

# Web应用开发 之 Servlet技术模型

# 本节内容

- Servlet简介
- Servlet API
- Servlet生命周期

## 2.1Servlet简介

### 1、什么是Servlet

- Servlet一般翻译成**服务器端小程序**，它是使用Servlet API以及相关的类编写的Java程序。
- Servlet的主要用途是**实现对Web服务器功能的扩充**。
- 它是一种动态加载的模块，采用请求——响应模式提供Web服务，以动态地生成的Web页面作为响应服务请求的返回结果。

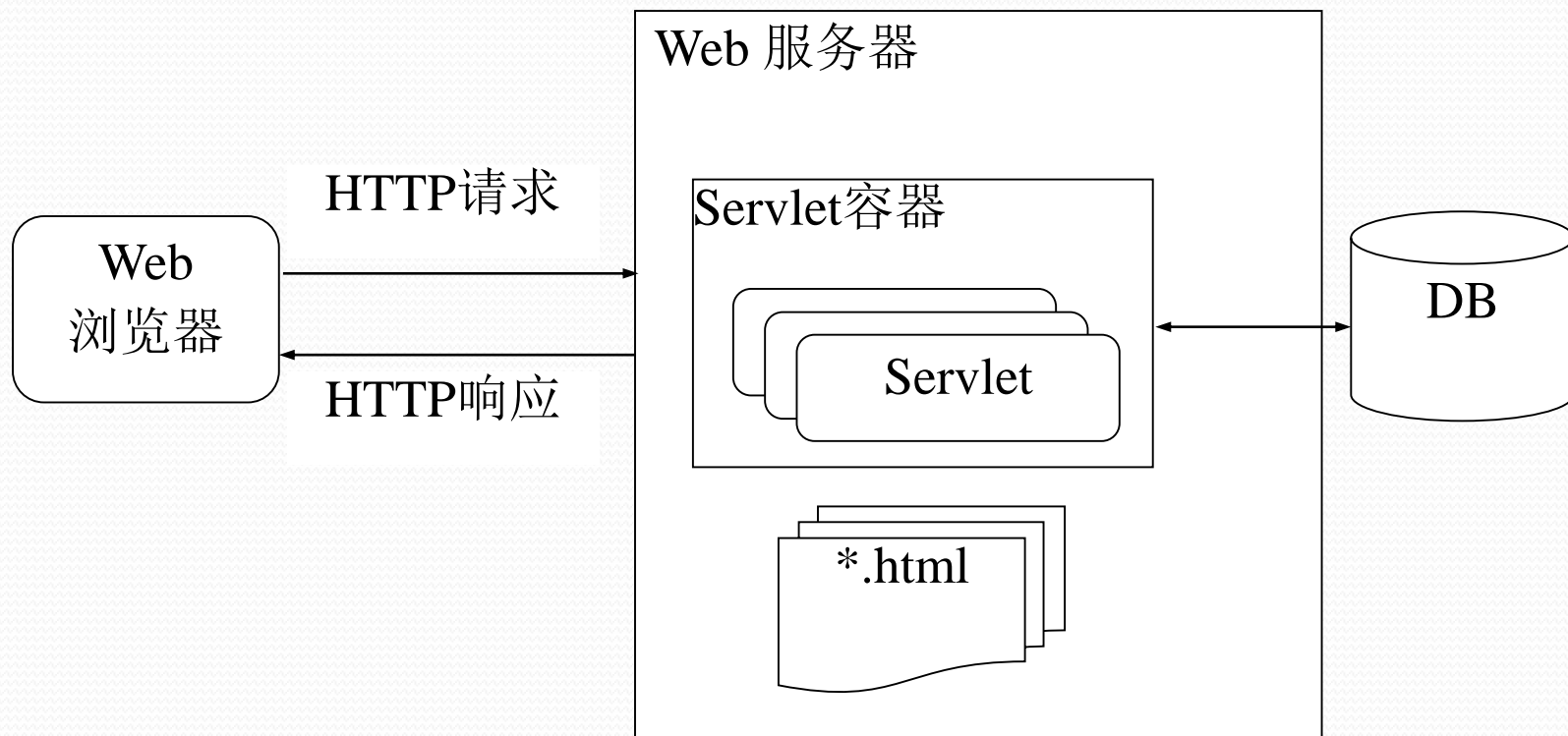
# 2.1Servlet简介

- 2、Servlet的功能

- 读取客户程序发送来的显式数据(表单数据)
- 读取客户程序发送来的隐式数据(请求报头)
- 生成相应的结果
- 发送显式的数据给客户端（以HTML格式）
- 发送隐式的数据给客户程序(状态代码和响应报头)
- 可以与其他服务器资源（如文件、数据库、Java应用程序等）进行通信

### 3、什么是Servlet容器

- Web服务器使用一个单独的模块装载和运行Servlet。这个专门用于Servlet管理的单独模块称为Servlet容器、或称Web容器。Tomcat含有Web容器。
- 各种不同的组件构成的一个示意图如下：

