**Servlet**

# 1 HTTP协议

## 协议的概念

规定被约束的对象需要遵守的规则。

## 1.2 HTTP协议

HTTP协议：超文本传输协议（HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的WWW文件都必须遵守这个标准。

HTTP协议规定 浏览器(客户端)向服务器发送 何种格式的数据. 服务器 会处理数据. 向浏览器(客户端)作出响应.(向客户端发送何种格式的数据)

HTTP协议的特点:

* HTTP协议遵守一个请求响应模型.

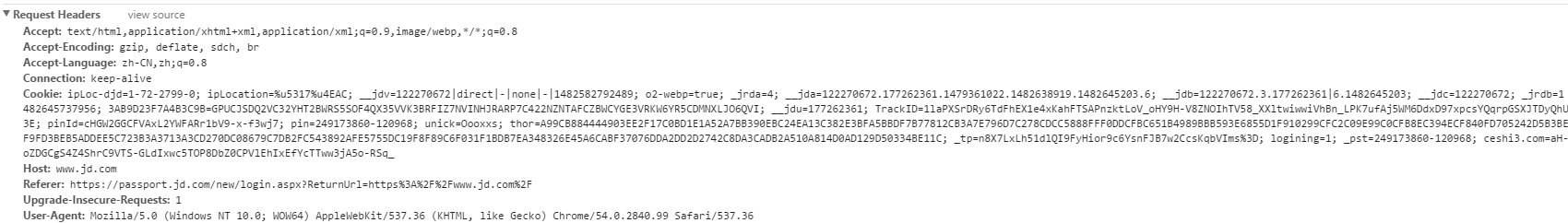
1. 请求和响应必须成对出现.
2. 必须先有请求后有响应.

* HTTP协议默认的端口:80

## 1.3 HTTP协议的请求部分

客户端向服务器发送的数据的格式：

GET请求方式抓包：



POST请求方式抓包：



【请求行】

|  |
| --- |
| 请求方式 请求路径 协议版本   * 请求方式：请求方式有很多种 常用的是GET和POST. * GET和POST区别?   GET：请求参数会显示到地址栏.GET方式有大小的限制.GET方式没有请求体  POST：请求参数不会显示到地址栏.在请求体中.POST没有大小限制.POST方式有请求体.   * 只有表单设置为method=”post”才是post请求.其他的都是get请求 |

【请求头】

|  |
| --- |
| 请求头通常都是key:value的键值对的形式.一般情况下一个key对应一个value但也有一个key对应多个value的情况.   * Referer :网站的来源.防盗链. * User-Agent :获得客户端浏览器的信息.(文件下载:IE:URL编码 火狐Base64) |

【请求体】

|  |
| --- |
| POST方式 提交的请求参数 |

## 1.4 HTTP协议的响应部分

服务器向客户端发送的数据的格式：

|  |
| --- |
| HTTP/1.1 200 OK  Server: Apache-Coyote/1.1  Accept-Ranges: bytes  ETag: W/"147-1455670867735"  Last-Modified: Wed, 17 Feb 2016 01:01:07 GMT  Content-Type: text/html  Content-Length: 147  Date: Wed, 17 Feb 2016 01:17:06 GMT  <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <h1>这是一个页面</h1>  </body>  </html> |

【响应行】

|  |
| --- |
| 协议版本 状态码 状态码描述  200:响应成功  302:重定向  304:查找本地缓存  404:浏览资源不存在.  500:服务器内部错误. |

【响应头】

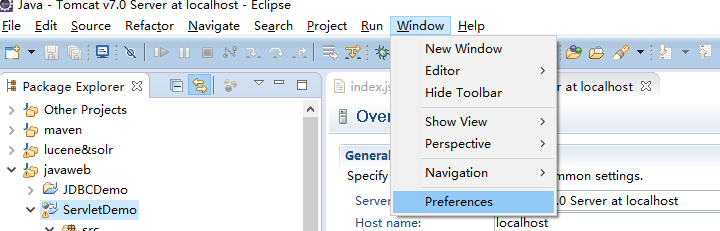
|  |
| --- |
| 一个key对应一个value，也有一个key对应多个value的头.  Last-Modified :最后的修改文件的事件.与If-Modified-Since一起使用.  Refresh :定时刷新.  Location :重定向的路径.  Content-Disposition:文件下载的时候使用的头信息.  禁用浏览器缓存:  Pragma  Expires  Cache-Control |

