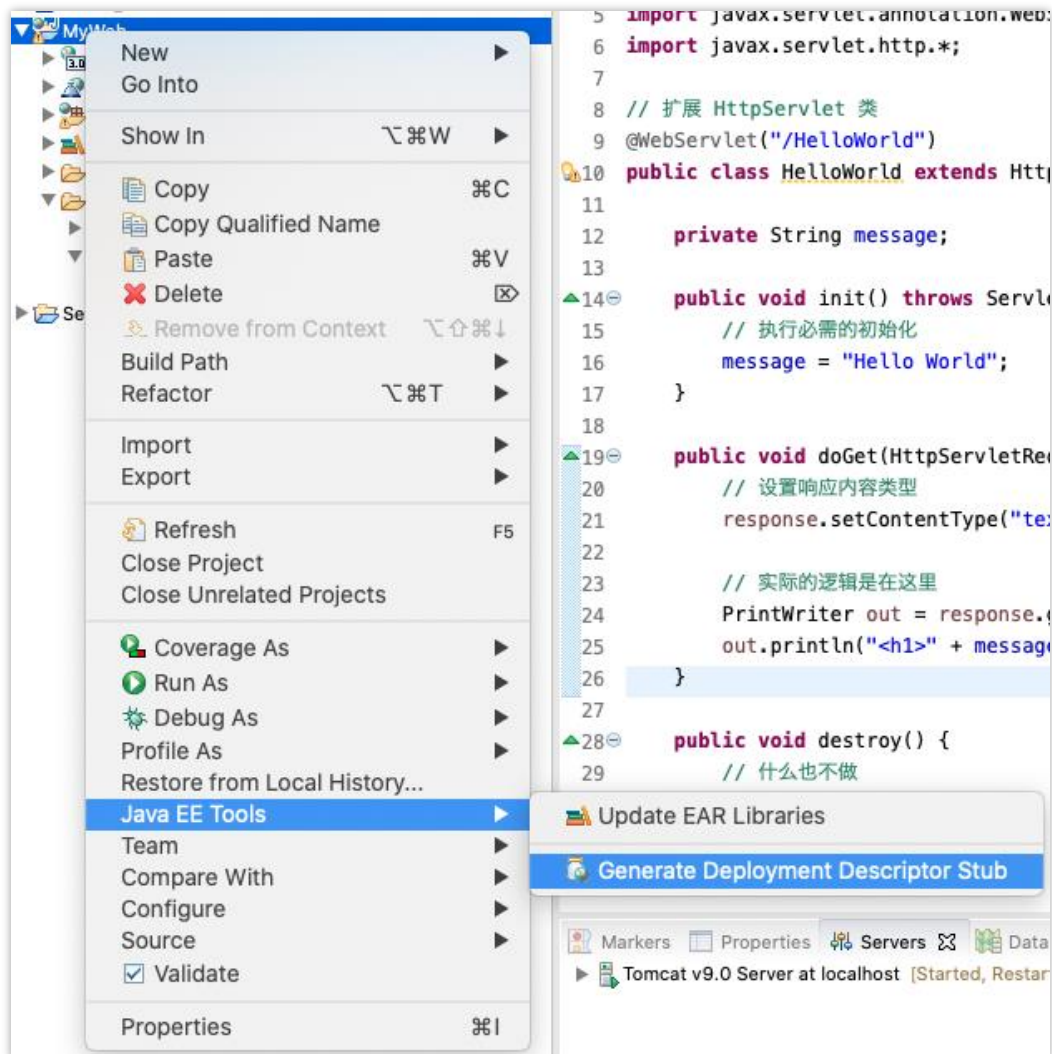
步骤 3：导入 Jar 包

- 代码报错是因为还没有导入 Jar 包。右键项目 → Build Path → Configure Build Path。
- 选择 Libraries，点击 Add External JARs。
- 将课件代码中的 Jar 包都添加进来。
- Open → Apply and Close。

