# 

**Servlet学习笔记**

**Java相关课程系列笔记之九**

**笔记内容说明**

Servlet（程祖红老师主讲，占笔记内容100%）；

**目 录**

[一、 Servlet概述 1](#_Toc17184)

[1.1 B/S架构（了解） 1](#_Toc19912)

[1.2什么是Servlet 1](#_Toc26)

[1.3什么是Tomcat 2](#_Toc7412)

[1.4如何写一个Servlet（不使用开发工具） 3](#_Toc27369)

[1.5使用MyEclipse开发Servlet 5](#_Toc5043)

[1.6 Servlet是如何运行的 6](#_Toc8393)

[1.7常见错误及解决方式 6](#_Toc25816)

[1.8案例：根据请求次数显示结果和显示当前时间 7](#_Toc4517)

[二、 HTTP协议 8](#_Toc12160)

[2.1什么是HTTP协议 8](#_Toc31642)

[2.2通讯的过程 8](#_Toc12028)

[2.3数据格式 8](#_Toc29621)

[2.4 TCP/IP Monitor的使用 9](#_Toc22305)

[2.5 get请求与post请求 9](#_Toc28041)

[2.6如何读取请求参数 9](#_Toc10236)

[2.7访问数据库（MySql） 10](#_Toc144)

[2.8案例：添加员工（访问MySql数据库） 10](#_Toc7227)

[2.9异常：IllegalStateException 11](#_Toc24948)

[三、 编码问题 12](#_Toc28620)

[3.1 Java语言在内存当中默认使用的字符集 12](#_Toc30381)

[3.2编码 12](#_Toc10365)

[3.3解码 12](#_Toc27426)

[3.4 Servlet如何输出中文 12](#_Toc22697)

[3.5如果表单有中文参数值，也需要注意编码问题 12](#_Toc30372)

[3.6案例：根据请求正确显示中文 12](#_Toc5169)

[3.7将中文数据插入到数据库 13](#_Toc4995)

[四、 重定向 14](#_Toc4687)

[4.1什么是重定向 14](#_Toc20880)

[4.2如何重定向 14](#_Toc2832)

[4.3注意两个问题 14](#_Toc14905)

[4.4两个特点 14](#_Toc5485)

[4.5重定向原理图：以2.8案例为例 14](#_Toc5815)

[五、 DAO 15](#_Toc13147)

[5.1什么是DAO 15](#_Toc12376)

[5.2如何写一个DAO 15](#_Toc25430)

[5.3工厂类 15](#_Toc9325)

[六、 工厂设计模式 16](#_Toc2855)

[6.1什么是工厂 16](#_Toc28778)

[6.2使用工厂设计模式的好处 16](#_Toc28153)

[6.3如何使用工厂模式 16](#_Toc16437)

[6.4案例：为2.8案例添加新功能，并使用DAO和工厂模式 16](#_Toc12281)

[七、 Servlet容器处理细节 19](#_Toc6071)

[7.1 Servlet容器如何处理请求资源路径以及匹配 19](#_Toc15379)

[7.2一个Servlet如何处理多种请求 19](#_Toc31274)

[八、 Servlet的生命周期 20](#_Toc17554)

[8.1 Servlet的生命周期的含义 20](#_Toc1219)

[8.2 Servlet生命周期的四个阶段 20](#_Toc7587)

[8.3实例化 20](#_Toc24477)

[8.4初始化 20](#_Toc26892)

[8.5就绪 20](#_Toc16151)

[8.6销毁 21](#_Toc1183)

[8.7 Servlet生命周期图 21](#_Toc16640)

[8.8 Servlet生命周期相关的几个接口与类 21](#_Toc31014)

[九、 JSP（简要介绍，详细内容见JSP笔记） 23](#_Toc3140)

[9.1什么是JSP 23](#_Toc21989)

[9.2为什么要使用JSP 23](#_Toc3842)

[9.3 JSP与Servlet的关系 23](#_Toc1245)

[9.4如何写一个JSP文件 23](#_Toc22054)

[9.5 JSP是如何运行的 23](#_Toc18036)

[9.6指令 24](#_Toc4860)

[9.7案例：创建emplist.jsp页面，将表示逻辑交给JSP处理 24](#_Toc14140)

[十、 请求转发 26](#_Toc18022)

[10.1什么是转发 26](#_Toc6364)

[10.2如何转发 26](#_Toc1676)

[10.3编程需要注意的两个问题 26](#_Toc8052)

[10.4转发原理图 26](#_Toc15170)

[10.5转发的特点 27](#_Toc20211)

[10.6转发和重定向的区别 27](#_Toc20095)

[10.7何时用重定向 27](#_Toc4143)

[10.8何时用转发 27](#_Toc31121)

[10.9案例：修改6.4案例中step7中的ListEmpServlet.java 28](#_Toc9162)

[十一、 异常的处理 30](#_Toc23797)

[11.1用输出流out 30](#_Toc2338)

[11.2用转发的方式 30](#_Toc16385)

[11.3让容器处理系统异常 30](#_Toc25055)

[11.4案例：将10.9案例中的step3中的所有catch块修改 30](#_Toc32447)