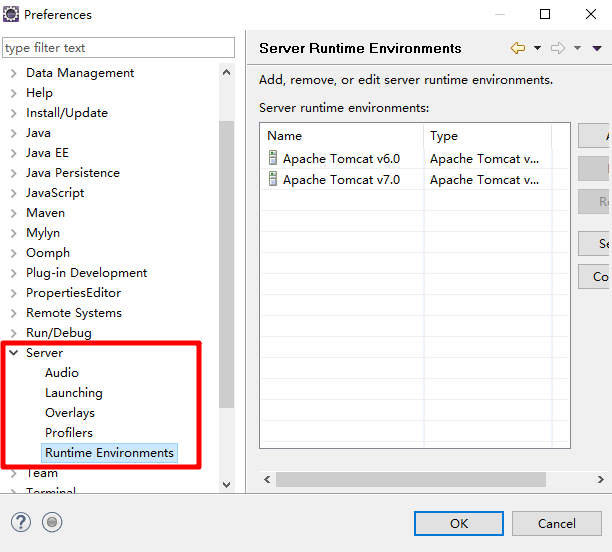
【响应体】

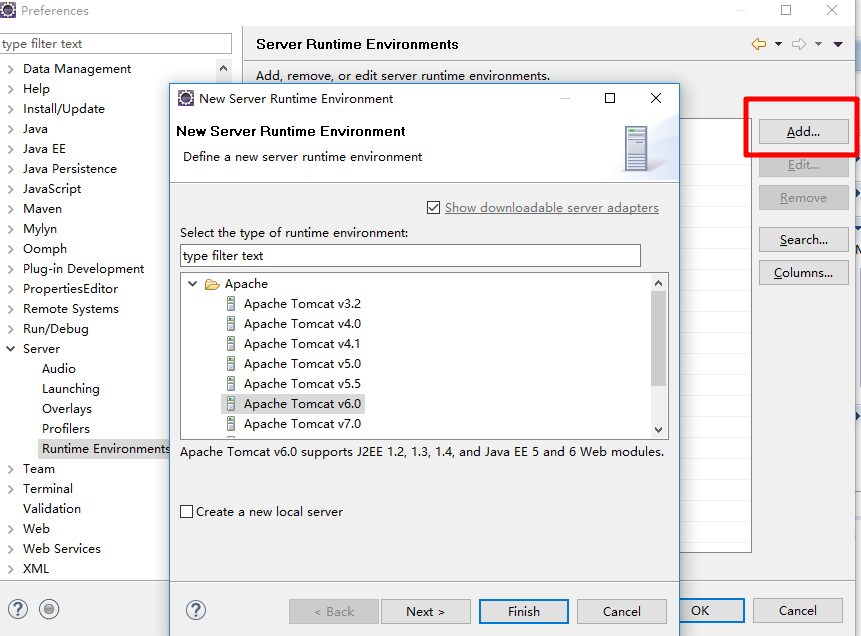
|  |
| --- |
| 页面要显示的内容. |

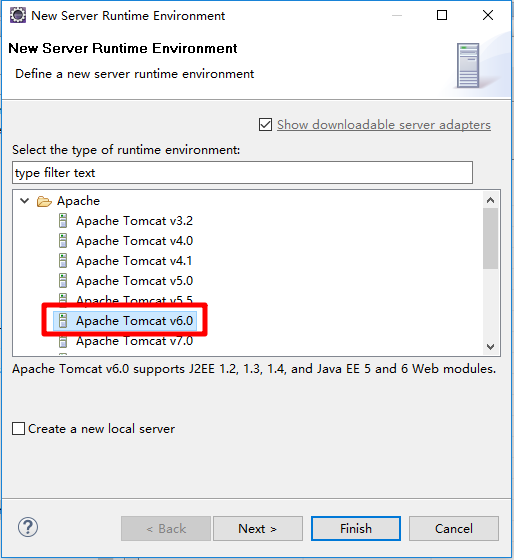
# 创建一个动态Servlet项目

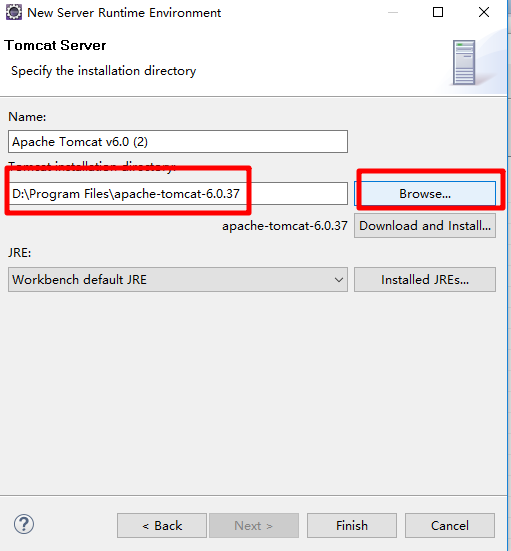
配置Eclipse的Tomcat和jdk:

