

第十章：Servlet

第1节：HTTP协议

1.1 概念

Hypertext Transfer Protocol, HTTP超文本传输协议, 是一个简单的请求-响应协议。规定了客户端和服务端之间 传输数据的一种格式

HTTP底层协议: TCP 协议

TCP 的特点: 面向连接, 保证可靠性

TCP三次握手, 因为HTTP是基于TCP协议的, 所有也可以认为是HTTP的三次握手:

第一次握手: 客户端向服务端发送一个请求消息, 服务端收到信息后知道自己与客户端是可以连接成功的;

第二次握手: 此时客户端并不知道服务端是否已经接收到了它的请求, 所以服务端接收到消息后要向客户端做出响应, 客户端得到服务端的反馈后, 才确定自己与服务端是可以连接上的。

第三次握手: 客户端确认与服务器建立连接后, 才开始向服务器端发送数据。

1.2 特点

基于TCP/IP的高级协议

默认端口号: 80

基于请求/响应模型: 一次请求对应一次响应

特点:

1. 简单快速。客户端向服务器请求服务时, 只需要传送请求方法和路径。
2. 灵活。HTTP协议允许传送任意格式的数据。正在传输的类型由, content-type标明。
3. 无连接。就是每个请求都建立一个连接, 请求处理完毕并发送至客户端之后就断开连接。这样明显有其缺点, 就是在需要连续发送请求时, 需要为每一个请求单独的再次建立连接, 很浪费时间和资源。于是在HTTP协议1.1版本之后引入了可持续连接, 也就是再每一个请求处理完毕后, 它不会立刻就断开连接, 而是再等待一段时间, 如果在此期间又有新的请求过来, 那么等待时间刷新, 如果没有, 则等待时间完毕后, 连接关闭。
4. 无状态。是指协议对于每次请求的处理没有记忆能力, 它不知道之前是否已经访问过, 不保留访问痕迹。主要目的是为了保证数据传输的安全性。

1.3 传输数据的特点

1.3.1 请求

请求分为三部分:

请求行 : 请求行中, 我们可以通过request对象的相应方法获取到比如协议名、服务名、端口号、项目名称、请求方式、参数列表等信息。

请求头 : 请求头是当前对用户发送的数据的描述信息。请求头信息在请求的时候不需要程序员手动添加, 是浏览器发送的时候已经处理好的。

请求体 : 请求体就是请求中携带的数据, 也就是我们需要获取的参数。

1.3.2 响应

响应分为三部分：

响应行：响应行中包含协议和版本

响应头：服务器想要告诉浏览器的一些信息。写法格式：头名称:值

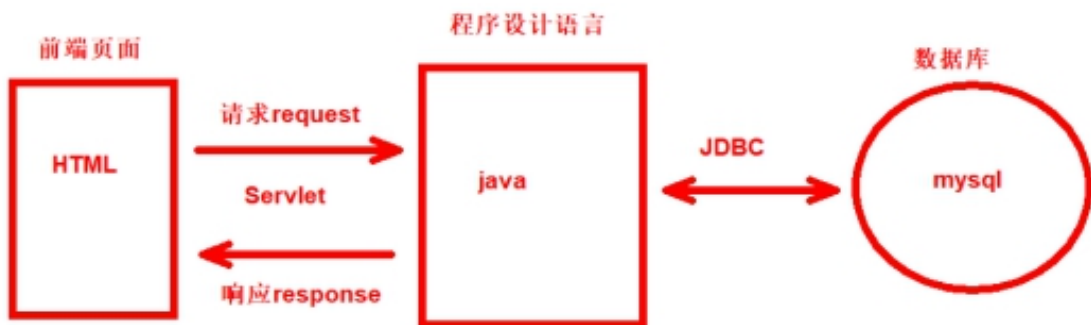
响应体：服务器响应给浏览器的源码信息和状态码

- 1xx -信息，服务器收到请求，需要请求者继续执行操作
- 2xx -成功，操作被成功接收并处理
- 3xx -重定向，需要进一步的操作以完成请求
- 4xx -客户端错误，请求包含语法错误或无法完成请求
- 5xx -服务器错误，服务器在处理请求的过程中发生了错误

第2节：Servlet详解

2.1 概念

servlet 是运行在 Web 服务器中的小型 Java 程序。servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求