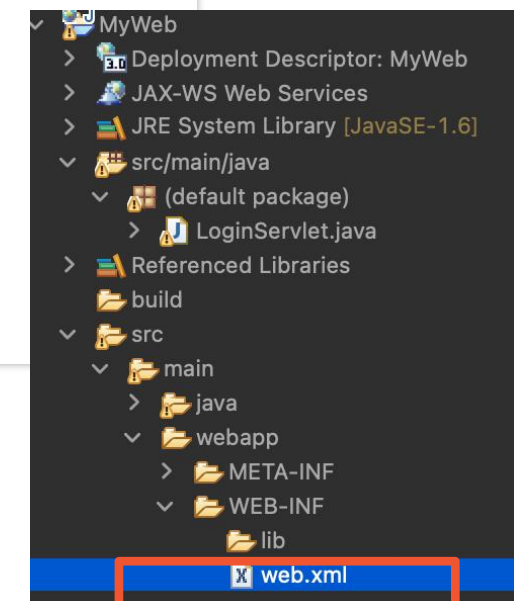


步骤 4：添加 web.xml 文件

- 该文件是项目配置文件，可用于声明哪个 Servlet 对应哪个网址。
- 右键项目 → Java EE Tools → Generate。



这里多出一个web.xml文件

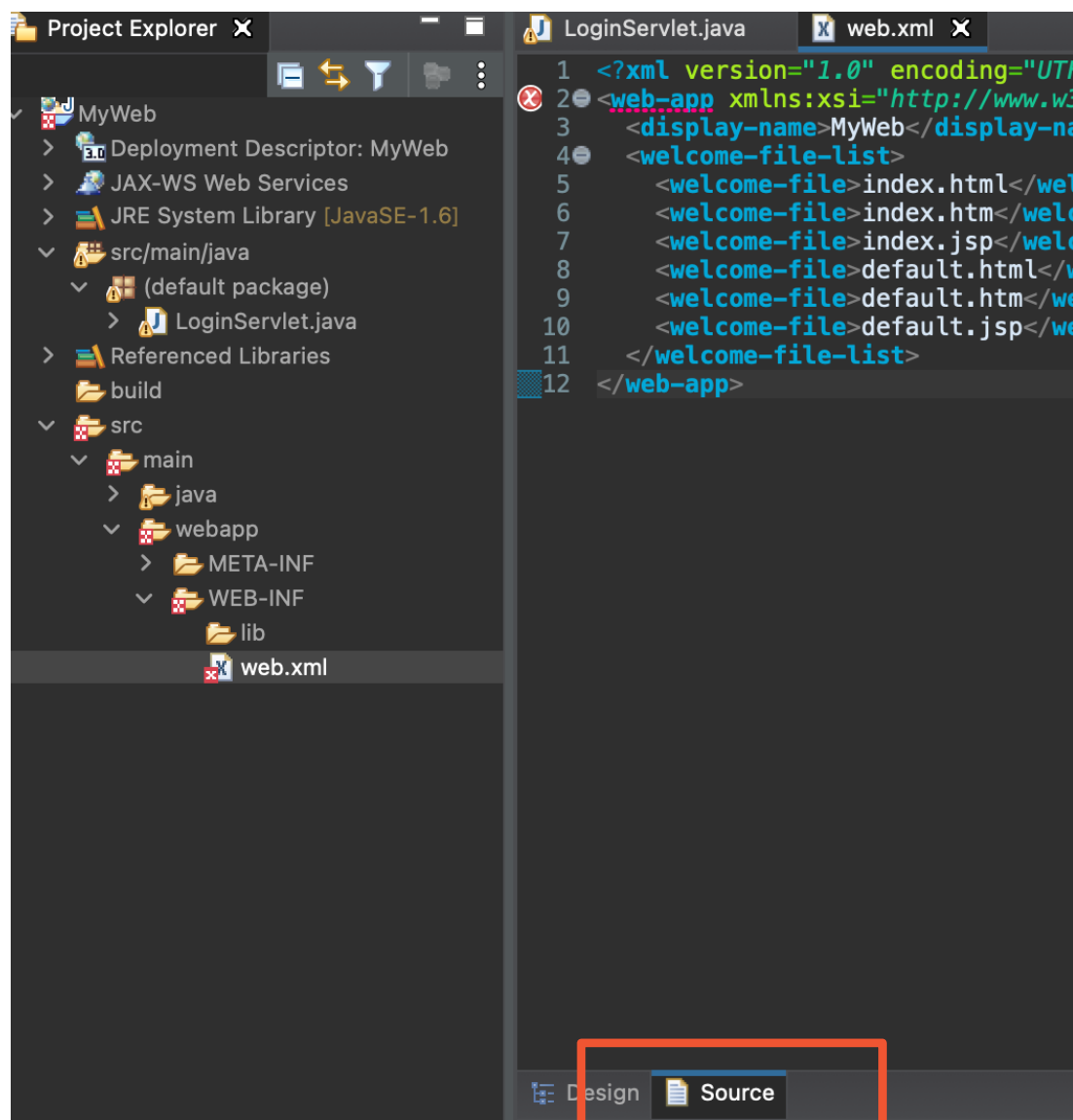


或者这里

接次页 ➡