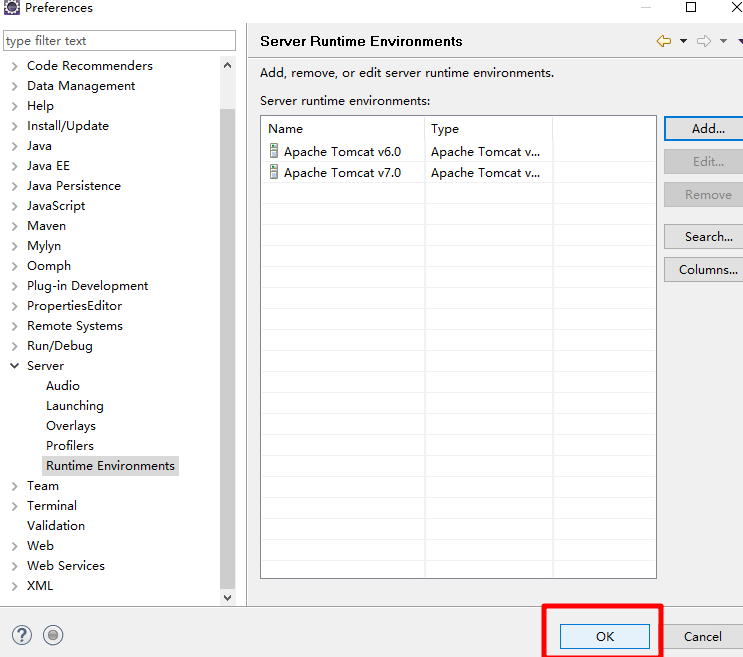




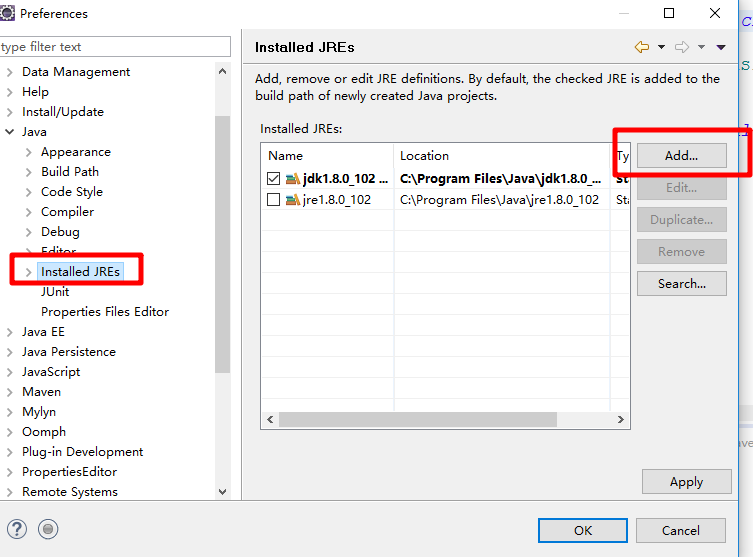


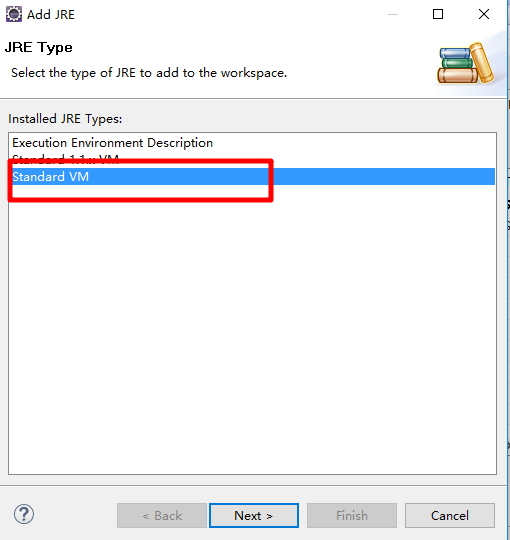


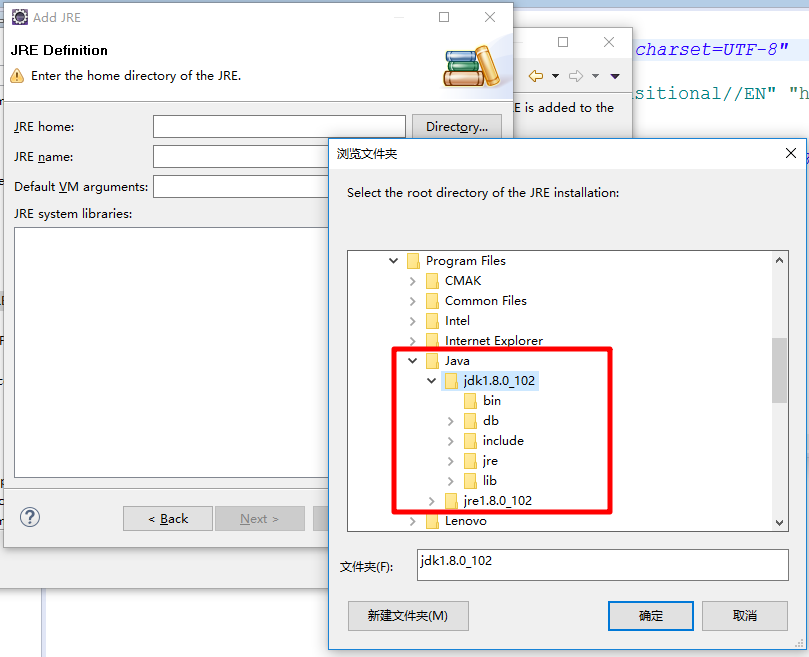


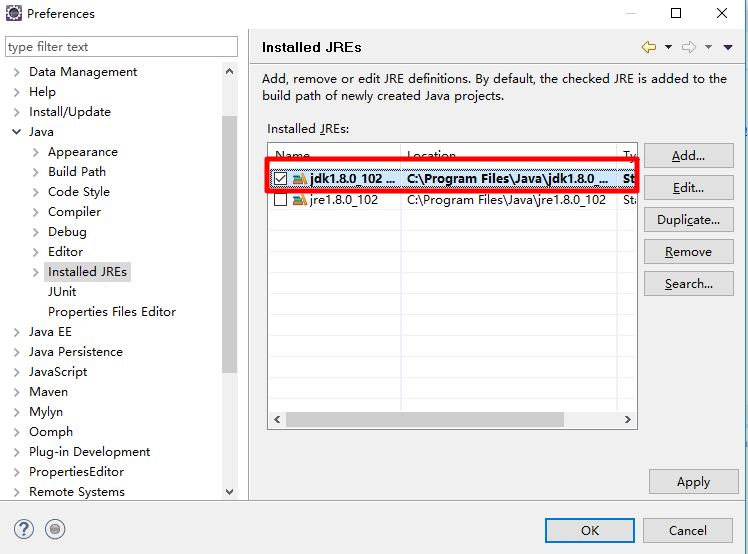


Eclipse jdk配置：

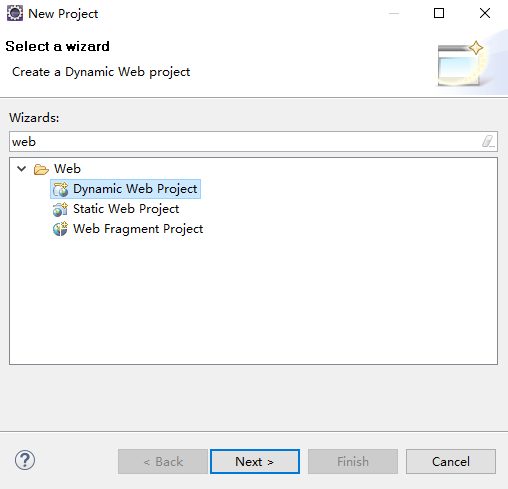


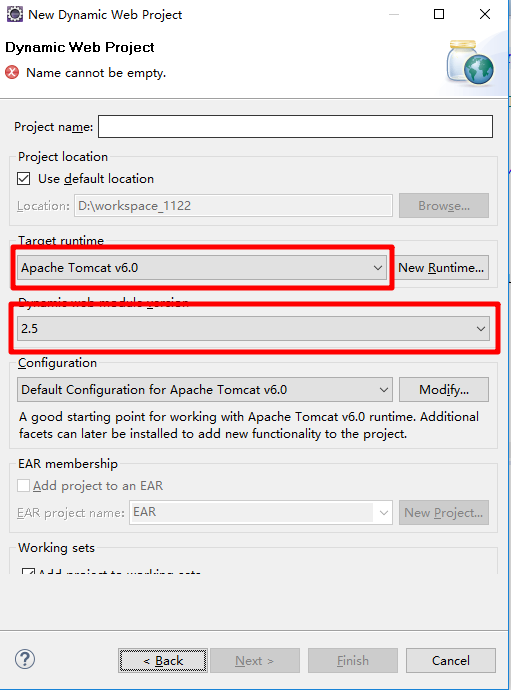


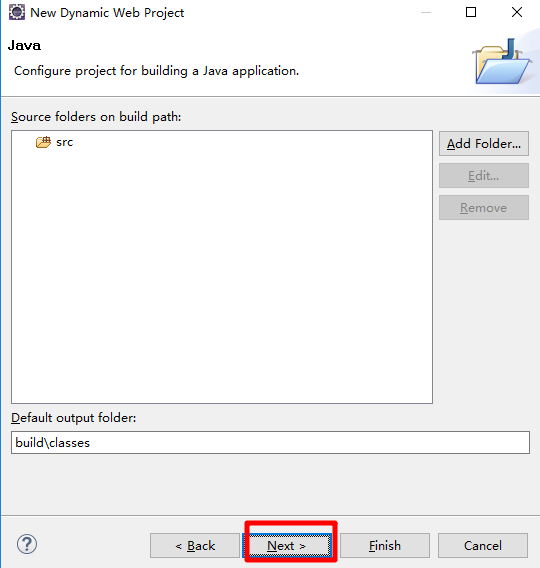


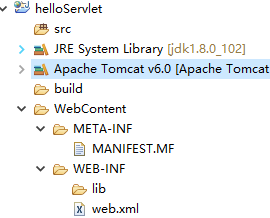


Eclipse创建 web项目：

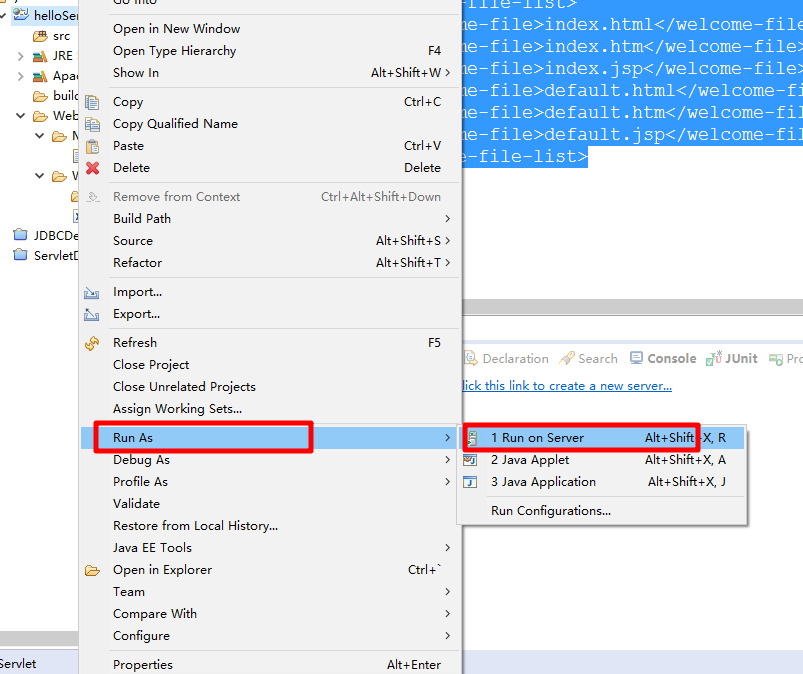








运行项目：



# Servlet简介

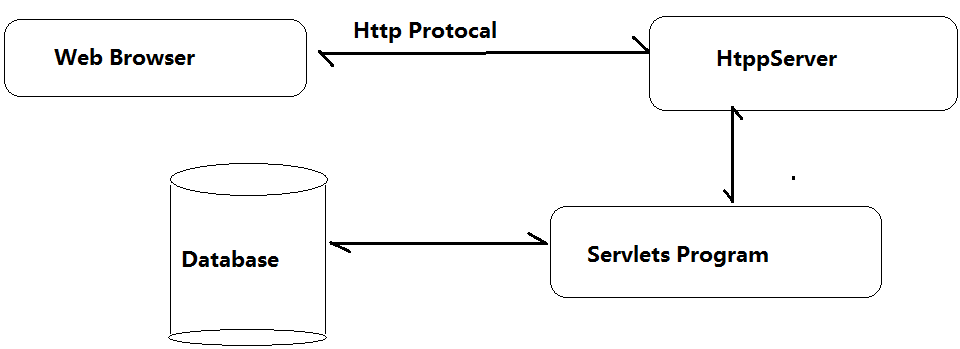
## 什么是Servlet

Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

使用 Servlet，可以收集来自网页表单的用户输入，呈现来自数据库或者其他源的记录，还可以动态创建网页。（Servlet的作用）

## Servlet架构

下图显示了Servlet在Web应用程序中的位置。

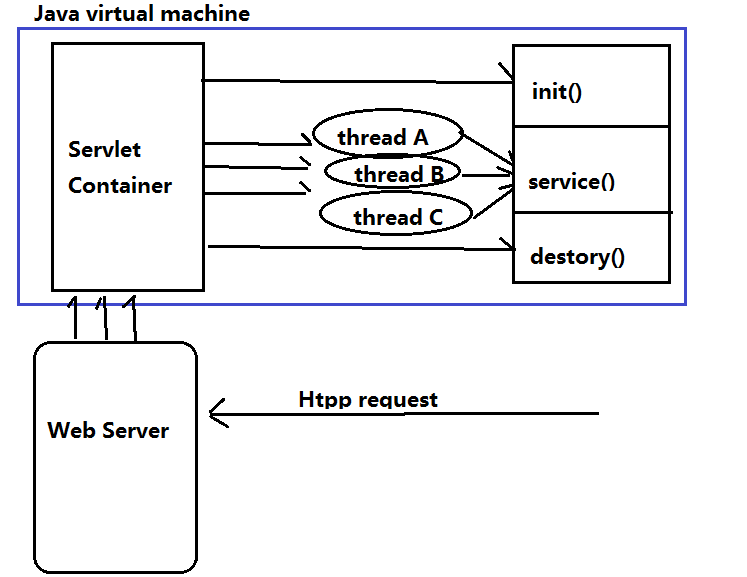


## Servlet的生命周期

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

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

架构图：



### init方法

init 方法被设计成只调用一次。它在第一次创建 Servlet 时被调用，在后续每次用户请求时不再调用。因此，它是用于一次性初始化。

Servlet 创建于用户第一次调用对应于该 Servlet 的 URL 时，但是您也可以指定 Servlet 在服务器第一次启动时被加载。

当用户调用一个 Servlet 时，就会创建一个 Servlet 实例，每一个用户请求都会产生一个新的线程，适当的时候移交给 doGet 或 doPost 方法。init() 方法简单地创建或加载一些数据，这些数据将被用于 Servlet 的整个生命周期。

### service方法

service() 方法是执行实际任务的主要方法。Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。

每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

service() 方法由容器调用，service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。

### destory方法

destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy() 方法可以让 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。

在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。

## 表单数据<form>

很多情况下，需要传递一些信息，从浏览器到 Web 服务器，最终到后台程序。浏览器使用两种方法可将这些信息传递到 Web 服务器，分别为 GET 方法和 POST 方法。

### GET方法

GET 方法向页面请求发送已编码的用户信息。页面和已编码的信息中间用 ? 字符分隔，如下所示：

<http://localhost:8080/ServletDemo/helloservlet?key1=value1&key2=value>

GET 方法是默认的从浏览器向 Web 服务器传递信息的方法，它会产生一个很长的字符串，出现在浏览器的地址栏中。如果您要向服务器传递的是密码或其他的敏感信息，不要使用 GET 方法。GET 方法有大小限制：请求字符串中最多只能有 1024 个字符。

Servlet 使用 doGet() 方法处理这种类型的请求。

### 2.4.2 POST方法

另一个向后台程序传递信息的比较可靠的方法是 POST 方法。POST 方法打包信息的方式与 GET 方法基本相同，但是 POST 方法不是把信息作为 URL 中，字符后的文本字符串进行发送，而是把这些信息作为一个单独的消息。消息以标准输出的形式传到后台程序，您可以解析和使用这些标准输出。Servlet 使用 doPost() 方法处理这种类型的请求。

### 使用Servlet读取表单数据

Servlet 处理表单数据，这些数据会根据不同的情况使用不同的方法自动解析：

* getParameter()：您可以调用 request.getParameter() 方法来获取表单参数的值。
* getParameterValues()：如果参数出现一次以上，则调用该方法，并返回多个值，例如复选框。
* getParameterNames()：如果您想要得到当前请求中的所有参数的完整列表，则调用该方法。

案例：

* 创建一个动态Web工程，编写一个Servlet，接受浏览器端的get请求参数和Post请求参数。

### 编写过滤器

Servlet 过滤器可以动态地拦截请求和响应，以变换或使用包含在请求或响应中的信息。