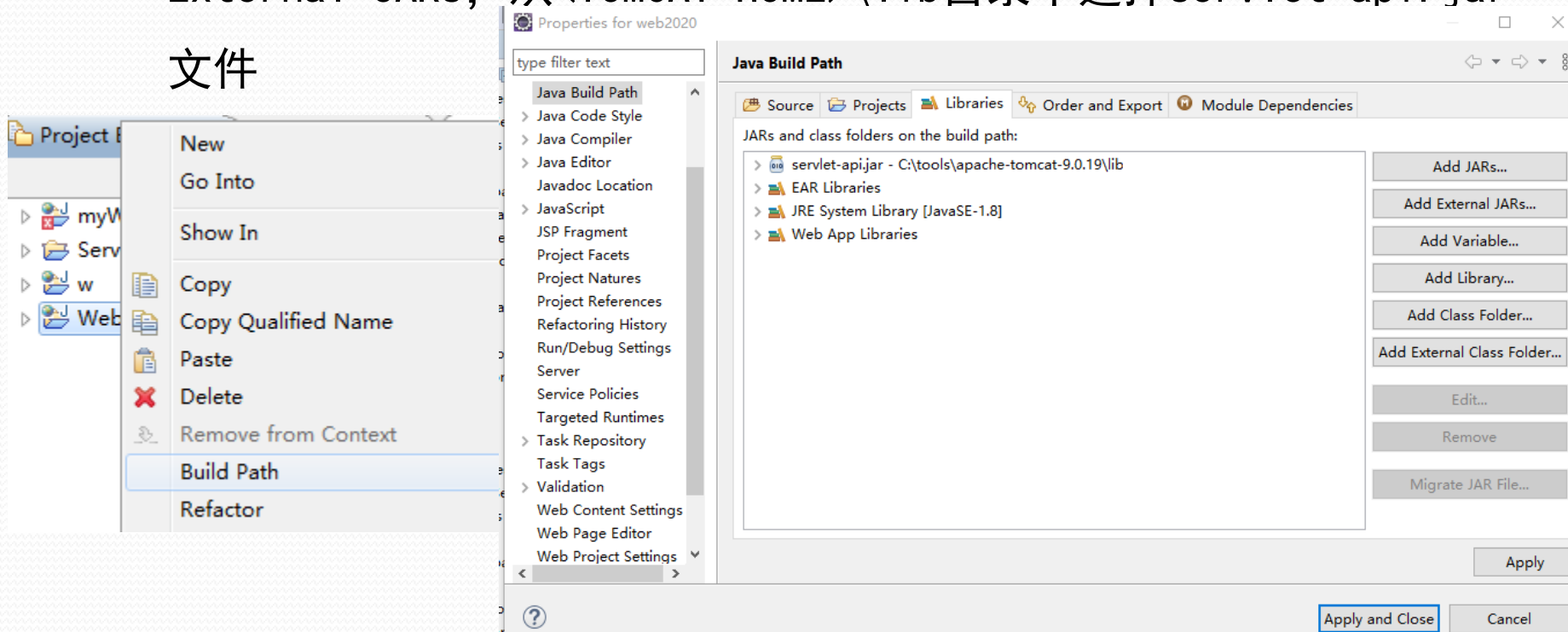


## 4、如何编写Servlet

- Servlet是Web应用程序的一个组件。Web应用程序具有严格定义的目录结构。
- 开发Servlet的步骤
  - ① 创建一个Web应用工程，如web
  - ② 项目配置
  - ③ 新建一个Servlet，如**HelloServlet.java**
  - ④ 编译、运行

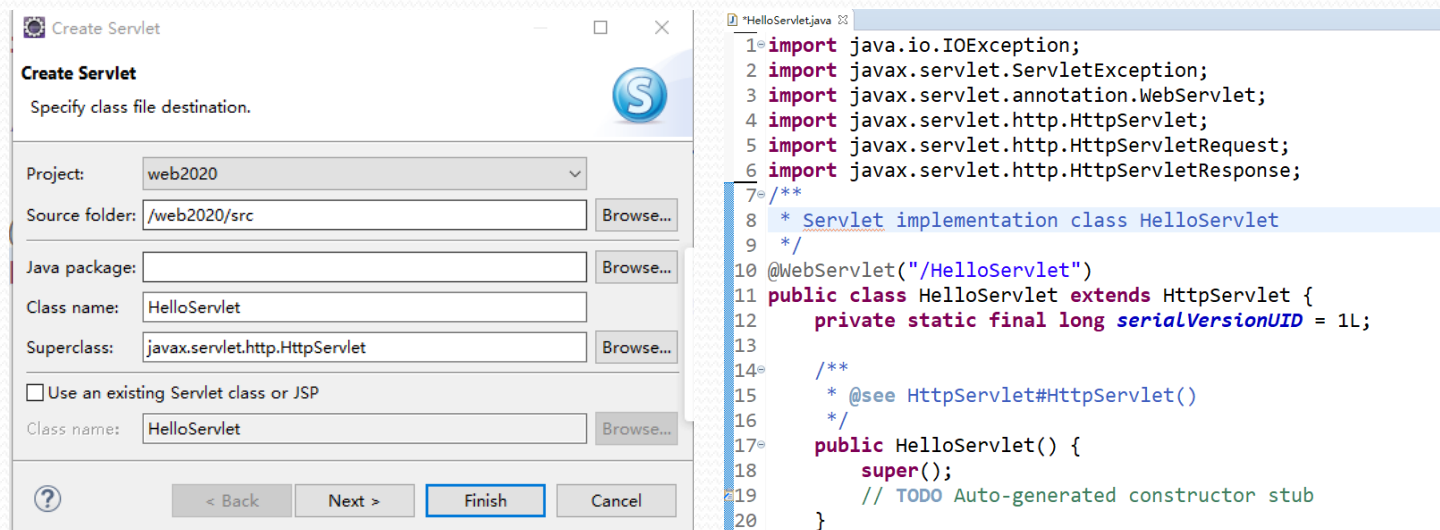
# 方法一：Eclipse中创建Servlet

- ① 创建一个Web应用工程，如web2020
- ② 项目配置。右键点击工程名web2020,选择Build Path->Configure Build Path...在弹出的对话框中选择Libraries，然后点击Add External JARs，从<TOMCAT HOME>\lib目录中选择servlet-api.jar文件



# 方法一：Eclipse中创建Servlet

- ③ 右键点击工程名web2020->new->Servlet，在Class name中输入servlet的类名,如HelloServlet



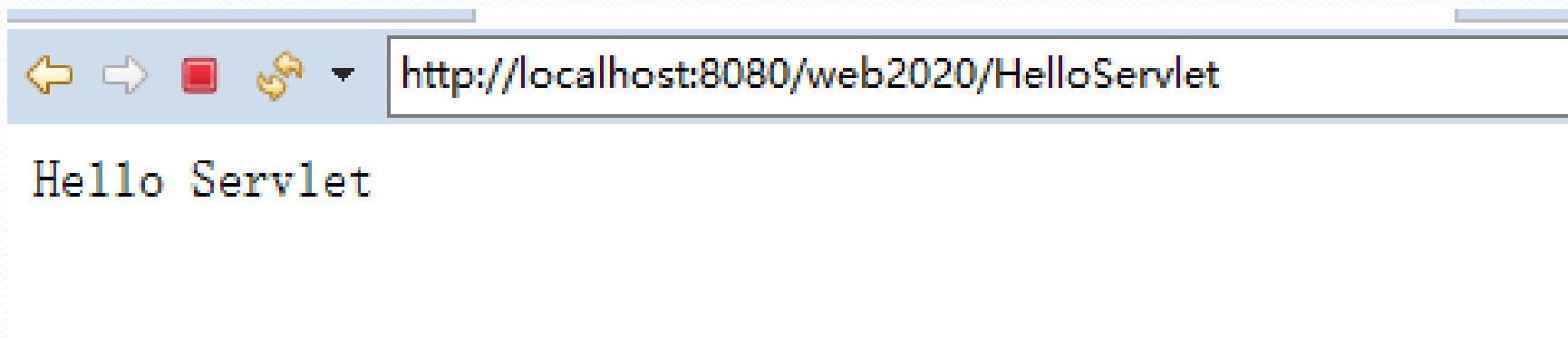
- ④ 在HelloServlet的doGet方法中输入测试代码

```
protected void doGet(HttpServletRequest request,
    // TODO Auto-generated method stub
    PrintWriter out=response.getWriter();
    out.println("Hello Servlet");
}
```



