

关于直播

1. 什么时间直播？

- 晚上8:00到10:00

2. 每周直播几天？

- 3天（周一、周三、周五）
- 本周比较特殊：周四周五周六三天直播，从下周开始就是一三五直播。

3. 直播什么内容？

- 从JavaWEB开始。（Servlet为核心，从Servlet开始学习。）
- JSP（JSP使用较少了，但是还有用，所以时间上少一些。快速地学习一下JSP。）
- AJAX（异步通信技术。）
- jQuery（JavaScript库。）
- MyBatis
- Spring
- SpringMVC
- SpringBoot
- SpringCloud
-

需要提前准备了哪些技术，接下来的课才能听懂？

- JavaSE (Java语言的标准版, Java提供的最基本的类库)
 - Java的开发环境搭建
 - Java的基础语法
 - Java的面向对象
 - 数组
 - 常用类
 - 异常
 - 集合
 - 多线程
 - IO流
 - 反射机制
 - 注解Annotation
 -
- MySQL (数据库)
 - 最基本的要求是：能够编写增删改查等简单的SQL语句即可。

- JDBC (Java语言链接数据库)
 - 这是一套Java语言链接数据库的接口。
- WEB前端 (会一些)
 - HTML (网页)
 - CSS (层叠样式表语言, 修饰HTML)
 - JavaScript (一种脚本语言, 运行在浏览器当中, 让浏览器中的元素可以增删改。让网页产生更强的交互效果)
- WEB后端
 - Servlet (Server Applet: 服务器端的Java小程序)
 - JSP
 - AJAX
 - jQuery
 - MyBatis
 - Spring
 - SpringMVC
 - SpringBoot
 - SpringCloud
 -

Typora软件介绍

- Markdown文本编辑器（可以编辑xxx.md文件）

```
1 public class Test{
2     public static void main(String[]
   args){
3         System.out.println("Test
   code!");
4     }
5 }
```

Servlet

关于系统架构

1. 系统架构包括什么形式？

- C/S架构
- B/S架构

2. C/S架构？

- Client / Server（客户端 / 服务器）
- C/S架构的软件或者说系统有哪些呢？
 - QQ（先去腾讯官网下载一个QQ软件，几十MB，然后把这个客户端软件安装上去，然后输入QQ号以及密码，登录之后，就可以和你的朋友聊天了，就可以使用这个软件了。）

- C/S架构的特点：需要安装特定的客户端软件。
- C/S架构的系统优点和缺点分别是什么？
 - 优点：
 - 速度快（软件中的数据大部分都是集成到客户端软件当中的，很少量的数据从服务器端传送过来，所以C/S结构的系统速度快）
 - 体验好（速度又快，界面又酷炫，当然体验好了。）
 - 界面酷炫（专门的语言去实现界面的，更加灵活。）
 - 服务器压力小（因为大量的数据都是集成在客户端软件当中，所以服务器只需要传送很少的数据量，当然服务器压力小。）
 - 安全（因为大量的数据是集成在客户端软件当中的，并且客户端有很多个，服务器虽然只有一个，就算服务器那边地震了，火灾了，服务器受损了，问题也不大，因为大量的数据在多个客户端上有缓存，有存储，所以从这个方面来说，C/S结构的系统比较安全。）
 -
 - 缺点：

- 升级维护比较差劲。（升级维护比较麻烦。成本比较高。每一个客户端软件都需要升级。有一些软件不是那么容易安装的。）

3. B/S架构?

- B/S (Browser / Server, 浏览器 / 服务器)
- <http://www.baidu.com>
- <http://www.jd.com>
- <http://www.126.com>
- B/S结构的系统是不是一个特殊的C/S系统?
 - 实际上B/S结构的系统还是一个C/S，只不过这个C比较特殊，这个Client是一个固定不变浏览器软件。
- B/S结构的系统优点和缺点是：
 - 优点：
 - 升级维护方便，成本比较低。（只需要升级服务器端即可。）
 - 不需要安装特定的客户端软件，用户操作极其方便。只需要打开浏览器，输入网址即可。
 - 缺点：

- 速度慢（不是因为带宽低的问题，是因为所有的数据都是在服务器上，用户发送的每一个请求都是需要服务器全身心的响应数据，所以B/S结构的系统在网络中传送的数据量比较大。）
- 体验差（界面不是那么酷炫，因为浏览器只支持三个语言HTML CSS JavaScript。在加上速度慢。）
- 不安全（所有的数据都在服务器上，只要服务器发生火灾，地震等不可抗力，最终数据全部丢失。）
-

4. C/S和B/S结构的系统，哪个好，哪个不好？

- 这个问题问的没有水平。并不是哪个好，哪个不好。不同结构的系统在不同的业务场景下有不同的适用场景。
- 娱乐性软件建议使用？
 - C/S 结构
- 公司内部使用的一些业务软件建议使用？
 - 公司内部使用的系统，需要维护成本低。
 - 公司内部使用的系统，不需要很酷炫。
 - 公司内部使用的企业级系统主要是能够进行数据的维护即可。

- B/S 结构。

5. 注意了：开发B/S结构的系统，其实就是开发网站，其实就是开发一个WEB系统。

○ 开发一个WEB系统你需要会哪些技术？

- WEB前端（运行在浏览器上的程序。）

- HTML

- CSS

- JavaScript

- WEB后端（WEB服务器端的程序。）

- Java可以（Java做WEB开发我们称为JavaWEB开发。JavaWEB开发最核心的规范：Servlet【Server Applet服务器端的Java小程序。】）

- C语言也可以

- C++也可以

- Python也行

- PHP也可以

-

6. JavaEE是什么？

○ Java包括三大块：

- JavaSE

- Java标准版（一套类库：别人写好的一套类库，只不过这个类库是标准类库，走EE，或者走ME，这个SE一定是基础，先学。）
- JavaEE（WEB方向，WEB系统。）
 - Java企业版（也是一套类库：也是别人写好的一套类库，只不过这套类库可以帮助我们完成企业级项目的开发，专门为企业内部提供解决方案的一套（多套）类库。）
 - 别人写好的，你用就行了，用它可以开发企业级项目。
 - 可以开发web系统。
 - Java比较火爆的就是这个JavaEE方向。
- JavaME
 - Java微型版（还是一套类库，只不过这套类库帮助我们进行电子微型设备内核程序的开发）
 - 机顶盒内核程序，吸尘器内核程序，电冰箱内核程序，电饭煲内核程序。。。。。
- JavaEE实际上包括很多种规范，13种规范，其中Servlet就是JavaEE规范之一。学Servlet还是Java语言。

B/S结构的系统通信原理（没有涉及到Java小程序）

- WEB系统的访问过程
 - 第一步：打开浏览器
 - 第二步：找到地址栏
 - 第三步：输入一个合法的网址
 - 第四步：回车
 - 第五步：在浏览器上会展示响应的结果。
- 关于域名：
 - <https://www.baidu.com/>（网址）
 - www.baidu.com 是一个域名
 - 在浏览器地址栏上输入域名，回车之后，域名解析器会将域名解析出来一个具体的IP地址和端口号等。
 - 解析结果也许是：<http://110.242.68.3:80/index.html>
- IP地址是啥？
 - 计算机在网络其中的一个身份证号。在同一个网络当中，IP地址是唯一的。
 - A计算机要想和B计算机通信，首先你需要知道B计算机的IP地址，有了IP地址才能建立连接。

- 端口号是啥？
 - 一个端口代表一个软件（一个端口代表一个应用，一个端口仅代表一个服务）。
 - 一个计算机当中有很多软件，每一个软件启动之后都有一个端口号。
 - 在同一个计算机上，端口号具有唯一性。
- 一个WEB系统的通信原理？通信步骤：
 - 第一步：用户输入网址（URL）
 - 第二步：域名解析器进行域名解析：<http://110.242.68.3:80/index.html>
 - 第三步：浏览器软件在网络中搜索110.242.68.3这一台主机，直到找到这台主机。
 - 第四步：定位110.242.68.3这台主机上的服务器软件，因为是80端口，可以很轻松的定位到80端口对应的服务器软件。
 - 第五步：80端口对应的服务器软件得知浏览器想要的资源名是：index.html
 - 第六步：服务器软件找到index.html文件，并且将index.html文件中的内容直接输出响应到浏览器上。
 - 第七步：浏览器接收到来自服务器的代码（HTML CSS JS）

- 第八步：浏览器渲染，执行HTML CSS JS代码，展示效果。
- 什么是URL?
 - 统一资源定位符 (<http://www.baidu.com>)
- 什么是请求，什么是响应？
 - 请求和响应实际上说的是数据的流向不同。
 - 从Browser端发送数据到Server端，我们称为请求。英语单词：request
 - 从Server端向浏览器Browser端发送数据，我们称为响应。英语单词：response
 - B --> S （请求request）
 - S --> B （响应response）

关于WEB服务器软件

- WEB服务器软件都有哪些呢？（这些软件都是提前开发好的。）
 - Tomcat （WEB服务器）
 - jetty （WEB服务器）
 - JBOSS （应用服务器）
 - WebLogic （应用服务器）
 - WebSphere （应用服务器）
- 应用服务器和WEB服务器的关系？

- 应用服务器实现了JavaEE的所有规范。(JavaEE有13个不同的规范。)
- WEB服务器只实现了JavaEE中的Servlet + JSP两个核心的规范。
- 通过这个讲解说明了：应用服务器是包含WEB服务器的。
- 用过JBoss服务器的同学应该很清楚，JBoss中内嵌了一个Tomcat服务器。
- Tomcat下载
 - apache官网地址：<https://www.apache.org/>
 - tomcat官网地址：<https://tomcat.apache.org>
 - tomcat开源免费的轻量级WEB服务器。
 - tomcat还有另外一个名字：catalina (catalina是美国的一个岛屿，风景秀丽，据说作者是在这个风景秀丽的小岛上开发了一个轻量级的WEB服务器，体积小，运行速度快，因此tomcat又被称为catalina)
 - tomcat的logo是一只公猫 (寓意表示Tomcat服务器是轻巧的，小巧的，果然，体积小，运行速度快，只实现了Servlet+JSP规范)
 - tomcat是java语言写的。
 - tomcat服务器要想运行，必须先有jre (Java的运行环境)

- **Tomcat服务器要想运行，需要先有jre，所以要先安装JDK，配置java运行环境。**
 - JAVA_HOME=C:\Program Files\Java\jdk-17.0.1
 - PATH=%JAVA_HOME%\bin
 - 目前JAVA_HOME没有配置，思考一个问题，这样行不行呢？目前只运行java程序是没问题的。真的没问题吗？
- Tomcat服务器的安装：
 - 绿色版本的安装很简单，直接zip包解压即可。解压就是安装。
 - 我有一个好习惯，在C盘的根目录下新建一个dev目录，java开发所有相关的工具都安装到dev目录下，这样比较方便管理。（你随意）
 - 启动Tomcat
 - bin目录下有一个文件：startup.bat,通过它可以启动Tomcat服务器。
 - xxx.bat文件是个什么文件？bat文件是windows操作系统专用的，bat文件是批处理文件，这种文件中可以编写大量的windows的dos命令，然后执行bat文件就相当于批量的执行dos命令。

- startup.sh, 这个文件在windows当中无法执行, 在Linux环境当中可以使用。在Linux环境下能够执行的是shell命令, 大量的shell命令编写在shell文件当中, 然后执行这个shell文件可以批量的执行shell命令。
- tomcat服务器提供了bat和sh文件, 说明了这个tomcat服务器的通用性。
- 分析startup.bat文件得出, 执行这个命令, 实际上最后是执行: catalina.bat文件。
- catalina.bat文件中有这样一行配置:
MAINCLASS=org.apache.catalina.startup.Bootstrap (这个类就是main方法所在的类。)
- tomcat服务器就是Java语言写的, 既然是java语言写的, 那么启动Tomcat服务器就是执行main方法。
- 我们尝试打开dos命令窗口, 在dos命令窗口中输入startup.bat来启动tomcat服务器。
- 启动Tomcat服务器只配置path对应的bin目录是不行的。有两个环境变量需要配置:
 - JAVA_HOME=JDK的根
 - CATALINA_HOME=Tomcat服务器的根
- 关于Tomcat服务器的目录

- bin：这个目录是Tomcat服务器的命令文件存放的目录，比如：启动Tomcat，关闭Tomcat等。
- conf：这个目录是Tomcat服务器的配置文件存放目录。（server.xml文件中可以配置端口号，默认Tomcat端口是8080）
- lib：这个目录是Tomcat服务器的核心程序目录，因为Tomcat服务器是Java语言编写的，这里的jar包里面都是class文件。
- logs: Tomcat服务器的日志目录，Tomcat服务器启动等信息都会在这个目录下生成日志文件。
- temp: Tomcat服务器的临时目录。存储临时文件。
- **webapps：这个目录当中就是用来存放大量的webapp（web application：web应用）**
- work：这个目录是用来存放JSP文件翻译之后的java文件以及编译之后的class文件。
- **配置Tomcat服务器需要哪些环境变量？**
 - JAVA_HOME=JDK的根
 - CATALINA_HOME=Tomcat服务器的根
 - PATH=%JAVA_HOME%\bin;%CATALINA_HOME%\bin
- **启动Tomcat： startup**

- **关闭Tomcat: stop (shutdown.bat文件重命名为stop.bat, 为什么? 原因是shutdown命令和windows中的关机命令冲突。所以修改一下。)**
- 怎么测试Tomcat服务器有没有启动成功呢?
 - 打开浏览器, 在浏览器的地址栏上输入URL即可:
 - <http://ip地址:端口号>
 - ip地址是什么? 端口号我知道, 是8080
 - 本机的IP地址是: 127.0.0.1, 或者是localhost, 都行。

实现一个最基本的web应用 (这个web应用中沒有java小程序)

- 第一步: 找到CATALINA_HOME\webapps目录
 - 因为所有的webapp要放到webapps目录下。(没有为什么, 这是Tomcat服务器的要求。如果不放到这里, Tomcat服务器找不到你的应用。)
- 第二步: 在CATALINA_HOME\webapps目录下新建一个子目录, 起名: oa
 - 这个目录名oa就是你这个webapp的名字。
- 第三步: 在oa目录下新建资源文件, 例如: index.html
 - 编写index.html文件的内容。

- 第四步：启动Tomcat服务器
- 第五步：打开浏览器，在浏览器地址栏上输入这样的URL：
- <http://127.0.0.1:8080/oa/index.html>
- 思考一个问题：
 - 我们在浏览器上直接输入一个URL，然后回车。这个动作和超链接一样吗？既然是一样的，我们完全可以使用超链接。

```
1 <!--注意以下的路径，以/开始，带项目名，是一个绝对路径。不需要添加：
   http://127.0.0.1:8080-->
2 <a href="/oa/login.html">user
  login2</a>
3
4 <!--多个层级也没有关系，正常访问即可。-->
5 <!--注意：我们目前前端上的路径都以“/”开始的，都是加项目名的。-->
6 <a href="/oa/test/debug/d.html">d
  page</a>
```

- <http://127.0.0.1:8080/oa/userList.html>
 - 访问这个地址，可以展示一个用户列表页面。但是这个用户列表页面是写死在HTML文件当中的。这种资源我们称为静态资源。怎么能变成动态资源。显然需要连接数据库。

- 连接数据库需要JDBC程序，也就是说需要编写Java程序连接数据库，数据库中有多少条记录，页面上就显示多少条记录，这种技术被称为动态网页技术。（动态网页技术并不是说页面中有flash动画。动态网页技术是说页面中的数据是动态的，根据数据库中数据的变化而变化。）

对于一个动态的web应用来说，一个请求和响应的过程有多少个角色参与，角色和角色之间有多少个协议

BS结构系统的通信原理2

- 有哪些角色（在整个BS结构的系统当中，有哪些人参与进去了）
 - 浏览器软件的开发团队（浏览器软件太多了：谷歌浏览器、火狐浏览器、IE浏览器....）
 - WEB Server的开发团队（WEB Server这个软件也是太多了：Tomcat、Jetty、WebLogic、JBOSS、WebSphere....）
 - DB Server的开发团队（DB Server这个软件也是太多了：Oracle、MySQL.....）
 - webapp的开发团队（WEB应用是我们做为JavaWEB程序员开发的）

- 角色和角色之间需要遵守哪些规范，哪些协议
 - webapp的开发团队 和 WEB Server的开发团队之间有一套规范: JavaEE规范之一Servlet规范。
 - Servlet规范的作用是什么?
 - WEB Server 和 webapp解耦合。
 - Browser 和 WebServer之间有一套传输协议: HTTP协议。（超文本传输协议。）
 - webapp开发团队 和 DB Server的开发团队之间有一套规范: JDBC规范。

BS结构系统的角色和协议

- Servlet规范是一个什么规范?
 - 遵循Servlet规范的webapp，这个webapp就可以放在不同的WEB服务器中运行。（因为这个webapp是遵循Servlet规范的。）
 - Servlet规范包括什么呢?
 - 规范了哪些接口
 - 规范了哪些类
 - 规范了一个web应用中应该有哪些配置文件
 - 规范了一个web应用中配置文件的名字
 - 规范了一个web应用中配置文件存放的路径
 - 规范了一个web应用中配置文件的内容

- 规范了一个合法有效的web应用它的目录结构应该是怎样的。
-

开发一个带有Servlet (Java小程序) 的webapp (重点)

- 开发步骤是怎样的？
 - 第一步：在webapps目录下新建一个目录，起名crm（这个crm就是webapp的名字）。当然，也可以是其它项目，比如银行项目，可以创建一个目录bank，办公系统可以创建一个oa。
 - 注意：crm就是这个webapp的根
 - 第二步：在项目的根下新建一个目录：WEB-INF
 - 注意：这个目录的名字是Servlet规范中规定的，必须全部大写，必须一模一样。必须的必须。
 - 第三步：在WEB-INF目录下新建一个目录：classes
 - 注意：这个目录的名字必须是全部小写的classes。这也是Servlet规范中规定的。另外这个目录下一定存放的是Java程序编译之后的class文件（这里存放的是字节码文件）。
 - 第四步：在WEB-INF目录下新建一个目录：lib

- 注意：这个目录不是必须的。但如果一个webapp需要第三方的jar包的话，这个jar包要放到这个lib目录下，这个目录的名字也不能随意编写，必须是全部小写的lib。例如java语言连接数据库需要数据库的驱动jar包。那么这个jar包就一定要放到lib目录下。这Servlet规范中规定的。
- 第五步：在WEB-INF目录下新建一个文件：
web.xml
 - 注意：这个文件是必须的，这个文件名必须叫做web.xml。这个文件必须放在这里。一个合法的webapp，web.xml文件是必须的，这个web.xml文件就是一个配置文件，在这个配置文件中描述了请求路径和Servlet类之间的对照关系。
 - 这个文件最好从其他的webapp中拷贝，最好别手写。没必要。复制粘贴

- ```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <web-app
 xmlns="https://jakarta.ee/xml/ns/jakartaee"
4
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5
 xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
6
 https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
7 version="5.0"
8 metadata-complete="true">
9
10
11 </web-app>
12
```