[十二、 路径问题 31](#_Toc24289)

[12.1什么是相对路径 31](#_Toc8100)

[12.2什么是绝对路径 31](#_Toc27028)

[12.3如何写绝对路径 31](#_Toc6156)

[12.4如何防止硬编码 31](#_Toc5368)

[12.5案例：相对、绝对路径对比 31](#_Toc15787)

[12.6四种情况下，正确的绝对路径写法 32](#_Toc31712)

[十三、 状态管理 33](#_Toc5189)

[13.1什么是状态管理 33](#_Toc7735)

[13.2如何进行状态管理 33](#_Toc25201)

[13.3 cookie 33](#_Toc7493)

[13.4如何创建一个cookie 33](#_Toc3381)

[13.5查询cookie 33](#_Toc30541)

[13.6编码问题 34](#_Toc31163)

[13.7 cookie的生存时间 34](#_Toc9474)

[13.8 cookie的路径问题 34](#_Toc28013)

[13.9 cookie的限制 35](#_Toc20377)

[13.10案例：写一个CookieUtil 35](#_Toc26686)

[13.11 session（会话） 36](#_Toc19593)

[13.12如何创建一个session对象 36](#_Toc4583)

[13.13 HttpSession接口中提供的常用方法 36](#_Toc13372)

[13.14 session的超时 37](#_Toc23137)

[13.15用户禁止cookie以后，如何继续使用session 37](#_Toc21299)

[13.16 url重写 37](#_Toc12009)

[13.17 session的优点 37](#_Toc29043)

[13.18 session的缺点 38](#_Toc28741)

[13.19案例：session验证和访问次数 38](#_Toc2611)

[13.20案例：验证码 39](#_Toc31788)

[13.21案例：购物车 41](#_Toc7716)

[十四、 过滤器 45](#_Toc27394)

[14.1什么是过滤器 45](#_Toc23753)

[14.2如何写一个过滤器 45](#_Toc27710)

[14.3案例：敏感字过滤器 45](#_Toc1462)

[14.4过滤器的优先级 46](#_Toc16180)

[14.5初始化参数 47](#_Toc10424)

[14.6优点 47](#_Toc29477)

[十五、 监听器 48](#_Toc25638)

[15.1什么是监听器 48](#_Toc12471)

[15.2容器会产生两大类事件 48](#_Toc31481)

[15.3如何写一个监听器 48](#_Toc8261)

[15.4 ServletContext（Servlet上下文） 48](#_Toc10327)

[15.5如何获得Servlet上下文 48](#_Toc23470)

[15.6 Servlet上下文的作用 48](#_Toc1201)

[15.7案例：统计在线人数 49](#_Toc3972)

[十六、 上传文件 51](#_Toc12458)

[16.1如何上传文件 51](#_Toc20170)

[16.2案例：上传文件 51](#_Toc23649)

[十七、 Servlet线程安全问题 53](#_Toc30123)

[17.1为何Servlet会有线程安全问题 53](#_Toc13775)

[17.2如何处理线程安全问题 53](#_Toc6857)

[十八、 Servlet小结 54](#_Toc24412)

[18.1 Servlet基础 54](#_Toc19471)

[18.2 Servlet核心 54](#_Toc22491)

[18.3状态管理 54](#_Toc31548)

[18.4数据库访问 54](#_Toc18303)

[18.5过滤器和监听器 54](#_Toc24434)

[18.6典型案例和扩展 54](#_Toc410)

[十九、 其他注意事项 55](#_Toc31886)

[19.1连接数据库的工具 55](#_Toc18716)

[19.2知名网站 55](#_Toc17956)

[19.3 C/S架构：Client/Server 55](#_Toc5670)

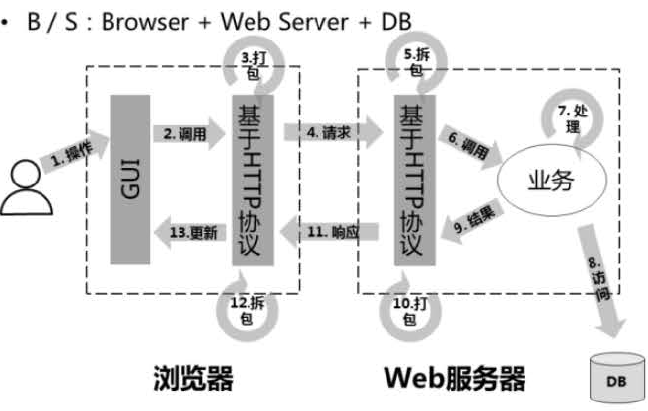
[19.4 B/S架构：Browser/Server 56](#_Toc17218)