# 方法一：Eclipse中创建Servlet

⑤ 保存（即时完成编译），运行Run as->Run on server

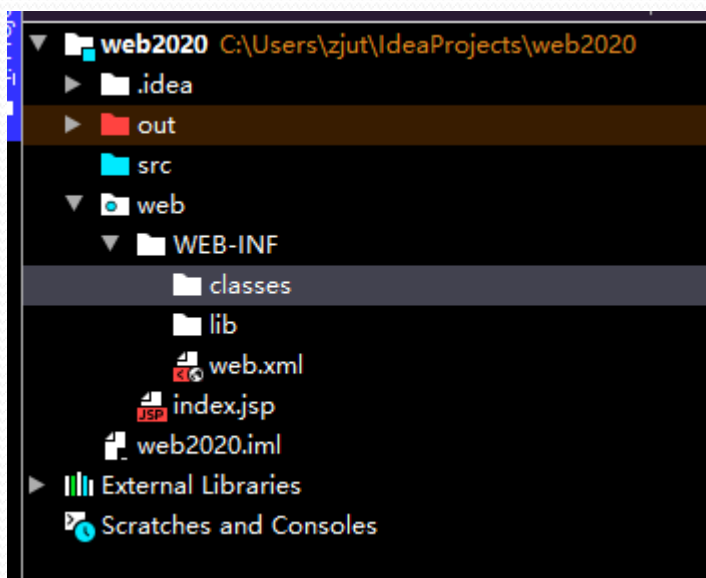


# 方法二：IDEA中创建Servlet

① 创建一个Web应用工程，如web2020

② 配置项目

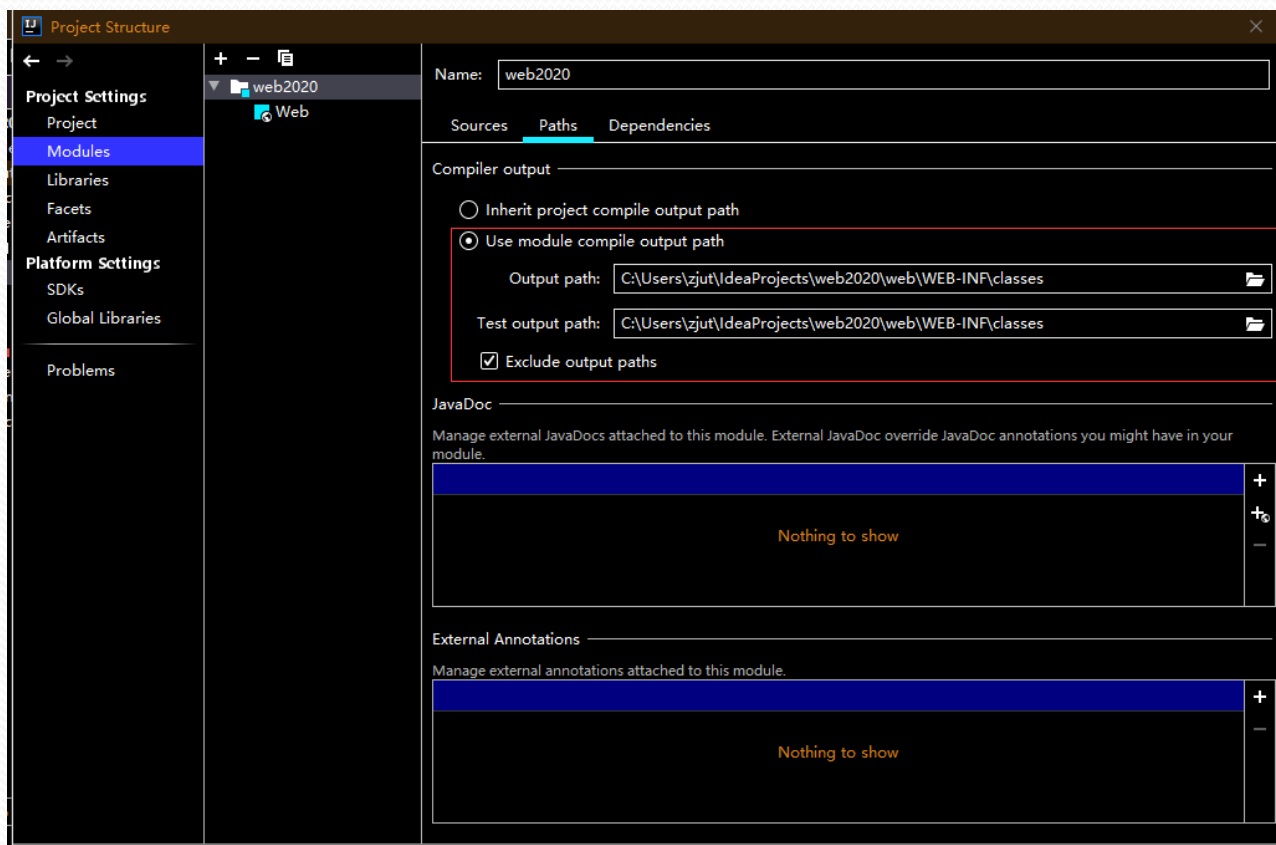
(1) 在WEB-INF目录下创建两个文件夹：classes和lib



# 方法二：IDEA中创建Servlet

## ② 配置项目

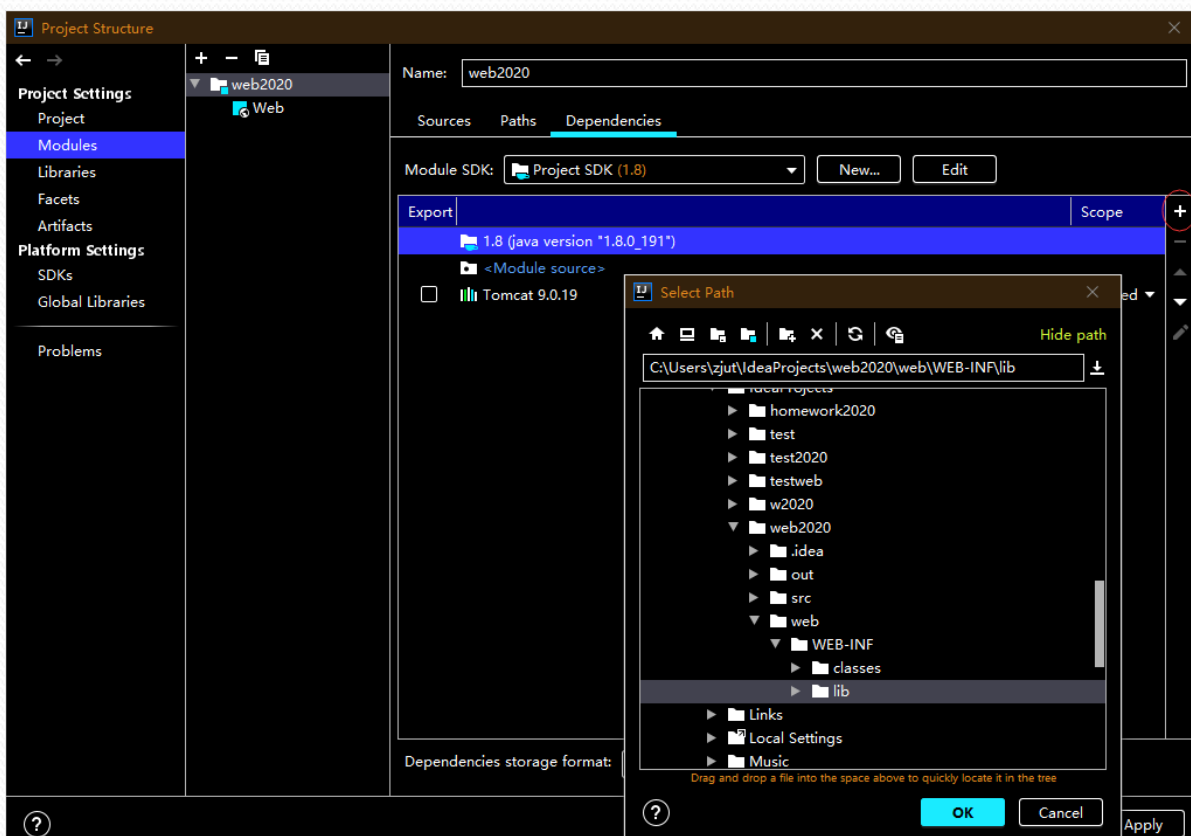
(2) 选择菜单File ---> Project Structure.. --->选择modules ---->path  
选项 将class文件的输出改为之前创建的classes目录