- 第六步：编写一个Java程序，这个小Java程序也不能随意开发，这个小Java程序必须实现Servlet接口。

- 这个Servlet接口不在JDK当中。（因为Servlet不是JavaSE了。Servlet属于JavaEE，是另外的一套类库。）

- Servlet接口（Servlet.class文件）是Oracle提供的。（最原始的是sun公司提供的。）
- Servlet接口是JavaEE的规范中的一员。
- Tomcat服务器实现了Servlet规范，所以Tomcat服务器也需要使用Servlet接口。  
Tomcat服务器中应该有这个接口，Tomcat服务器的CATALINA\_HOME\lib目录下有一个servlet-api.jar，解压这个servlet-api.jar之后，你会看到里面有一个Servlet.class文件。
- 重点：从JakartaEE9开始，Servlet接口的全名变了：jakarta.servlet.Servlet
- 注意：编写这个Java小程序的时候，java源代码你愿意在哪里就在哪里，位置无所谓，你只需要将java源代码编译之后的class文件放到classes目录下即可。

- ```
1 HelloServlet.java
2 package com.bjpowernode.servlet;
3 import javax.servlet.*; //注意我的版本不是tomcat10而是9，所以任然还是导入javax包而不是jakarta
4 import java.io.*;
5 public class HelloServlet
  implements Servlet{
6     //5个方法
7     public void init(ServletConfig config) throws ServletException{
```



```

8
9     }
10    public void
service(ServletRequest request,
ServletResponse response)
11        throws
ServletException, IOException{
12        System.out.println("hello
my first servlet!");
13    }
14    public void destroy(){
15        System.out.println("hello
my first servlet!");
16
17    }
18    public String getServletInfo()
{
19        return null;
20    }
21    public ServletConfig
getServletConfig(){
22        return null;
23    }
24 }

```

- 第七步：编译我们编写的HelloServlet
 - 重点：你怎么能让你的HelloServlet编译通过呢？配置环境变量CLASSPATH

CLASSPATH=.;C:\dev\apache-tomcat-10.0.12\lib\servlet-api.jar

- 思考问题：以上配置的CLASSPATH和Tomcat服务器运行有没有关系？
 - 没有任何关系，以上配置这个环境变量只是为了让你的HelloServlet能够正常编译生成class文件。
- 第八步：将以上编译之后的HelloServlet.class文件(带包)拷贝到WEB-INF\classes目录下。
- 第九步：在web.xml文件中编写配置信息，让“请求路径”和“Servlet类名”关联在一起。
 - 这一步用专业术语描述：在web.xml文件中注册Servlet类。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <web-app
  xmlns="https://jakarta.ee/xml/ns/jakartaee"
4
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5
   xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
```

```
6      https://jakarta.ee/xml/ns/jakarta
ee/web-app_5_0.xsd"
7      version="5.0"
8      metadata-complete="true">
9
10     <!--servlet描述信息-->
11     <!--任何一个servlet都对应一个
servlet-mapping -->
12     <servlet>
13         <servlet-
name>HelloServlet</servlet-name>
14         <!--这个位置必须是带有包名的全
限定类名-->
15         <servlet-
class>com.bjpowernode.servlet.HelloServlet</servlet-class>
16     </servlet>
17
18     <!--servlet映射信息-->
19     <servlet-mapping>
20         <!--这个也是随便的，不过这里写的
内容要和上面的一样。-->
21         <servlet-
name>HelloServlet</servlet-name>
22         <!--这里需要一个路径-->
23         <!--这个路径唯一的要求是必须以
/ 开始-->
24         <!--当前这个路径可以随便写-->
```

```
25         <url-  
        pattern>/helloservlet</url-  
        pattern>  
26     </servlet-mapping>  
27  
28 </web-app>  
29
```

- 第十步：启动Tomcat服务器
- 第十一步：打开浏览器，在浏览器地址栏上输入一个url，这个URL必须是：
 - <http://127.0.0.1:8080/crm/helloservlet> //注意是http而不是https
 - 非常重要的一件事：浏览器上的请求路径不能随便写，这个请求路径必须和web.xml文件中的url-pattern一致。
 - **注意：浏览器上的请求路径和web.xml文件中的url-pattern的唯一区别就是：浏览器上的请求路径带项目名：/crm**
- 浏览器上编写的路径太复杂，可以使用超链接(Hello Servlet!)。 (**非常重要：html页面只能放到WEB-INF目录外面。**)

- 以后不需要我们编写main方法了。tomcat服务器负责调用main方法，Tomcat服务器启动的时候执行的就是main方法。我们javaweb程序员只需要编写Servlet接口的实现类，然后将其注册到web.xml文件中，即可。
- 总结一下：一个合法的webapp目录结构应该是怎样的？

```
1 webapproot
2     |-----WEB-INF
3         |-----classes(存放字节
   码)
4         |-----lib(第三方jar包)
5         |-----web.xml(注册
   Servlet)
6     |-----html
7     |-----css
8     |-----javascript
9     |-----image
10    ....
```

- 浏览器发送请求，到最终服务器调用Servlet中的方法，是怎样的一个过程？（以下这个过程描述的很粗糙。其中还有很多步骤我省略了。）
 - 用户输入URL，或者直接点击超链接：<http://127.0.0.1:8080/crm/helloservlet>
 - 然后Tomcat服务器接收到请求，截取路径：/crm/helloservlet

- Tomcat服务器找到crm项目
- Tomcat服务器在web.xml文件中查找/crm/helloservlet 对应的Servlet是：
com.bjpowernode.servlet.HelloServlet
- Tomcat服务器通过反射机制，创建
com.bjpowernode.servlet.HelloServlet的对象。
- Tomcat服务器调用
com.bjpowernode.servlet.HelloServlet对象的
service方法。

关于JavaEE的版本

- JavaEE目前最高版本是 JavaEE8
- JavaEE被Oracle捐献给， Oracle将JavaEE规范捐献给 Apache了。
- Apache把JavaEE换名了，以后不叫JavaEE了，以后叫做 Jakarta EE。
- 以后没有JavaEE了。以后都叫做Jakarta EE。
- JavaEE8版本升级之后的"JavaEE 9"，不再是"JavaEE9"这个名字了，叫做JakartaEE9
- JavaEE8的时候对应的Servlet类名是：
javax.servlet.Servlet

- JakartaEE9的时候对应的Servlet类名是：
jakarta.servlet.Servlet （包名都换了）
- 如果你之前的项目还是在使用javax.servlet.Servlet，那么你的项目无法直接部署到Tomcat10+版本上。你只能部署到Tomcat9-版本上。在Tomcat9以及Tomcat9之前的版本中还是能够识别javax.servlet这个包。

解决Tomcat服务器在DOS命令窗口中的乱码问题（控制台乱码）

将CATALINA_HOME/conf/logging.properties文件中的内容修改如下：

```
java.util.logging.ConsoleHandler.encoding = GBK
```

向浏览器响应一段HTML代码

```
1 public void service(ServletRequest
   request, ServletResponse response){
2
   response.setContentType("text/html");//
   要在获取输出流之前就要设置好内容Type
3   PrintWriter out =
   response.getWriter();
4   out.print("<h1>hello servlet!</h1>");
5 }
```

在Servlet中连接数据库，怎么做？

- Servlet是Java程序，所以在Servlet中完全可以编写JDBC代码连接数据库。
- 在一个webapp中去连接数据库，需要将驱动jar包放到WEB-INF/lib目录下。（com.mysql.cj.jdbc.Driver这个类就在驱动jar包当中。）
- com.mysql.jdbc.Driver在新版本中已过时,需要替换成com.mysql.cj.jdbc.Driver

在集成开发环境当中开发Servlet程序

- 集成开发工具很多，其中目前使用比较多的是：
 - IntelliJ IDEA（这个居多，IDEA在提示功能方面要强于Eclipse，也就是说IDEA使用起来比Eclipse更加智能，更好用。JetBrain公司开发的。收费的。）
 - Eclipse（这个少一些），Eclipse目前还是有团队使用，只不过处于减少的趋势，自己从事工作之后，可能会遇到。Eclipse是IBM团队开发的。Eclipse寓意是“日食”。“日食”表示将太阳吃掉。太阳是SUN。IBM团队开发Eclipse的寓意是吞并SUN公司，但是2009年的时候SUN公司被Oracle公司并购了。IBM并没有成功并购SUN公司。
- 使用IDEA集成开发工具开发Servlet
 - 第一步：New Project（我比较习惯先创建一个Empty Project【空工程】，然后在空工程下新建Module【模块】，这不是必须的，只是一种习惯，你可以直接新建非空的Project），这个Empty Project起名为：javaweb（不是必须的，只是一个名字而已。一般情况下新建的Project的名字最好和目录的名字一致。）
 - 第二步：新建模块（File --> new --> Module...）

- 这里新建的是一个普通的JavaSE模块（这里先不要新建Java Enterprise模块）
- 这个Module自动会被放在javaweb的project下面。
- 这个Module起名：servlet01
- 第三步：让Module变成JavaEE的模块。（让Module变成webapp的模块。符合webapp规范。符合Servlet规范的Module）
 - 在Module上点击右键：Add Framework Support...（添加框架支持）
 - 在弹出的窗口中，选择Web Application（选择的是webapp的支持）
 - 选择了这个webapp的支持之后，IDEA会自动给你生成一个符合Servlet规范的webpp目录结构。
 - **重点，需要注意的：在IDEA工具中根据Web Application模板生成的目录中有一个web目录，这个目录就代表webapp的根**
- 第四步（非必须）：根据Web Application生成的资源中有index.jsp文件，这里我选择删除这个index.jsp文件。
- 第五步：编写Servlet（StudentServlet）
 - `class StudentServlet implements Servlet`

- 这个时候发现Servlet.class文件没有。怎么办？
将CATALINA_HOME/lib/servlet-api.jar和jsp-api.jar添加到classpath当中（这里的classpath说的是IDEA的classpath）
 - File --> Project Structure --> Modules --> + 加号 --> Add JARS....
- 实现jakarta.servlet.Servlet接口中的5个方法。
- 第六步：在Servlet当中的service方法中编写业务代码（我们这里连接数据库了。）
- 第七步：在WEB-INF目录下新建了一个子目录：lib（这个目录名可不能随意，必须是全部小写的lib），并且将连接数据库的驱动jar包放到lib目录下。
- 第八步：在web.xml文件中完成StudentServlet类的注册。（请求路径和Servlet之间对应起来）

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
      http://xmlns.jcp.org/xml/ns/javaee/web
      -app_4_0.xsd"
5          version="4.0">
6
7      <servlet>
8          <servlet-
name>studentServlet</servlet-name>
9          <servlet-
class>com.bjpowernode.javaweb.servlet.
StudentServlet</servlet-class>
10      </servlet>
11      <servlet-mapping>
12          <servlet-
name>studentServlet</servlet-name>
13          <url-
pattern>/servlet/student</url-pattern>
14      </servlet-mapping>
15
16 </web-app>
```

- 第九步：给一个html页面，在HTML页面中编写一个超链接，用户点击这个超链接，发送请求，Tomcat执行后台的StudentServlet。
 - student.html

- 这个文件不能放到WEB-INF目录里面，只能放到WEB-INF目录外面。
- student.html文件的内容

- ```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>student page</title>
6 </head>
7 <body>
8 <!--这里的项目名是 /xmm ，无法动态
 获取，先写死-->
9 student
 list
10 </body>
11 </html>
```


- 第十步：让IDEA工具去关联Tomcat服务器。关联的过程当中将webapp部署到Tomcat服务器当中。
  - IDEA工具右上角，绿色小锤子右边有一个：Add Configuration
  - 左上角加号，点击Tomcat Server --> local
  - 在弹出的界面中设置服务器Server的参数（基本上不用动）

- 在当前窗口中有一个Deployment（点击这个用来部署webapp），继续点击加号，部署即可。
- 修改 Application context为：/xmm
- 第十一步：启动Tomcat服务器
  - 在右上角有绿色的箭头，或者绿色的小虫子，点击这个绿色的小虫子，可以采用debug的模式启动Tomcat服务器。
  - 我们开发中建议适用debug模式启动Tomcat
- 第十二步：打开浏览器，在浏览器地址栏上输入：  
<http://localhost:8080/xmm/student.html>

## Servlet对象的生命周期

---

- 什么是Servlet对象生命周期？
  - Servlet对象什么时候被创建。
  - Servlet对象什么时候被销毁。
  - Servlet对象创建了几个？
  - Servlet对象的生命周期表示：一个Servlet对象从出生在最后的死亡，整个过程是怎样的。
- 思考：Servlet对象是由谁来维护的？
  - Servlet对象的创建，对象上方法的调用，对象最终的销毁，Javaweb程序员是无权干预的。

- Servlet对象的生命周期是由Tomcat服务器（WEB Server）全权负责的。
- Tomcat服务器通常我们又称为：WEB容器。（这个叫法你要知道【WEB Container】）
- WEB容器来管理Servlet对象的死活。
- 思考：我们自己new的Servlet对象受WEB容器的管理吗？
  - 我们自己new的Servlet对象是不受WEB容器管理的。
  - WEB容器创建的Servlet对象，这些Servlet对象都会被放到一个集合当中（HashMap），只有放到这个HashMap集合中的Servlet才能够被WEB容器管理，自己new的Servlet对象不会被WEB容器管理。（自己new的Servlet对象不在容器当中）
  - web容器底层应该有一个HashMap这样的集合，在这个集合当中存储了Servlet对象和请求路径之间的关系
  -  WEB容器中的Map集合
- 研究：服务器在启动的Servlet对象有没有被创建出来（默认情况下）？
  - 在Servlet中提供一个无参数的构造方法，启动服务器的时候看看构造方法是否执行。
  - 经过测试得出结论：默认情况下，服务器在启动的时候Servlet对象并不会被实例化。

- 这个设计是合理的。用户没有发送请求之前，如果提前创建出来所有的Servlet对象，必然是耗费内存的，并且创建出来的Servlet如果一直没有用户访问，显然这个Servlet对象是一个废物，没必要先创建。
- 研究：怎么让服务器启动的时候创建Servlet对象呢？
  - 在servlet标签中添加子标签，在该子标签中填写整数，越小的整数优先级越高。

- ```
1 <servlet>
2     <servlet-name>aservlet</servlet-
  name>
3     <servlet-
  class>com.bjpowernode.javaweb.servlet
  .AServlet</servlet-class>
4     <load-on-startup>1</load-on-
  startup>
5 </servlet>
6 <servlet-mapping>
7     <servlet-name>aservlet</servlet-
  name>
8     <url-pattern>/a</url-pattern>
9 </servlet-mapping>
```

- Servlet对象生命周期
 - 默认情况下服务器启动的时候AServlet对象并没有被实例化

- 用户发送第一次请求的时候，控制台输出了以下内容：

```
1 AServlet无参数构造方法执行了
2 AServlet's init method execute!
3 AServlet's service method execute!
```

- 根据以上输出内容得出结论：
 - 用户在发送第一次请求的时候Servlet对象被实例化（AServlet的构造方法被执行了。并且执行的是无参数构造方法。）
 - AServlet对象被创建出来之后，Tomcat服务器马上调用了AServlet对象的init方法。（init方法在执行的时候，AServlet对象已经存在了。已经被创建出来了。）
 - 用户发送第一次请求的时候，init方法执行之后，Tomcat服务器马上调用AServlet对象的service方法。
- 用户继续发送第二次请求，控制台输出了以下内容：

```
1 AServlet's service method execute!
```

- 根据以上输出结果得知，用户在发送第二次，或者第三次，或者第四次请求的时候，Servlet对象并没有新建，还是使用之前创建好的Servlet对象，直接调用该Servlet对象的service方法，这说明：

- 第一：Servlet对象是单例的（单实例的。但是要注意：Servlet对象是单实例的，但是Servlet类并不符合单例模式。我们称之为假单例。之所以单例是因为Servlet对象的创建我们javaweb程序员管不着，这个对象的创建只能是Tomcat来说了算，Tomcat只创建了一个，所以导致了单例，但是属于假单例。真单例模式，构造方法是私有化的。）
- 第二：无参数构造方法、init方法只在第一次用户发送请求的时候执行。也就是说无参数构造方法只执行一次。init方法也只被Tomcat服务器调用一次。
- 第三：只要用户发送一次请求：service方法必然会被Tomcat服务器调用一次。发送100次请求，service方法会被调用100次。
- 关闭服务器的时候，控制台输出了以下内容：

```
1 AServlet's destroy method execute!
```

- 通过以上输出内容，可以得出以下结论：
 - Servlet的destroy方法只被Tomcat服务器调用一次。
 - destroy方法是在什么时候被调用的？
 - 在服务器关闭的时候。
 - 因为服务器关闭的时候要销毁AServlet对象的内存。

- 服务器在销毁AServlet对象内存之前，Tomcat服务器会自动调用AServlet对象的destroy方法。
- 请问：destroy方法调用的时候，对象销毁了还是没有销毁呢？
 - destroy方法执行的时候AServlet对象还在，没有被销毁。destroy方法执行结束之后，AServlet对象的内存才会被Tomcat释放。
- Servlet对象更像一个人的一生：
 - Servlet的无参数构造方法执行：标志着你出生了。
 - Servlet对象的init方法的执行：标志着你正在接受教育。
 - Servlet对象的service方法的执行：标志着你已经开始工作了，已经开始为人类提供服务了。
 - Servlet对象的destroy方法的执行：标志着临终。有什么遗言，抓紧的。要不然，来不及了。
- 关于Servlet类中方法的调用次数？
 - 构造方法只执行一次。
 - init方法只执行一次。
 - service方法：用户发送一次请求则执行一次，发送N次请求则执行N次。
 - destroy方法只执行一次。

- 当我们Servlet类中编写一个有参数的构造方法，如果没有手动编写无参数构造方法会出现什么问题？
 - 报错了：500错误。
 - 注意：500是一个HTTP协议的错误状态码。
 - 500一般情况下是因为服务器端的Java程序出现了异常。（服务器端的错误都是500错误：服务器内部错误。）
 - 如果没有无参数的构造方法，会导致出现500错误，无法实例化Servlet对象。
 - 所以，一定要注意：在Servlet开发当中，不建议程序员来定义构造方法，因为定义不当，一不小心就会导致无法实例化Servlet对象。
- 思考：Servlet的无参数构造方法是在对象第一次创建的时候执行，并且只执行一次。init方法也是在对象第一次创建的时候执行，并且只执行一次。那么这个无参数构造方法可以代替掉init方法吗？
 - 不能。
 - Servlet规范中有要求，作为javaweb程序员，编写Servlet类的时候，不建议手动编写构造方法，因为编写构造方法，很容易让无参数构造方法消失，这个操作可能会导致Servlet对象无法实例化。所以init方法是有存在的必要的。
- init、service、destroy方法中使用最多的是哪个方法？

- 使用最多就是service方法，service方法是一定要实现的，因为service方法是处理用户请求的核心方法。
- 什么时候使用init方法呢？
 - init方法很少用。
 - 通常在init方法当中做初始化操作，并且这个初始化操作只需要执行一次。例如：初始化数据库连接池，初始化线程池....
- 什么时候使用destroy方法呢？
 - destroy方法也很少用。
 - 通常在destroy方法当中，进行资源的关闭。马上对象要被销毁了，还有什么没有关闭的，抓紧时间关闭资源。还有什么资源没保存的，抓紧时间保存一下。

GenericServlet(适配器设计模式)

- 我们编写一个Servlet类直接实现Servlet接口有什么缺点？
 - 我们只需要service方法，其他方法大部分情况下是不需要使用的。代码很丑陋。
- **适配器设计模式Adapter**

- 比喻: 手机直接插到220V的电压上, 手机直接就报废了。怎么办? 可以找一个充电器。这个充电器就是一个适配器。手机连接适配器。适配器连接220V的电压。这样问题就解决了
- 让一个类去实现, 然后把主要的设置成抽象方法(那么这个类就变了抽象类)。其他servlet类只需要去继承这个GenericServlet即可
- 编写一个GenericServlet类, 这个类是一个抽象类, 其中有一个抽象方法service。
 - GenericServlet实现Servlet接口。
 - GenericServlet是一个适配器。
 - 以后编写的所有Servlet类继承GenericServlet, 重写service方法即可。
- 思考: GenericServlet类是否需要改造一下? 怎么改造? 更利于子类程序的编写?
 - 思考第一个问题: 我提供了一个GenericServlet之后, init方法还会执行吗?
 - 还会执行。动态绑定机制-会执行GenericServlet类中的init方法。
 - 思考第二个问题: init方法是谁调用的?
 - Tomcat服务器调用的。
 - 思考第三个问题: init方法中的ServletConfig对象是谁创建的? 是谁传过来的?