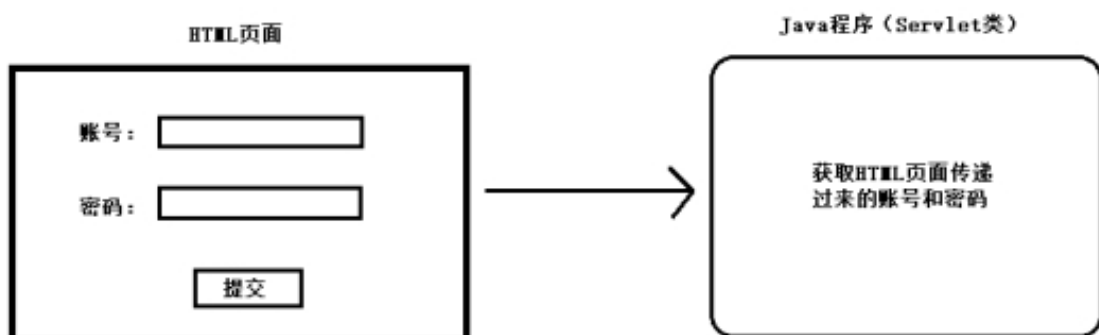
简单理解：

我们之前讲的JDBC技术是用来实现Java语言和数据库之间的连接技术，而现在又涉及到Java和HTML进行连接，此时可以由HTML做前台，java做后台进行连接，这时候就需要一个中间件（Servlet）。

2.2 使用

2.2.1 准备工作

需求：写一个HTML页面将一个表单中的账号和密码传递到java类中并打印输出到控制台。



2.2.2 编写步骤

编写步骤：

1. 创建HTML页面并设计一个表单
2. 创建一个普通类，并继承HTTPServlet抽象类，并重写Service方法
3. 在web.xml文件中注册servlet，作为前后台连接的中间件
4. 修改前台表单中action的请求地址
5. 通过浏览器输入对应的地址来访问测试

2.2.3 实现方式

1. 创建HTML页面并设计一个表单

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<!-- 设计一个页面，并设计表单 /项目名/访问地址 -->
<form action="" method="post">
    <p>
        账号: <input type="text" name="username" id="username">
    </p>
    <p>
        密码: <input type="password" name="password" id="password">
    </p>
    <p>
        <input type="submit" value="登录">
    </p>
</form>
</body>
</html>
```

2. 创建一个普通类，并继承HTTPServlet抽象类，并重写Service方法

```
package com.ujiuye.servlet;

import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import java.io.IOException;

//创建一个普通类，让其普通类继承HttpServlet抽象类
public class LoginServlet extends HttpServlet {
    //并重写Service方法
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

```

String username = request.getParameter("username");
String password = request.getParameter("password");
System.out.println("账号为: " + username + " ==> 密码为: " + password);
}
}

```

3. 在web.xml文件中注册servlet，作为前后台连接的中间件

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

<!--注册servlet-->
<servlet>
    <servlet-name>login</servlet-name><!--servlet名称-->
    <servlet-class>com.ujiuye.servlet.LoginServlet</servlet-class><!--全路径-->
</servlet>
<!--映射路径/虚拟路径-->
<servlet-mapping>
    <servlet-name>login</servlet-name><!--servlet名称-->
    <url-pattern>/login</url-pattern><!--访问路径，以 / 开头-->
</servlet-mapping>
</web-app>

```

4. 修改前台表单中action的请求地址

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
<!-- 设计一个页面，并设计表单 /项目名/访问地址 -->
<form action="login" method="post">
    <p>
        账号: <input type="text" name="username" id="username">
    </p>
    <p>
        密码: <input type="password" name="password" id="password">
    </p>
    <p>
        <input type="submit" value="登录">
    </p>
</form>
</body>
</html>

```

5. 通过浏览器输入对应的地址来访问测试

← → ↻ ⓘ localhost:8080/day10_servlet/login.html

账号:

密码:

登录

效果展示:

Output

Connected to server

[2021-02-04 10:18:18,661] Artifact day10_servlet:war exploded: Artifact is being deployed, please wait...