# 方法二：IDEA中创建Servlet

## ② 配置项目

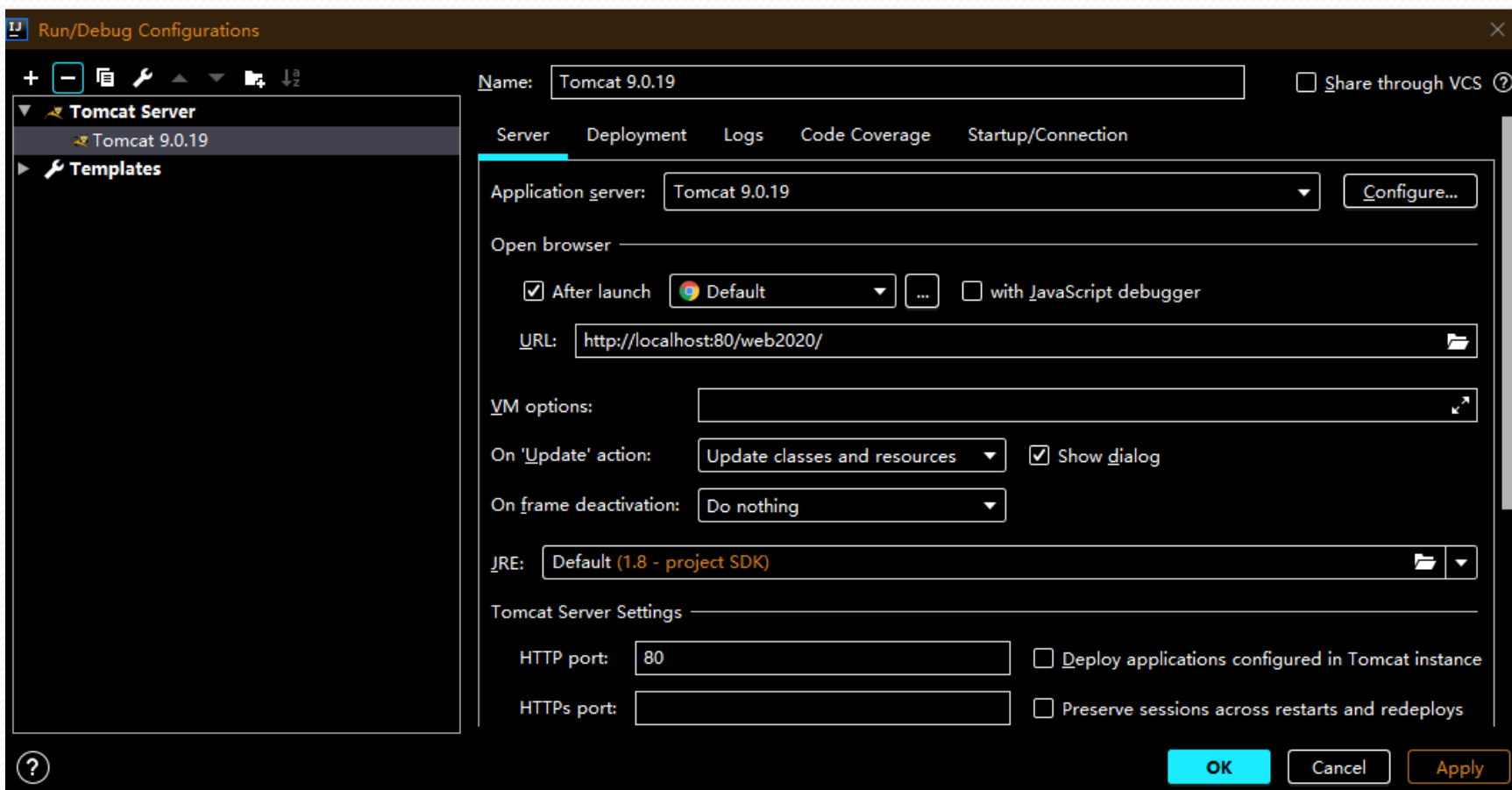
(3) 修改jar的存储目录。在modules菜单下，选择Dependencies 选项卡 ---> 点击右侧的 “+”号，选择 “JARs or directories...”，选择创建的 lib目录