- 都是Tomcat干的。
- Tomcat服务器先创建了ServletConfig对象，然后调用init方法，将ServletConfig对象传给了init方法。
- 思考一下Tomcat服务器伪代码：

```
1 public class Tomcat {
2     public static void
main(String[] args){
3         // .....
4         // Tomcat服务器伪代码
5         // 创建LoginServlet对象（通
        过反射机制，调用无参数构造方法来实例化
        LoginServlet对象）
6         Class clazz =
Class.forName("com.bjpowernode.jav
aweb.servlet.LoginServlet");
7         Object obj =
clazz.newInstance();
8
9         // 向下转型
10        Servlet servlet =
(Servlet)obj;
11
12        // 创建ServletConfig对象
13        // Tomcat服务器负责将
        ServletConfig对象实例化出来。
14        // 多态（Tomcat服务器完全实现
        了Servlet规范）
```

```

15         ServletConfig
        servletConfig = new
        org.apache.catalina.core.StandardW
        rapperFacade(); //接口的引用指向实现类
        的对象
16
17         // 调用Servlet的init方法
18
        servlet.init(servletConfig);
19
20         // 调用Servlet的service方法
21         // ....
22
23     }
24 }

```

改造GenericServlet

- init方法的ServletConfig servletConfig参数, 小猫咪创建好ServletConfig实现对象后调用init()
 - 这个ServletConfig对象目前在init方法的参数上, 属于局部变量, 那么ServletConfig对象肯定以后要在service方法中使用, 怎样才能保证ServletConfig对象在service()中使用?
 - 创建一个ServletConfig属性, 在init方法中赋给属性

- 由于属性私有封装,我想在GenericServlet的子类的service()中使用,那么要在getServletConfig()方法中return
- 思考: 我又没有可能需要我在LoginServlet中重写init方法
 - 如果子类重写父类init方法,那么势必走子类init方法, 如果走子类的init方法, 父类init方法不走,也就意味着小猫咪传不了ServletConfig对象, 子类service方法的ServletConfig就没有了
 - 我给父类init方法,用final修饰,子类重写不了父类init(), 但我还是希望能够重写init()的等价效果(就在init执行的时候执行一些代码)呢
 - 然后我在父类写一个无参数的init方法, 在原本init()里调用无参的init方法, 让子类重写无参init() 就达到了异曲同工之妙---模板方法设计模式
- transient关键字de1zuo'y
- 序列化版本号的作用

ServletConfig

- 研究: 各个Servlet对象是否共享同一个ServletConfig对象?
 - 经过测试, 每个servlet对象是单独与一个ServletConfig一一对应
- 什么是ServletConfig?

- Servlet对象的配置信息对象。
- ServletConfig对象中封装了标签中的配置信息。
(web.xml文件中servlet的配置信息)
- 一个Servlet对应一个ServletConfig对象。
- Servlet对象是Tomcat服务器创建，并且ServletConfig对象也是Tomcat服务器创建。并且默认情况下，他们都是在用户发送第一次请求的时候创建。
- Tomcat服务器调用Servlet对象的init方法的时候需要传一个ServletConfig对象的参数给init方法。
- ServletConfig接口的实现类是Tomcat服务器给实现的。(Tomcat服务器说的就是WEB服务器。)
- ServletConfig接口有哪些常用的方法?

- ```
1 public String getInitParameter(String
 name); // 通过初始化参数的name获取value
2 public Enumeration<String>
 getInitParameterNames(); // 获取所有的
 初始化参数的name
3 public ServletContext
 getServletContext(); // 获取
 ServletContext对象
4 public String getServletName(); // 获
 取Servlet的name
```

- 以上方法在Servlet类当中，都可以使用this去调用。因为GenericServlet实现了ServletConfig接口。

## ServletContext

---

- 研究：ServletContext是否是共享的
  - 经过测试， AServlet和BServlet都打印出同一个对象，所以是共享的
- 一个Servlet对象对应一个ServletConfig。100个Servlet对象则对应100个ServletConfig对象。
- 只要在同一个webapp当中，只要在同一个应用当中，所有的Servlet对象都是共享同一个ServletContext对象的。
- ServletContext对象在服务器启动阶段创建，在服务器关闭的时候销毁。这就是ServletContext对象的生命周期。ServletContext对象是应用级对象。
- Tomcat服务器中有一个webapps，这个webapps下可以存放webapp，可以存放多个webapp，假设有100个webapp，那么就有100个ServletContext对象。但是，总之，一个应用，一个webapp肯定是只有一个ServletContext对象。
- ServletContext被称为Servlet上下文对象。（Servlet对象的四周环境对象。）

- 一个ServletContext对象通常对应的是一个web.xml文件。
- ServletContext对应显示生活中的什么例子呢？
  - 一个教室里有多个学生，那么每一个学生就是一个Servlet，这些学生都在同一个教室当中，那么我们可以把这个教室叫做ServletContext对象。那么也就是说放在这个ServletContext对象（环境）当中的数据，在同一个教室当中，物品都是共享的。比如：教室中有一个空调，所有的学生都可以操作。可见，空调是共享的。因为空调放在教室当中。教室就是ServletContext对象。
- ServletContext是一个接口，Tomcat服务器对ServletContext接口进行了实现。
  - ServletContext对象的创建也是Tomcat服务器来完成的。启动webapp的时候创建的。
- ServletContext接口中有哪些常用的方法？

```
1 //这两个在ServletConfig中也有
2 public String getInitParameter(String
 name); // 通过初始化参数的name获取value
3 public Enumeration<String>
 getInitParameterNames(); // 获取所有的
 初始化参数的name
```

○ 1 <!--以上两个方法是ServletContext对象的方法，这个方法获取的是什么信息？是以下的配置信息-->

```
2 <context-param>
3 <param-name>pageSize</param-name>
4 <param-value>10</param-value>
5 </context-param>
6 <context-param>
7 <param-name>startIndex</param-name>
8 <param-value>0</param-value>
9 </context-param>
```

10 <!--注意：以上的配置信息属于应用级的配置信息，一般一个项目中共享的配置信息会放到以上的标签当中。-->

11 <!--如果你的配置信息只是想给某一个servlet作为参考，那么你配置到servlet标签当中即可，使用ServletConfig对象来获取。-->

○ 1 // 获取应用的根路径（非常重要），因为在java源代码当中有一些地方可能会需要应用的根路径，这个方法可以动态获取应用的根路径（/表示从web下开始找）

2 // 在java源码当中，不要将应用的根路径写死，因为你永远都不知道这个应用在最终部署的时候，起一个什么名字。

```
3 public String getContextPath();
4 //String contextPath =
 application.getContextPath();
```

- ```
1 // 获取文件的绝对路径（真实路径）
2 public String getRealPath(String
  path);
```

- ```
1 // 通过ServletContext对象也是可以记录日
 志的
2 public void log(String message);
3 public void log(String message,
 Throwable t);
4 // 这些日志信息记录到哪里了？
5 // localhost.2021-11-05.log
6
7 // Tomcat服务器的logs目录下都有哪些日志文
 件？
8 //catalina.2021-11-05.log 服务器端的
 java程序运行的控制台信息。
9 //localhost.2021-11-05.log
 ServletContext对象的log方法记录的日志信
 息存储到这个文件中。
10 //localhost_access_log.2021-11-
 05.txt 访问日志
```

- ```
1 // ServletContext对象还有另一个名字：应
  用域（后面还有其他域，例如：请求域、会话
  域）
2
3 // 如果所有的用户共享一份数据，并且这个数据
  很少的被修改，并且这个数据量很少，可以将这
  些数据放到ServletContext这个应用域中
4
```

5 // 为什么是所有用户共享的数据？ 不是共享的
没有意义。因为`ServletContext`这个对象只有一个。只有共享的数据放进去才有意义。

6

7 // 为什么数据量要小？ 因为数据量比较大的
话，太占用堆内存，并且这个对象的生命周期比较长，服务器关闭的时候，这个对象才会被销毁。大数据量会影响服务器的性能。占用内存较小的数据量可以考虑放进去。

8

9 // 为什么这些共享数据很少的修改，或者说几乎
不修改？

10 // 所有用户共享的数据，如果涉及到修改操作，
必然会存在线程并发所带来的安全问题。所以放在`ServletContext`对象中的数据一般都是只读的。

11

12 // 数据量小、所有用户共享、又不修改，这样的
数据放到`ServletContext`这个应用域当中，会大大提升效率。因为应用域相当于一个缓存，放到缓存中的数据，下次在用的时候，不需要从数据库中再次获取，大大提升执行效率。

13

14 // 存（怎么向`ServletContext`应用域中存数据）

15 `public void setAttribute(String`
`name, Object value); // map.put(k,`
`v)`

16 // 取（怎么从`ServletContext`应用域中取数据）

```
17 public Object getAttribute(String  
    name); // Object v = map.get(k)  
18 // 删（怎么删除ServletContext应用域中的  
    数据）  
19 public void removeAttribute(String  
    name); // map.remove(k)
```

- 注意：以后我们编写Servlet类的时候，实际上是不会去直接继承GenericServlet类的，因为我们是B/S结构的系统，这种系统是基于HTTP超文本传输协议的，在Servlet规范当中，提供了一个类叫做HttpServlet，它是专门为HTTP协议准备的一个Servlet类。我们编写的Servlet类要继承HttpServlet。（HttpServlet是HTTP协议专用的。）使用HttpServlet处理HTTP协议更便捷。但是你需要直到它的继承结构：

- - 1 jakarta.servlet.Servlet（接口）【爷爷】
 - 2 jakarta.servlet.GenericServlet
 implements Servlet（抽象类）【儿子】
 - 3 jakarta.servlet.http.HttpServlet
 extends GenericServlet（抽象类）【孙子】
 - 4
 - 5 我们以后编写的Servlet要继承HttpServlet
 类。

- 大家到目前为止都接触过哪些缓存机制了？
 - 堆内存当中的字符串常量池。（jdk1.8新特性）

- "abc" 先在字符串常量池中查找，如果有，直接拿来用。如果没有则新建，然后再放入字符串常量池。
- 堆内存当中的整数型常量池。
 - [-128 ~ 127] 一共256个Integer类型的引用，放在整数型常量池中。没有超出这个范围的话，直接从常量池中取。
- 连接池(Connection Cache)
 - 这里所说的连接池中的连接是java语言连接数据库的连接对象：java.sql.Connection对象。
 - JVM是一个进程。MySQL数据库是一个进程。进程和进程之间建立连接，打开通道是很费劲的。是很耗费资源的。怎么办？可以提前先创建好N个Connection连接对象，将连接对象放到一个集合当中，我们把这个放有Connection对象的集合称为连接池。每一次用户连接的时候不需要再新建连接对象，省去了新建的环节，直接从连接池中获取连接对象，大大提升访问效率。
 - 连接池
 - 最小连接数
 - 最大连接数
 - 连接池可以提高用户的访问效率。当然也可以保证数据库的安全性。
- 线程池

- Tomcat服务器本身就是支持多线程的。
- Tomcat服务器是在用户发送一次请求，就新建一个Thread线程对象吗？
 - 当然不是，实际上是在Tomcat服务器启动的时候，会先创建好N多个线程Thread对象，然后将线程对象放到集合当中，称为线程池。用户发送请求过来之后，需要有一个对应的线程来处理这个请求，这个时候线程对象就会直接从线程池中拿，效率比较高。
 - 所有的WEB服务器，或者应用服务器，都是支持多线程的，都有线程池机制。
- redis
 - NoSQL数据库。非关系型数据库。缓存数据库。
- 向ServletContext应用域中存储数据，也等于是将数据存放到缓存cache当中了。

HTTP协议

- 什么是协议？
 - 协议实际上是某些人，或者某些组织提前制定好的一套规范，大家都按照这个规范来，这样可以做到沟通无障碍。
 - 协议就是一套规范，就是一套标准。由其他人或其他组织来负责制定的。

- 我说的话你能听懂，你说的话，我也能听懂，这说明我们之间是有一套规范的，一套协议的，这套协议就是：中国普通话协议。我们都遵守这套协议，我们之间就可以沟通无障碍。
- 什么是HTTP协议？
 - HTTP协议：是W3C制定的一种超文本传输协议。（通信协议：发送消息的模板提前被制定好。）
 - W3C：
 - 万维网联盟组织
 - 负责制定标准的：HTTP HTML4.0 HTML5 XML DOM等规范都是W3C制定的。
 - 万维网之父：蒂姆·伯纳斯·李
 - 什么是超文本？
 - 超文本说的就是：不是普通文本，比如流媒体：声音、视频、图片等。
 - HTTP协议支持：不但可以传送普通字符串，同样支持传递声音、视频、图片等流媒体信息。
 - 这种协议游走在B和S之间。B向S发数据要遵循HTTP协议。S向B发数据同样需要遵循HTTP协议。这样B和S才能解耦合。
 - 什么是解耦合？
 - B不依赖S。
 - S也不依赖B。

- B/S表示：B/S结构的系统（浏览器访问WEB服务器的系统）
- 浏览器 向 WEB服务器发送数据，叫做：请求（request）
- WEB服务器 向 浏览器发送数据，叫做：响应（response）
- HTTP协议包括：
 - 请求协议
 - 浏览器 向 WEB服务器发送数据的时候，这个发送的数据需要遵循一套标准，这套标准中规定了发送的数据具体格式。
 - 响应协议
 - WEB服务器 向 浏览器发送数据的时候，这个发送的数据需要遵循一套标准，这套标准中规定了发送的数据具体格式。
- HTTP协议就是提前制定好的一种消息模板。
 - 不管你是哪个品牌的浏览器，都是这么发。
 - 不管你是哪个品牌的WEB服务器，都是这么发。
 - FF浏览器 可以向 Tomcat发送请求，也可以向 Jetty服务器发送请求。浏览器不依赖具体的服务器品牌。

- WEB服务器也不依赖具体的浏览器品牌。可以是FF浏览器，也可以是Chrome浏览器，可以是IE，都行。
- HTTP的请求协议 (B --> S)
 - HTTP的请求协议包括：4部分
 - 请求行
 - 请求头
 - 空白行
 - 请求体
 - HTTP请求协议的具体报文：GET请求

- | | |
|---|---|
| 1 | GET /servlet05/getServlet?
username=lucy&userpwd=1111
HTTP/1.1 |
| | 请求行 |
| 2 | Host: localhost:8080 |
| | 请求头 |
| 3 | Connection: keep-alive |
| 4 | sec-ch-ua: "Google Chrome";v="95",
"Chromium";v="95", ";Not A
Brand";v="99" |
| 5 | sec-ch-ua-mobile: ?0 |
| 6 | sec-ch-ua-platform: "Windows" |
| 7 | Upgrade-Insecure-Requests: 1 |

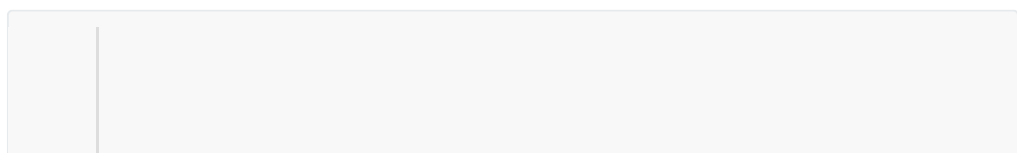
```
8 User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/95.0.4638.54
Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer:
http://localhost:8080/servlet05/index.html
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: zh-CN,zh;q=0.9
17

空白行

18

请求体
```

- HTTP请求协议的具体报文：POST请求





1 POST /servlet05/postServlet
HTTP/1.1

请求行

2 Host: localhost:8080

请求头

3 Connection: keep-alive

4 Content-Length: 25

5 Cache-Control: max-age=0

6 sec-ch-ua: "Google Chrome";v="95",
"Chromium";v="95", ";Not A
Brand";v="99"

7 sec-ch-ua-mobile: ?0

8 sec-ch-ua-platform: "Windows"

9 Upgrade-Insecure-Requests: 1

10 Origin: http://localhost:8080

11 Content-Type: application/x-www-
form-urlencoded

12 User-Agent: Mozilla/5.0 (Windows
NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/95.0.4638.54
Safari/537.36

13 Accept:
text/html,application/xhtml+xml,ap
plication/xml;q=0.9,image/avif,ima
ge/webp,image/apng,*/*;q=0.8,appli
cation/signed-exchange;v=b3;q=0.9

14 Sec-Fetch-Site: same-origin

15 Sec-Fetch-Mode: navigate

```
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer:
    http://localhost:8080/servlet05/index.html
19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: zh-CN,zh;q=0.9
21
```