# 一、Servlet概述

## 1.1 B/S架构（了解）

### 1.1.1 什么是B/S架构

客户端使用浏览器，服务端使用web浏览器，客户端跟服务器之间使用HTTP协议进行通讯。



1. 数据库只负责数据的管理

2. Web服务器负责业务逻辑的处理

3. 浏览器负责提供操作页面

### 1.1.2 B/S特点

1. 客户端不需要单独安装，因为使用浏览器。C/S架构比较麻烦的是需要单独安装每个客户端，并且一旦客户端版本发生改变，就需要再次安装。

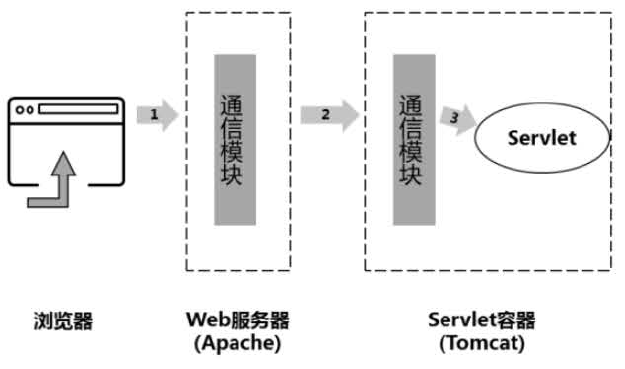
2. 开发相对简单，C/S架构需要我们分别在，客户端和服务器端编写相应的通信处理模块和自定义协议，而B/S架构使用标准的HTTP协议（即不再需要自定义协议），而且浏览器与Web服务器已经包含了相应的通信模块了。

## 1.2什么是Servlet

### 1.2.1 servlet概述

Sun公司制订的一种用来扩展Web服务器功能的组件规范。早期（2000年左右）的Web服务器：apache的web server、微软的iis。只能够处理静态资源，即需要事先将html文件写好，不能够处理动态资源的请求，即需要依据请求参数然后进行计算，生成相应的页面。

为了让这些web服务器能够处理动态资源的请求，需要扩展他们的功能。现在，可以使用Servlet来扩展。



当浏览器将请求发送给Web服务器，Web服务器会向Servlet容器发送请求，Servlet容器负责解析请求数据包。当然，也包括网络通讯相关的一些处理，然后将解析之后的数据交给Servlet来处理。因此Servlet只需要关注具体的业务处理，不用关心网络通讯相关的问题。

注意事项：可以不使用服务器，而直接向Servlet容器发送请求，因为Servlet容器里面也有个通信模块，所以也可直接把Servlet容器当作简单的Web服务器来使用。

### 1.2.2 组件规范

组件：是符合一定规范，并且实现部分功能的可以单独部署的软件模块。组建必须要部署到容器里面才能运行。

容器：也是符合一定规范，并且提供组件的运行环境的程序。

注意事项：单个的组件、单个的容器都是没意义的，都不能单独运行，需要放在一起才能运行。

## 1.3什么是Tomcat

### 1.3.1 Tomcat概述

Tomcat本身是一个Servlet容器，即可以提供Servlet运行环境的一个程序，但是Tomcat还提供了Web服务器所具有的所有功能，所以我们也称Tomcat是一个Web服务器。Tomcat的默认端口是8080。

### 1.3.2 Tomcat目录结构简介

bin目录：存放启动和关闭服务器的一些脚本（命令）。

common目录：共享(部署在该服务器上的所有程序都可以使用)的一些jar包。

conf目录：存放服务器的一些配置文件。

webapps目录：部署目录。

work目录：服务器运行时，生成的一些临时文件。

## 1.4如何写一个Servlet（不使用开发工具）

**step1：**先写一个**Java类**，实现Servlet接口或者继承HttpServlet抽象类。

1. Tomcat会自动调用service，自己不用再去写方法调用了

2. 设置消息头content-type，告诉浏览器返回的数据类型是html文档，以及编码格3. 式此外，还可以告诉服务器，在使用out输出时，使用指定的编码格式进行编码

4. 通过响应对象，获得一个输出流

5. 调用流的方法进行输出，其实质是将处理结果写到了response对象上

**step2：**执行**编译操作**：javac -d . HelloWorldServlet.java

**step3**：**打包**，即创建一个具有如下结构的文件夹：将编译好的.class文件放入classes文件夹

appname（文件夹名起应用名）

|－－WEB-INF（必须大写）

｜－－classes（文件夹，放.class文件）

｜－－lib（文件夹，放.jar文件，可没有）

｜－－web.xml（部署描述文件 url-pattern）

**step4**：**部署**，将step3创建的文件夹拷贝到Servlet容器特定的文件夹下面注意事项：也可以将step3创建的文件夹使用jar命令进行压缩，生成.war为后缀的文件，然后拷贝。

**step5**：**启动**Servlet容器，**访问**Servlet。

访问格式：http://ip:port/appname/url-pattern

比如在浏览器地址栏输入：http://localhost:8080/web01/hello

## 1.6 Servlet是如何运行的

比如，在浏览器地址栏输入：http://ip:port/web01/hello（

step1：浏览器依据ip和port建立与Servlet容器之间的连接。

step2：浏览器将请求数据打包，并发送给Servlet容器。

step3：**Servlet容器**解析请求数据包，并将解析数据放到已创建的request对象上。

同时，容器也已经创建好了一个response对象。

step4：**Servlet容器**依据请求资源路径找到Servlet的配置，然后创建Servlet对象

根据xml文件里配置的类而创建。

step5：**Servlet容器**接下来调用Servlet对象的service方法，并且会将事先创建好的

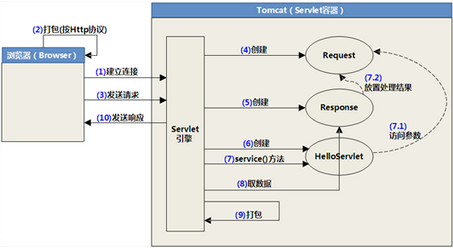
request对象和response对象作为service方法的参数传递给Servlet对象。