# 方法二：IDEA中创建Servlet

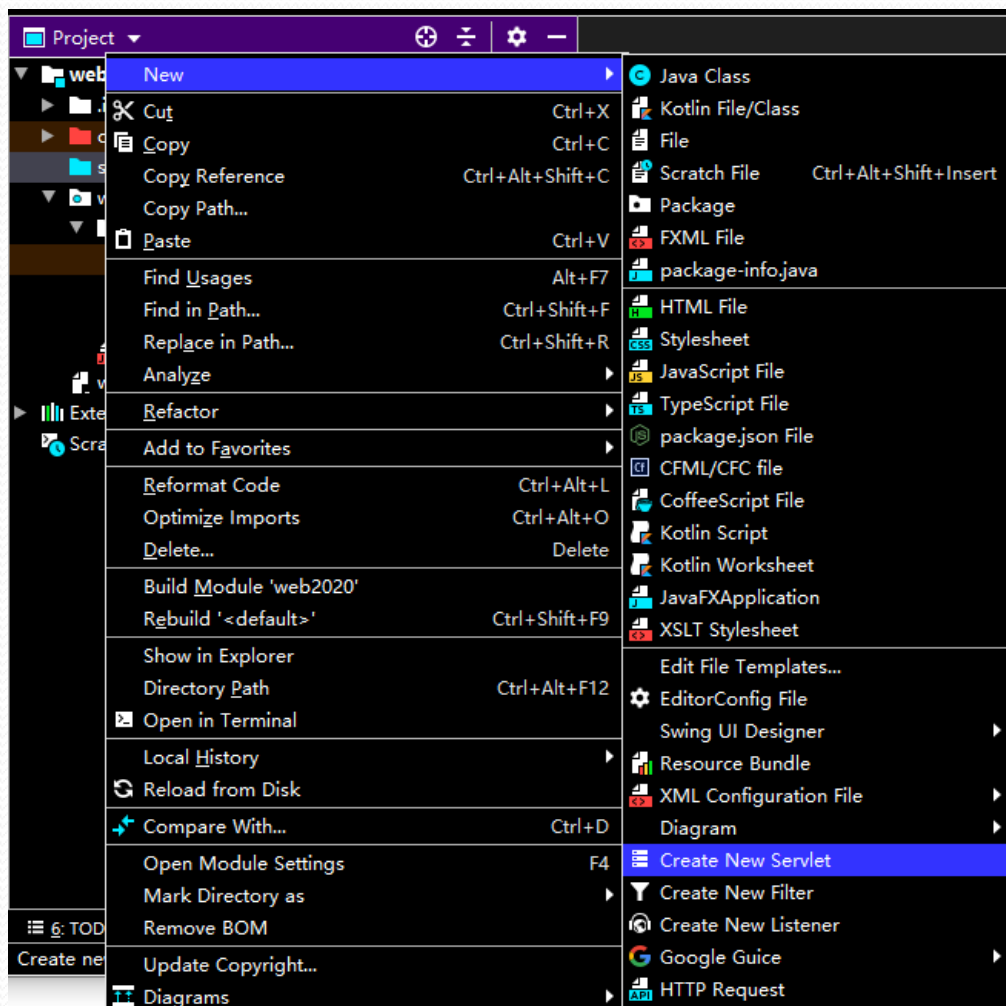
② 配置项目。

(4)在Run→Edit Configurations..配置Tomcat



# 方法二：IDEA中创建Servlet

③ 创建Servlet。右键src→new→Create New Servlet




## 方法二：IDEA中创建Servlet

- ④ 编写Servlet代码。在doGet中添加测试代码。

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
    PrintWriter out=response.getWriter();  
    out.println("Hello Servlet");  
}
```

## 方法二：IDEA中创建Servlet

⑤ 在WEB-INF中的web.xml添加Servlet的映射地址代码

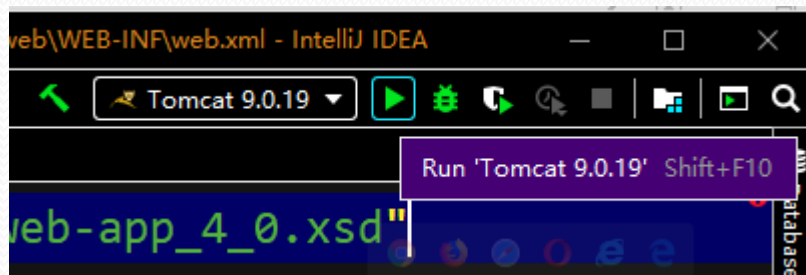


```
5      http://java.sun.com/xml/ns/javaee/web-app_4_0.xsd"
6      version="4.0">
7      <servlet>
8          <servlet-name>HelloServlet</servlet-name>
9          <servlet-class>HelloServlet</servlet-class>
10     </servlet>
11     <servlet-mapping>
12         <servlet-name>HelloServlet</servlet-name>
13         <url-pattern>/helloworld</url-pattern>
14     </servlet-mapping>
15 </web-app>
```

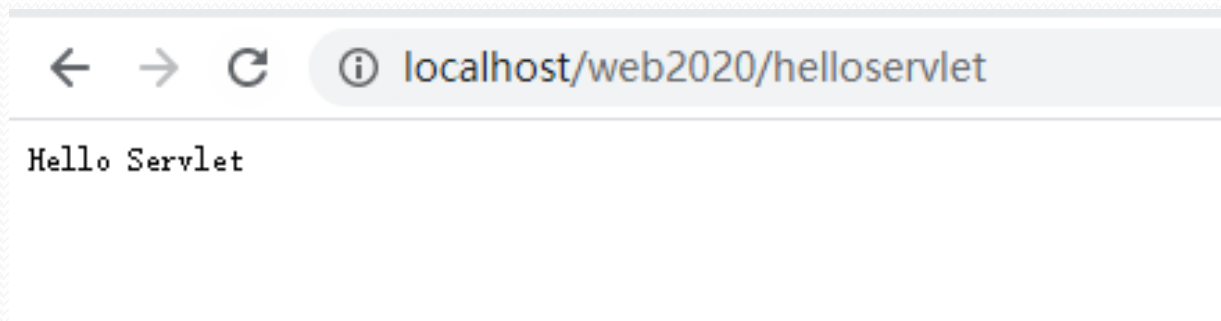


## 方法二：IDEA中创建Servlet

### ⑥ 启动tomcat



### ⑦ 浏览器地址栏输入http://localhost/web2020/helloservlet, 测试servlet



# HelloServlet.java

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "HelloServlet")
/* Eclipse默认的注解代码为: @WebServlet("/HelloServlet") */
public class HelloServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

    }

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out=response.getWriter();
        out.println("Hello Servlet");
    }
}
```

## 5、运行Servlet常见的错误

- 在测试Servlet时，由于各种原因可能会出现错误，下面是最经常出现的错误：

HTTP Status **404** -/myWeb/Hello.

The requested resource(/myWeb/Hello) is not available.

- 404错误是最常见的一种错误，它表示请求的资源不可用。有多种原因可导致该错误。你可以从这几方面检查：
  - 查看给定的路径名是否正确（包括大小写）；
  - 查看Servlet类文件是否在WEB-INF\classes目录中；
  - Servlet的注解是否有冲突导致tomcat启动失败
  - 查看web.xml文件内容是否正确；
  - 查看Tomcat服务器是否启动。
- 另一种常见错误是500，主要是Servlet代码有问题导致抛出各种异常了

## 6、Servlet的优缺点

优点：

- (1) 高效性。每个请求由一个轻量级的Java线程处理
- (2) 方便性。提供了大量的实用工具例程
- (3) 功能强大。许多使用传统CGI程序很难完成的任务都可以轻松地完成。
- (4) 可移植性好 。为一个服务器编写的Servlet无需任何实质上的改动即可移植到其他服务器上。
- (5) 节省投资 。有许多廉价甚至免费的Web服务器可供个人或小规模网站使用

## 6、Servlet的优缺点

缺点：

- 缺点是它经常既包含业务逻辑又包含表示逻辑。
- 表示逻辑（presentation logic）是展示给用户的信息，在Servlet中产生HTML响应就是表示逻辑。
- 业务逻辑（business logic）是完成某种数据处理和存储任务的功能。
- JSP技术可以实现业务逻辑和表示逻辑的分离：
  - Servlet专门处理业务逻辑
  - 用JSP实现表示逻辑。

## 2.2 Servlet API

- Servlet是Java Web应用开发的基础，Servlet API定义了若干接口和类。
- Servlet规范提供了一个标准的，平台独立的框架实现在Servlet和容器之间的通信。该框架是由一组Java接口和类组成的，它们称为Servlet API。