空白行

```
22 username=lisi&userpwd=123
```

请求体

- 请求行

- 包括三部分：

- 第一部分：请求方式（7种）

- get（常用的）
 - post（常用的）
 - delete
 - put
 - head
 - options
 - trace

- 第二部分：URI

- 什么是URI? 统一资源标识符。代表网络中某个资源的名字。但是通过URI是无法定位资源的。
- 什么是URL? 统一资源定位符。代表网络中某个资源, 同时, 通过URL是可以定位到该资源的。
- URI和URL什么关系, 有什么区别?
 - URL包括URI
 - <http://localhost:8080/servlet05/index.html> 这是URL。
 - /servlet05/index.html 这是URI。
- 第三部分: HTTP协议版本号
- 请求头
 - 请求的主机
 - 主机的端口
 - 浏览器信息
 - 平台信息
 - cookie等信息
 -
- 空白行
 - 空白行是用来区分“请求头”和“请求体”
- 请求体

- 向服务器发送的具体数据。
- HTTP的响应协议 (S --> B)
 - HTTP的响应协议包括：4部分
 - 状态行
 - 响应头
 - 空白行
 - 响应体
 - HTTP响应协议的具体报文：

```
1 HTTP/1.1 200 ok
                                     状态行
2 Content-Type:
  text/html; charset=UTF-8
                                     响应头
3 Content-Length: 160
4 Date: Mon, 08 Nov 2021 13:19:32
  GMT
5 Keep-Alive: timeout=20
6 Connection: keep-alive
7
                                     空白行
8 <!doctype html>
                                     响应体
9 <html>
10     <head>
11         <title>from get
      servlet</title>
```

```
12     </head>
13     <body>
14         <h1>from get servlet</h1>
15     </body>
16 </html>
```

- 状态行

- 三部分组成

- 第一部分：协议版本号（HTTP/1.1）
 - 第二部分：状态码（HTTP协议中规定的响应状态号。不同的响应结果对应不同的号码。）
 - 200 表示请求响应成功，正常结束。
 - 404表示访问的资源不存在，通常是因为要么是你路径写错了，要么是路径写对了，但是服务器中对应的资源并没有启动成功。总之404错误是前端错误。
 - 405表示前端发送的请求方式与后端请求的处理方式不一致时发生：
 - 比如：前端是POST请求，后端的处理方式按照get方式进行处理时，发生405
 - 比如：前端是GET请求，后端的处理方式按照post方式进行处理时，发生405
 - 500表示服务器端的程序出现了异常。一般会认为是服务器端的错误导致的。

- 以4开始的，一般是浏览器端的错误导致的。
- 以5开始的，一般是服务器端的错误导致的。
- 第三部分：状态的描述信息
 - ok 表示正常成功结束。
 - not found 表示资源找不到。
- 响应头：
 - 响应的内容类型
 - 响应的内容长度
 - 响应的时间
 -
- 空白行：
 - 用来分隔“响应头”和“响应体”的。
- 响应体：
 - 响应体就是响应的正文，这些内容是一个长的字符串，这个字符串被浏览器渲染，解释并执行，最终展示出效果。
- 怎么查看的协议内容？
 - 使用chrome浏览器：F12。然后找到network，通过这个面板可以查看协议的具体内容。
- 怎么向服务器发送GET请求，怎么向服务器发送POST请求？

- 到目前为止，只有一种情况可以发送POST请求：
使用form表单，并且form标签中的method属性值为：method="post"。
- 其他所有情况一律都是get请求：
 - 在浏览器地址栏上直接输入URL，敲回车，属于get请求。
 - 在浏览器上直接点击超链接，属于get请求。
 - 使用form表单提交数据时，form标签中没有写method属性，默认就是get
 - 或者使用form的时候，form标签中method属性值为：method="get"
 -
- GET请求和POST请求有什么区别？
 - 参数位置
 - get请求发送数据的时候，数据会挂在URI的后面，并且在URI后面添加一个"?"，"?"后面是数据。这样会导致发送的数据回显在浏览器的地址栏上。（get请求在“请求行”上发送数据）
 - <http://localhost:8080/servlet05/getServlet?username=zhangsan&userpwd=1111>
 - post请求发送数据的时候，在请求体当中发送。不会回显到浏览器的地址栏上。也就是说post发送的数据，在浏览器地址栏上看不到。（post在“请求体”当中发送数据）

○ 数据量大小

- get请求只能发送普通的字符串。并且发送的字符串长度有限制，不同的浏览器限制不同。这个没有明确的规范。
- get请求无法发送大数据量。
- post请求可以发送任何类型的数据，包括普通字符串，流媒体等信息：视频、声音、图片。
- post请求可以发送大数据量，理论上没有长度限制。

○ W3C

- get请求在W3C中是这样说的：get请求比较适合从服务器端获取数据。
- post请求在W3C中是这样说的：post请求比较适合向服务器端传送数据。

○ 安全性和使用场景

- get请求是安全的。get请求是绝对安全的。为什么？因为get请求只是为了从服务器上获取数据。不会对服务器造成威胁。（get本身是安全的，你不要用错了。用错了之后又冤枉人家get不安全，你这样不好（太坏了），那是你自己的问题，不是get请求的问题。）

- post请求是危险的。为什么？因为post请求是向服务器提交数据，如果这些数据通过后门的方式进入到服务器当中，服务器是很危险的。另外post是为了提交数据，所以一般情况下拦截请求的时候，大部分会选择拦截（监听）post请求。
- 缓存
 - get请求支持缓存。
 - <https://n.sinaimg.cn/finance/590/w240h350/20211101/b40c-b425eb67cab342ff5b9dc018b4b00cc.jpg>
 - 任何一个get请求最终的“响应结果”都会被浏览器缓存起来。在浏览器缓存当中：
 - 一个get请求的路径a 对应 一个资源。
 - 一个get请求的路径b 对应 一个资源。
 - 一个get请求的路径c 对应 一个资源。
 -
 - 实际上，你只要发送get请求，浏览器做的第一件事都是先从本地浏览器缓存中找，找不到的时候才会去服务器上获取。这种缓存机制目的是为了提升用户的体验。
 - 有没有这样一个需求：我们不想让get请求走缓存，怎么办？怎么避免走缓存？我希望每一次这个get请求都去服务器上找资源，我不想从本地浏览器的缓存中取。

- 只要每一次get请求的请求路径不同即可。
- <https://n.sinaimg.cn/finance/590/w240h350/20211101/7cab342ff5b9dc018b4b00cc.jpg?t=789789787897898>
- <https://n.sinaimg.cn/finance/590/w240h350/20211101/7cab342ff5b9dc018b4b00cc.jpg?t=789789787897899>
- <https://n.sinaimg.cn/finance/590/w240h350/20211101/7cab342ff5b9dc018b4b00cc.jpg?t=系统毫秒数>
- 怎么解决？可以在路径的后面添加一个每时每刻都在变化的“时间戳”，这样，每一次的请求路径都不一样，浏览器就不走缓存了。
- post请求不支持缓存。（POST是用来修改服务器端的资源的。）
 - post请求之后，服务器“响应的结果”不会被浏览器缓存起来。因为这个缓存没有意义。
- GET请求和POST请求如何选择，什么时候使用GET请求，什么时候使用POST请求？
 - 如何选择GET请求和POST请求呢？衡量标准是什么呢？你这个请求是想获取服务器端的数据，还是想向服务器发送数据。如果你是想从服务器上获取资源，建议使用GET请求，如果你这个请求是为了向服务器提交数据，建议使用POST请求。

- 大部分的form表单提交，都是post方式，因为form表单中要填写大量的数据，这些数据是收集用户的信息，一般是需要传给服务器，服务器将这些数据保存/修改等。
- 如果表单中有敏感信息，还是建议适用post请求，因为get请求会回显敏感信息到浏览器地址栏上。
(例如：密码信息)
- 做文件上传，一定是post请求。要传的数据不是普通文本。
- 其他情况都可以使用get请求。
- 不管你是get请求还是post请求，发送的请求数据格式是完全相同的，只不过位置不同，格式都是统一的：
 - name=value&name=value&name=value&name=value
 - name是什么？
 - 以form表单为例：form表单中input标签的name。
 - value是什么？
 - 以form表单为例：form表单中input标签的value。

模板方法设计模式

- 什么是设计模式？

- 某个问题的固定的解决方案。(可以被重复使用。)
- 你知道哪些设计模式？
 - GoF设计模式：
 - 通常我们所说的23种设计模式。（Gang of Four：4人组提出的设计模式）
 - 单例模式
 - 工厂模式
 - 代理模式
 - 门面模式
 - 责任链设计模式
 - 观察者模式
 - 模板方法设计模式
 -
 - JavaEE设计模式：
 - DAO
 - DTO
 - VO
 - PO
 - pojo
 -
 -

- 什么是模板方法设计模式？
 - 在模板类的模板方法当中定义核心算法骨架，具体的实现步骤可以延迟到子类当中完成。
- 模板类通常是一个抽象类，模板类当中的模板方法定义核心算法，这个方法通常是final的（但也可以不是final的）
- 模板类当中的抽象方法就是不确定实现的方法，这个不确定怎么实现的事儿交给子类去做。

HttpServlet源码分析

- HttpServlet类是专门为HTTP协议准备的。比GenericServlet更加适合HTTP协议下的开发。
- HttpServlet在哪个包下？
 - jakarta.servlet.http.HttpServlet
- 到目前为止我们接触了servlet规范中哪些接口？
 - jakarta.servlet.Servlet 核心接口（接口）
 - jakarta.servlet.ServletConfig Servlet配置信息接口（接口）
 - jakarta.servlet.ServletContext Servlet上下文接口（接口）
 - jakarta.servlet.ServletRequest Servlet请求接口（接口）

- jakarta.servlet.ServletResponse Servlet响应接口 (接口)
- jakarta.servlet.ServletException Servlet异常 (类)
- jakarta.servlet.GenericServlet 标准通用的Servlet 类 (抽象类)
- http包下都有哪些类和接口呢?
jakarta.servlet.http.*;
 - jakarta.servlet.http.HttpServlet (HTTP协议专用的Servlet类, 抽象类)
 - jakarta.servlet.http.HttpServletRequest (HTTP协议专用的请求对象)
 - jakarta.servlet.http.HttpServletResponse (HTTP协议专用的响应对象)
- HttpServletRequest对象中封装了什么信息?
 - HttpServletRequest, 简称request对象。
 - HttpServletRequest中封装了请求协议的全部内容。
 - Tomcat服务器 (WEB服务器) 将“请求协议”中的数据全部解析出来, 然后将这些数据全部封装到request对象当中了。
 - 也就是说, 我们只要面向HttpServletRequest, 就可以获取请求协议中的数据。

- HttpServletResponse对象是专门用来响应HTTP协议到浏览器的。
- 回忆Servlet生命周期？（根据Servlet生命周期学HttpServlet）
 - 用户第一次请求
 - Tomcat服务器通过反射机制，调用无参数构造方法。创建Servlet对象。（web.xml文件中配置的Servlet类对应的对象。）
 - Tomcat服务器调用Servlet对象的init方法完成初始化。
 - Tomcat服务器调用Servlet对象的service方法处理请求。
 - 用户第二次请求
 - Tomcat服务器调用Servlet对象的service方法处理请求。
 - 用户第三次请求
 - Tomcat服务器调用Servlet对象的service方法处理请求。
 -
 - Tomcat服务器调用Servlet对象的service方法处理请求。
 - 服务器关闭

- Tomcat服务器调用Servlet对象的destroy方法，做销毁之前的准备工作。
- Tomcat服务器销毁Servlet对象。
- HttpServlet源码分析：

```
1 public class HelloServlet extends
  HttpServlet {
2     // 用户第一次请求，创建HelloServlet对象
  的时候，会执行这个无参数构造方法。
3     public HelloServlet() {
4     }
5
6     //override 重写 doGet方法
7     //override 重写 doPost方法
8 }
9
10 public abstract class GenericServlet
  implements Servlet, ServletConfig,
11     java.io.Serializable {
12
13     // 用户第一次请求的时候，HelloServlet对
  象第一次被创建之后，这个init方法会执行。
14     public void init(ServletConfig
  config) throws ServletException {
15         this.config = config;
16         this.init();
17     }
```

```
18      // 用户第一次请求的时候，带有参数的
      init(ServletConfig config)执行之后，会执行
      这个没有参数的init()
19      public void init() throws
      ServletException {
20          // NOOP by default
21      }
22  }
23
24  // HttpServlet模板类。
25  public abstract class HttpServlet
      extends GenericServlet {
26      // 用户发送第一次请求的时候这个service会
      执行
27      // 用户发送第N次请求的时候，这个service方
      法还是会执行。
28      // 用户只要发送一次请求，这个service方法
      就会执行一次。
29      @Override
30      public void service(ServletRequest
      req, ServletResponse res)
31          throws ServletException,
      IOException {
32
33          HttpServletRequest request;
34          HttpServletResponse response;
35
36          try {
```

```

37         // 将ServletRequest和
        ServletResponse向下转型为带有Http的
        HttpServletRequest和HttpServletResponse
38         request =
        (HttpServletRequest) req;
39         response =
        (HttpServletResponse) res;
40     } catch (ClassCastException e)
    {
41         throw new
        ServletException(1Strings.getString("ht
        tp.non_http"));
42     }
43     // 调用重载的service方法。
44     service(request, response);
45 }
46
47     // 这个service方法的两个参数都是带有Http
    的。
48     // 这个service是一个模板方法。
49     // 在该方法中定义核心算法骨架，具体的实现
    步骤延迟到子类中去完成。
50     protected void
    service(HttpServletRequest req,
    HttpServletResponse resp)
51         throws ServletException,
    IOException {
52         // 获取请求方式
53         // 这个请求方式最终可能是：""

```



```
54      // 注意: request.getMethod()方法
    获取的是请求方式, 可能是七种之一:
55      // GET POST PUT DELETE HEAD
    OPTIONS TRACE
56      String method =
    req.getMethod();
57
58      // 如果请求方式是GET请求, 则执行
    doGet方法。
59      if (method.equals(METHOD_GET))
    {
60          long lastModified =
    getLastModified(req);
61          if (lastModified == -1) {
62              // servlet doesn't
    support if-modified-since, no reason
63              // to go through
    further expensive logic
64              doGet(req, resp);
65          } else {
66              long ifModifiedSince;
67              try {
68                  ifModifiedSince =
    req.getDateHeader(HEADER_IFMODSINCE);
69              } catch
    (IllegalArgumentException iae) {
70                  // Invalid date
    header - proceed as if none was set
71                  ifModifiedSince =
    -1;
```

```
72         }
73         if (ifModifiedSince <
74 (lastModified / 1000 * 1000)) {
75         // If the servlet
76 mod time is later, call doGet()
77         // Round down to
78 the nearest second for a proper compare
79         // A
80 ifModifiedSince of -1 will always be
81 less
82
83         maybeSetLastModified(resp,
84 lastModified);
85         doGet(req, resp);
86     } else {
87
88         resp.setStatus(HttpServletResponse.SC_
89 NOT_MODIFIED);
90
91         }
92     }
93
94     } else if
95 (method.equals(METHOD_HEAD)) {
96         long lastModified =
97 getLastModified(req);
98         maybeSetLastModified(resp,
99 lastModified);
100         doHead(req, resp);
101     }
102 }
```

```
89         } else if
(method.equals(METHOD_POST)) {
90             // 如果请求方式是POST请求，则执
行doPost方法。
91             doPost(req, resp);
92
93         } else if
(method.equals(METHOD_PUT)) {
94             doPut(req, resp);
95
96         } else if
(method.equals(METHOD_DELETE)) {
97             doDelete(req, resp);
98
99         } else if
(method.equals(METHOD_OPTIONS)) {
100             doOptions(req, resp);
101
102         } else if
(method.equals(METHOD_TRACE)) {
103             doTrace(req, resp);
104
105         } else {
106             //
107             // Note that this means NO
servlet supports whatever
108             // method was requested,
anywhere on this server.
109             //
110
```

```
111         String errMsg =
112             lStrings.getString("http.method_not_implemented");
113         Object[] errArgs = new
114             Object[1];
115         errArgs[0] = method;
116         errMsg =
117             MessageFormat.format(errMsg, errArgs);
118
119         resp.sendError(HttpServletResponse.SC_
120             NOT_IMPLEMENTED, errMsg);
121     }
122 }
123
124     protected void
125     doGet(HttpServletRequest req,
126         HttpServletResponse resp)
127         throws ServletException,
128         IOException{
129         // 报405错误
130         String msg =
131             lStrings.getString("http.method_get_not_
132                 _supported");
133         sendMethodNotAllowed(req, resp,
134             msg);
135     }
136 }
```

```

128         protected void
doPost(HttpServletRequest req,
HttpServletRequest resp)
129         throws ServletException,
IOException {
130             // 报405错误
131             String msg =
1Strings.getString("http.method_post_no
t_supported");
132             sendMethodNotAllowed(req, resp,
msg);
133         }
134
135     }
136
137     /*

```

138 通过以上源代码分析：

139 假设前端发送的请求是**get**请求，后端程序员重
写的方法是**doPost**

140 假设前端发送的请求是**post**请求，后端程序员重
写的方法是**doGet**

141 会发生什么呢？

142 发生**405**这样的错误。

143 **405**表示前端的错误，发送的请求方式不
对。和服务器不一致。不是服务器需要的请求方式。

144

145 通过以上源代码可以知道：只要**HttpServlet**类
中的**doGet**方法或**doPost**方法执行了，必然**405**。

146

147 怎么避免**405**的错误呢？

148 后端重写了**doGet**方法，前端一定要发**get**请求。

149 后端重写了**doPost**方法，前端一定要发**post**请求。

150 这样可以避免**405**错误。

151

152 这种前端到底需要发什么样的请求，其实应该后端说了算。后端让发什么方式，前端就得发什么方式。

153

154 有的人，你会看到为了避免**405**错误，在**Servlet**类当中，将**doGet**和**doPost**方法都进行了重写。

155 这样，确实可以避免**405**的发生，但是不建议，**405**错误还是有用的。该报错的时候就应该让他报错。

156 如果你要是同时重写了**doGet**和**doPost**，那还不如你直接重写**service**方法好了。这样代码还能

157 少写一点。

158 ***/**

159

160

- 我们编写的HelloServlet直接继承HttpServlet，直接重写HttpServlet类中的service()方法行吗？
 - 可以，只不过你享受不到405错误。享受不到HTTP协议专属的东西。
- 到今天我们终于得到了最终的一个Servlet类的开发步骤：
 - 第一步：编写一个Servlet类，直接继承HttpServlet

- 第二步：重写doGet方法或者重写doPost方法，到底重写谁，javaweb程序员说了算。
- 第三步：将Servlet类配置到web.xml文件当中。
- 第四步：准备前端的页面（form表单），form表单中指定请求路径即可。