[2021-02-04 10:18:19,269] Artifact day10_servlet:war exploded: Artifact is deployed successfully

[2021-02-04 10:18:19,272] Artifact day10_servlet:war exploded: Deploy took 608 milliseconds

04-Feb-2021 10:18:28.183 洪℃佬 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory

04-Feb-2021 10:18:28.245 洪℃佬 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory

账号为: liuyan ==> 密码为: 123

2.3 实现过程

实现过程:

- ① 当服务器接收到客户端浏览器的请求后, 先解析请求的url路径, 获取访问Servlet的资源路径
- ② 查找项目的web.xml, 根据资源路径匹配web.xml中的 `<servlet-mapping>` 中的 `<url-pattern>`
- ③ 如果没有匹配到报 404 如果匹配到了 根据 `<servlet-mapping>` 中的 `<servlet-name>`
- ④ 再去匹配 `<servlet>` 中的 `<servlet-name>` 如果没有匹配到 404
- ⑤ 如果匹配到了执行 `<servlet>` 中的 `<servlet-class>` 从而以反射的方式访问到指定的Servlet, 调用其方法

2.4 生命周期

servlet生命周期: 从创建到销毁的全过程。共分为三个阶段:

- 1、初始化
- 2、使用 (提供服务)
- 3、销毁

1. 编写Servlet

```
package com.ujiuye.servlet;
```

```
import javax.servlet.*;
```

```
import java.io.IOException;
```

```
public class LifeCycleServlet implements Servlet {
```

```
//初始化
```

```

@Override
public void init(ServletConfig servletConfig) throws ServletException {
    System.out.println("servlet初始化了...");
}

@Override
public ServletConfig getServletConfig() {
    return null;
}

//服务
@Override
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
    System.out.println("servlet中的service方法执行了....");
}

@Override
public String getServletInfo() {
    return null;
}

//销毁
@Override
public void destroy() {
    System.out.println("servlet销毁了...");
}
}

```

2. 配置Servlet

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">

    <!--注册servlet-->
    <servlet>
        <servlet-name>lifeCycle</servlet-name><!--servlet名称-->
        <servlet-class>com.ujiuye.servlet.LifeCycleServlet</servlet-class><!--全路径-->
    </servlet>
    <!--映射路径/虚拟路径-->
    <servlet-mapping>
        <servlet-name>lifeCycle</servlet-name><!--servlet名称-->
        <url-pattern>/lifeCycle</url-pattern><!--访问路径，以 / 开头-->
    </servlet-mapping>
</web-app>

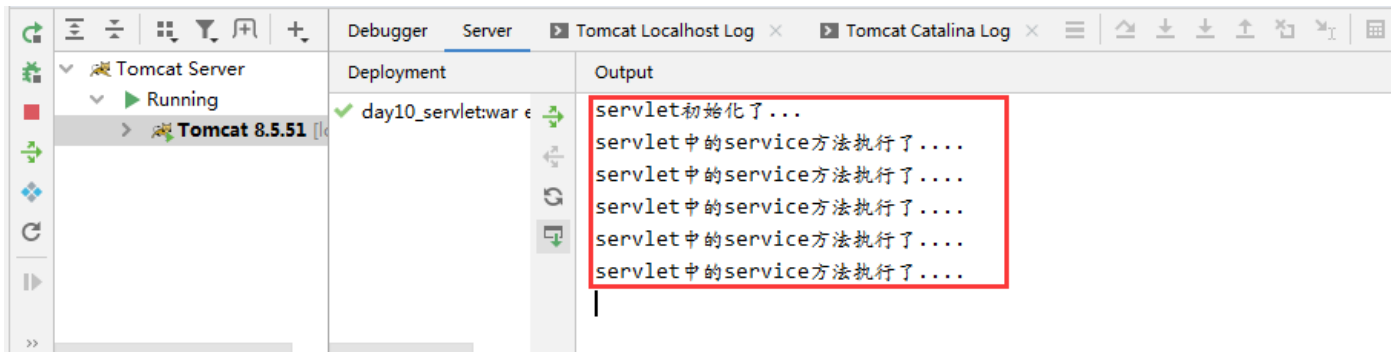
```

3. 测试观察：

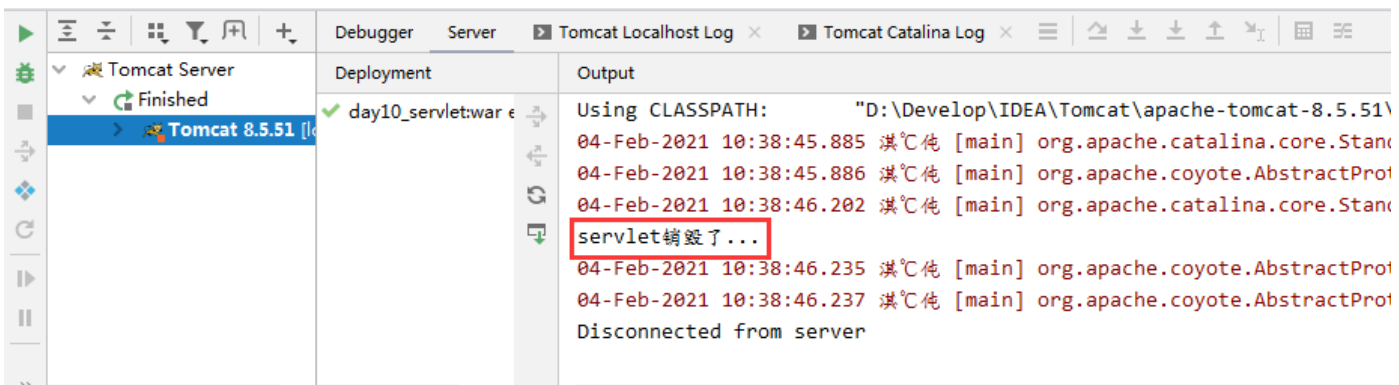
当我们启动tomcat服务器时，观察控制台并无打印消息，表示Servlet默认情况下在启动服务器时并未被创建。在浏览器地址栏中直接输入MyServlet的虚拟访问地址：

`http://localhost:8080/day10_servlet/lifeCycle`

此时观察控制台打印输出初始化与服务的相关信息，当多次访问该地址，发现服务方法被多次调用执行。



当我们关闭tomcat服务，会在控制台打印销毁方法的信息。



通过以上案例发现Servlet默认情况下是在第一次访问时被创建并初始化的，为了减少内存的压力，我们可否改变它的创建时机呢？

答案是肯定的。我们可以在web.xml文件中进行配置，使得Servlet在服务器启动时直接创建并初始化。

load-on-startup节点必须写在servlet节点的内部，且作为最后一个节点。取值必须是整数，如果是大于等于0的整数表示在服务器启动时就被创建并初始化，如果有多个Servlet设置了load-on-startup节点的值，那么值越小越优先执行；如果是负数则表示第一次被访问时创建并初始化，也就是默认情况，可以省略不写。