## 2.2 Servlet API

- Servlet API由下面4个包组成:
- `javax.servlet`包, 定义了开发独立于协议的服务器小程序的接口和类。
- `javax.servlet.http`包, 定义了开发采用HTTP协议通信的服务器小程序的接口和类。
- `javax.servlet.annotation`包, 定义9个注解类型和2个枚举类型。
- `javax.servlet.descriptor`包, 定义了访问Web应用程序配置信息的类型。

## 2.2.1 javax.servlet包

| 接口名                             | 说 明                                    |
|---------------------------------|--|
| Filter                          | 在请求和响应之间执行过滤任务的过滤器对象                   |
| FilterChain                     | Servlet容器向开发人员提供的一个过滤器链对象              |
| FilterConfig                    | Servlet容器使用的过滤器配置对象                    |
| RequestDispatcher               | 将请求转发到其他资源的对象                          |
| Servlet                         | 所有Servlet的根接口                          |
| ServletConfig                   | Servlet容器使用的Servlet配置对象，用来向Servlet传递信息 |
| ServletContext                  | 该接口定义了一些方法，Servlet可以与Servlet容器通信       |
| ServletRequest                  | 提供客户请求的对象                              |
| ServletResponse                 | 提供服务器响应的对象                             |
| ServletContextListener          | 用于监听Web应用程序的监听器接口                      |
| ServletContextAttributeListener | 用于监听Web应用程序属性的监听器接口                    |
| ServletRequestListener          | 用于监听请求对象的监听器接口                         |
| ServletRequestAttributeListener | 用于监听请求对象属性的监听器接口                       |
| SingleThreadModel               | 实现单线程的接口，已不推荐使用                        |



## 2.2.1 javax.servlet包

| 类 名                          | 说 明                         |
|------------------------------|-----------------------------|
| GenericServlet               | 定义了一般的、独立于协议的Servlet        |
| ServletContextAttributeEvent | Servlet环境属性的事件类             |
| ServletContextEvent          | Servlet环境的事件类               |
| ServletInputStream           | 从客户请求读取二进制数据的类              |
| ServletOutputStream          | 向客户发送二进制数据的类                |
| ServletRequestAttributeEvent | 请求属性事件类                     |
| ServletRequestEvent          | 请求事件类                       |
| ServletRequestWrapper        | 请求对象包装类                     |
| ServletResponseWrapper       | 响应对象包装类                     |
| ServletException             | 当Servlet遇到一般错误时抛出该异常        |
| UnavailableException         | Servlet或过滤器在其永久或临时不可用时抛出的异常 |

# 1. Servlet接口

- Servlet接口是Servlet API中的核心接口，每个Servlet必须直接或间接实现该接口。
- Servlet接口定义了如下5个方法：
  - ① `public void init(ServletConfig config)`
  - ② `public void service(ServletRequest req, ServletResponse res)  
throws ServletException, IOException`
  - ③ `public ServletConfig getServletConfig()`
  - ④ `public String getServletInfo()`
  - ⑤ `public void destroy()`

## 2. ServletConfig接口

- ServletConfig接口为用户提供了有关Servlet配置信息。
- Servlet配置包括Servlet名称、Servlet上下文对象、Servlet初始化参数等。

### 3. GenericServlet类

- GenericServlet抽象类实现了Servlet接口和ServletConfig接口，提供了Servlet接口中除了service()方法外的所有方法的实现，同时增加了几个支持日志的方法。
- 可以扩展该类并实现service()方法来创建任何类型的Servlet。

# 例：GenericServlet示例

```
package com.demo;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
@WebServlet(name = "genericServlet", urlPatterns = { "/generic-servlet" })
public class GenericDemoServlet extends GenericServlet{
    private transient ServletConfig servletConfig;
    @Override
    public void service(ServletRequest request,
                        ServletResponse response)
                        throws ServletException, IOException {
        servletConfig = getServletConfig();
        String servletName = servletConfig.getServletName();
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.print("<!DOCTYPE html>" + "<html>"
                + "<body>Hello from " + servletName + "<br>"
                + "世界那么大，我想去看看。"
                + "</body></html>");
    }
}
```

## 4. ServletRequest接口

- ServletRequest接口是独立于任何协议的请求对象，定义了获取客户请求信息的方法，如 `getParameter()`、`getProtocol()`、`getRemoteHost()` 等。

## 5. ServletResponse接口

- ServletResponse接口是独立于任何协议的响应对象，定义了向客户发送响应的方法，如 `setContentType()` 方法、`sendRedirect()` 方法、`getWriter()` 方法等。

## 2.2.2 javax.servlet.http包

- 该包提供创建使用HTTP协议的Servlet所需要的接口和类。
- 该包共定义8个接口和7个类，其中某些接口和类扩展了javax.servlet包中对应的接口和类来实现对HTTP协议的支持。