可以将一个或多个 Servlet 过滤器附加到一个 Servlet 或一组 Servlet。Servlet 过滤器也可以附加到 JavaServer Pages (JSP) 文件和 HTML 页面。调用 Servlet 前调用所有附加的 Servlet 过滤器。

Servlet 过滤器可以实现以下目的：

* 在客户端的请求访问后端资源之前，拦截这些请求。
* 在服务器的响应发送回客户端之前，处理这些响应。

过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类。javax.servlet.Filter 接口定义了三个方法：

|  |  |
| --- | --- |
| **序号** | **说明** |
| **1** | public void doFilter (ServletRequest, ServletResponse, FilterChain)  该方法完成实际的过滤操作，当客户端请求方法与过滤器设置匹配的URL时，Servlet容器将先调用过滤器的doFilter方法。FilterChain用户访问后续过滤器。 |
| **2** | public void init(FilterConfig filterConfig)  web 应用程序启动时，web 服务器将创建Filter 的实例对象，并调用其init方法，读取web.xml配置，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（filter对象只会创建一次，init方法也只会执行一次）。开发人员通过init方法的参数，可获得代表当前filter配置信息的FilterConfig对象。 |
| **3** | public void destroy()  Servlet容器在销毁过滤器实例前调用该方法，在该方法中释放Servlet过滤器占用的资源。 |

* FilterConfig 使用

Filter 的 init 方法中提供了一个 FilterConfig 对象。

如 web.xml 文件配置如下：

|  |
| --- |
| <!-- 配置过滤器 -->  <filter>  <filter-name>charsetFilter</filter-name>  <filter-class>cn.edu360.servlet.Filter.CharsetFilter</filter-class>  <init-param>  <param-name>charset</param-name>  <param-value>utf-8</param-value>  </init-param>  </filter>  <!-- 配置需要拦截的URL, \*表示所有 -->  <filter-mapping>  <filter-name>charsetFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

拦截器代码

|  |
| --- |
| package cn.edu360.servlet.Filter;  import java.io.IOException;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  public class CharsetFilter implements Filter {    private String charset = null;  @Override  public void destroy() {  // TODO Auto-generated method stub  }  @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)  throws IOException, ServletException {  System.out.println("拦截器过滤字符集格式.......");  request.setCharacterEncoding(charset);  response.setCharacterEncoding(charset);  response.setContentType("text/html;charset="+charset);    filterChain.doFilter(request, response);  }  @Override  public void init(FilterConfig filterConfig) throws ServletException {  charset = filterConfig.getInitParameter("charset");  System.out.println("设定字符集格式为UTF-8");  }  } |

注意：在web.xml中可以配置多个拦截器，当配置了多个拦截器时，拦截器的执行是有序的，即web.xml配置的越靠前，优先级越高。

web.xml拦截器配置各节点说明

* <filter>指定一个过滤器。
* <filter-name>用于为过滤器指定一个名字，该元素的内容不能为空。
* <filter-class>元素用于指定过滤器的完整的限定类名。
* <init-param>元素用于为过滤器指定初始化参数，它的子元素<param-name>指定参数的名字，<param-value>指定参数的值。
* 在过滤器中，可以使用FilterConfig接口对象来访问初始化参数。
* <filter-mapping>元素用于设置一个 Filter 所负责拦截的资源。一个Filter拦截的资源可通过两种方式来指定：Servlet 名称和资源访问的请求路径
* <filter-name>子元素用于设置filter的注册名称。该值必须是在<filter>元素中声明过的过滤器的名字
* <url-pattern>设置 filter 所拦截的请求路径(过滤器关联的URL样式)
* <servlet-name>指定过滤器所拦截的Servlet名称。