- web.xml 文件可以呈现 Design 和 Source 两种模式，选择 Source 模式。



 接前页

- 在 web-app 开始结束标签中添加以下标签并保存:

```
<?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 类名

name 一致才能匹配上

接次页 

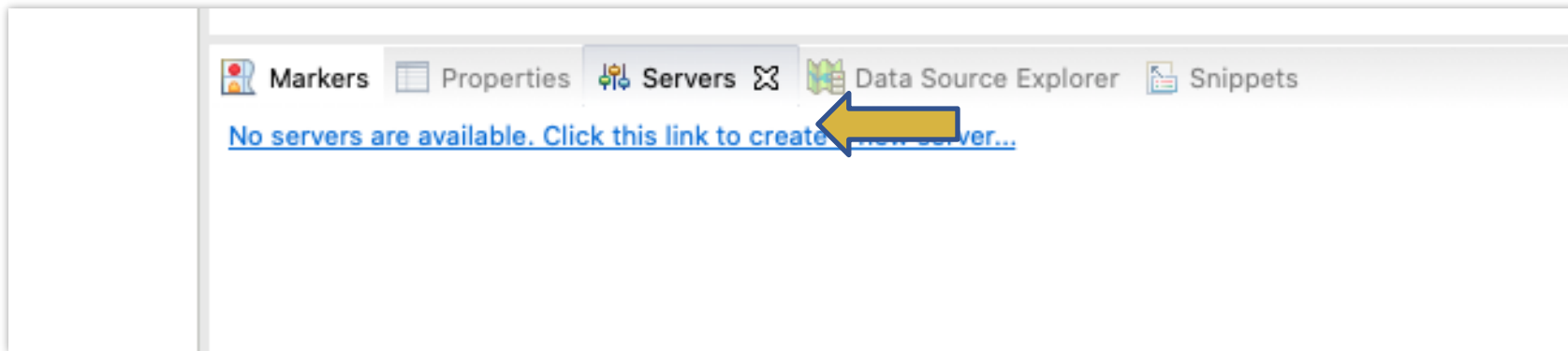
 接前页