step6：Servlet可以通过request对象获得请求参数，进行相应的处理

然后将处理结果写到response对象上

step7：**Servlet容器**读取response对象上的数据，然后将处理结果打包并发送给浏览器

step8：**浏览器**解析响应数据包，将返回的数据展现给用户。



## 1.7常见错误及解决方式

### 1.7.1 错误404

**概述**：404，是一个状态码，是一个三位数字，由服务器发送给浏览器，告诉浏览器是否正确处理了请求，404的意思是说：服务器依据请求资源路径，**找不到对应的资源**。

**解决**：依据http://ip:port/appname/url-pattern检查你的请求地址是否正确。

仔细检查web.xml，特别要注意servlet-name是否一致。

### 1.7.2 错误500

**概述**：500，是服务器处理出错，一般是因为**程序运行出错**。

**解决**：检查程序的代码，比如：是否继承。

检查web.xml，类名要填写正确。

### 1.7.3 错误405

**概述**：405，是服务器**找不到对应的service方法**。

**解决**：检查service方法的签名（方法名、参数类型、返回类型、异常类型）。

## 1.8案例：根据请求次数显示结果和显示当前时间

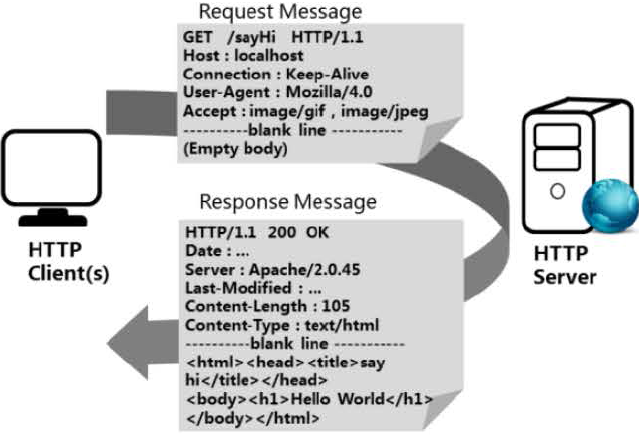
eg1：在地址栏输入http://localhost:8080/web01/hello?qty=5后，获得5个Hello World

eg2：在网页上显示当前时间

# 二、HTTP协议

## 2.1什么是HTTP协议

HTTP（Hypertext transport protocol）是超文本传输协议。是一种应用层协议，由W3C制定，它定义了浏览器与Web服务器之间通讯的过程及数据格式。



## 2.2通讯的过程

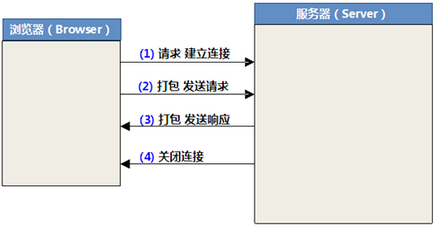
step1：浏览器建立与Web服务器之间的连接（Socket）。

step2：浏览器要将请求数据打包（请求数据包），然后发送给Web服务器。

step3：Web服务器将处理结果打包（响应数据包），然后发送给浏览器。

step4：Web服务器关闭连接。

注意事项：一次请求，一次连接。优点：Web服务器可以利用有限的连接个数为尽可能多的客户服务（效率高）。如果浏览器要再发请求，就必须重新建立一个新的连接。



## 2.3数据格式

### 2.3.1 请求数据包

1. **请求行**：请求方式+请求资源路径+协议描述。

2. **若干消息头**：消息头是一些键值对，一般由W3C定义，有特定的含义。浏览器和服务器之间，可以通过发送消息头来传递一些特定的信息，比如：浏览器可以通过user-agent消息头来通知服务器浏览器的类型和版本。

3. **实体内容**：只有当请求方式为post时，浏览器才会将请求参数添加到实体内容里面，如果请求方式为get，浏览器会将请求参数添加到请求资源路径的后面。



### 2.3.2 响应数据包

**状态行**：协议描述+状态码+状态描述。

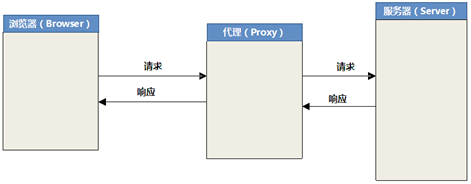
**若干消息头**：服务器也可以发送一些消息头给浏览器，比如content-type，告诉浏览器服务器返回的数据类型和编码格式（字符集，比如：UTF-8、ISO-8859-1）。