### 网页重定向

当文档移动到新的位置，我们需要向客户端发送这个新位置时，我们需要用到网页重定向。当然，也可能是为了负载均衡，或者只是为了简单的随机，这些情况都有可能用到网页重定向。

重定向请求到另一个网页的最简单的方式是使用 response 对象的 sendRedirect() 方法。

|  |
| --- |
| ……  request.setAttribute("msg", "网页跳转，当表单使用新增、修改、删除的时候不能使用该方法，可能造成表单数据重复提交问题");  // 网页跳转，当表单使用新增、修改、删除的时候不能使用该方法，可能造成表单数据重复提交问题  request.getRequestDispatcher("temp2.jsp").forward(request, response);    // 重定向  request.setAttribute("msg", "重定向");  response.sendRedirect("temp.jsp"); |

### HttpSession会话

HTTP 是一种"无状态"协议，这意味着每次客户端检索网页时，客户端打开一个单独的连接到 Web 服务器，服务器会自动不保留之前客户端请求的任何记录。

但是仍然有以下三种方式来维持 Web 客户端和 Web 服务器之间的 session 会话：

|  |  |  |
| --- | --- | --- |
| **1** | **Cookies** | 一个 Web 服务器可以分配一个唯一的 session 会话 ID 作为每个 Web 客户端的 cookie，对于客户端的后续请求可以使用接收到的 cookie 来识别。  这可能不是一个有效的方法，因为很多浏览器不支持 cookie，所以我们建议不要使用这种方式来维持 session 会话。 |
| **2** | 隐藏表单字段 | 一个 Web 服务器可以发送一个隐藏的 HTML 表单字段，以及一个唯一的 session 会话 ID，如下所示：  <input type="hidden" name="sessionid" value="12345">  这意味着，当表单被提交时，指定的名称和值会被自动包含在 GET 或 POST 数据中。每次当 Web 浏览器发送回请求时，session\_id 值可以用于保持不同的 Web 浏览器的跟踪。  这可能是一种保持 session 会话跟踪的有效方式，但是点击常规的超文本链接（<a href>）不会导致表单提交，因此隐藏的表单字段也不支持常规的 session 会话跟踪 |
| **3** | URL重写 | 可以在每个 URL 末尾追加一些额外的数据来标识 session 会话，服务器会把该 session 会话标识符与已存储的有关 session 会话的数据相关联。  例如，http://localhost:8080/ServletDemo/helloServlet;sessionid=12345，session 会话标识符被附加为 sessionid=12345，标识符可被 Web 服务器访问以识别客户端。  URL 重写是一种更好的维持 session 会话的方式，它在浏览器不支持 cookie 时能够很好地工作，但是它的缺点是会动态生成每个 URL 来为页面分配一个 session 会话 ID，即使是在很简单的静态 HTML 页面中也会如此。 |

* HttpSession 对象

除了上述的三种方式，Servlet 还提供了 HttpSession 接口，该接口提供了一种跨多个页面请求或访问网站时识别用户以及存储有关用户信息的方式。

Servlet 容器使用这个接口来创建一个 HTTP 客户端和 HTTP 服务器之间的 session 会话。会话持续一个指定的时间段，跨多个连接或页面请求。

通过调用 HttpServletRequest 的getSession()方法来获取 HttpSession 对象，如下所示：

HttpSession session = request.getSession();

下面总结了 HttpSession 对象中可用的几个重要的方法：

|  |  |
| --- | --- |
| **序号** | **描述** |
| **1** | public Object getAttribute(String name)  该方法返回在该 session 会话中具有指定名称的对象，如果没有指定名称的对象，则返回 null。 |
| **2** | public void removeAttribute(String name)  该方法将从该 session 会话移除指定名称的对象。 |
| **3** | public void setAttribute(String name, Object value)  该方法使用指定的名称绑定一个对象到该 session 会话。 |

* 删除 Session 会话数据

1. 移除一个特定的属性：调用 public void removeAttribute(String name) 方法来删除与特定的键相关联的值。
2. 删除整个 session 会话：调用 public void invalidate() 方法来丢弃整个 session 会话。
3. 设置 session 会话过期时间：您可以调用 public void setMaxInactiveInterval(int interval) 方法来单独设置 session 会话超时。
4. web.xml 配置：如果您使用的是 Tomcat，除了上述方法，您还可以在 web.xml 文件中配置 session 会话超时，如下所示：

<session-config>

<session-timeout>15</session-timeout>

</session-config>

上面实例中的超时时间是以分钟为单位，将覆盖 Tomcat 中默认的 30 分钟超时时间。

## Servlet文件上传

准备工作：

* 需要引入的 jar 文件：commons-fileupload-1.3.2、commons-io-2.5.jar。
* fileUpload.jsp: 文件上传页面
* uploadmsg.jsp: 文件上传后跳转页面
* FileUpload.java: 上传处理Servlet

FileUpload.java 上传处理Servlet

|  |
| --- |
| **package** cn.edu360.servlet;  **import** java.io.File;  **import** java.io.IOException;  **import** java.util.List;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.apache.commons.fileupload.FileItem;  **import** org.apache.commons.fileupload.disk.DiskFileItemFactory;  **import** org.apache.commons.fileupload.servlet.ServletFileUpload;  /\*\*  \* Servlet文件上传  \*/  **public** **class** FileUpload **extends** HttpServlet {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;    **public** FileUpload() {  **super**();  }  **protected** **void** service(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  // 检测是否为多媒体文件上传  **if**(!ServletFileUpload.*isMultipartContent*(request)) {  response.getWriter().append("Error: 表单必须包含 enctype=multipart/form-data");  **return**;  }  // 构建文件上传路径  String path = getServletContext().getRealPath(File.***separator*** + "upload");    // 获取磁盘文件条目工厂  DiskFileItemFactory factory = **new** DiskFileItemFactory();    /\*  \* setSizeThreshold 设置内存缓存大小  \* setRepository 设置临时文件存储目录，如果上传的文件超过设置的缓存大小，则会溢出成临时文件  \* 如果不设置这两个参数将会导致占用过多服务器资源  \*/  // 设置内存缓存大小，超过后将产生临时文件并存储于临时目录中  factory.setSizeThreshold(5\*1024\*1024); // 10MB 默认为1MB  // 设置临时目录, 文件在上传过程中会产生临时文件  factory.setRepository(**new** File(path));    ServletFileUpload fileUpload = **new** ServletFileUpload(factory);  // 设置最大文件上传值 10MB  fileUpload.setFileSizeMax(10 \* 1024 \* 1024);  // 设置最大请求值，包含文件和表单数据 20MB  fileUpload.setSizeMax(20 \* 1024 \* 1024);    // 判断上传的目录是否存在，不存在则创建  File uploadDir = **new** File(path);  **if** (!uploadDir.exists()) {  uploadDir.mkdirs();  }    // 解析请求的内容，提取文件数据  **try** {  List<FileItem> formItems = fileUpload.parseRequest(request);    **for**(FileItem item : formItems) {  // 如果是普通的form表单项，不处理，这里只处理上传的  **if**(!item.isFormField()){  String fileName = **new** File(item.getName()).getName();  String filePath = path + File.***separator*** + fileName;  System.***out***.println("文件上传存储路径："+filePath);    // 保存文件到磁盘  File storeFile = **new** File(filePath);  item.write(storeFile);  //request.setAttribute("uploadMessage", "文件上传成功！路径【"+filePath+"】");  request.getSession().setAttribute("uploadMessage", "文件上传成功！路径【"+filePath+"】");  }  }  } **catch** (Exception e) {  //request.setAttribute("uploadMessage", "文件上传失败！");  request.getSession().setAttribute("uploadMessage", "文件上传失败！");  e.printStackTrace();  }  //request.getRequestDispatcher("uploadmsg.jsp").forward(request, response);  response.sendRedirect("uploadmsg.jsp");  }  } |