```
<!--注册servlet-->
<servlet>
    <servlet-name>lifeCycle</servlet-name><!--servlet名称-->
    <servlet-class>com.ujiuye.servlet.LifeCycleServlet</servlet-class><!--全路径-->
    <!--配置servlet在服务器启动时被创建-->
    <load-on-startup>0</load-on-startup>
</servlet>
<!--映射路径/虚拟路径-->
<servlet-mapping>
    <servlet-name>lifeCycle</servlet-name><!--servlet名称-->
    <url-pattern>/lifeCycle</url-pattern><!--访问路径，以 / 开头-->
</servlet-mapping>
```

总结：

1、init()初始化阶段：默情况当第一次访问Servlet时，Servlet容器（tomcat）会加载Servlet，加载完成后，Servlet容器（tomcat）会创建一个Servlet实例并调用init()方法进行初始化；如果需要改变创建时机，只需要在web.xml文件中配置load-on-startup节点值为大于等于0的整数即可。init()方法只会执行一次，也说明一个Servlet只有一个实例，即单例模式。


2、service()服务阶段：当Servlet每被访问一次，service方法就会自动被调用一次，所以该方法可执行多次。

3、Destroy()销毁阶段：当关闭服务或资源重新加载时，destroy方法会被执行，这就标识着整个Servlet的生命周期到此结束。该方法仅被执行一次。

2.5 进化史

Servlet也是由复杂难用、功能单一，逐步发展到目前的操作简单，功能强大的。
Servlet的发展史，完全就是一部人类由猿人的进化史。

2.5.1 第一阶段



第一阶段：猿人阶段

JavaEE规定了javax.servlet.Servlet接口，**只要实现这个接口，那么实现类就是一个Servlet程序。**

接口中有五个方法：

- init();**//servlet程序出生时会执行的方法
- service();**//Servlet程序执行一次，一定会执行的方法
- destroy();**//servlet程序销毁时，会执行的方法
- getServletConfig();//无关方法，获取Servlet配置信息
- getServletInfo();//无关方法，获取Servlet说明信息

程序员慢慢发现，Servlet接口中我们**仅关注编写service()方法**，其他四个方法几乎用不到，每次实现接口又必须实现，特别麻烦

案例演示：

```
package com.ujiuye.servlet;

import javax.servlet.*;
import java.io.IOException;

public class ApeManStageServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }
}
```