**实体内容**：程序处理之后，返回的结果。



### 2.3.3截取数据包

截获数据包，使用MyEclipse中的TCP/IP Monitor，TCP/IP Monitor相当于一个代理服务器，它的原理图如下：



## 2.4 get请求与post请求

### 2.4.1 使用get请求方式

浏览器会使用get方式发请求，情况如下：

1. 直接在浏览器地址输入某个地址。

2. 点击链接地址。

3. 表单默认的提交方法：<form method="get(默认)/post">。

浏览器使用，get请求的特点如下：

1. get请求会将请求参数添加到请求资源路径的后面，因为请求行存放的数据大小有限，也就是地址栏的最长字节数，所以get请求只能提交少量的数据。

2. get请求会将请求参数显示在浏览器地址栏，不安全（比如，路由器会记录整个地址）。

### 2.4.2 使用post请求

浏览器使用post方法发请求，情况如下：

设置表单的method属性值为“post”。

浏览器使用，post请求的特点如下：

post请求会将请求参数添加到实体内容里面，所以，可以提交大量的数据。

post请求不会将请求参数显示在浏览器地址栏，相对安全一些。但是，post请求并不会对请求参数进行加密处理。用HTTPS协议进行加密处理。

注意：

* 服务器不关心是用浏览器还是Java程序发送的请求，只要符合协议格式，都会处理。
* get在地址栏中传值，post在实体中传值

## 2.5读取请求参数

方法一：String request.getParameter(String paraName);

如果paraName（即参数名称）与实际的参数名称不一致，会获得null（不报错）。

在使用表单提交数据时，如果用户没有填写任何的值，会获得空字符串 "" 。

方法二：String[] request.getParameterValues(String paraName);

当有多个参数且名称相同时，使用该方法。比如：?city=bj&city=cs&city=wh

注意事项：getParameterValues方法也可用于只有一个参数的情况。

## 2.6访问数据库（MySql）

### 2.6.1 使用MySql数据库

**登录MySql**：mysql -uroot;//登录mysql，使用root用户权限

**查看**当前所有的**数据库**：show databases;

**创建**一个新的**数据库**：create database db\_chang default character set utf8;

**使用**某个**数据库**：use db\_chang;

**查看**当前数据库有哪些**表**：show tables; drop table tablname;//删表

建表：

create table chang\_emp(

id int primary key auto\_increment,

name varchar(50),

salary double,

age int );

insert into chang\_emp(name,salary,age) values("tom",10000,23);

### 2.6.2 使用JDBC访问数据库

step1：将JDBC驱动放到WEB-INF\lib下

step2：编写JDBC代码，需要注意异常的处理！

# 三、编码问题

## 3.1 Java语言在内存当中默认使用的字符集

默认会用“Unicode”编码格式（字符集）来保存字符。

## 3.2编码

把Unicode这种编码格式对应的字节数组，转换成某种本地编码格式（比如GBK）对应的字节数组，从而保存。Unicode-->GBK。

## 3.3解码

把某种本地编码格式的字节数组转换成Unicode这种编码格式对应的字节数组。GBK-->Unicode。

服务器默认使用ISO-8859-1编码格式，使用1个字节保存，无法存中文！因此中文会出现乱码。

## 3.4 Servlet如何输出中文

需要调用：response.setContentType("text/html;charset=utf-8");其中charset=utf-8表示：

用来指定编码格式，只要支持中文即可，比如也可设置为charset=gbk。

作用两个：① 生成一个content-type消息头，告诉浏览器返回的数据类型和编码格式。

② 服务器在输出时，会使用指定的编码格式进行编码。

## 3.5如果表单有中文参数值，也需要注意编码问题

因为，当表单提交的时候，浏览器会对表单中的数据进行编码，会使用打开表单时的编码格式进行编码，而服务器默认情况下，会使用ISO-8859-1去解码，所以，会产生乱码问题。

**解决方式一：**

step1：先保证表单所在的页面按照指定的编码格式打开。即：

<meta http-equiv="content-type" content="text/html;charset=utf-8" />

已是一种规范模拟content-type消息头，告诉浏览器正在解析的数据类型和编码格式。

step2：调用request.setCharacterEncoding("utf-8");

意思是告诉服务器，使用指定的编码格式进行解码。

注意事项：该方法只能用于"post"请求！注意代码放置顺序，在request.getParameter()方法前。

**解决方式二：**

step1：同方式一的第一步。

step2：使用new String(str.getBytes("iso-8859-1"),"utf-8");

## 3.6将中文数据插入到数据库

程序从内存Unicode编码格式--->数据库中的某种本地格式。

step1：要确保数据库支持中文，即正确设置数据库的字符集。

例如，建MySql的数据库时设置编码：create database db\_chang default character set utf8;

step2：JDBC驱动必须能够正确地进行编码和解码。有些MySql的驱动不能够正确进行编码和解码（默认使用ISO-8859-1进行编码和解码），可以在JDBC的URL后添加：

“useUnicode=true&characterEncoding=utf8”。

例如：conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/db\_chang?" +

"useUnicode=true&characterEncoding=utf8","root","");

注意事项：

* MySql中utf不能写“-”，记得数据库名后有个“？”。
* MySql客户端也是一个程序，也会有编码问题，若你的程序中数据显示正常，而MySql客户端中查询的数据为乱码，则更改下编码，使用：set names gbk;命令。