fileUpload.jsp 文件上传页面

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Servlet文件上传</title>  </head>  <body>  <h1>文件上传实例 </h1>  <form method=*"post"* action=*"*${pageContext.request.contextPath }*/fileUpload"* enctype=*"multipart/form-data"*>  选择一个文件:  <input type=*"file"* name=*"uploadFile"* /><br>  选择一个文件:  <input type=*"file"* name=*"uploadFile"* />  <br/><br/>  <input type=*"submit"* value=*"上传"* />  </form>  </body>  </html> |

uploadmsg.jsp 上传后跳转页面

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>文件上传结果</title>  </head>  <body>  <h5>${requestScope.uploadMessage }</h5>  <h5>${sessionScope.uploadMessage }</h5>  </body>  </html> |

# JSP&JSTL&Ajax(掌握)

## 4.1 JSP

### 4.1.1 什么是JSP

JSP全名为Java Server Pages，中文名叫java服务器页面，其根本是一个简化的Servlet设计，是由Sun公司倡导、许多公司参与一起建立的一种动态网页技术标准。JSP技术有点类似ASP技术，它是在传统的网页HTML（标准通用标记语言的子集）文件(\*.htm,\*.html)中插入Java程序段(Scriptlet)和JSP标记(tag)，从而形成JSP文件，后缀名为(\*.jsp)。 用JSP开发的Web应用是跨平台的，既能在Linux下运行，也能在其他操作系统上运行。

### 4.1.2 JSP与HTML的区别和优势

HTML页面是静态页面，也就是事先由用户写好放在服务器上，JSP页面是动态页,有JSP容器执行该页面的Java代码部分然后实时生成的页面（JSP底层是Servlet实现的）

### 4.1.3 JSP语法

可以在JSP中使用java的代码，如引用其他的类需要导包

* 使用<%= %>输出内容
* 在<% %>编写java代码
* 导入依赖的jar包
* 导包：<%@page import=*"java.util.\*"*%>

|  |
| --- |
| 1. <%@page import=*"org.apache.commons.dbutils.handlers.BeanListHandler"*%> 2. <%@page import=*"cn.edu360.jdbc.DataPool"*%> 3. <%@page import=*"cn.edu360.domain.User"*%> 4. <%@page import=*"java.util.\*"*%> 5. <%@page import=*"org.apache.commons.dbutils.QueryRunner"*%> 6. <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"* pageEncoding=*"UTF-8"*%> 7. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"> 8. <html> 9. <head> 10. <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*> 11. <title>Insert title here</title> 12. </head> 13. <body> 14. <h1> 15. <% 16. out.println("你好世界"); 17. QueryRunner queryRunner = **new** QueryRunner(DataPool.getDataSource()); 18. String sql = "SELECT \* FROM user"; 19. List<User> users = queryRunner.query(sql, **new** BeanListHandler<User>(User.**class**)); 20. out.println(users.size()); 21. %> 22. </h1> 23. </body> 24. </html> |

### 4.1.4 JSP整合Servlet

1. 让Servlet接收请求，并处理业务逻辑，然后用jsp进行展现
2. 创建一个注册页面login.jsp
3. 提交注册信息后给UserServlet，用于处理用户相关的请求
4. 然后将所有用户的信息展现出来

### 4.1.5 JSP内置对象和域对象(了解)

#### 4.1.5.1 JSP中的9个内置对象

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象名称** | **对应Servlet中的对象** | **常用方法** |
| 1 | request | HttpServletRequest | getParameter(),setAttribute() |
| 2 | response | HttpServletResponse | sendRedirect(),getOutputStream(); |
| 3 | session | HttpSession | setAttribute(),getAttribute(), |
| 4 | application | ServletContext | setAttribute(),getAttribute() |
| 5 | page | Object | toString(),wait(), |
| 6 | pageContext | PageContext | setAttribute(),getAttribute() |
| 7 | out | JspWriter | write(),print(), |
| 8 | config | ServletConfig | getInitParamter() |
| 9 | exception | Throwable | getMessage(),getCause() |

1、request对象

request 对象是 javax.servlet.httpServletRequest类型的对象。 该对象代表了客户端的请求信息，主要用于接受通过HTTP协议传送到服务器的数据。（包括头信息、系统信息、请求方式以及请求参数等）。request对象的作用域为一次请求。

2、response对象

response 代表的是对客户端的响应，主要是将JSP容器处理过的对象传回到客户端。response对象也具有作用域，它只在JSP页面内有效。

3、session对象

session 对象是由服务器自动创建的与用户请求相关的对象。服务器为每个用户都生成一个session对象，用于保存该用户的信息，跟踪用户的操作状态。session对象内部使用Map类来保存数据，因此保存数据的格式为 “Key/value”。 session对象的value可以使复杂的对象类型，而不仅仅局限于字符串类型。

4、application对象

application 对象可将信息保存在服务器中，直到服务器关闭，否则application对象中保存的信息会在整个应用中都有效。与session对象相比，application对象生命周期更长，类似于系统的“全局变量”。

5、out 对象

out 对象用于在Web浏览器内输出信息，并且管理应用服务器上的输出缓冲区。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据，为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

6、pageContext 对象

pageContext 对象的作用是取得任何范围的参数，通过它可以获取 JSP页面的out、request、reponse、session、application 等对象。pageContext对象的创建和初始化都是由容器来完成的，在JSP页面中可以直接使用 pageContext对象。

7、config 对象

config 对象的主要作用是取得服务器的配置信息。通过 pageConext对象的 getServletConfig() 方法可以获取一个config对象。当一个Servlet 初始化时，容器把某些信息通过 config对象传递给这个 Servlet。 开发者可以在web.xml 文件中为应用程序环境中的Servlet程序和JSP页面提供初始化参数。

8、page 对象

page 对象代表JSP本身，只有在JSP页面内才是合法的。 page隐含对象本质上包含当前 Servlet接口引用的变量，类似于Java编程中的 this 指针。

9、exception 对象

exception 对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的JSP页面中使用该对象将无法编译JSP文件。excepation对象和Java的所有对象一样，都具有系统提供的继承结构。exception 对象几乎定义了所有异常情况。在Java程序中，可以使用try/catch关键字来处理异常情况； 如果在JSP页面中出现没有捕获到的异常，就会生成 exception 对象，并把 exception 对象传送到在page指令中设定的错误页面中，然后在错误页面中处理相应的 exception 对象。

#### 4.1.5.2 JSP中的4个域对象

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **对象名称** | **范围** | **中文翻译** |
| 1 | pageContext | pageScope | 页面范围 |
| 2 | request | requestScope | 请求范围 |
| 3 | session | sessionScope | 会话范围 |
| 4 | application | applicationScope | 应用范围 |

Request.setAttribuite(“jsplist”,user);

Request......forwed(req,resp)

用户名：${sessionScope.jsplist.account}

1、如果把变量放到pageContext里，就说明它的作用域是page，它的有效范围只在当前jsp页面里。 从把变量放到pageContext开始，到jsp页面结束，你都可以使用这个变量。