## 2.2.2 javax.servlet.http包

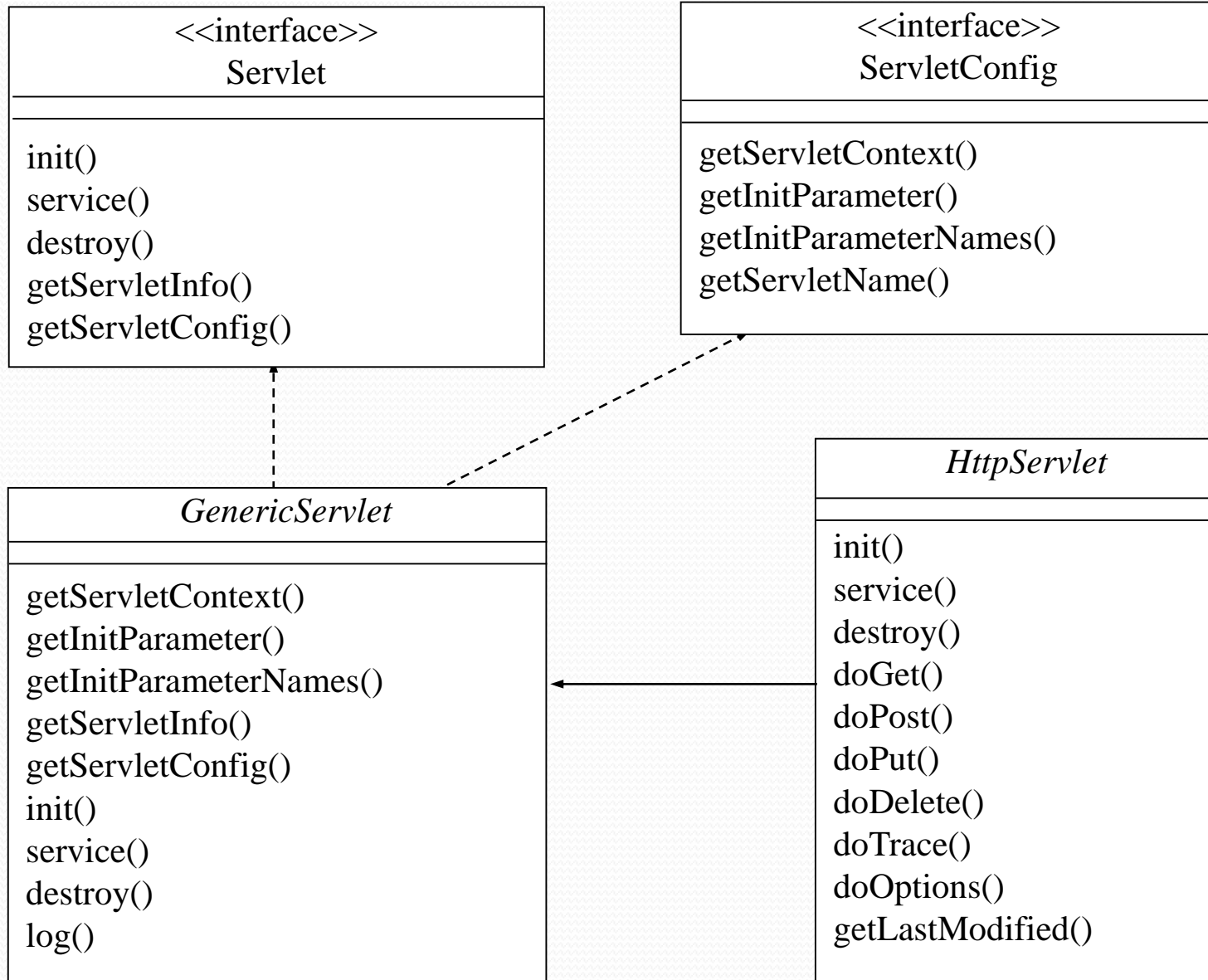
提供创建使用HTTP协议的Servlet所需要的接口和类。

| 接口名                           | 说 明                 |
|-------------------------------|---------------------|
| HttpServletRequest            | 该接口提供了有关HTTP请求的信息   |
| HttpServletResponse           | 该接口提供了有关HTTP响应的信息   |
| HttpSession                   | 实现会话管理的接口，也用来存储用户信息 |
| HttpSessionActivationListener | HTTP会话启动监听器接口       |
| HttpSessionAttributeListener  | HTTP会话属性监听器接口       |
| HttpSessionBindingListener    | HTTP会话绑定监听器接口       |
| HttpSessionListener           | HTTP会话监听器接口         |
| HttpSessionContext            | 该接口已不推荐使用           |

## 2.2.2 javax.servlet.http包

| 类 名                        | 说 明                       |
|----------------------------|---------------------------|
| HttpServlet                | 用于创建HTTP Servlet的抽象类      |
| Cookie                     | 创建Cookie对象的一个实现类          |
| HttpServletRequestWrapper  | HttpServletRequest接口的实现类  |
| HttpServletResponseWrapper | HttpServletResponse接口的实现类 |
| HttpSessionEvent           | 会话事件类                     |
| HttpSessionBindingEvent    | 会话绑定事件或会话属性事件类            |
| HttpUtils                  | 一个工具类，已不推荐使用              |

# Servlet API的层次结构



# 1. `HttpServlet`类

- `HttpServlet`抽象类用来实现针对HTTP协议的`Servlet`，它扩展了`GenericServlet`类。
- 在`HttpServlet`类中增加了一新的`service()`方法，格式如下：

```
protected void service (HttpServletRequest,  
                        HttpServletResponse)  
    throws ServletException, IOException
```

- `service`是`Servlet`向客户提供服务的一个方法，我们编写的`Servlet`可以覆盖该方法。

## 例 : ServiceImplServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
@WebServlet("/service.do")
public class ServiceImplServlet extends HttpServlet{
    public void service(HttpServletRequest request,
                        HttpServletResponse response)
                        throws ServletException,IOException{
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<font color = '#0000ff'>");
        out.println("<h3>Hello,World!</h3>");
        out.println("The time now is:"+new java.util.Date());
        out.println("</body>");
        out.println("</html>");
    }
}
```

# 1. HttpServlet类

- 在HttpServlet中针对不同的HTTP请求方法定义了不同的处理方法，如处理GET请求的doGet () 方法格式如下：

```
protected void doGet(HttpServletRequestRequest,  
                        HttpServletResponse)
```