# 四、重定向

## 4.1什么是重定向

服务器发送一个302状态码及一个Location消息头（值是一个地址，称为重定向地址），通知浏览器立即向重定向地址发请求。

## 4.2如何重定向

使用response.sendRedirect(String url);

## 4.3注意两个问题

1）重定向之前，不要调用out.close();会报错！

out.println("添加成功");//能看到这个输出！（如果不写这个输出，则响应为空白页）

out.close();

response.sendRedirect("list");//看不到重定向结果！

2）重定向之前，服务器会先清空response对象上缓存的数据。Servlet只允许同时发送一个响应。

out.println("添加成功");//看不到这个输出！缓存数据被清空（响应一）

response.sendRedirect("list");//能看到重定向结果！（响应二）

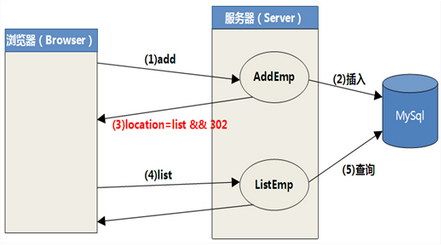
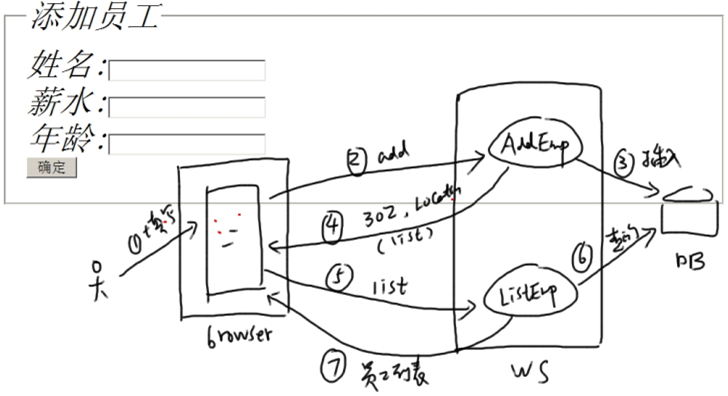
out.close();

## 4.4两个特点

1）重定向的地址是任意的（前提要存在否则报404）。

2）重定向之后，浏览器地址栏的地址会变成重定向地址。

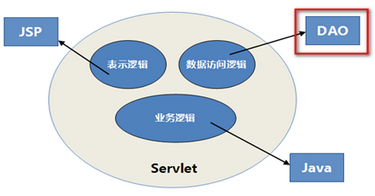
## 4.5重定向原理图：以2.8案例为例



# DAO

## 5.1什么是DAO

Data Access Object，即封装了数据访问逻辑的一个模块。



## 5.2如何写一个DAO

step1：实体类

为了方便对数据库中的记录进行操作而封装的一个普通的Java类。比如，有一个表chang\_emp，可以根据表中的字段设计一个Employee类，属性名与表中的字段名相同，且有get/set方法。

step2：声明DAO接口

声明一系列方法，这些方法不应该涉及任何具体的数据库访问技术。比如，EmployeeDAO接口：

public ResultSet findAll();//错误的！因为ResultSet限制了只能是用JDBC来实现

public List<Employee> findAll() throws SQLException;//错误的！因为限制了只能是用JDBC来实现，SQLException是JDBC的错误

public List<Employee> findAll();//正确的！

public List<Employee> findAll() throws Exception;//正确的！

step3：DAO实现

使用具体的技术来实现DAO接口。比如，EmployeeDAOJdbcImpl类

step4：DAO工厂

Connection conn=DriverManager.getConnection...//DriverManager就是个工厂类，调用者通过DriverManager工厂类获得一个符合Connection接口要求的对象conn

## 5.3工厂类

封装了对象的创建细节，为调用者提供符合要求的对象。详细介绍见第六章。

# 工厂设计模式

## 6.1什么是工厂

工厂是一个设计模式，所谓设计模式，是为解决一类相同或者相似的问题提出的一套解决方案，并且会对这个解决方案命名，比如工厂就是一个常用的设计模式。

## 6.2使用工厂设计模式的好处

1）工厂为调用者提供符合接口要求的对象（对这对象一般称为产品），这样做的好处是，调用者不用了解对象的创建细节，这样，当对象创建过程发生改变，不会影响到调用者。

2）工厂也可以从配置文件来获得接口与其对应的实现类的配置信息。这样，当实现类发生改变时，也不用再去修改工厂类的源代码了。

## 6.3如何使用工厂模式

现在做这样一个假设，假如我们的底层数据访问不再使用JDBC，而采用Hibernate（框架技术，后续会学习，此处做了解）。

那么我们需要为EmployeeDAO增加一个实现EmployeeDAOHibernateImpl.java

并且修改之前的代码，DelEmpServlet、AddEmpServlet、ListEmpServlet、LoadEmpServlet、ModifyEmpServlet都要做相应修改（new的地方都要换成EmployeeDAOHibernateImpl对象）。