2、如果把变量放到request里，就说明它的作用域是request，它的有效范围是当前请求周期。所谓请求周期，就是指从http请求发起，到服务器处理结束，返回响应的整个过程。在这个过程中可能使用forward的方式跳转了多个jsp页面，在这些页面里你都可以使用这个变量。

3、如果把变量放到session里，就说明它的作用域是session，它的有效范围是当前会话。所谓当前会话，就是指从用户打开浏览器开始，到用户关闭浏览器这中间的过程。这个过程可能包含多个请求响应。也就是说，只要用户不关浏览器，服务器就有办法知道这些请求是一个人发起的，整个过程被称为一个会话（session），而放到会话中的变量。

4、如果把变量放到application里，就说明它的作用域是application，它的有效范围是整个应用。整个应用是指从应用启动，到应用结束。我们没有说“从服务器启动，到服务器关闭”是因为一个服务器可能部署多个应用，当然你关闭了服务器，就会把上面所有的应用都关闭了。application作用域里的变量，它们的存活时间是最长的，如果不进行手工删除，它们就一直可以使用。与上述三个不同的是，application里的变量可以被所有用户共用。如果用户甲的操作修改了application中的变量，用户乙访问时得到的是修改后的值。这在其他scope中都是不会发生的，page, request, session都是完全隔离的，无论如何修改都不会影响其他

## 4.2 EL表达式（掌握）

### 4.2.1 什么是EL表达式

EL全名Expression Language，EL语法很简单，它最大的特点就是可以从相应的作用域中**取值**和进行简单的**逻辑运算**

### 4.2.2 EL表达式的语法格式

所有EL都是以${为起始、以}为结尾的。

<%=user.getGender( ) ;%> 等价于 ${user.gender}

${sessionScope.user.gender}

使用java语法获得属性

|  |
| --- |
| User user =(User)session.getAttribute("user");  String nickname =user.getNickname(); |

EL存取变量数据的方法很简单，例如：${username}。它的意思是取出某一范围中名称为username的变量。

因为我们并没有指定哪一个范围的username，所以它会依序从Page、Request、Session、Application范围查找。假如途中找到username，就直接回传，不再继续找下去。

### 4.2.3 EL中的运算符

1.算术运算符有五个：+ - \* /

2.关系运算符有六个：==或eq !=或ne <或lt >或gt <=或le >=或ge

3.逻辑运算符有三个：&&或and ||或or !或not

4.其它运算符有三个：Empty运算符、条件运算符、()运算符

例：${empty param.name}、${A?B:C}、${A\*(B+C)}

## 4.3 JSTL

### 4.3.1 什么是JSTL

JSTL（JSP Standard Tag Library，JSP标准标签库)是一个不断完善的开放源代码的JSP标签库，是由apache的jakarta小组来维护的。

### 4.3.2 JSTL的优点

比EL表达式更加强大，在展现层取值和计算更加方便。

### 4.3.3 JSTL的标签库

JSTL中有五大标签库:core、fmt、fn、xml、sql

### 4.2.4 如何使用JSTL

使用步骤：

1.导入依赖jar包

2引入标签库，给标签库取别名

3.在JSP中使用

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>

|  |
| --- |
| <c:if test="${requestScope.username == 'admin' }"> |

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/fmt"* prefix=*"fmt"* %>

|  |
| --- |
| <%@page import=*"java.util.Date"*%>  <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <%@ taglib prefix=*"fmt"* uri=*"http://java.sun.com/jsp/jstl/fmt"* %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>JSTL实例演示</title>  </head>  <body>  <c:set var=*"data"* value=*"*<%=**new** Date() %>*"*></c:set>  <h4>formatDate默认 =><fmt:formatDate value=*"*${data }*"* /></h4>  <h4>formatDate type="time" =><fmt:formatDate value=*"*${data }*"* type=*"time"*/></h4>  <h4>formatDate type="both" =><fmt:formatDate value=*"*${data }*"* type=*"both"*/></h4>  <h4>formatDate pattern="yyyy-MM-dd HH:mm:ss" =><fmt:formatDate value=*"*${data }*"* pattern=*"yyyy-MM-dd HH:mm:ss"*/></h4>  <h4>formatDate type="both" dateStyle="long" timeStyle="long" =><fmt:formatDate value=*"*${data }*"* type=*"both"* dateStyle=*"long"* timeStyle=*"long"*/></h4>  <h4>formatDate type="both" dateStyle="medium" timeStyle="medium" =><fmt:formatDate type=*"both"* dateStyle=*"medium"* timeStyle=*"medium"* value=*"*${data}*"* /></h4>  </body>  </html> |

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/functions"* prefix=*"fn"* %>

|  |
| --- |
| ${ fn:length("HelloWorld") }  ${ fn:toLowerCase("HADOOP") }  ${ fn:toUpperCase("spark") }  ${ fn:contains("www.apache.org","apache") } |

## Ajax

AJAX即“Asynchronous Javascript And XML”（异步JavaScript和XML），是指一种创建交互式网页应用的网页开发技术，即不用刷新页面即可更改页面中的内容。

案例：JQuery Ajax实现二级级联

AjaxServlet.java

|  |
| --- |
| **package** cn.edu360.servlet;  **import** java.io.IOException;  **import** java.io.PrintWriter;  **import** java.util.ArrayList;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** com.alibaba.fastjson.JSONObject;  **import** cn.edu360.beans.City;  /\*\*  \* Ajax 实现二级级联  \*/  **public** **class** AjaxServlet **extends** HttpServlet {  /\*\*  \* K:省份Id, Value:地市集合  \*/  **private** Map<String, List<City>> provincesCity = **null**;  /\*\*  \* 改方法只会被调用一次  \*/  @Override  **public** **void** init() **throws** ServletException {  provincesCity = **new** HashMap<>();  List<City> bjCity = **new** ArrayList<>();  City cp1\_1 = **new** City("11", "朝阳区");  City cp1\_2 = **new** City("12", "海淀区");  City cp1\_3 = **new** City("13", "东城区");  City cp1\_4 = **new** City("14", "西城区");  bjCity.add(cp1\_1);  bjCity.add(cp1\_2);  bjCity.add(cp1\_3);  bjCity.add(cp1\_4);  provincesCity.put("1", bjCity);    List<City> hnCity = **new** ArrayList<>();  City hnc1 = **new** City("21", "郑州市");  City hnc2 = **new** City("22", "信阳市");  City hnc3 = **new** City("23", "洛阳市");  City hnc4 = **new** City("34", "开封市");  hnCity.add(hnc1);  hnCity.add(hnc2);  hnCity.add(hnc3);  hnCity.add(hnc4);  provincesCity.put("3", hnCity);  }  **protected** **void** service(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  // 接收请求传递过来的参数， 参数名称provinceId  String provinceId = request.getParameter("provinceId");  List<City> cities = provincesCity.get(provinceId);  String citiesJsonStr = cities == **null**? "": JSONObject.*toJSONString*(cities);    PrintWriter out = response.getWriter();  out.write(citiesJsonStr);  out.flush();  out.close();  }  } |