关于一个web站点的欢迎页面

- 什么是一个web站点的欢迎页面？
 - 对于一个webapp来说，我们是可以设置它的欢迎页面的。
 - 设置了欢迎页面之后，当你访问这个webapp的时候，或者访问这个web站点的时候，没有指定任何“资源路径”，这个时候会默认访问你的欢迎页面。
 - 我们一般的访问方式是：
 - <http://localhost:8080/servlet06/login.html> 这种方式是指定了要访问的就是login.html资源。
 - 如果我们访问的方式是：
 - <http://localhost:8080/servlet06> 如果我们访问的就是这个站点，没有指定具体的资源路径。它默认会访问谁呢？
 - 默认会访问你设置的欢迎页面。
- 怎么设置欢迎页面呢？

- 第一步：我在IDEA工具的web目录下新建了一个文件login.html
- 第二步：在web.xml文件中进行了以下的配置

- ```
1 <welcome-file-list>
2 <welcome-
3 file>login.html</welcome-file>
 </welcome-file-list>
```

- 注意：设置欢迎页面的时候，这个路径不需要以“/”开始。并且这个路径默认是从webapp的根下查找。
- 第三步：启动服务器，浏览器地址栏输入地址
  - <http://localhost:8080/servlet07>
- 如果在webapp的根下新建一个目录，目录中再给一个文件，那么这个欢迎页该如何设置呢？
  - 在webapp根下新建page1
  - 在page1下新建page2目录
  - 在page2目录下新建page.html页面
  - 在web.xml文件中应该这样配置

- ```
1 <welcome-file-list>
2     <welcome-
    file>page1/page2/page.html</welcome-
    file>
3 </welcome-file-list>
```


- 注意：路径不需要以“/”开始，并且路径默认从webapp的根下找。
- 一个webapp是可以设置多个欢迎页面的

```
1 <welcome-file-list>
2     <welcome-
  file>page1/page2/page.html</welcome-
  file>
3     <welcome-
  file>login.html</welcome-file>
4 </welcome-file-list>
```

- 注意：越靠上的优先级越高。找不到的继续向下找。
- 你有没有注意一件事：当我的文件名设置为index.html的时候，不需要在web.xml文件中进行配置欢迎页面。这是为什么？
 - 这是因为小猫咪Tomcat服务器已经提前配置好了。
 - 实际上配置欢迎页面有两个地方可以配置：
 - 一个是在webapp内部的web.xml文件中。（在这个地方配置的属于局部配置）
 - 一个是在CATALINA_HOME/conf/web.xml文件中进行配置。（在这个地方配置的属于全局配置）

- ```
1 <welcome-file-list>
2 <welcome-
file>index.html</welcome-file>
3 <welcome-
file>index.htm</welcome-file>
4 <welcome-
file>index.jsp</welcome-file>
5 </welcome-file-list>
```

- Tomcat服务器的全局欢迎页面是：  
index.html index.htm index.jsp。如果你一个web站点没有设置局部的欢迎页面，  
Tomcat服务器就会以index.html index.htm  
index.jsp作为一个web站点的欢迎页面。

- 注意原则：局部优先原则。（就近原则）

- 欢迎页可以是一个Servlet吗？

- 当然可以。
- 你不要多想，欢迎页就是一个资源，既然是一个资源，那么可以是静态资源，也可以是动态资源。
- 静态资源：index.html welcome.html .....
- 动态资源：Servlet类。
- 步骤：

- 第一步：写一个Servlet

- ```
1 public class welcomeServlet
  extends HttpServlet {
2     @Override
3     protected void
doGet(HttpServletRequest
request, HttpServletResponse
response) throws
ServletException, IOException {
4
    response.setContentType("text/h
tml");
5        PrintWriter out =
response.getWriter();
6        out.print("<h1>welcome
to bjpowernode!</h1>");
7    }
8 }
```

- 第二步：在web.xml文件中配置servlet

- ```
1 <servlet>
2 <servlet-
name>welcomeServlet</servlet-
name>
3 <servlet-
class>com.bjpowernode.javaweb.se
rvlet.welcomeServlet</servlet-
class>
4 </servlet>
5 <servlet-mapping>
6 <servlet-
name>welcomeServlet</servlet-
name>
7 <url-
pattern>/fdsa/fds/a/fds/af/ds/af
/dsafdsafdsa</url-pattern>
8 </servlet-mapping>
```

- 第三步：在web.xml文件中配置欢迎页

- ```
1      <welcome-file-list>
2          <welcome-
file>fdsa/fds/a/fds/af/ds/af/dsa
fsafdsa</welcome-file>
3      </welcome-file-list>
```

关于WEB-INF目录

- 在WEB-INF目录下新建了一个文件：welcome.html
- 打开浏览器访问：<http://localhost:8080/servlet07/WEB-INF/welcome.html> 出现了404错误。
- 注意：放在WEB-INF目录下的资源是受保护的。在浏览器上不能够通过路径直接访问。所以像HTML、CSS、JS、image等静态资源一定要放到WEB-INF目录之外。

HttpServletRequest接口 详解

- HttpServletRequest是一个接口，全限定名称：
jakarta.servlet.http.HttpServletRequest
- HttpServletRequest接口是Servlet规范中的一员。
- HttpServletRequest接口的父接口：ServletRequest
 - ```
1 public interface HttpServletRequest
 extends ServletRequest {}
```
- HttpServletRequest接口的实现类谁写的？  
HttpServletRequest对象是谁给创建的？
  - 通过测试：  
org.apache.catalina.connector.RequestFacade  
实现了 HttpServletRequest接口

- 1 `public class RequestFacade`  
`implements HttpServletRequest {}`

- 测试结果说明：Tomcat服务器（WEB服务器、WEB容器）实现了HttpServletRequest接口，还是说明了Tomcat服务器实现了Servlet规范。而对于我们javaweb程序员来说，实际上不需要关心这个，我们只需要面向接口编程即可。我们关心的是HttpServletRequest接口中有哪些方法，这些方法可以完成什么功能！！！！
- HttpServletRequest对象中都有什么信息？都包装了什么信息？
  - HttpServletRequest对象是Tomcat服务器负责创建的。这个对象中封装了什么信息？封装了HTTP的请求协议。
  - 实际上是用户发送请求的时候，遵循了HTTP协议，发送的是HTTP的请求协议，Tomcat服务器将HTTP协议中的信息以及数据全部解析出来，然后Tomcat服务器把这些信息封装到HttpServletRequest对象当中，传给了我们javaweb程序员。
  - javaweb程序员面向HttpServletRequest接口编程，调用方法就可以获取到请求的信息了。
- request和response对象的生命周期？

- request对象和response对象，一个是请求对象，一个是响应对象。这两个对象只在当前请求中有效。
- 一次请求对应一个request。
- 两次请求则对应两个request。
- .....
- HttpServletRequest接口中有哪些常用的方法？
  - 怎么获取前端浏览器用户提交的数据？

- - 1 `Map<String,String[]>`  
`getParameterMap()` 这个是获取Map
  - 2 `Enumeration<String>`  
`getParameterNames()` 这个是获取Map集合中所有的key
  - 3 `String[] getParameterValues(String name)` 根据key获取Map集合的value
  - 4 `String getParameter(String name)`  
获取value这个一维数组当中的第一个元素。  
这个方法最常用。
  - 5 `//` 以上的4个方法，和获取用户提交的数据有关系。

- 思考：如果是你，前端的form表单提交了数据之后，你准备怎么存储这些数据，你准备采用什么样的数据结构去存储这些数据呢？

- 前端提交的数据格式：  
username=abc&userpwd=111&aihao=s&aihao=d&aihao=tt
- 我会采用Map集合来存储：

- ```
1 Map<String,String>
2     key存储String
3     value存储String
4     这种想法对吗？ 不对。
5     如果采用以上的数据结构存储会发现key重复的时候value覆盖。
6     key           value
7     -----
8     username      abc
9     userpwd       111
10    aihao         s
11    aihao         d
12    aihao         tt
13    这样是不行的，因为map的key不能重复。
14 Map<String, String[]>
15     key存储String
16     value存储String[]
17     key           value
18     -----
19     -----
19     username      {"abc"}
20     userpwd       {"111"}
```



```
{"s","d","tt"}
```

- 注意：前端表单提交数据的时候，假设提交了120这样的“数字”，其实是以字符串"120"的方式提交的，所以服务器端获取到的一定是一个字符串的"120"，而不是一个数字。（前端永远提交的是字符串，后端获取的也永远是字符串。）
- 手工开发一个webapp。测试HttpServletRequest接口中的相关方法。
 - 先测试了4个常用的方法，获取请求参数的四个方法。

```
1      Map<String,String[]>
    parameterMap =
    request.getParameterMap();
2      Enumeration<String> names =
    request.getParameterNames();
3      String[] values =
    request.getParameterValues("name");
4      String value =
    request.getParameter("name");
```

- request对象实际上又称为“请求域”对象。
 - 应用域对象是什么？

- ServletContext (Servlet上下文对象。)
- 什么情况下会考虑向ServletContext这个应用域当中绑定数据呢？
 - 第一：所有用户共享的数据。
 - 第二：这个共享的数据量很小。
 - 第三：这个共享的数据很少的修改操作。
 - 在以上三个条件都满足的情况下，使用这个应用域对象，可以大大提高我们程序执行效率。
 - 实际上向应用域当中绑定数据，就相当于把数据放到了缓存 (Cache) 当中，然后用户访问的时候直接从缓存中取，减少IO的操作，大大提升系统的性能，所以缓存技术是提高系统性能的重要手段。
- 你见过哪些缓存技术呢？
 - 字符串常量池
 - 整数型常量池 [-128~127]，但凡是在这个范围当中的Integer对象不再创建新对象，直接从这个整数型常量池中获取。大大提升系统性能。

- 数据库连接池（提前创建好N个连接对象，将连接对象放到集合当中，使用连接对象的时候，直接从缓存中拿。省去了连接对象的创建过程。效率提升。）
- 线程池（Tomcat服务器就是支持多线程的。所谓的线程池就是提前先创建好N个线程对象，将线程对象存储到集合中，然后用户请求过来之后，直接从线程池中获取线程对象，直接拿来用。提升系统性能）
- 后期你还会学习更多的缓存技术，例如：redis、mongoDB.....
- ServletContext当中有三个操作域的方法：

- ```
1 void setAttribute(String
 name, Object obj); // 向域当
 中绑定数据。
2 Object getAttribute(String
 name); // 从域当中根据name获取
 数据。
3 void removeAttribute(String
 name); // 将域当中绑定的数据移
 除
4
5 // 以上的操作类似于Map集合的操
 作。
6 Map<String, Object> map;
7 map.put("name", obj); // 向
 map集合中放key和value
8 Object obj =
 map.get("name"); // 通过map集
 合的key获取value
9 map.remove("name"); // 通过
 Map集合的key删除key和value这个
 键值对。
```

- “请求域”对象

- “请求域”对象要比“应用域”对象范围小很多。生命周期短很多。请求域只在一次请求内有效。
- 一个请求对象request对应一个请求域对象。一次请求结束之后，这个请求域就销毁了。

- 请求域对象也有这三个方法：

- - 1 `void setAttribute(String name, Object obj);` // 向域当中绑定数据。
  - 2 `Object getAttribute(String name);` // 从域当中根据`name`获取数据。
  - 3 `void removeAttribute(String name);` // 将域当中绑定的数据移除

- 请求域和应用域的选用原则？

- 尽量使用小的域对象，因为小的域对象占用的资源较少。

- 跳转

- 转发（一次请求）

- ```
1 // 第一步：获取请求转发器对象
2 RequestDispatcher dispatcher
  =
  request.getRequestDispatcher
    ("/b");
3 // 第二步：调用转发器的forward
  方法完成跳转/转发
4 dispatcher.forward(request, r
  esponse);
5
6 // 第一步和第二步代码可以联合在一
  起。
7 request.getRequestDispatcher
  ("/b").forward(request, respo
  nse);
8
```

- 两个Servlet怎么共享数据？
 - 将数据放到ServletContext应用域当中，当然是可以的，但是应用域范围太大，占用资源太多。不建议使用。
 - 可以将数据放到request域当中，然后AServlet转发到BServlet，保证AServlet和BServlet在同一次请求当中，这样就可以做到两个Servlet，或者多个Servlet共享同一份数据。
- 转发的下一个资源必须是一个Servlet吗？

- 不一定，只要是Tomcat服务器当中的合法资源，都是可以转发的。例如：html....
- 注意：转发的时候，路径的写法要注意，转发的路径以“/”开始，不加项目名。
- 关于request对象中两个非常容易混淆的方法：

- ```
1 // uri?
2 username=zhangsan&userpwd=123
3 &sex=1
4 String username =
5 request.getParameter("username");
6
7 // 之前一定是执行过：
8 request.setAttribute("name",
9 new Object())
10 Object obj =
11 request.getAttribute("name");
12
13 // 以上两个方法的区别是什么？
14 // 第一个方法：获取的是用户在浏览器
15 // 上提交的数据。
16 // 第二个方法：获取的是请求域当中
17 // 绑定的数据。
```

- HttpServletRequest接口的其他常用方法：

- ```
1 // 获取客户端的IP地址
```

```
2 String remoteAddr =
  request.getRemoteAddr();
3
4 // get请求在请求行上提交数据。
5 // post请求在请求体中提交数据。
6 // 设置请求体的字符集。（显然这个方法
  是处理POST请求的乱码问题。这种方式
  并不能解决get请求的乱码问题。）
7 // Tomcat10之后，request请求体
  当中的字符集默认就是UTF-8，不需要
  设置字符集，不会出现乱码问题。
8 // Tomcat9前（包括9在内），如果前
  端请求体提交的是中文，后端获取之后
  出现乱码，怎么解决这个乱码？执行以
  下代码。
9 request.setCharacterEncoding(
  "UTF-8");
10
11 // 在Tomcat9之前（包括9），响应中
  文也是有乱码的，怎么解决这个响应的
  乱码？
12 response.setContentType("text
  /html; charset=UTF-8");
13 // 在Tomcat10之后，包括10在内，
  响应中文的时候就不在出现乱码问题
  了。以上代码就不需要设置UTF-8了。
14
15 // 注意一个细节
```



```
16 // 在Tomcat10包括10在内之后的版本，中文将不再出现乱码。（这也体现了中文地位的提升。）
17
18 // get请求乱码问题怎么解决？
19 // get请求发送的时候，数据是在请求行上提交的，不是在请求体当中提交的。
20 // get请求乱码怎么解决
21 // 方案：修改
    CATALINA_HOME/conf/server.xml
    配置文件
22 <Connector URIEncoding="UTF-8" />
23 // 注意：从Tomcat8之后，
    URIEncoding的默认值就是UTF-8，所以GET请求也没有乱码问题了。
24
25 // 获取应用的根路径
26 String contextPath =
    request.getContextPath();
27
28 // 获取请求方式
29 String method =
    request.getMethod();
30
31 // 获取请求的URI
32 String uri =
    request.getRequestURI(); //
    /aaa/testRequest
```

```
33
34 // 获取servlet path
35 String servletPath =
    request.getServletPath(); //
    /testRequest
36
```

使用纯Servlet做一个单表的CRUD操作

- 使用纯粹的Servlet完成单表【对部门的】的增删改查操作。（B/S结构的。）
- 实现步骤
 - 第一步：准备一张数据库表。（sql脚本）

- ```
1 # 部门表
2 drop table if exists dept;
3 create table dept(
4 deptno int primary key,
5 dname varchar(255),
6 loc varchar(255)
7);
8 insert into dept(deptno, dname,
9 loc) values(10, 'XiaoShouBu',
10 'BEIJING');
11 insert into dept(deptno, dname,
12 loc) values(20, 'YanFaBu',
13 'SHANGHAI');
14 insert into dept(deptno, dname,
15 loc) values(30, 'JiShuBu',
16 'GUANGZHOU');
17 insert into dept(deptno, dname,
18 loc) values(40, 'MeiTiBu',
19 'SHENZHEN');
20 commit;
21 select * from dept;
```

- 第二步：准备一套HTML页面（项目原型）【前端开发工具使用HBuilder】

- 把HTML页面准备好
- 然后将HTML页面中的链接都能够跑通。（页面流转没问题。）
- 应该设计哪些页面呢？

- 欢迎页面：index.html
- 列表页面：list.html（以列表页面为核心，展开其他操作。）
- 新增页面：add.html
- 修改页面：edit.html
- 详情页面：detail.html
- 第三步：分析我们这个系统包括哪些功能？
  - 什么叫做一个功能呢？
    - 只要 这个操作连接了数据库，就表示一个独立的功能。
  - 包括哪些功能？
    - 查看部门列表
    - 新增部门
    - 删除部门
    - 查看部门详细信息
    - 跳转到修改页面
    - 修改部门
- 第四步：在IDEA当中搭建开发环境
  - 创建一个webapp（给这个webapp添加servlet-api.jar和jsp-api.jar到classpath当中。）
  - 向webapp中添加连接数据库的jar包（mysql驱动）

- 必须在WEB-INF目录下新建lib目录，然后将mysql的驱动jar包拷贝到这个lib目录下。这个目录名必须叫做lib，全部小写的。
- JDBC的工具类
- 将所有HTML页面拷贝到web目录下。
- 第五步：实现第一个功能：查看部门列表
  - 我们应该怎么去实现一个功能呢？
    - 建议：你可以从后端往前端一步一步写。也可以从前端一步一步往后端写。都可以。但是千万要记住不要想起来什么写什么。你写代码的过程最好是程序的执行过程。也就是说：程序执行到哪里，你就写哪里。这样一个顺序流下来之后，基本上不会出现什么错误、意外。
    - 从哪里开始？
      - 假设从前端开始，那么一定是从用户点击按钮那里开始的。
  - 第一：先修改前端页面的超链接，因为用户先点击的就是这个超链接。

```
1 | 查看部门列表
```
  - 第二：编写web.xml文件

- ```
1 <servlet>
2     <servlet-name>list</servlet-
  name>
3     <servlet-
  class>com.bjpowernode.oa.web.act
  ion.DeptListServlet</servlet-
  class>
4 </servlet>
5 <servlet-mapping>
6     <servlet-name>list</servlet-
  name>
7     <!--web.xml文件中的这个路径也是
  以“/”开始的，但是不需要加项目名-->
8     <url-
  pattern>/dept/list</url-pattern>
9 </servlet-mapping>
```

- 第三：编写DeptListServlet类继承HttpServlet类。然后重写doGet方法。

- ```
1 package
 com.bjpowernode.oa.web.action;
2
3 import
 jakarta.servlet.ServletException;
4 import
 jakarta.servlet.http.HttpServlet;
```

```

5 import
 jakarta.servlet.http.HttpServlet
 tRequest;
6 import
 jakarta.servlet.http.HttpServlet
 tResponse;
7
8 import java.io.IOException;
9
10 public class DeptListServlet
 extends HttpServlet {
11 @Override
12 protected void
 doGet(HttpServletRequest
 request, HttpServletResponse
 response)
13 throws
 ServletException, IOException {
14 }
15 }

```

- 第四：在DeptListServlet类的doGet方法中连接数据库，查询所有的部门，动态的展示部门列表页面。
  - 分析list.html页面中哪部分是固定死的，哪部分是需要动态展示的。
  - list.html页面中的内容所有的双引号要替换成单引号，因为out.print("")这里有一个双引号，容易冲突。