## 6.4案例：为2.8案例添加新功能，并使用DAO和工厂模式

step1：创建DBUtil.java类，专门用于连接和关闭数据库：

public static Connection getConnection() throws Exception{

Connection conn=null;

try { Class.forName("com.mysql.jdbc.Driver");

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/db\_chang?" +

"useUnicode=true&characterEncoding=utf8","root","");

} catch (Exception e) { e.printStackTrace(); throw e; }

return conn; }

public static void close(Connection conn){

if(conn!=null){ try { conn.close(); } catch (SQLException e) {

e.printStackTrace(); } } }

step2：创建Employee.java实体类：有id、name、salary、age四个属性，以及get/set方法。

step3：创建DAO接口EmployeeDAO

public List<Employee> findAll() throws Exception;

public void add(Employee e)throws Exception;

public void delete(long index)throws Exception;

public void modify(Employee e)throws Exception;

public Employee findById(long id) throws Exception;

step4：创建DAO接口的实现类EmployeeDAOJdbcImpl：

public List<Employee> findAll() throws Exception{//查找所有员工

List<Employee> employees=new ArrayList<Employee>();

Connection conn=DBUtil.getConnection();

Statement state=conn.createStatement();

ResultSet rs=state.executeQuery("select \* from chang\_emp order by id");

while(rs.next()){ int id=rs.getInt("id"); String name=rs.getString("name");

double salary=rs.getDouble("salary"); int age=rs.getInt("age");

Employee e=new Employee(); e.setId(id); e.setName(name);

e.setSalary(salary); e.setAge(age); employees.add(e); }

DBUtil.close(conn); return employees; }

public void add(Employee e) throws Exception{//添加员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement(

"insert into chang\_emp(name,salary,age) value(?,?,?)");

prep.setString(1, e.getName()); prep.setDouble(2, e.getSalary());

prep.setInt(3, e.getAge()); prep.executeUpdate(); DBUtil.close(conn); }

public void delete(long index) throws Exception{//删除员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement("delete from chang\_emp where id=?");

prep.setLong(1, index); prep.executeUpdate(); DBUtil.close(conn); }

public void modify(Employee e) throws Exception{//修改员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement(

"update chang\_emp set name=?,salary=?,age=? "where id=?");

prep.setString(1, e.getName()); prep.setDouble(2, e.getSalary());

prep.setInt(3, e.getAge()); prep.setLong(4, e.getId()); prep.executeUpdate();

DBUtil.close(conn); }

public Employee findById(long id) throws Exception {//根据id查找员工

Connection conn=DBUtil.getConnection();

PreparedStatement prep=conn.prepareStatement("select \* from chang\_emp where id=?");

prep.setLong(1, id); ResultSet rs=prep.executeQuery(); Employee e=null;

if(rs.next()){ e=new Employee(); e.setId(id); e.setName(rs.getString("name"));

e.setSalary(rs.getDouble("salary")); e.setAge(rs.getInt("age")); }

DBUtil.close(conn); return e; }

step5：创建Factory.java工厂类：

public static Object getInstance(String type){ Object obj=null;

if("EmployeeDAO".equals(type)){

obj=new EmployeeDAOJdbcImpl();

/\*\* obj=new EmployeeDAOHibernateImpl();当不再使用JDBC连接数据库时，如HibernateI连接时，就需要更改每个new EmployeeDAOJdbcImpl();不方便维护。使用工厂模式就可以统一更改了 \*/ }

return obj; }

step6：修改AddEmpMySql.java中的try块，finally块也不需要了（DAO中调用了DBUtil的关闭）

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=new Employee(); e.setName(name); e.setSalary(salary);

e.setAge(age); dao.add(e); response.sendRedirect("list"); }

step7：修改ListEmpServlet.java中的try块，finally块也不需要了（DAO中调用了DBUtil的关闭）

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

List<Employee> Employees=new ArrayList(); Employees=dao.findAll();

out.println("<style>tr:hover{color:red;background:silver;}</style>"+

"<table border='1' width='60%' cellpadding='1' cellspacing='1'>");

out.println("<tr><td>id</td><td>姓名</td><td>薪水</td><td>年龄</td>" +

"<td>操作</td></tr>");

for(int i=0;i<Employees.size();i++){ Employee e=Employees.get(i);

out.println("<tr><td>"+e.getId()+"</td><td>"+e.getName()+"</td>" +

"<td>"+e.getSalary()+"</td><td>"+e.getAge()+"</td>"+

"<td><a href='del?id="+e.getId()+">删除</a>&nbsp;" +

"<a href='load?id="+e.getId()+">修改</a></td></tr>"); }

out.println("</table>");

out.println("<a href='addEmpMySql.html'>再次添加新员工</a>"); }

step8：增加DelEmpServlet.java类用于删除员工，其中service方法内容为：

response.setCharacterEncoding("utf-8"); PrintWriter out=response.getWriter();

long id=Long.parseLong(request.getParameter("id"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

dao.delete(id); response.sendRedirect("list");