Ajax.jsp

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"*  pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <script type=*"text/javascript"* src=*"js/jquery-1.11.1.min.js"*></script>  <title>二级联动-Ajax</title>  <script type=*"text/javascript"*>  $(document).ready(**function**(){  $("#provinceId").change(**function**(){  $.ajax({  type: "GET", // 请求方式  url: "${pageContext.request.contextPath}/ajaxServlet", // 请求的url  data: "provinceId="+$(**this**).val(), // 请求携带的参数  success: **function**(data){ // 服务器端响应结果  // alert( "Data: " +data);  $("#cityId").empty();  **if**(""!=data){  $.each(JSON.parse(data), **function**(index, obj){  // alert(this.cityId + " " + this.cityName);  $("<option value='"+obj.cityId+"'>"+obj.cityName+"</option>").appendTo($("#cityId"));  });  } **else** {  alert("没有找到匹配项！");  }  }  });  });  });  </script>  </head>  <body>  <br>  <h1>下拉列表的Ajax二级联动<h1>  省份：<select id=*"provinceId"*>  <option value=*"0"*>--请选择--</option>  <option value=*"1"*>北京市</option>  <option value=*"2"*>山西省</option>  <option value=*"3"*>河南省</option>  <option value=*"4"*>湖北省</option>  </select>&nbsp;&nbsp;&nbsp;&nbsp;  地市：<select id=*"cityId"*></select>  </body>  </html> |

# MVC设计模式

**MVC模式**（Model-View-Controller）是[软件工程](http://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E5%B7%A5%E7%A8%8B" \t "_blank" \o "软件工程)中的一种[软件架构](http://zh.wikipedia.org/wiki/%E8%BD%AF%E4%BB%B6%E6%9E%B6%E6%9E%84" \t "_blank" \o "软件架构)模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

MVC模式最早由[Trygve Reenskaug](http://zh.wikipedia.org/w/index.php?title=Trygve_Reenskaug&action=edit&redlink=1" \t "_blank" \o "Trygve Reenskaug（页面不存在）)在1978年提出[[1]](http://zh.wikipedia.org/wiki/MVC" \l "cite_note-1" \t "_blank) ，是[施乐帕罗奥多研究中心](http://zh.wikipedia.org/wiki/%E5%B8%95%E7%BE%85%E5%A5%A7%E5%A4%9A%E7%A0%94%E7%A9%B6%E4%B8%AD%E5%BF%83" \t "_blank" \o "帕罗奥多研究中心)（Xerox PARC）在20世纪80年代为程序语言[Smalltalk](http://zh.wikipedia.org/wiki/Smalltalk" \t "_blank" \o "Smalltalk)发明的一种软件设计模式。**MVC模式**的目的是实现一种动态的程序设计，使后续对程序的修改和扩展简化，并且使程序某一部分的重复利用成为可能。除此之外，此模式通过对复杂度的简化，使程序结构更加直观。软件系统通过对自身基本部分分离的同时也赋予了各个基本部分应有的功能。专业人员可以通过自身的专长分组：

* （控制器Controller）- 负责转发请求，对请求进行处理。
* （视图View） - 界面设计人员进行图形界面设计。
* （模型Model） - 程序员编写程序应有的功能或程序的业务逻辑代码。

[](http://zh.wikipedia.org/wiki/File:ModelViewControllerDiagramZh.png)  
**模型**：

模型表示企业数据和业务规则。在MVC的三个部件中，**模型拥有最多的处理任务**。被模型返回的数据是中立的，就是说**模型与数据格式无关，这样一个模型能为多个视图提供数据**。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

**控制器**

控制器**接受用户的输入并调用模型和视图去完成用户的需求**。所以当单击Web页面中的超链接和发送HTML表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

* MVC的优点

1.低耦合性

视图层和业务层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的业务流程或者业务规则的改变只需要改动MVC的模型层即可。因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

2.高重用性和可适用性

随着技术的不断进步，现在需要用越来越多的方式来访问应用程序。MVC模式允许你使用各种不同样式的视图来访问同一个服务器端的代码。它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用HTML来表示，但是也有可能用WAP来表示，而这些表示所需要的命令是改变视图层的实现方式，而控制层和模型层无需做任何改变。

3.较低的生命周期成本

MVC使开发和维护用户接口的技术含量降低。

4.快速的部署

使用MVC模式使开发时间得到相当大的缩减，它使程序员（Java开发人员）集中精力于业务逻辑，界面程序员（HTML和JSP开发人员）集中精力于表现形式上。

5.可维护性

分离视图层和业务逻辑层也使得WEB应用更易于维护和修改。

# 单例模式和工厂模式

单例模式：单例模式保证一个类仅有一个实例。它确保一个类在java虚拟机里只有一个实例，使一个类只有一个对象，整个系统共享这个对象。

单例模式的四大特征：

懒汉式：

1.声明一个私有的，静态的本类对象，但并不在声明的时候就初始化，因此，它的值为null。

2.私有化构造器

3.对外提供一个全局的，公有的，静态的，唯一的方法，用来获得该实例。

4.在该方法里，判断对象是否为null，如果是null的话，表示这个类还没有被实例化，就会初始化这个对象，再返回如果不是null的话，就直接返回。

|  |
| --- |
| **package** cn.edu360.utils;  /\*\*  \* 单例模式  \* 懒汉式：第一次本调用时创建，下次调用不再创建  \*/  **public** **class** Singleton1 {  **private** Singleton1() { }  **private** **static** Singleton1 *instance* = **null**;  **public** **synchronized** **static** Singleton1 getInstance() {  **if** (*instance* == **null**){  *instance* = **new** Singleton1();  }  **return** *instance*;  }  } |

饿汉式：

1.声明一个私有的，静态的本类对象，并在声明的时候就初始化

2.私有构造器

3.对外提供一个全局的，共有的，静态的，唯一的方法，用来获得该实例。

4.在该方法里，直接返回该对象即可。

|  |
| --- |
| **package** cn.edu360.utils;  /\*\*  \* 单例模式  \* 饿汉式：类创建时，就实例化好  \*/  **public** **class** Singleton2 {    **private** **static** Singleton2 *instance* = **new** Singleton2();    **public** **static** Singleton2 getInstance(){  **return** *instance*;  }  } |

工厂模式：

|  |
| --- |
| **package** cn.edu360.utils;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** org.dom4j.Document;  **import** org.dom4j.Element;  **import** org.dom4j.io.SAXReader;  /\*\*  \* 通过工厂模式创建实例  \*/  **public** **class** BeanFactory {  // 存储单例模式对象  **private** **static** Map<String, Object> *beans* = **new** HashMap<>();  // 存储类名称，通过反射创建多实例  **private** **static** Map<String, Object> *beansName* = **new** HashMap<>();    **static** {  SAXReader reader = **new** SAXReader();  **try** {  Document document = reader.read(BeanFactory.**class**.getClassLoader().getResourceAsStream("applicationBeans.xml"));  List<Element> selectNodes = document.selectNodes("/beans/bean");  **for**(Element elm : selectNodes){  String id = elm.attributeValue("id");  String className = elm.attributeValue("class");  String type = elm.attributeValue("type");  // 单例  **if** ("singleton".equals(type)) {  Class<?> clazz = Class.*forName*(className);  Object newInstance = clazz.newInstance();    *beans*.put(id, newInstance);  } **else** {  *beansName*.put(id, className);  }  }  } **catch** (Exception e) {  e.printStackTrace();  }  }    **public** **static** Object getBean(String beanId){  Object object = *beans*.get(beanId);  **if** (**null** == object) {  // 多例  Object className = *beansName*.get(beanId);  **try** {  Class<?> clazz = Class.*forName*(className.toString());  object = clazz.newInstance();  } **catch** (Exception e) {  e.printStackTrace();  }  }  **return** object;  }  } |

# 案例演练

* 将HTML静态注册登录页面改造成Servlet
* 使用Servlet实现注册功能
* 使用Servlet实现登录功能
* 使用Servlet实现查询列表
* 综合练习BBS论坛