- 现在写完这个功能之后，你会有一种感觉，感觉开发很繁琐，只使用servlet写代码太繁琐了。

- ```
1 while(rs.next()){
2     String deptno =
rs.getString("a");
3     String dname =
rs.getString("dname");
4     String loc =
rs.getString("loc");
5
6     out.print("                <tr>");
7     out.print("
<td>" ++ i + "</td>");
8     out.print("
<td>" + deptno + "</td>");
9     out.print("
<td>" + dname + "</td>");
10    out.print("
<td>");
11    out.print("
<a href=' ' >删除</a>");
12    out.print("
<a href='edit.html' >修改</a>");
13    out.print("
<a href='detail.html' >详情
</a>");
14    out.print("
</td>");
```



```
15     out.print("
    </tr>");
16 }
```

○ 第六步：查看部门详情。

- 建议：从前端往后端一步一步实现。首先要考虑的是，用户点击的是什么？用户点击的东西在哪里？

- 一定要先找到用户点的“详情”在哪里。找了半天，终于在后端的java程序中找到了

- 1 | `详情`

- 详情 是需要连接数据库的，所以这个超链接点击之后也是需要执行一段java代码的。所以要将这个超链接的路径修改一下。

- 注意：修改路径之后，这个路径是需要加项目名的。"/oa/dept/detail"

- 技巧：

- 1 | `out.print("详情");`

- 重点：向服务器提交数据的格式：uri?name=value&name=value&name=value&name=value

- 这里的问号，必须是英文的问号。不能中文的问号。
- 解决404的问题。写web.xml文件。

- ```
1 <servlet>
2 <servlet-
name>detail</servlet-name>
3 <servlet-
class>com.bjpowernode.oa.web.act
ion.DeptDetailServlet</servlet-
class>
4 </servlet>
5 <servlet-mapping>
6 <servlet-
name>detail</servlet-name>
7 <url-
pattern>/dept/detail</url-
pattern>
8 </servlet-mapping>
```

- 编写一个类：DeptDetailServlet继承HttpServlet，重写doGet方法。

- ```
1 package
com.bjpowernode.oa.web.action;
2
3 import
jakarta.servlet.ServletException;
```

```
4 import
jakarta.servlet.http.HttpServlet;
5 import
jakarta.servlet.http.HttpServletRequest;
6 import
jakarta.servlet.http.HttpServletResponse;
7
8 import java.io.IOException;
9
10 public class DeptDetailServlet
    extends HttpServlet {
11     @Override
12     protected void
doGet(HttpServletRequest
request, HttpServletResponse
response)
13         throws
ServletException, IOException {
14         //中文思路（思路来源于：你要
做什么？目标：查看部门详细信息。）
15         // 第一步：获取部门编号
16         // 第二步：根据部门编号查询
数据库，获取该部门编号对应的部门信息。
17         // 第三步：将部门信息响应到
浏览器上。（显示一个详情。）
18     }
19 }
```

- 在doGet方法当中：连接数据库，根据部门编号查询该部门的信息。动态展示部门详情页。
- 第七步：删除部门
 - 怎么开始？从哪里开始？从前端页面开始，用户点击删除按钮的时候，应该提示用户是否删除。因为删除这个动作是比较危险的。任何系统在进行删除操作之前，是必须要提示用户的，因为这个删除的动作有可能是用户误操作。（在前端页面上写JS代码，来提示用户是否删除。）

```
1 <a href="javascript:void(0)"
  onclick="del(30)" >删除</a>
2 <script type="text/javascript">
3     function del(dno){
4         if(window.confirm("亲，删
  了不可恢复哦！")){
5
6         document.location.href =
  "/oa/dept/delete?deptno=" + dno;
7     }
8 </script>
```

- 以上的前端程序要写到后端的java代码当中：
 - DeptListServlet类的doGet方法当中，使用out.print()方法，将以上的前端代码输出到浏览器上。
- 解决404的问题：

- <http://localhost:8080/oa/dept/delete?deptno=30>
- web.xml文件

- ```
1 <servlet>
2 <servlet-
 name>delete</servlet-name>
3 <servlet-
 class>com.bjpowernode.oa.web.a
 ction.DeptDelServlet</servlet-
 class>
4 </servlet>
5 <servlet-mapping>
6 <servlet-
 name>delete</servlet-name>
7 <url-
 pattern>/dept/delete</url-
 pattern>
8 </servlet-mapping>
```

- 编写DeptDelServlet继承HttpServlet，重写doGet方法。

- ```
1 package
  com.bjpowernode.oa.web.action;
2
3 import
  jakarta.servlet.ServletException;
```

```

4  import
   jakarta.servlet.http.HttpServlet
   t;
5  import
   jakarta.servlet.http.HttpServlet
   tRequest;
6  import
   jakarta.servlet.http.HttpServlet
   tResponse;
7
8  import java.io.IOException;
9
10 public class DeptDelServlet
    extends HttpServlet {
11     @Override
12     protected void
        doGet(HttpServletRequest
            request, HttpServletResponse
                response)
13         throws
            ServletException, IOException {
14         // 根据部门编号，删除部门。
15
16     }
17 }

```

- 删除成功或者失败的时候的一个处理（这里我们选择了转发，并没有使用重定向机制。）

```
■ 1 // 判断删除成功了还是失败了。
2 if (count == 1) {
3     //删除成功
4     //仍然跳转到部门列表页面
5     //部门列表页面的显示需要执行另一个Servlet。怎么办？转发。
6
7     request.getRequestDispatcher(
8         "/dept/list").forward(request, response);
9 }else{
10     // 删除失败
11
12     request.getRequestDispatcher(
13         "/error.html").forward(request, response);
14 }
```

○ 第八步：新增部门

- 注意：最后保存成功之后，转发到 /dept/list 的时候，会出现405，为什么？
 - 第一：保存用的是post请求。底层要执行doPost方法。
 - 第二：转发是一次请求，之前是post，之后还是post，因为它是一次请求。
 - 第三：/dept/list Servlet当中只有一个doGetMethod。
 - 怎么解决？两种方案

- 第一种：在/dept/list Servlet中添加doPost方法，然后在doPost方法中调用doGet。
- 第二种：重定向。
 - 第九步：跳转到修改部门的页面
 - 第十步：修改部门
- oa项目是存在一些可以改进的地方的
 - 资源跳转用转发不合理
 - xml文件过大，书写麻烦
 - 做一个单表crud操作就有6个类，类太多了
 - 纯servlet存在前端不好维护的问题

在一个web应用中应该如何完成资源的跳转

- 在一个web应用中通过两种方式，可以完成资源的跳转：
 - 第一种方式：转发
 - 第二种方式：重定向
- 转发和重定向有什么区别？
 - 代码上有什么区别？
 - 转发

- 1 // 获取请求转发器对象
- 2 RequestDispatcher dispatcher =
request.getRequestDispatcher("/
dept/list");
- 3 // 调用请求转发器对象的forward方法完
成转发
- 4 dispatcher.forward(request,
response);
- 5
- 6 // 合并一行代码
- 7 request.getRequestDispatcher("/
dept/list").forward(request,
response);
- 8 // 转发的时候是一次请求，不管你转发了
多少次。都是一次请求。
- 9 // AServlet转发到BServlet，再转发
到CServlet，再转发到DServlet，不管
转发了多少次，都在同一个request当
中。
- 10 // 这是因为调用forward方法的时候，会
将当前的request和response对象传递给
下一个Servlet。

■ 重定向

- ```
// 转发
//request.getRequestDispatcher("/b").forward(request, response);

// 重定向（重新定方向）
// 重定向时的路径当中需要以项目名开始，或者说需要添加项目名。
// response对象将这个路径："/servlet10/b"响应给浏览器了。
// 浏览器又自发的向服务器发送了一次全新的请求：http://localhost:8080/servlet10/b
// 所以浏览器一共发送了两次请求：
// 第一次请求：http://localhost:8080/servlet10/a
// 第二次请求：http://localhost:8080/servlet10/b
// 最终浏览器地址栏上显示的地址当然是最后那一次请求的地址。所以重定向会导致浏览器地址栏上的地址发生改变。
response.sendRedirect(location: request.getContextPath() + "/b");
```

- - 1 // 注意：路径上要加一个项目名。为什么？
  - 2 // 浏览器发送请求，请求路径上是需要添加项目名的。
  - 3 // 以下这一行代码会将请求路径“/oa/dept/list”发送给浏览器
  - 4 // 浏览器会自发的向服务器发送一次全新的请求：/oa/dept/list
  - 5 response.sendRedirect("/oa/dept/list");

○ 形式上有什么区别？

■ 转发（一次请求）

- 在浏览器地址栏上发送的请求是：<http://localhost:8080/servlet10/a>，最终请求结束之后，浏览器地址栏上的地址还是这个。没变。

■ 重定向（两次请求）

- 在浏览器地址栏上发送的请求是：<http://localhost:8080/servlet10/a>，最终在浏览器地址栏上显示的地址是：<http://localhost:8080/servlet10/b>

○ 转发和重定向的本质区别？

- 转发：是由WEB服务器来控制的。A资源跳转到B资源，这个跳转动作是Tomcat服务器内部完成的。浏览器并不知道
- 重定向：是浏览器完成的。具体跳转到哪个资源，是浏览器说了算。
  - 重定向的资源共享用session会话域来实现
- 使用一个例子去描述这个转发和重定向
  - 借钱（转发：发送了一次请求）
    - 杜老师没钱了，找张三借钱，其实张三没有钱，但是张三够义气，张三自己找李四借了钱，然后张三把这个钱给了杜老师，杜老师不知道这个钱是李四的，杜老师只求了一个人。杜老师以为这个钱就是张三的。
  - 借钱（重定向：发送了两次请求）
    - 杜老师没钱了，找张三借钱，张三没有钱，张三有一个好哥们，叫李四，李四是个富二代，于是张三将李四的家庭住址告诉了杜老师，杜老师按照这个地址去找到李四，然后从李四那里借了钱。显然杜老师在这个过程中，求了两个人。并且杜老师知道最终这个钱是李四借给俺的。
- 转发和重定向应该如何选择？什么时候使用转发，什么时候使用重定向？

- 如果在上一个Servlet当中向request域当中绑定了数据，希望从下一个Servlet当中把request域里面的数据取出来，使用转发机制。
- 剩下所有的请求均使用重定向。（重定向使用较多。）
- 跳转的下一个资源有没有要求呢？必须是一个Servlet吗？
  - 不一定，跳转的资源只要是服务器内部合法的资源即可。包括：Servlet、JSP、HTML.....
- 转发会存在浏览器的刷新问题：因为转发后地址仍然是之前的，如果之前的地址是向后端插入数据，跳转成功页面后一直刷新会引起数据库数据一直被插入

## 将oa项目中的资源跳转修改为合适的跳转方式

---

- 删除之后，重定向
- 修改之后，重定向
- 保存之后，重定向
- 重定向：
  - 成功
  - 失败

# Servlet注解，简化配置

---

- 分析oa项目中的web.xml文件
  - 现在只是一个单标的CRUD，没有复杂的业务逻辑，很简单的一丢丢功能。web.xml文件中就有如此多的配置信息。如果采用这种方式，对于一个大的项目来说，这样的话web.xml文件会非常庞大，有可能最终会达到几十兆。
  - 在web.xml文件中进行servlet信息的配置，显然开发效率比较低，每一个都需要配置一下。
  - 而且在web.xml文件中的配置是很少被修改的，所以这种配置信息能不能直接写到java类当中呢？可以的。
- Servlet3.0版本之后，推出了各种Servlet基于注解式开发。优点是什么？
  - 开发效率高，不需要编写大量的配置信息。直接在java类上使用注解进行标注。
  - web.xml文件体积变小了。
- 并不是说注解有了之后，web.xml文件就不需要了：
  - 有一些需要变化的信息，还是要配置到web.xml文件中。一般都是 注解+配置文件 的开发模式。
  - 一些不会经常变化修改的配置建议使用注解。一些可能会被修改的建议写到配置文件中。
- 我们的第一个注解：

- 1 | `jakarta.servlet.annotation.WebServlet`
- 在Servlet类上使用：@WebServlet, WebServlet 注解中有哪些属性呢？
  - name属性：用来指定Servlet的名字。等同于：
  - urlPatterns属性：用来指定Servlet的映射路径。可以指定多个字符串。
  - loadOnStartup属性：用来指定在服务器启动阶段是否加载该Servlet。等同于：
  - value属性：当注解的属性名是value的时候，使用注解的时候，value属性名是可以省略的。模糊匹配@webServlet("/dept/\*")
  - 注意：不是必须将所有属性都写上，只需要提供需要的。（需要什么用什么。）
  - 注意：属性是一个数组，如果数组中只有一个元素，使用该注解的时候，属性值的大括号可以省略。
- 注解对象的使用格式：
  - @注解名称(属性名=属性值, 属性名=属性值, 属性名=属性值....)

# 使用模板方法设计模式优化oa项目

---

- 上面的注解解决了配置文件的问题。但是现在的oa项目仍然存在一个比较臃肿的问题。
  - 一个单标的CRUD，就写了6个Servlet。如果一个复杂的业务系统，这种开发方式，显然会导致类爆炸。（类的数量太大。）
  - 怎么解决这个类爆炸问题？可以使用模板方法设计模式。
- 怎么解决类爆炸问题？
  - 以前的设计是一个请求一个Servlet类。1000个请求对应1000个Servlet类。导致类爆炸。
  - 可以这样做：一个请求对应一个方法。一个业务对应一个Servlet类。
  - 处理部门相关业务的对应一个DeptServlet。处理用户相关业务的对应一个UserServlet。处理银行卡卡片业务对应一个CardServlet。

## 分析使用纯粹Servlet开发web应用的缺陷

---

- 在Servlet当中编写HTML/CSS/JavaScript等前端代码。存在什么问题？

- java程序中编写前端代码，编写难度大。麻烦。
- java程序中编写前端代码，显然程序的耦合度非常高。
- java程序中编写前端代码，代码非常不美观。
- java程序中编写前端代码，维护成本太高。（非常难于维护）
  - 修改小小的一个前端代码，只要有改动，就需要重新编译java代码，生成新的class文件，打一个新的war包，重新发布。
- 思考一下，如果是你的话，你准备怎么解决这个问题？
  - 思路很重要。使用什么样的思路去做、去解决这个问题
    - 上面的那个Servlet (Java程序) 能不能不写了，让机器自动生成。我们程序员只需要写这个Servlet程序中的“前端的那段代码”，然后让机器将我们写的“前端代码”自动翻译生成“Servlet这种java程序”。然后机器再自动将“java”程序编译生成"class"文件。然后再使用JVM调用这个class中的方法。



# 关于B/S结构系统的会话机制 (session机制)

---

- 什么是会话？
  - 会话对应的英语单词：session
  - 用户打开浏览器，进行一系列操作，然后最终将浏览器关闭，这个整个过程叫做：一次会话。会话在服务器端也有一个对应的java对象，这个java对象叫做：session。
  - 什么是一次请求：用户在浏览器上点击了一下，然后到页面停下来，可以粗略认为是一次请求。请求对应的服务器端的java对象是：request。
  - 一个会话当中包含多次请求。（一次会话对应N次请求。）
- 在java的servlet规范当中，session对应的类名：  
HttpSession (jarkata.servlet.http.HttpSession)
- session机制属于B/S结构的一部分。如果使用php语言开发WEB项目，同样也是有session这种机制的。session机制实际上是一个规范。然后不同的语言对这种会话机制都有实现。
- session对象最主要的作用是：保存会话状态。（用户登录成功了，这是一种登录成功的状态，你怎么把登录成功的状态一直保存下来呢？使用session对象可以保留会话状态。）

- 为什么需要session对象来保存会话状态呢？
  - 因为HTTP协议是一种无状态协议。
  - 什么是无状态：请求的时候，B和S是连接的，但是请求结束之后，连接就断了。为什么要这么做？  
HTTP协议为什么要设计成这样？因为这样的无状态协议，可以降低服务器的压力。请求的瞬间是连接的，请求结束之后，连接断开，这样服务器压力小。
  - 只要B和S断开了，那么关闭浏览器这个动作，服务器知道吗？
    - 不知道。服务器是不知道浏览器关闭的。
- 张三打开一个浏览器A，李四打开一个浏览器B，访问服务器之后，在服务器端会生成：
  - 张三专属的session对象
  - 李四专属的session对象
- 为什么不使用request对象保存会话状态？为什么不使用ServletContext对象保存会话状态？
  - request.setAttribute()存，request.getAttribute()取，ServletContext也有这个方法。request是请求域。ServletContext是应用域。
  - request是一次请求一个对象。

- ServletContext对象是服务器启动的时候创建，服务器关闭的时候销毁，这个ServletContext对象只有一个。
- ServletContext对象的域太大。
- request请求域 (HttpServletRequest) 、 session会话域 (HttpSession) 、 application域 (ServletContext)
- request < session < application
- 思考一下：session对象的实现原理。
  - HttpSession session = request.getSession();
  - 这行代码很神奇。张三访问的时候获取的session对象就是张三的。李四访问的时候获取的session对象就是李四的。
- session的实现原理：
  - JSESSIONID=xxxxxx 这个是以Cookie的形式保存在浏览器的内存中的。浏览器只要关闭。这个cookie就没有了。
  - session列表是一个Map，map的key是sessionid，map的value是session对象。
  - 用户第一次请求，服务器生成session对象，同时生成id，将id发送给浏览器。
  - 用户第二次请求，自动将浏览器内存中的id发送给服务器，服务器根据id查找session对象。

- 关闭浏览器，内存消失，cookie消失，sessionid消失，会话等同于结束。
- Cookie禁用了，session还能找到吗？
  - cookie禁用是什么意思？服务器正常发送cookie给浏览器，但是浏览器不要了。拒收了。并不是服务器不发了。
  - 找不到了。每一次请求都会获取到新的session对象。
  - cookie禁用了，session机制还能实现吗？
    - 可以。需要使用URL重写机制。
    - <http://localhost:8080/servlet12/test/session;jsessionid=19D1C99560DCBF84839FA43D58F56E16>
    - URL重写机制会提高开发者的成本。开发人员在编写任何请求路径的时候，后面都要添加一个sessionid，给开发带来了很大的难度，很大的成本。所以大部分的网站都是这样设计的：你要是禁用cookie，你就别用了。
- 总结一下到目前位置我们所了解的域对象：
  - request（对应的类名：HttpServletRequest）
    - 请求域（请求级别的）
  - session（对应的类名：HttpSession）
    - 会话域（用户级别的）