```

@Override
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {

}

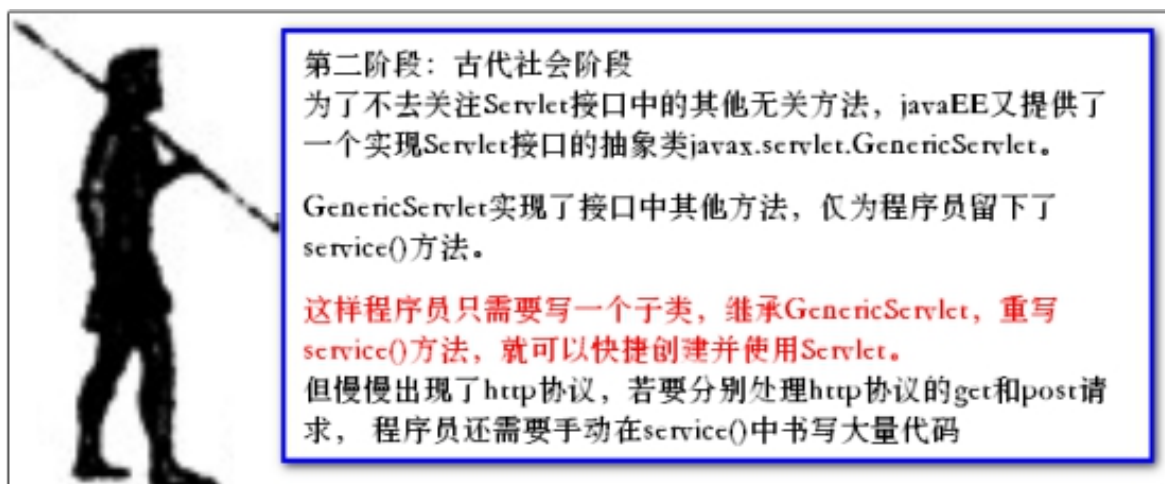
@Override
public String getServletInfo() {
    return null;
}

@Override
public void destroy() {

}
}

```

2.5.2 第二阶段



案例演示:

```

package com.ujiuye.servlet;


import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.io.IOException;

public class AncientSocialStageServlet extends GenericServlet {
    @Override
    public void service(ServletRequest request, ServletResponse response) throws
        ServletException, IOException {

    }
}

```

2.5.3 第三阶段



第三阶段：现代社会阶段

为了能够自动分开处理http协议的get和post请求，javaEE又提供了GenericServlet的子类，javax.servlet.http.HttpServlet

HttpServlet仍是抽象类，但将service()方法进行了重写，并且判断请求方式是get还是post。

若为get请求，调用doGet()方法；
若为post请求，调用doPost()方法；

程序员仅需书写一个子类，继承HttpServlet，重写doGet和doPost方法，就可以轻松写出一个能够处理http协议的强大Servlet。

案例演示:

```
package com.ujiuye.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class ModernityStage extends HttpServlet {

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

    }

}
```

2.6 注解使用

2.6.1 介绍

Java 注解 (Annotation) 又称 Java 标注，是 JDK1.5 引入的新特性。其主要作用是简化复杂的编码工作。Java 语言中的类、方法、变量、参数和包等都可以被标注，被标注的元素就具有了一些特定的功能。Servlet的注解 (@WebServlet) 是Servlet3.0及以上版本支持的，主要目的是简化web.xml配置。

2.6.2 实现

```
package com.ujiuye.servlet;

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;

/*此注解配置相当于web.xml文件中url-pattern设置的访问路径*/
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        System.out.println("注解版Servlet更加简明");
    }
}
```

2.6.3 注意事项

1. 一个Servlet的配置可以使用注解，也可以使用web.xml文件，但二者不能共存，只能任选其一，否则会报错。
2. 为什么可以执行servlet?
Servlet被实例化了
3. 谁实例化的servlet?
tomcat(servlet容器)实例化的Servlet
4. servlet什么时候实例化的?
第一次访问servlet的时候实例化的
5. Servlet被实例化了几次?
只被实例化一次
6. 访问路径 一个servlet可以指定多少个访问路径?
可以指定多个访问路径

2.7 多元化路径

问题：一个Servlet是否可以配置多个访问路径？ 答案是可以的
url-pattern配置方式共有三种：

1. 完全路径匹配(一层路径/指定多层路径)：以 / 开始
例如： /hello , /hello1, /aa/hello2
2. 目录匹配(多种路径都可以访问 排除已经指定的路径)：以 / 开始 需要以 * 结束。
例如： /* (所有) , /aaa/* (aaa目录下的所有) , /aaa/bbb/*
3. 扩展名匹配(后缀名)：不能以 / 开始 以 * 开始的。
例如： *.do , *.action 注意：错误的写法：/*.do

案例演示

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    version="3.0">
```

```
<!--注册servlet-->
<servlet>
    <servlet-name>hello</servlet-name><!--servlet名称-->
    <servlet-class>com.ujiuye.servlet.HelloServlet</servlet-class><!--全路径-->
</servlet>
<!--映射路径/虚拟路径-->
<servlet-mapping>
    <servlet-name>hello</servlet-name><!--servlet名称-->
    <url-pattern>/hello</url-pattern><!--访问路径，以 / 开头-->
    <!-- 目录匹配 -->
    <url-pattern>/aa/bb/cc</url-pattern>
    <url-pattern>/aa/bb/cc/dd/*</url-pattern>
    <!-- 扩展名匹配 -->
    <url-pattern>*.do</url-pattern>
    <url-pattern>*.action</url-pattern>
</servlet-mapping>
</web-app>
```