```
throws ServletException,IOException
```

- 一般Servlet覆盖doGet () 方法或doPost () 方法。

## 2. HttpServletRequest接口

- HttpServletRequest接口扩展了ServletRequest接口并提供了针对HTTP请求操作方法，如定义了从请求对象中获取HTTP请求头、Cookie等信息的方法。

### 3. HttpServletResponse接口

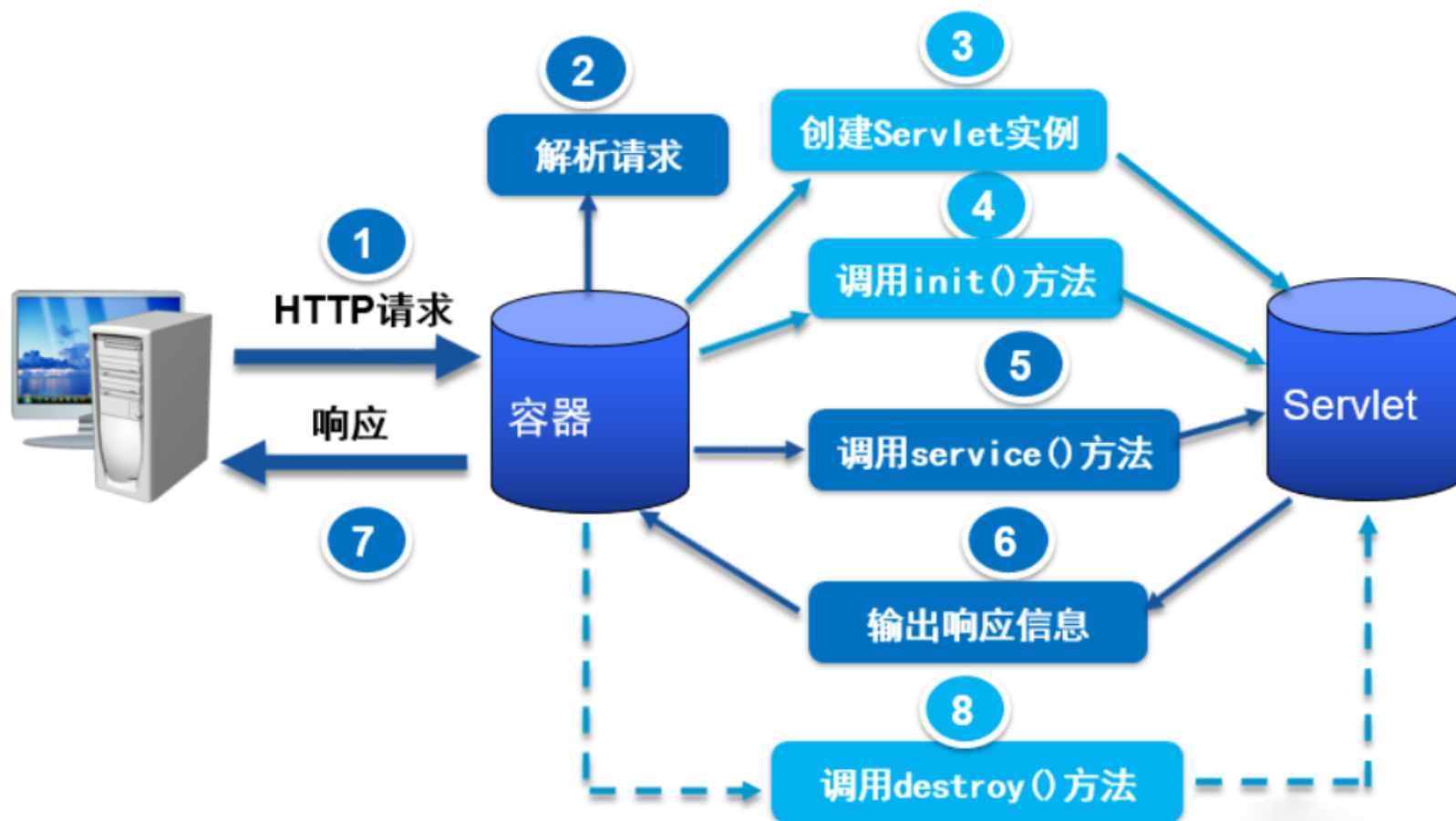
- HttpServletResponse接口扩展了ServletResponse接口并提供了针对HTTP的发送响应的方法。它定义了为响应设置如HTTP头、Cookie信息的方法。



## 2.3 Servlet生命周期

- Servlet作为一种在容器中运行的组件，有一个从创建到销毁的过程，这个过程被称为Servlet生命周期。
- Servlet生命周期包括以下几个阶段：
  - 加载和实例化Servlet类
  - 调用`init()`方法初始化Servlet实例
  - 一旦初始化完成，容器从客户收到请求时就将调用它的`service()`方法
  - 最后容器在Servlet实例上调用`destroy()`方法使它进入销毁状态。

## 2.3 Servlet生命周期



## 2.3.1 加载和实例化Servlet

- 当Servlet容器启动或客户端发送一个请求时，Servlet容器会查找内存中是否存在该Servlet实例，若存在，则直接读取该实例响应请求；如果不存在，就创建一个Servlet实例。
- 对每个Servlet，容器使用Class.forName()方法对其加载并实例化。
- 容器创建了Servlet实例后就进入生命周期阶段，Servlet生命周期方法包括
  - init()方法
  - service()方法
  - destroy()方法

## 2.3.2 初始化Servlet

- 实例化后，Servlet容器将调用Servlet的 `init (ServletConfig)` 方法进行初始化（一些准备工作或资源预加载工作）
- 调用 `init (ServletConfig)` 方法后，容器将调用无参数的 `init ()` 方法，之后Servlet就完成初始化。在Servlet生命周期中 `init ()` 方法仅被调用一次。

## 2.3.3 为客户提供服务

- 初始化后，Servlet处于能响应请求的就绪状态，即为客户提供服务。
- 当容器接收到对Servlet的请求时，容器根据请求中的URL找到正确的Servlet。
  - ① 创建两个对象 (请求和响应)
  - ② 创建一个新的线程，在该线程中调用`service()`方法，同时将请求对象和响应对象作为参数传递给该方法
  - ③ `service()`方法会根据不同的HTTP决定调用`doGet()`或`doPost()`方法。

## 2.2.3 为客户提供服务

- Servlet使用响应对象（response）获得输出流对象，调用有关方法将响应发送给客户浏览器。
- 线程将被销毁或者返回到容器管理的线程池。
- 请求和响应对象已经离开其作用域，也将被销毁。
- 最后客户得到响应。

## 2.3.4 销毁和卸载Servlet

- 当容器决定不再需要Servlet实例时，它将在Servlet实例上调用`destroy()`方法，Servlet在该方法中释放资源，如它在`init()`方法中获得的数据库连接。一旦该方法被调用，Servlet实例不能再提供服务。
- 一旦Servlet实例被销毁，它将作为垃圾被回收。如果Web容器关闭，Servlet也将被销毁和卸载。

# 模拟异步处理展示 Servlet生命周期的例子

```
@WebServlet(urlPatterns = "/demo", asyncSupported = true)
public class AsyncDemoServlet extends HttpServlet {
    @Override
    public void init(ServletConfig servletConfig) {
        System.out.println("初始化Servlet");
    }

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        out.println("进入Servlet的时间: " + new Date() + "<br>");
        System.out.println("进入Servlet的时间: " + new Date() );
        out.flush();

        //在子线程中执行业务调用，并由其负责输出响应，主线程退出
        AsyncContext ctx = req.startAsync();
        new Thread(new Executor(ctx)).start();

        out.println("结束Servlet的时间: " + new Date() + "<br>");
        System.out.println("结束Servlet的时间: " + new Date() );
        out.flush();
    }
}
```



# 模拟异步处理展示 Servlet生命周期的例子

```
class Executor implements Runnable {
    private AsyncContext ctx = null;
    public Executor(AsyncContext ctx){
        this.ctx = ctx;
    }

    public void run(){
        try {
            //等待十秒钟，以模拟业务方法的执行
            Thread.sleep(10000);
            PrintWriter out = ctx.getResponse().getWriter();
            out.println("业务处理完毕的时间: " + new Date() + "<br>");
            System.out.println("业务处理完毕的时间: " + new Date() );
            out.flush();
            ctx.complete();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# 运行结果

## ● 第一次运行

进入Servlet的时间: Sat Mar 07 13:19:48 CST 2020  
结束Servlet的时间: Sat Mar 07 13:19:48 CST 2020  
业务处理完毕的时间: Sat Mar 07 13:19:58 CST 2020

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0\_191\bin\javaw.exe (2020年三月 07, 2020 1:19:47 下午 org.apache.catalina.startup.Catalina: Server startup in [1,243] milliseconds  
初始化Servlet  
进入Servlet的时间: Sat Mar 07 13:19:48 CST 2020  
结束Servlet的时间: Sat Mar 07 13:19:48 CST 2020  
业务处理完毕的时间: Sat Mar 07 13:19:58 CST 2020

## ● 第二次运行

进入Servlet的时间: Sat Mar 07 13:20:24 CST 2020  
结束Servlet的时间: Sat Mar 07 13:20:24 CST 2020  
业务处理完毕的时间: Sat Mar 07 13:20:34 CST 2020

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0\_191\bin\javaw.exe (2020年三月 07, 2020 1:19:47 下午 org.apache.catalina.startup.Catalina: Server startup in [1,243] milliseconds  
初始化Servlet  
进入Servlet的时间: Sat Mar 07 13:19:48 CST 2020  
结束Servlet的时间: Sat Mar 07 13:19:48 CST 2020  
业务处理完毕的时间: Sat Mar 07 13:19:58 CST 2020  
进入Servlet的时间: Sat Mar 07 13:20:24 CST 2020  
结束Servlet的时间: Sat Mar 07 13:20:24 CST 2020  
业务处理完毕的时间: Sat Mar 07 13:20:34 CST 2020

**Thank You !**