```
<?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
```

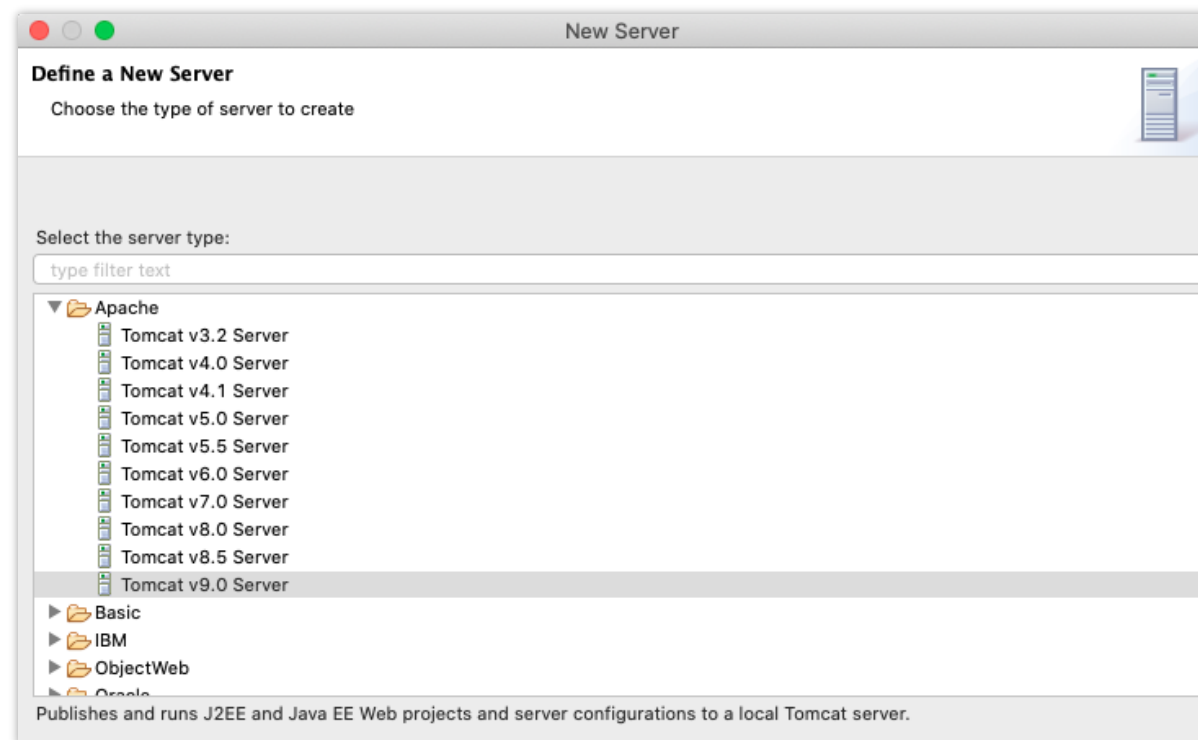
如果配置文件代码报错，可以尝试把
这里的“java”改成大写再保存

步骤 5：建立服务器

- Eclipse 下方区域选择 Servers，创建新服务器。



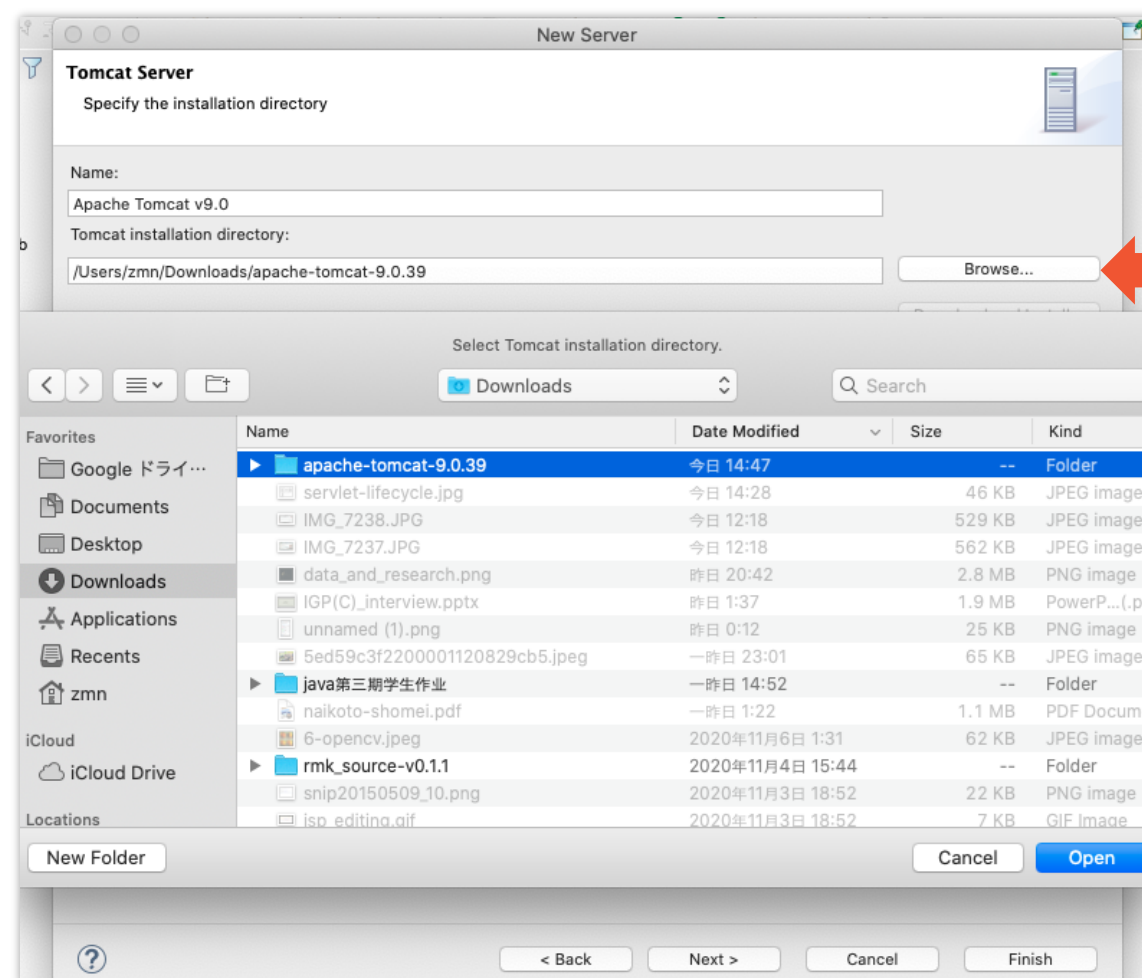
- 选择 Apache Tomcat v9.0, 点击 Next。



接次页 ➞

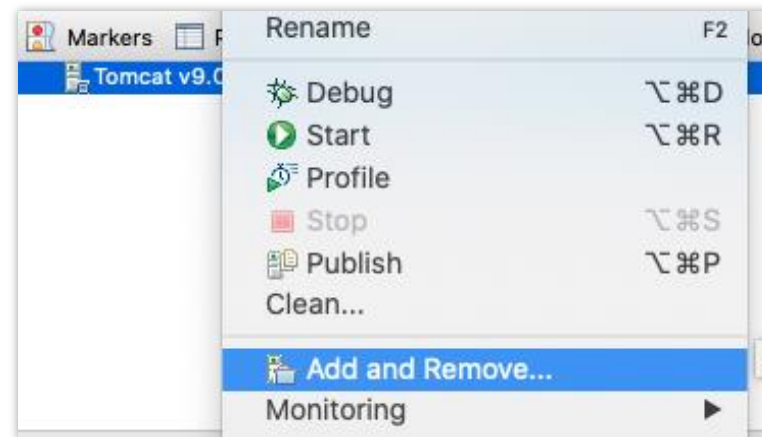


- 因为是第一次用 Eclipse 创建服务器，需要把 Tomcat 的包传给 Eclipse。