第3节：Servlet处理请求

3.1 介绍

Servlet处理请求是由核心方法中的request参数处理，处理响应是由核心方法的response参数处理。
request：定义一个向servlet提供客户端请求信息的对象。servlet容器创建一个ServletRequest对象，并将其作为参数传递给servlet的服务方法。

3.2 作用

接收客户端的请求，获取请求中的信息。除了可以获取请求中携带的数据之外，还可以获取比如主机地址、端口、请求方式、项目名称等一系列信息。

请求分类：

请求行、请求头、请求体。

3.3 体系

ServletRequest 接口

↑ 继承

HttpServletRequest 子接口

↑ 实现

org.apache.catalina.connector.RequestFacade 实现类

3.4 使用

3.4.1 获取请求行数据

```
String scheme = request.getScheme();
System.out.println("协议名: " + scheme);
String serverName = request.getServerName();
System.out.println("服务器: " + serverName);
int serverPort = request.getServerPort();
System.out.println("端口号: " + serverPort);
String contextPath = request.getContextPath();
System.out.println("项目名: " + contextPath);
System.out.println(scheme + "://" + serverName + ":" + serverPort +
contextPath);
String method = request.getMethod();
System.out.println("请求方式: " + method);
String queryString = request.getQueryString();
System.out.println("参数字符串: " + queryString);
String requestURI = request.getRequestURI();
System.out.println("项目名+具体地址: " + requestURI);
StringBuffer requestURL = request.getRequestURL();
System.out.println("获取请求的URL: " + requestURL);
String servletPath = request.getServletPath();
System.out.println("获取请求的URL: " + servletPath);
```

3.4.2 获取请求头数据

```
// 通过请求头关键字获取值
String header = request.getHeader("host");
System.out.println(header);
System.out.println("=====");
// 读取所有请求头信息
Enumeration<String> headerNames = request.getHeaderNames();
while (headerNames.hasMoreElements()) {
    String key = (String) headerNames.nextElement();
    System.out.println(key + ":" + request.getHeader(key));
}
```