- application（对应的类名：ServletContext）
  - 应用域（项目级别的，所有用户共享的。）
- 这三个域对象的大小关系
  - request < session < application
- 他们三个域对象都有以下三个公共的方法：
  - setAttribute（向域当中绑定数据）
  - getAttribute（从域当中获取数据）
  - removeAttribute（删除域当中的数据）
- 使用原则：尽量使用小的域。
- session掌握之后，我们怎么解决oa项目中的登录问题，怎么能让登录起作用。
  - 登录成功之后，可以将用户的登录信息存储到session当中。也就是说session中如果有用户的信息就代表用户登录成功了。session中没有用户信息，表示用户没有登录过。则跳转到登录页面。
- 销毁session对象：
  - ```
1 session.invalidate();
```

Cookie

- session的实现原理中，每一个session对象都会关联一个sessionid，例如：

- JSESSIONID=41C481F0224664BDB28E95081D23D5B8
- 以上的这个键值对数据其实就是cookie对象。
- 对于session关联的cookie来说，这个cookie是被保存在浏览器的“运行内存”当中。
- 只要浏览器不关闭，用户再次发送请求的时候，会自动将运行内存中的cookie发送给服务器。
- 例如，这个Cookie：
JSESSIONID=41C481F0224664BDB28E95081D23D5B8就会再次发送给服务器。
- 服务器就是根据
41C481F0224664BDB28E95081D23D5B8这个值来找到对应的session对象的。
- cookie怎么生成？ cookie保存在什么地方？ cookie有啥用？ 浏览器什么时候会发送cookie，发送哪些cookie给服务器？ ? ? ? ? ? ? ?
- cookie最终是保存在浏览器客户端上的。
 - 可以保存在运行内存中。（浏览器只要关闭cookie就消失了。）
 - 也可以保存在硬盘文件中。（永久保存。）
- cookie有啥用呢？
 - cookie和session机制其实都是为了保存会话的状态。

- cookie是将会话的状态保存在浏览器客户端上。
(cookie数据存储在浏览器客户端上的。)
- session是将会话的状态保存在服务器端上。
(session对象是存储在服务器上。)
- 为什么要有cookie和session机制呢？因为HTTP协议是无状态 无连接协议。
- cookie的经典案例
 - 京东商城，在未登录的情况下，向购物车中放几件商品。然后关闭商城，再次打开浏览器，访问京东商城的时候，购物车中的商品还在，这是怎么做的？我没有登录，为什么购物车中还有商品呢？
 - 将购物车中的商品编号放到cookie当中，cookie保存在硬盘文件当中。这样即使关闭浏览器。硬盘上的cookie还在。下一次再打开京东商城的时候，查看购物车的时候，会自动读取本地硬盘中存储的cookie，拿到商品编号，动态展示购物车中的商品。
 - 京东存储购物车中商品的cookie可能是这样的：productIds=xxxxx,yyyy,zzz,kkkk
 - 注意：cookie如果清除掉，购物车中的商品就消失了。
 - 126邮箱中有一个功能：十天内免登录
 - 这个功能也是需要cookie来实现的。
 - 怎么实现的呢？

- 用户输入正确的用户名和密码，并且同时选择十天内免登录。登录成功后。浏览器客户端会保存一个cookie，这个cookie中保存了用户名和密码等信息，这个cookie是保存在硬盘文件当中的，十天有效。在十天内用户再次访问126的时候，浏览器自动提交126的关联的cookie给服务器，服务器接收到cookie之后，获取用户名和密码，验证，通过之后，自动登录成功。
- 怎么让cookie失效？
 - 十天过后自动失效。
 - 或者改密码。
 - 或者在客户端浏览器上清除cookie。
- cookie机制和session机制其实都不属于java中的机制，实际上cookie机制和session机制都是HTTP协议的一部分。php开发中也有cookie和session机制，只要是你是做web开发，不管是什么编程语言，cookie和session机制都是需要的。
- HTTP协议中规定：任何一个cookie都是由name和value组成的。name和value都是字符串类型的。
- 在java的servlet中，对cookie提供了哪些支持呢？
 - 提供了一个Cookie类来专门表示cookie数据。
jakarta.servlet.http.Cookie;

- java程序怎么把cookie数据发送给浏览器呢?
`response.addCookie(cookie);`
- 在HTTP协议中是这样规定的：当浏览器发送请求的时候，会自动携带该path下的cookie数据给服务器。
(URL。)
- 关于cookie的有效时间
 - 怎么用java设置cookie的有效时间
 - `cookie.setMaxAge(60 * 60);` 设置cookie在一小时之后失效。
 - 没有设置有效时间：默认保存在浏览器的运行内存中，浏览器关闭则cookie消失。
 - 只要设置cookie的有效时间 > 0，这个cookie一定会存储到硬盘文件当中。
 - 设置cookie的有效时间 = 0 呢？
 - cookie被删除，同名cookie被删除。
 - 设置cookie的有效时间 < 0 呢？
 - 保存在运行内存中。和不设置一样。
- 关于cookie的path，cookie关联的路径：
 - 假设现在发送的请求路径是“<http://localhost:8080/servlet13/cookie/generate>”生成的cookie，如果cookie没有设置path，默认的path是什么？
 - 默认的path是：<http://localhost:8080/servlet13/cookie> 以及它的子路径。

- 也就是说，以后只要浏览器的请求路径是<http://localhost:8080/servlet13/cookie> 这个路径以及这个路径下的子路径，cookie都会被发送到服务器。
- 手动设置cookie的path
 - `cookie.setPath("/servlet13");` 表示只要是这个servlet13项目的请求路径，都会提交这个cookie给服务器。
- 浏览器发送cookie给服务器了，服务器中的java程序怎么接收？

```
1 Cookie[] cookies =  
    request.getCookies(); // 这个方法可能  
    返回null  
2 if(cookies != null){  
3     for(Cookie cookie : cookies){  
4         // 获取cookie的name  
5         String name =  
        cookie.getName();  
6         // 获取cookie的value  
7         String value =  
        cookie.getValue();  
8     }  
9 }  
10
```

- 使用cookie实现一下十天内免登录功能。
 - 先实现登录功能

- 登录成功
 - 跳转到部门列表页面
- 登录失败
 - 跳转到登录失败页面
- 修改前端页面
 - 在登录页面给一个复选框，复选框后面给一句话：十天内免登录。
 - 用户选择了复选框：表示要支持十天内免登录。
 - 用户没有选择复选框：表示用户不想使用十天内免登录功能。
- 修改Servlet中的login方法
 - 如果用户登录成功了，并且用户登录时选择了十天内免登录功能，这个时候应该在Servlet的login方法中创建cookie，用来存储用户名和密码，并且设置路径，设置有效期，将cookie响应给浏览器。（浏览器将其自动保存在硬盘文件中10天）
- 用户再次访问该网站的时候，访问这个网站的首页的时候，有两个走向：
 - 要么跳转到部门列表页面
 - 要么跳转到登录页面
 - 以上分别有两个走向，这显然是需要编写java程序进行控制的。

JSP

- 我的第一个JSP程序：
 - 在WEB-INF目录之外创建一个index.jsp文件，然后这个文件中没有任何内容。
- 将上面的项目部署之后，启动服务器，打开浏览器，访问以下地址：
 - <http://localhost:8080/jsp/index.jsp> 展现在大家面前的的是一个空白。
 - 实际上访问以上的这个：index.jsp，底层执行的是：index_jsp.class 这个java程序。
 - 这个index.jsp会被tomcat翻译生成index_jsp.java文件，然后tomcat服务器又会将index_jsp.java编译生成index_jsp.class文件
 - 访问index.jsp，实际上执行的是index_jsp.class中的方法。
- JSP实际上就是一个Servlet。
 - index.jsp访问的时候，会自动翻译生成index_jsp.java，会自动编译生成index_jsp.class，那么index_jsp 这就是一个类。
 - index_jsp 类继承 HttpJspBase，而HttpJspBase类继承的是HttpServlet。所以index_jsp类就是一个Servlet类。

- jsp的生命周期和Servlet的生命周期完全相同。完全就是一个东西。没有任何区别。
- jsp和servlet一样，都是单例的。（假单例。）
- jsp文件第一次访问的时候是比较慢的，为什么？
 - 为什么大部分的运维人员在给客户演示项目的时候，为什么提前先把所有的jsp文件先访问一遍。
 - 第一次比较麻烦：
 - 要把jsp文件翻译生成java源文件
 - java源文件要编译生成class字节码文件
 - 然后通过class去创建servlet对象
 - 然后调用servlet对象的init方法
 - 最后调用servlet对象的service方法。
 - 第二次就比较快了，为什么？
 - 因为第二次直接调用单例servlet对象的service方法即可。
- JSP是什么？
 - JSP是java程序。（JSP本质还是一个Servlet）
 - JSP是：JavaServer Pages的缩写。（基于Java语言实现的服务器端的页面。）
 - Servlet是JavaEE的13个子规范之一，那么JSP也是JavaEE的13个子规范之一。

- JSP是一套规范。所有的web容器/web服务器都是遵循这套规范的，都是按照这套规范进行的“翻译”
- 每一个web容器/web服务器都会内置一个`JSP翻译引擎。
- 对JSP进行错误调试的时候，还是要直接打开JSP文件对应的java文件，检查java代码。 放在
CATALINA_BASE路径
- 开发JSP的最高境界：
 - 眼前是JSP代码，但是脑袋中呈现的是java代码。
- JSP既然本质上是一个Servlet，那么JSP和Servlet到底有什么区别呢？
 - 职责不同：
 - Servlet的职责是什么：收集数据。（Servlet的强项是逻辑处理，业务处理，然后链接数据库，获取/收集数据。）
 - JSP的职责是什么：展示数据。（JSP的强项是做数据的展示）
- JSP的基础语法
 - 在jsp文件中直接编写文字，都会自动被翻译到哪里？
 - 翻译到servlet类的service方法的out.write("翻译到这里")，直接翻译到双引号里，被java程序当做普通字符串打印输出到浏览器。

- 在JSP中编写的HTML CSS JS代码，这些代码对于JSP来说只是一个普通的字符串。但是JSP把这个普通的字符串一旦输出到浏览器，浏览器就会对HTML CSS JS进行解释执行。展现一个效果。
- JSP的page指令（这个指令后面再详细说，这里先解决一下中文乱码问题），解决响应时的中文乱码问题：
 - 通过page指令来设置响应的内容类型，在内容类型的最后面添加：charset=UTF-8
 - `<%@page contentType="text/html;charset=UTF-8"%>`，表示响应的内容类型是text/html，采用的字符集UTF-8
 - `<%@page import="java.util.List,java.util.ArrayList"%>`
- 怎么在JSP中编写Java程序：
 - `<% java语句; %>`
 - 在这个符号当中编写的被视为java程序，被翻译到Servlet类的service方法内部。
 - 这里你要细心点，你要思考，在`<% %>`这个符号里面写java代码的时候，你要时时刻刻的记住你正在“方法体”当中写代码，方法体中可以写什么，不可以写什么，你心里是否明白呢？

- `<!-- -->` jsp脚本块的注释
- 在service方法当中编写的代码是有顺序的，方法体当中的代码要遵循自上而下的顺序依次逐行执行。
- service方法当中不能写静态代码块，不能写方法，不能定义成员变量。。。。。
- 在同一个JSP当中 `<%%>` 这个符号可以出现多个。
- `<%! %>`
 - 在这个符号当中编写的java程序会自动翻译到service方法之外。
 - 这个语法很少用，为什么？不建议使用，因为在service方法外面写静态变量和实例变量，都会存在线程安全问题，因为JSP就是servlet，servlet是单例的，多线程并发的环境下，这个静态变量和实例变量一旦有修改操作，必然会存在线程安全问题。
- JSP的输出语句
 - 怎么向浏览器上输出一个java变量。
 - `<% String name = "jack"; out.write("name = " + name); %>`
 - 注意：以上代码中的out是JSP的九大内置对象之一。可以直接拿来用。当然，必须只能在service方法内部使用。

- 如果向浏览器上输出的内容中没有“java代码”，例如输出的字符串是一个固定的字符串，可以直接在jsp中编写，不需要写到`<%%>` 这里。
- 如果输出的内容中含有“java代码”，这个时候可以使用以下语法格式：
 - `<%= %>` 注意：在=的后面编写要输出的内容。
 - `<%= %>` 这个符号会被翻译到哪里？最终翻译成什么？
 - 翻译成了这个java代码： `out.print();`
 - 翻译到service方法当中了。
 - 什么时候使用`<%= %>` 输出呢？输出的内容中含有java的变量，输出的内容是一个动态的内容，不是一个死的字符串。如果输出的是一个固定的字符串，直接在JSP文件中编写即可。
- 在JSP中如何编写JSP的专业注释
 - `<%--JSP的专业注释，不会被翻译到java源代码当中。--%>`
 -
- JSP基础语法总结：
 - JSP中直接编写普通字符串
 - 翻译到service方法的`out.write("这里")`

- `<% %>`
 - 翻译到service方法体内部，里面是一条一条的java语句。
- `<%! %>`
 - 翻译到service方法之外。
- `<%= %>`
 - 翻译到service方法体内部，翻译为：
`out.print();`
- `<%@page`
`contentType="text/html;charset=UTF-8"%>`
 - page指令，通过contentType属性用来设置响应的内容类型。
- 使用Servlet + JSP完成oa项目的改造。
 - 使用Servlet处理业务，收集数据。使用JSP展示数据。
 - 将之前原型中的html文件，全部修改为jsp，然后在jsp文件头部添加page指令（指定contentType防止中文乱码），将所有的JSP直接拷贝到web目录下。
 - 完成所有页面的正常流转。（页面仍然能够正常的跳转。修改超链接的请求路径。）
 - `<%=request.getContextPath() %>` 在JSP中动态的获取应用的根路径。

- Servlet中连接数据库，查询所有的部门，遍历结果集。
 - 遍历结果集的过程中，取出部门编号、部门名、位置等信息，封装成java对象。
 - 将java对象存放到List集合中。
 - 将List集合存储到request域当中。
 - 转发forward到jsp。
- 在JSP中：
 - 从request域当中取出List集合。
 - 遍历List集合，取出每个部门对象。动态生成tr。
- 思考一个问题：如果我只用JSP这一个技术，能不能开发web应用？
 - 当然可以使用JSP来完成所有的功能。因为JSP就是Servlet，在JSP的<%%>里面写的代码就是在service方法当中的，所以在<%%>当中完全可以编写JDBC代码，连接数据库，查询数据，也可以在这个方法当中编写业务逻辑代码，处理业务，都是可以的，所以使用单独的JSP开发web应用完全没问题。
 - 虽然JSP一个技术就可以完成web应用，但是不建议，还是建议采用servlet + jsp的方式进行开发。这样都能将各自的优点发挥出来。JSP就是做数据展示。Servlet就是做数据的收

集。（JSP中编写的Java代码越少越好。）一定要职责分明。

- JSP文件的扩展名必须是xxx.jsp吗？
 - jsp文件的扩展名是可以配置的。不是固定的。
 - 在CATALINA_HOME/conf/web.xml，在这个文件当中配置jsp文件的扩展名。

```
1 <servlet-mapping>
2     <servlet-name>jsp</servlet-
  name>
3     <url-pattern>*.jsp</url-
  pattern>
4     <url-pattern>*.jspx</url-
  pattern>
5 </servlet-mapping>
```

- xxx.jsp文件对于小猫咪来说，只是一个普通的文本文件，web容器会将xxx.jsp文件最终生成java程序，最终调用的是java对象相关的方法，真正执行的时候，和jsp文件就没有关系了。
 - 小窍门：JSP如果看不懂，建议把jsp翻译成java代码，就能看懂了。
- 同学问：包名bean是什么意思？
 - javabean（java的logo是一杯冒着热气的咖啡。javabean被翻译为：咖啡豆）

- java是一杯咖啡，咖啡又是由一粒一粒的咖啡豆研磨而成。
- 整个java程序中有很多bean的存在。由很多bean组成。
- 什么是javabean? 实际上javabean你可以理解为符合某种规范的java类，比如：
 - 有无参数构造方法
 - 属性私有化
 - 对外提供公开的set和get方法
 - 实现java.io.Serializable接口
 - 重写toString
 - 重写hashCode+equals
 -
- javabean其实就是java中的实体类。负责数据的封装。
- 由于javabean符合javabean规范，具有更强的通用性。
- 完成剩下所有功能的改造。
- 当前的oa应用存在的问题：
 - 任何一个用户都可以访问这个系统，都可以对这个系统当中的数据进行增删改这些危险的操作。我只想让合法的用户去使用这个系统，不合法的用户不能访问这个系统，怎么办？

- 加一个登录功能。登录成功的可以访问该系统，登录失败不能访问。
- 实现登录功能：
 - 步骤1：数据库当中添加一个用户表：t_user
 - t_user表当中存储的是用户的登录信息，最基本的也包括：登录的用户名和登录的密码。
 - 密码一般在数据库表当中存储的是密文。一般不以明文的形式存储。（这里先使用明文方式。）
 - 向t_user表中插入数据。
 - 步骤2：再实现一个登录页面。
 - 登录页面上应该有一个登录的表单。有用户名和密码输入的框。
 - 用户点击登录，提交表单，提交用户名和密码。form是post方式提交。
 - 步骤3：后台要有一个对应的Servlet来处理登录的请求。
 - 登录成功：跳转到部门列表页面。
 - 登录失败：跳转到失败的页面。
 - 步骤4：再提供一个登录失败的页面。
- 登录功能实现了，目前存在的最大的问题：

- 这个登录功能目前只是一个摆设，没有任何作用。只要用户知道后端的请求路径，照样可以在不登录的情况下访问。
- 这个登录没有真正起到拦截的作用。怎么解决？
- JSP的指令
 - 指令的作用：指导JSP的翻译引擎如何工作（指导当前的JSP翻译引擎如何翻译JSP文件。）
 - 指令包括哪些呢？
 - include指令：包含指令，在JSP中完成静态包含，很少用了。（这里不讲）
 - taglib指令：引入标签库的指令。这个到JSTL标签库的时候再学习。现在先不管。
 - page指令：目前重点学习一个page指令。
 - 指令的使用语法是什么？
 - <%@指令名 属性名=属性值 属性名=属性值 属性名=属性值....%>
 - 关于page指令当中都有哪些常用的属性呢？

- - 1 `<%@page session="true|false" %>`
 - 2 `true`表示启用JSP的内置对象`session`，表示一定启动`session`对象。没有`session`对象会创建。
 - 3 如果没有设置，默认值就是`session="true"`
 - 4 `session="false"` 表示不启动内置对象`session`。当前JSP页面中无法使用内置对象`session`。

- - 1 `<%@page contentType="text/json" %>`
 - 2 `contentType`属性用来设置响应的内容类型
 - 3 但同时也可以设置字符集。
 - 4 `<%@page
contentType="text/json;charset=UTF-8" %>`

- - 1 `<%@page pageEncoding="UTF-8" %>`
 - 2 `pageEncoding="UTF-8"` 表示设置响应时采用的字符集。