- 点击 Browse，找到刚下载好的Tomcat，点击 Open → Finish。

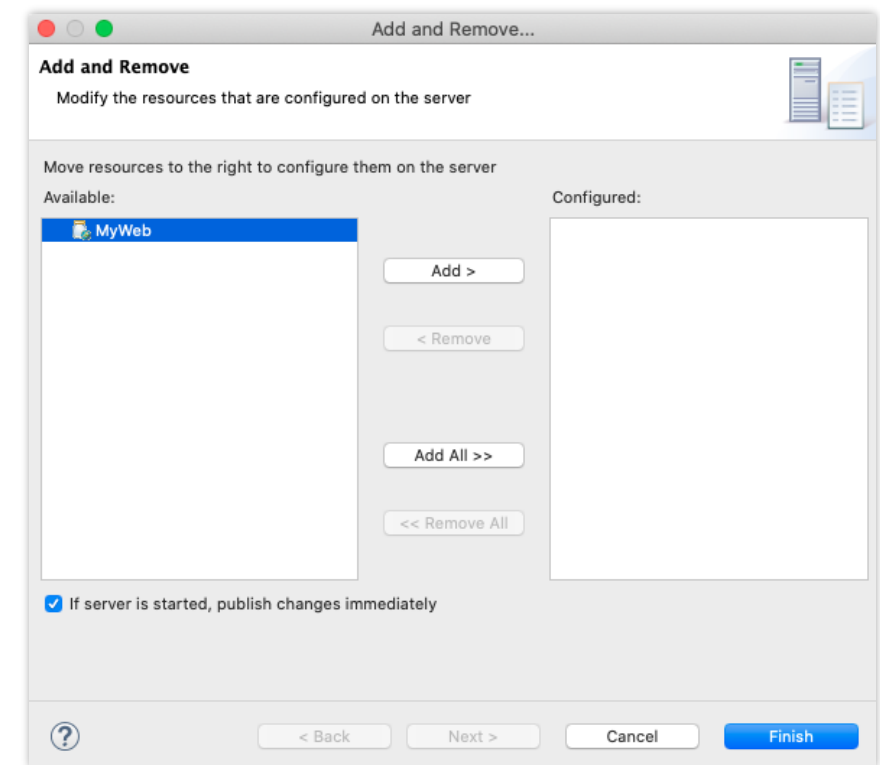


步骤 6：将项目添加到服务器

- 右键创建好的 Server，选择 Add and Remove。

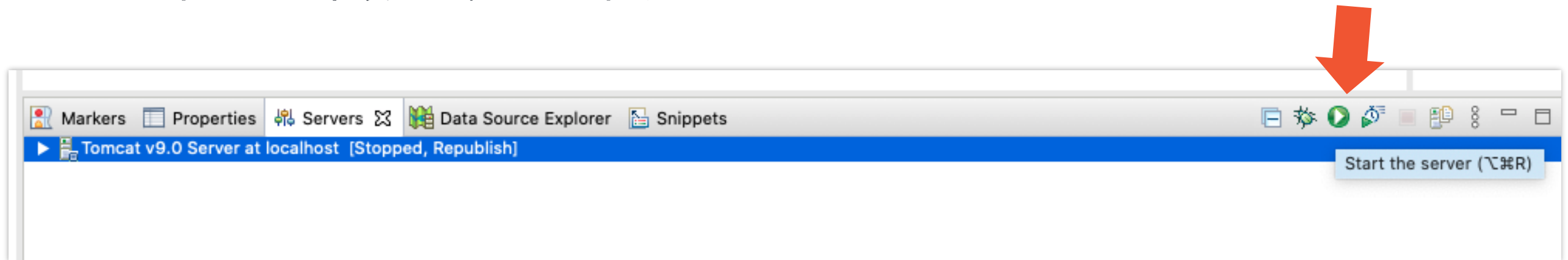


- 选择项目名，点击 Add → Finish。

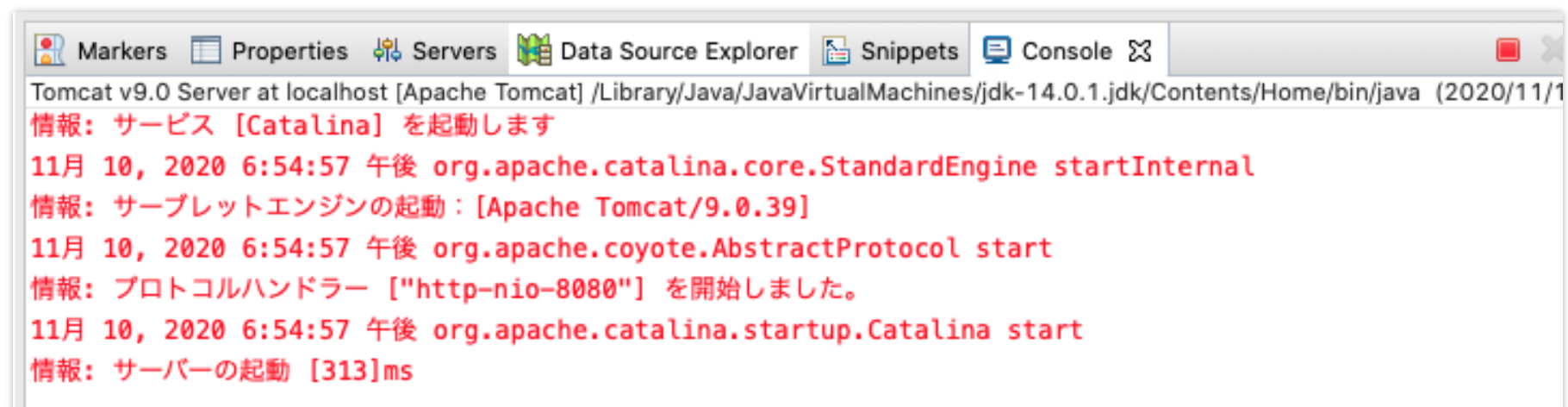


步骤 7：启动服务器

- 选择下方服务器， 点击绿色按钮。



- 随后控制台会显示服务器状态：



使用浏览器查看网页

- <http://localhost:8080/MyWeb/HelloWorld>

↑
端口号[ポート番号]

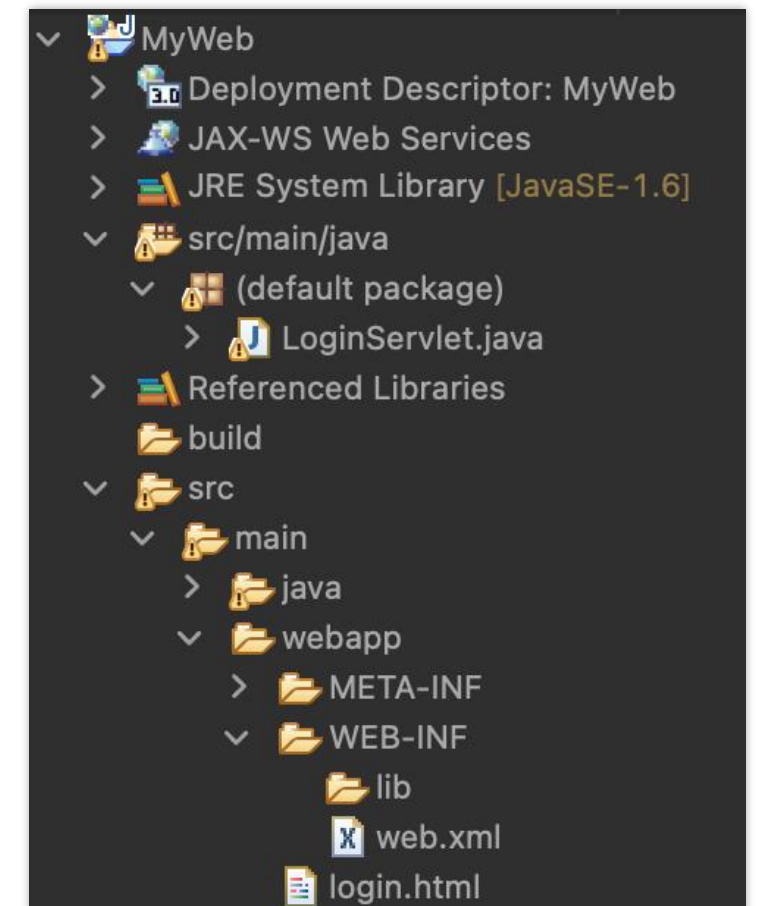