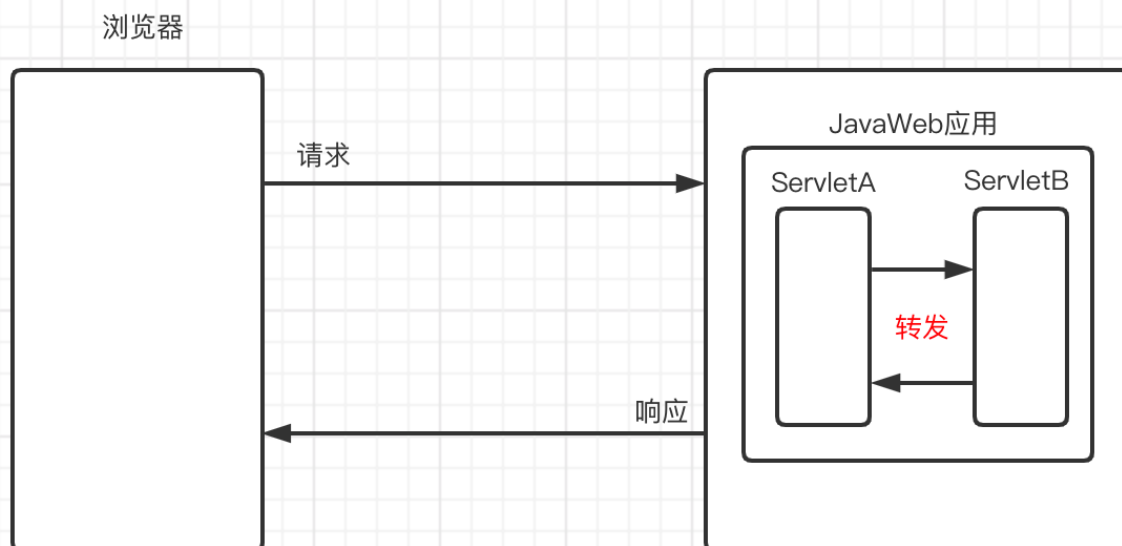
3.4.3 获取请求正文数据

- ① get方式提交 将数据拼接在url后面
`http://IP地址:端口号/项目名/文件名?name=xiaoming&password=123`
- ② post方式提交
使用form表单
- ③ 常用方法
 1. `request.getParameter("name值")`; 获取一个name值对应单个数据
 2. `request.getParameterValues("name值")`; 获取一个name值对应多个数据
 3. `request.getParameterNames()`; 获取所有的name值

- 4. `request.getParameterMap()`; 获取所有信息 放入map中
- ④ 将form数据封装到实体类对象中
 - BeanUtils 工具类
 - `BeanUtils.populate(bean, map);`
- ⑤ 其他方法
 - 1. `getContextPath()` ; 获取当前访问的项目名
 - 2. `getCookies()`; 获取浏览器请求时携带的cookie
 - 3. `getMethod()` ; 获取请求方式
 - 4. `getSession()`; 获取session对象
 - 5. `request.setCharacterEncoding("UTF-8")`; 设置请求编码

3.4.4 转发

- ① 理解
 - 一种在服务器内部资源跳转的方式。浏览器请求ServletA资源时, ServletA资源不足或者没有资源, ServletA请求其他的资源然后响应给浏览器, 这个过程叫转发;
- ② 实现
 - `request.getRequestDispatcher("转发路径").forward/include(request, response);`
- ③ 场景
 - 借钱:
 - 1. 全部直接借
 - 2. 一部分自己的钱 一部分找别人借
 - 3. 全部找别人借
- ④ 特点:
 - 1. 转发发生在服务器内部
 - 2. 转发的过程中url地址不变 浏览器不知道
 - 3. 转发可以访问到 WEB-INF 中的资源
 - 4. 转发的路径是当前项目下 因此转发不可以访问项目以外的资源
 - 5. 转发的路径一般是相对路径
 - 6. 转发发生一次请求中
- ⑤ 作用
 - 实现Servlet与页面的跳转



第4节：Servlet处理响应

4.1 介绍

`response`：定义一个对象来帮助servlet向客户端发送响应。servlet容器创建一个`ServletResponse`对象，并将其作为参数传递给servlet的服务方法。

4.2 作用

针对页面发送的请求做出数据响应，向页面输出信息，包括文本、图片、视频等。

响应分类：

响应行、响应头、响应体。

4.3 体系

`ServletResponse` 接口

↑ 继承

`HttpServletResponse` 子接口

↑ 实现

`org.apache.catalina.connector.ResponseFacade` 实现类

4.4 使用

4.4.1 设置响应行

1. 设置响应行：响应行中包含协议和状态码

可以通过`response.sendError(sc, msg)`来设置状态信息，但一般不会手动设置，仅用来做测试。

4.4.2 设置响应头

设置响应头信息可以通过以下两种方法：

```
response.setHeader("Content-Type", "text/html;charset=utf-8");
```

```
response.addHeader("Content-Type", "text/html;charset=utf-8");
```

二者的区别：

`response.setHeader(String name, String value)`；一个关键字对应一个值，如果设置了多个值，则会覆盖。

`response.addHeader(String name, String value)`；一个关键字可以对应多个值
在实际开发中，同一个响应头信息只会对应一个值，所以在使用时一般没什么区别。

4.4.3 设置响应体

响应的数据就是响应体。响应对象`response`在返回数据、响应数据的时候，会将一些HTML、text、流数据等信息通过响应主体返回给页面，而响应体绝大多数都是文本类型。

响应数据需要通过流来进行数据传输，而`response`自带的流有两个：

`response.getWriter()` ==> `PrintWriter` 输出文本信息

`response.getOutputStream()` ==> `ServletOutputStream` 输出字节信息，比如图片、音频、视频

需要注意：

这两个流不能同时存在。

4.4.4 重定向

① 理解

重定向是客户端行为，当客户端浏览器向`AServlet`发送一个请求，经过处理后向客户端做出响应，这个响应就是向服务器再次发送新的请求，去请求`BServlet`，而这个新请求的地址将为浏览器做出第二次响应，此时浏览器地址栏会发生改变，由于一次请求/响应结束后，`request`对象会自动销毁，所以两次请求的`request`对象并非同一个，所以两次请求域中的数据信息不会共享。由此可见，重定向是做了两次请求和响应。