- - 1 `<%@page import="java.util.List,
java.util.Date,
java.util.ArrayList" %>`
 - 2 `<%@page import="java.util.*" %>`
 - 3 `import`语句，导包。

- - 1 <%@page errorPage="/error.jsp" %>
 - 2 当前页面出现异常之后，跳转到error.jsp页面。
 - 3 errorPage属性用来指定出错之后的跳转位置。

- - 1 <%@page isErrorPage="true" %>
 - 2 表示启用JSP九大内置对象之一：exception
 - 3 默认值是false。

- JSP的九大内置对象

- jakarta.servlet.jsp.PageContext pageContext 页面作用域
- jakarta.servlet.http.HttpServletRequest request 请求作用域
- jakarta.servlet.http.HttpSession session 会话作用域
- jakarta.servlet.ServletContext application 应用作用域
 - pageContext < request < session < application
 - 以上四个作用域都有：setAttribute、getAttribute、removeAttribute方法。
 - 以上作用域的使用原则：尽可能使用小的域。
- java.lang.Throwable exception
- jakarta.servlet.ServletConfig config

- java.lang.Object page （其实是this，当前的servlet对象）
- jakarta.servlet.jsp.JspWriter out （负责输出）
- jakarta.servlet.http.HttpServletResponse response （负责响应）

EL表达式

- EL表达式是干什么用的？
 - Expression Language （表达式语言）
 - EL表达式可以代替JSP中的java代码，让JSP文件中的程序看起来更加整洁，美观。
 - JSP中夹杂着各种java代码，例如<% java代码 %>、<%= %>等，导致JSP文件很混乱，不好看，不好维护。所以才有了后期的EL表达式。
 - EL表达式可以算是JSP语法的一部分。EL表达式属于JSP。
- EL表达式出现在JSP中主要是：
 - 从某个作用域中取数据，然后将其转换成字符串，然后将其输出到浏览器。这就是EL表达式的功效。
三大功效：
 - 第一功效：从某个域中取数据。
 - 四个域：

- pageContext
- request
- session
- application
- 第二功效：将取出的数据转成字符串。
 - 如果是一个java对象，也会自动调用java对象的toString方法将其转换成字符串。
- 第三功效：将字符串输出到浏览器。
 - 和这个一样：<%= %>，将其输出到浏览器。
- EL表达式很好用，基本的语法格式：
 - \${表达式}
- EL表达式的使用：

```
1 <%
2     // 创建User对象
3     User user = new User();
4     user.setUsername("jackson");
5     user.setPassword("1234");
6     user.setAge(50);
7
8     // 将User对象存储到某个域当中。一定要
    存，因为EL表达式只能从某个范围中取数据。
9     // 数据是必须存储到四大范围之一的。
10    request.setAttribute("userObj",
    user);
11 %>
```

12

13 `<%--使用EL表达式取--%>`

14 `${这个位置写什么? ? ? ? 这里写的一定是存储到域对象当中时的name}`

15 要这样写:

16 `${userObj}`

17 等同于java代码:

`<%=request.getAttribute("userObj")%>`

18 你不要这样写: `${"userObj"}`

19

20 面试题:

21 `${abc}` 和 `${"abc"}`的区别是什么?

22 `${abc}`表示从某个域中取出数据, 并且被取的这个数据的name是"abc", 之前一定有这样的代码: 域.`setAttribute("abc", 对象)`;

23 `${"abc"}` 表示直接将"abc"当做普通字符串输出到浏览器。不会从某个域中取数据了。

24

25 `${userObj}` 底层是怎么做的? 从域中取数据, 取出user对象, 然后调用user对象的toString方法, 转换成字符串, 输出到浏览器。

26

27 `<%--如果想输出对象的属性值, 怎么办? --%>`

28 `${userObj.username}` 使用这个语法的前提是: User对象有getUsername()方法。

29 `${userObj.password}` 使用这个语法的前提是: User对象有getPassword()方法。

```
30  ${userObj.age} 使用这个语法的前提是：  
    User对象有getAge()方法。  
31  ${userObj.email} 使用这个语法的前提是：  
    User对象有getEmail()方法。  
32  EL表达式中的. 这个语法，实际上调用了底层的  
    getXxx()方法。  
33  注意：如果没有对应的get方法，则出现异常。报  
    500错误。  
34  
35  ${userObj.addr222.zipcode}  
36  以上EL表达式对应的java代码：  
37  user.getAddr222().getZipcode()
```

- EL表达式优先从小范围中读取数据。
 - pageContext < request < session < application
- EL表达式中有四个隐含的隐式的范围：
 - pageScope 对应的是 pageContext范围。
 - requestScope 对应的是 request范围。
 - sessionScope 对应的是 session范围。
 - applicationScope 对应的是 application范围。
- EL表达式对null进行了预处理。如果是null，则向浏览器输出一个空字符串。
- EL表达式取数据的时候有两种形式：
 - 第一种：. （大部分使用这种方式）

- 第二种：[]（如果存储到域的时候，这个name中含有特殊字符，可以使用[]）
 - request.setAttribute("abc.def", "zhangsan");
 - \${requestScope.abc.def} 这样是无法取值的。
 - 应该这样：\${requestScope["abc.def"]}
- 掌握使用EL表达式，怎么从Map集合中取数据：
 - \${map.key}
- 掌握使用EL表达式，怎么从数组和List集合中取数据：
 - \${数组[0]}
 - \${数组[1]}
 - \${list[0]}
- page指令当中，有一个属性，可以忽略EL表达式

- - 1 <%@page
contentType="text/html; charset=UTF-8" isELIgnored="true" %>
 - 2 isELIgnored="true" 表示忽略EL表达式
 - 3 isELIgnored="false" 表示不忽略EL表达式。（这是默认值）
 - 4
 - 5 isELIgnored="true" 这个是全球的控制。
 - 6
 - 7 可以使用反斜杠进行局部控制：
\\${username} 这样也可以忽略EL表达式。

○ 通过EL表达式获取应用的根：

- \${pageContext.request.contextPath}

○ EL表达式中其他的隐式对象：

- pageContext
- param
- paramValues
- initParam

○ EL表达式的运算符

- 算术运算符

- +、-、*、/、%

- 关系运算符

- == eq != > >= < <=

- 逻辑运算符

- ! && || not and or
- 条件运算符
 - ?:
- 取值运算符
 - []和.
- empty运算符
 - empty运算符的结果是boolean类型
 - \${empty param.username}
 - \${not empty param.username}
 - \${!empty param.password}

JSTL标签库

- 什么是JSTL标签库?
 - Java Standard Tag Lib (Java标准的标签库)
 - JSTL标签库通常结合EL表达式一起使用。目的是让JSP中的java代码消失。
 - 标签是写在JSP当中的，但实际上最终还是要执行对应的java程序。（java程序在jar包当中。）
- 使用JSTL标签库的步骤：
 - 第一步：引入JSTL标签库对应的jar包。
 - tomcat10之后引入的jar包是：

- jakarta.servlet.jsp.jstl-2.0.0.jar
- jakarta.servlet.jsp.jstl-api-2.0.0.jar
- 在IDEA当中怎么引入?
 - 在WEB-INF下新建lib目录，然后将jar包拷贝到lib当中。然后将其“Add Lib...”
 - 一定是要和mysql的数据库驱动一样，都是放在WEB-INF/lib目录下的。
 - 什么时候需要将jar包放到WEB-INF/lib目录下？如果这个jar是tomcat服务器没有的。
- 第二步：在JSP中引入要使用标签库。（使用taglib指令引入标签库。）
 - JSTL提供了很多种标签，你要引入哪个标签？ ？ ？ ？ 重点掌握核心标签库。
 - ```
1 <%@taglib prefix="c"
 uri="http://java.sun.com/jsp/jstl/core" %>
2 这个就是核心标签库。
3 prefix="这里随便起一个名字就行了，核心标签库，大家默认的叫做c，你随意。"
```
- 第三步：在需要使用标签的位置使用即可。表面使用的是标签，底层实际上还是java程序。
- JSTL标签的原理

- - 1 `<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
  - 2 以上uri后面的路径实际上指向了一个xxx.tld文件。
  - 3 tld文件实际上是一个xml配置文件。
  - 4 在tld文件中描述了“标签”和“java类”之间的关系。
  - 5 以上核心标签库对应的tld文件是：c.tld文件。它在哪里。
  - 6 在jakarta.servlet.jsp.jstl-2.0.0.jar里面META-INF目录下，有一个c.tld文件。

- 源码解析：配置文件tld解析

- - 1 `<tag>`
  - 2     `<description>`对该标签的描述
  - 3     `</description>`
  - 4     `<name>`catch`</name>` 标签的名字
  - 5     `<tag-class>`org.apache.taglibs.standard.tag.common.core.CatchTag`</tag-class>` 标签对应的java类。
  - 6     `<body-content>`JSP`</body-content>` 标签体当中可以出现的内容，如果是JSP，就表示标签体中可以出现符合JSP所有语法的代码。例如EL表达式。
  - 7     `<attribute>`
  - 8         `<description>`
  - 9         对这个属性的描述

```

9 </description>
10 <name>var</name> 属性名
11 <required>false</required>
 false表示该属性不是必须的。true表示该属性
 是必须的。
12
 <rtexprvalue>false</rtexprvalue>
 这个描述说明了该属性是否支持EL表达式。
 false表示不支持。true表示支持EL表达式。
13 </attribute>
14 </tag>
15
16 <c:catch var="">
17 JSP....
18 </c:catch>

```

○ jstl中的核心标签库core当中有哪些常用的标签呢?

- c:if

- <c:if test="boolean类型, 支持EL表达式">  
</c: if>

- c:forEach

- <c:forEach items="集合, 支持EL表达式"  
var="集合中的元素" varStatus="元素状态对象"> \${元素状态对象.count} </c: forEach>

- <c:forEach var="i" begin="1" end="10"  
step="2"> \${i} </c: forEach>

- c:choose c:when c:otherwise

```
1 <c:choose>
2 <c:when test="${param.age <
3 18}">
4 青少年
5 </c:when>
6 <c:when test="${param.age <
7 35}">
8 青年
9 </c:when>
10 <c:when test="${param.age <
11 55}">
12 中年
13 </c:when>
14 <c:otherwise>
15 老年
16 </c:otherwise>
17 </c:choose>
```

## 改造OA

- 使用什么技术改造呢？
  - Servlet + JSP + EL表达式 + JSTL标签。进行改造。
- 在前端HTML代码中，有一个标签，叫做base标签，这个标签可以设置整个网页的基础路径。

- 这是Java的语法，也不是JSP的语法。是HTML中的一个语法。HTML中的一个标签。通常出现在head标签中。
- `< base href="http://localhost:8080/oa/">`
- 在当前页面中，凡是路径没有以“/”开始的，都会自动将base中的路径添加到这些路径之前。
  - `< a href="ab/def"></ a>`
  - 等同于：`< a href="http://localhost:8080/oa/a/b/def"></ a>`
- 需要注意：在JS代码中的路径，保险起见，最好不要依赖base标签。JS代码中的路径最好写上全路径。
- ```
1 <base
  href="${pageContext.request.scheme}:/
  /${pageContext.request.serverName}:${
  pageContext.request.serverPort}${page
  Context.request.contextPath}"/>
```

Filter过滤器

- 当前的OA项目存在什么缺陷？
 - DeptServlet、EmpServlet、OrderServlet。每一个Servlet都是处理自己相关的业务。在这些Servlet执行之前都是需要判断用户是否登录了。如果用户登录了，可以继续操作，如果没有登录，需

要用户登录。这段判断用户是否登录的代码是固定的，并且在每一个Servlet类当中都需要编写，显然代码没有得到重复利用。包括每一个Servlet都要解决中文乱码问题，也有公共的代码。这些代码目前都是重复编写，并没有达到复用。怎么解决这个问题？

- 可以使用Servlet规范中的Filter过滤器来解决这个问题。
- Filter是什么，有什么用，执行原理是什么？
 - Filter是过滤器。
 - Filter可以在Servlet这个目标程序执行之前添加代码。也可以在目标Servlet执行之后添加代码。之前之后都可以添加过滤规则。
 - 一般情况下，都是在过滤器当中编写公共代码。
- 一个过滤器怎么写呢？
 - 第一步：编写一个Java类实现一个接口：`javax.servlet.Filter`。并且实现这个接口当中所有的方法。
 - `init`方法：在Filter对象第一次被创建之后调用，并且只调用一次。
 - `doFilter`方法：只要用户发送一次请求，则执行一次。发送N次请求，则执行N次。在这个方法中编写过滤规则。

- destroy方法：在Filter对象被释放/销毁之前调用，并且只调用一次。
- 第二步：在web.xml文件中对Filter进行配置。这个配置和Servlet很像。

```
1 <filter>
2     <filter-name>filter2</filter-
   name>
3     <filter-
   class>com.bjpowernode.javaweb.servl
   et.Filter2</filter-class>
4 </filter>
5 <filter-mapping>
6     <filter-name>filter2</filter-
   name>
7     <url-pattern>*.do</url-pattern>
8 </filter-mapping>
```

- 或者使用注解：@WebFilter({"*.do"})
- 注意：
 - Servlet对象默认情况下，在服务器启动的时候是不会新建对象的。
 - Filter对象默认情况下，在服务器启动的时候会新建对象。
 - Servlet是单例的。Filter也是单例的。（单实例。）
- 目标Servlet是否执行，取决于两个条件：

- 第一：在过滤器当中是否编写了：
`chain.doFilter(request, response);` 代码。
- 第二：用户发送的请求路径是否和Servlet的请求路径一致。
- `chain.doFilter(request, response);` 这行代码的作用：
 - 执行下一个过滤器，如果下面没有过滤器了，执行最终的Servlet。
- 注意：Filter的优先级，天生的就比Servlet优先级高。
 - `/a.do` 对应一个Filter，也对应一个Servlet。那么一定是先执行Filter，然后再执行Servlet。
- 关于Filter的配置路径：
 - `/a.do`、`/b.do`、`/dept/save`。这些配置方式都是精确匹配。
 - `/*` 匹配所有路径。
 - `*.do` 后缀匹配。不要以 `/` 开始
 - `/dept/*` 前缀匹配。
- 在web.xml文件中进行配置的时候，Filter的执行顺序是什么？
 - 依靠filter-mapping标签的配置位置，越靠上优先级越高。
- 过滤器的调用顺序，遵循栈数据结构。

- 使用@WebFilter的时候，Filter的执行顺序是怎样的呢？
 - 执行顺序是：比较Filter这个类名。
 - 比如：FilterA和FilterB，则先执行FilterA。
 - 比如：Filter1和Filter2，则先执行Filter1。
- Filter的生命周期？
 - 和Servlet对象生命周期一致。
 - 唯一的区别：Filter默认情况下，在服务器启动阶段就实例化。Servlet不会。
- Filter过滤器这里有一个设计模式：
 - 责任链设计模式。
 - 过滤器最大的优点：
 - 在程序编译阶段不会确定调用顺序。因为Filter的调用顺序是配置到web.xml文件中的，只要修改web.xml配置文件中filter-mapping的顺序就可以调整Filter的执行顺序。显然Filter的执行顺序是在程序运行阶段动态组合的。那么这种设计模式被称为责任链设计模式。
 - 责任链设计模式最大的核心思想：
 - 在程序运行阶段，动态的组合程序的调用顺序。
- 使用过滤器改造OA项目。

Listener监听器

- 什么是监听器？
 - 监听器是Servlet规范中的一员。就像Filter一样。Filter也是Servlet规范中的一员。
 - 在Servlet中，所有的监听器接口都是以“Listener”结尾。
- 监听器有什么用？
 - 监听器实际上是Servlet规范留给我们javaweb程序员的特殊时机。
 - 特殊的时刻如果想执行这段代码，你需要想到使用对应的监听器。
- Servlet规范中提供了哪些监听器？
 - jakarta.servlet包下：
 - ServletContextListener
 - ServletContextAttributeListener
 - ServletRequestListener
 - ServletRequestAttributeListener
 - jakarta.servlet.http包下：
 - HttpSessionListener
 - HttpSessionAttributeListener

- 该监听器需要使用@WebListener注解进行标注。
- 该监听器监听的是什么？是session域中数据的变化。只要数据变化，则执行相应的方法。主要监测点在session域对象上。
- HttpSessionBindingListener
 - 该监听器不需要使用@WebListener进行标注。
 - 假设User类实现了该监听器，那么User对象在被放入session的时候触发bind事件，User对象从session中删除的时候，触发unbind事件。
 - 假设Customer类没有实现该监听器，那么Customer对象放入session或者从session删除的时候，不会触发bind和unbind事件。
- HttpSessionIdListener
 - session的id发生改变的时候，监听器中的唯一一个方法就会被调用。
- HttpSessionActivationListener
 - 监听session对象的钝化和活化的。
 - 钝化：session对象从内存存储到硬盘文件。
 - 活化：从硬盘文件把session恢复到内存。
- 实现一个监听器的步骤：以ServletContextListener为例。

- 第一步：编写一个类实现ServletContextListener接口。并且实现里面的方法。

- ```
1 void
 contextInitialized(ServletContextEvent event)
2 void
 contextDestroyed(ServletContextEvent event)
```

- 第二步：在web.xml文件中对ServletContextListener进行配置，如下：

- ```
1 <listener>
2   <listener-
  class>com.bjpowernode.javaweb.listener.MyServletContextListener</listener-
  class>
3 </listener>
```

- 当然，第二步也可以不使用配置文件，也可以用注解，例如：@WebListener
- 注意：所有监听器中的方法都是不需要javaweb程序员调用的，由服务器来负责调用？什么时候被调用呢？
 - 当某个特殊的事件发生（特殊的事件发生其实就是某个时机到了。）之后，被web服务器自动调用。
- 思考一个业务场景：

- 请编写一个功能，记录该网站实时的在线用户的个数。
- 我们可以通过服务器端有没有分配session对象，因为一个session代表了一个用户。有一个session就代表有一个用户。如果你采用这种逻辑去实现的话，session有多少个，在线用户就有多少个。这种方式的话：HttpSessionListener够用了。
session对象只要新建，则count++，然后将count存储到ServletContext域当中，在页面展示在线人数即可。
- 业务发生改变，只统计登录的用户的在线数量，这个该怎么办？
 - session.setAttribute("user", userObj);
 - 用户登录的标志是什么？session中曾经存储过User类型的对象。那么这个时候可以让User类型的对象实现HttpSessionBindingListener监听器，只要User类型对象存储到session域中，则count++，然后将count++存储到ServletContext对象中。页面展示在线人数即可。
- 实现oa项目中当前登录在线的人数。
 - 什么代表着用户登录了？
 - session.setAttribute("user", userObj); User类型的对象只要往session中存储过，表示有新用户登录。

- 什么代表着用户退出了?
 - `session.removeAttribute("user");` User类型的对象从session域中移除了。
 - 或者有可能是session销毁了。 (session超时)