↑
项目名

↑
Servlet URL

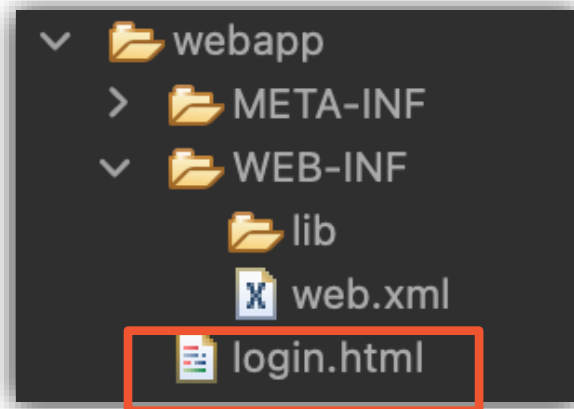
```
@WebServlet("/HelloWorld")
```

基于 Servlet 的动态登录网站

1. 仍然使用 MyWeb 项目。
2. 在 webapp 或 webContent 下创建一个 login.html，将课件中的 login.html 代码复制进去。
3. 索引到 src，创建 LoginServlet.java，将课件的 LoginServlet.java 的代码复制进去。



页面加载及跳转流程



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

Username

Password

```
<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>
```

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

http://localhost:8080/MyWeb/LoginServlet

请再次确认用户名和密码的正确性。-_-

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

Q & A

Question and answer

总结

Sum Up

1. 网络应用原理：
 - ① 动态与静态网页；
 - ② 网络服务器（硬件与软件）；
 - ③ HTTP 请求。
2. Tomcat 基本概念与使用方法。
3. Servlet 基本原理与搭建项目的方法。

THANK YOU!