② 实现

```
response.sendRedirect("重定向地址");
```

③ 场景

借钱：

不借，但告诉你谁有钱并指定地址

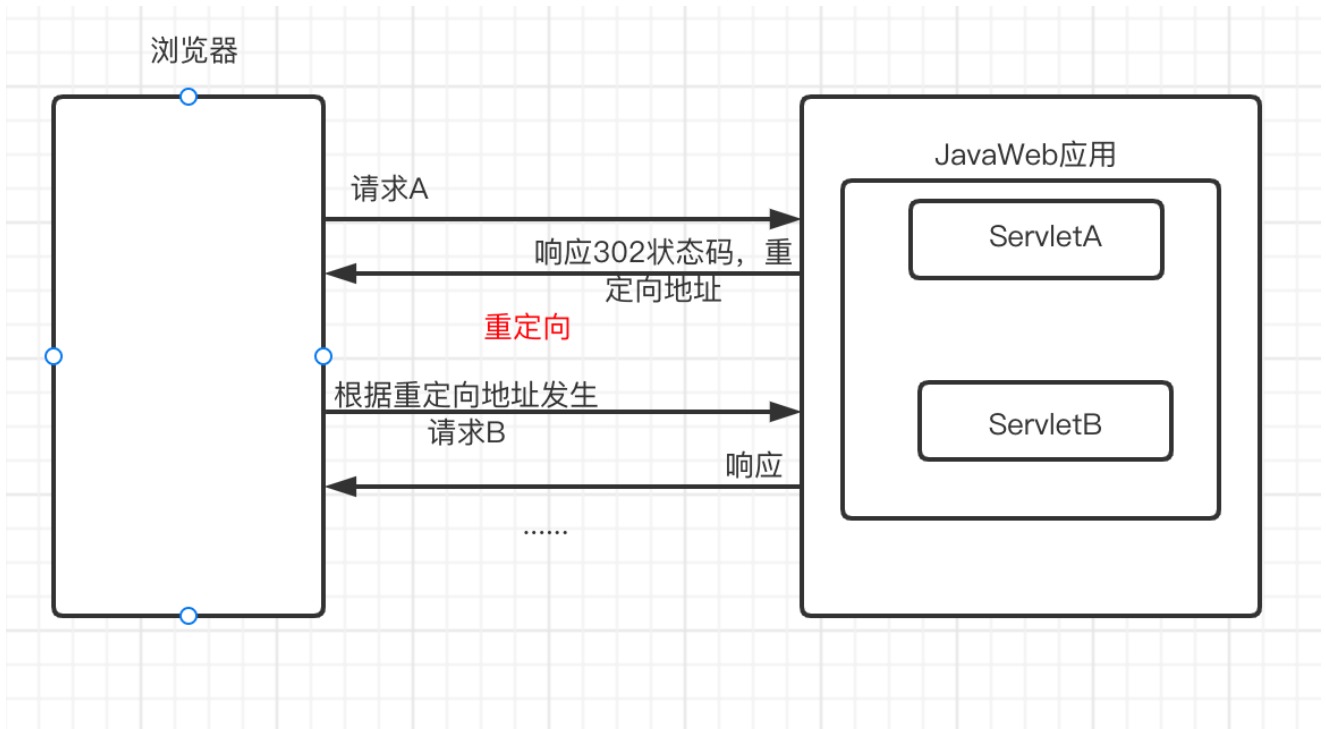
④ 特点

1. 重定向发生在浏览器和服务器之间
2. 重定向的过程url地址发生变化
3. 重定向发生至少两次请求中
4. 重定向访问不到WEB-INF的资源

5. 重定向可以访问当前应用外的资源
6. 重定向一般使用绝对路径

⑤ 作用

实现Servlet与页面的跳转



第5节：域对象

5.1 理解

可以共享的对象 提供给servlet之间数据共享 作用于服务器

5.2 分类

1. ServletContext 最大的共享域 作用范围整个项目
2. session 作用范围一次会话
3. request 作用范围一次请求

5.3 使用

1. 给共享域中放值 `setAttribute("string", "object");`
2. 获取共享域的值 `getAttribute("string");`
3. 删除共享域的值 `removeAttribute("string");`