}catch(Exception e){ e.printStackTrace(); out.println("删除失败！请稍后再试！"); }

step9：增加LoadEmpServlet.java类用于读取员工，其中service方法内容为：

response.setCharacterEncoding("utf-8"); PrintWriter out=response.getWriter();

long id = Long.parseLong(request.getParameter("id"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=dao.findById(id); response.setContentType("text/html;charset=utf-8");

out.println("<form action='modify?id="+id+"' method='post'>");

out.println("id："+id+"<br />");

out.println("姓名：<input name='name' value='"+e.getName()+"' /><br />");

out.println("薪水：<input name='salary' value='"+e.getSalary()+"' /><br />");

out.println("年龄：<input name='age' value='"+e.getAge()+"' /><br />");

//或out.println("<input type='hidden' name='id' value='"+id+" />");

out.println("<input type='submit' value='确定' />"); out.println("</form>");

}catch(Exception e){ e.printStackTrace(); out.println("读取失败！稍后再试！"); }

step10：增加ModifyEmpServlet.java类用于修改员工，其中service方法内容为：

response.setCharacterEncoding("utf-8"); request.setCharacterEncoding("utf-8");

PrintWriter out = response.getWriter(); long id = Long.parseLong(request.getParameter("id"));

String name=request.getParameter("name");

double salary=Double.parseDouble(request.getParameter("salary"));

int age=Integer.parseInt(request.getParameter("age"));

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=new Employee(); e.setId(id); e.setName(name); e.setSalary(salary);

e.setAge(age); dao.modify(e); response.sendRedirect("list");

} catch (Exception e) { e.printStackTrace(); out.println("修改失败！稍后重试"); }

step11：在浏览器地址栏输入http://localhost:8080/应用名/list则访问出员工列表页面。

# Servlet容器处理细节

## 7.1 Servlet容器如何处理请求资源路径以及匹配

比如：我们在浏览器地址栏输入：http://ip:port/appname/abc.html浏览器会将

"/appname/abc.html"作为请求资源路径发送给Servlet容器。

step1：Servlet容器会先假设访问的是一个Servlet，会依据应用名（appname）找到应用所在的文件夹，然后找到web.xml文件。

step2：匹配<url-pattern>

1）精确匹配（完全匹配）：“/”、大小写、名字完全一样。

2）通配符匹配：使用 “\*” 来匹配任意长度的字符串，比如：<url-pattern>/\*</url-pattern>

3）后缀匹配：使用“\*.”开头，后接任意的字符串，比如：<url-pattern>\*.do</url-pattern>

* 注意事项：\*.do表示匹配所有以.do结尾的请求，注意不写“/”。

比如地址栏输入：/del.do 正确 /aaaaaaa/bbbbb.do 正确

step3：如果都不匹配，则容器认为访问的是一个静态资源文件（比如html文件），然后容器会查找该文件，如果找到则返回，否则会返回404。

## 7.2一个Servlet如何处理多种请求

step1：使用后缀匹配模式，比如<url-pattern>\*.do</url-pattern>

step2：分析请求资源路径，然后依据分析的结果分别进行不同的处理，需使用String request.getRequestURI();

* 注意事项：是URI不是URL。

# Servlet的生命周期

## 8.1 Servlet的生命周期的含义

Servlet容器如何去创建Servlet对象，如何给Servlet对象分配资源，如何调用Servlet对象的方法来处理请求，以及如何去销毁Servlet对象的整个过程。

## 8.2 Servlet生命周期的四个阶段

实例化、初始化、就绪、销毁。

## 8.3实例化

1）指的是容器调用Servlet的构造器，创建Servlet对象。

2）何时实例化？

情况1：容器收到请求之后才创建Servlet对象。在默认情况下，容器只会为Servlet创建唯一的一个实例（多线程，有安全问题。每次请求创建一个线程，由线程去调用方法）。

情况2：容器事先（容器启动时）将某些Servlet（需要配置load-on-startup参数）对象创建好。load-on-startup参数值必须是>=0的整数，越小，优先级越高（即先被实例化）。参数加在web.xml配置文件里的某个<servlet>标签里，如<load-on-startup>1</load-on-startup>

* 注意事项：注意标签的先后顺序，是有要求的，要符合.xsd文档要求。

<servlet><servlet-name></servlet-name><servlet-class></servlet-class>

<load-on-startup>1</load-on-startup></servlet>

## 8.4初始化

1）指的是容器在创建好Servlet对象之后，会立即调用Servlet对象的init方法。

2）init方法：①init方法只会执行一次。

②GenericServlet已经实现了init方法，该方法会将容器创建好的ServletConfig对象作为参数传递给init方法。

③ServletConfig对象提供了一个getInitParameter方法来访问Servlet的初始化参数。 step1：在web.xml文件里，使用<init-param>来配置初始化参数，加在某个<servlet>标签里。

<servlet><init-param>

<param-name>name</param-name>

<param-value>常</param-value>

</init-param></servlet>

step2：使用String getInitParameter(String paraName)获取初始化参数。

④如果GenericServlet的init方法提供的初始化操作不满足需要，可以覆盖（override） init()方法，覆盖的是不带参数的，对有参数的无影响。有参数的init方法执行时，同时无参的也被执行（有参的init方法里调用了无参的init方法），叫钩子方法。

## 8.5就绪

1）就绪指的是Servlet对象可以接受调用了，容器收到请求之后，会调用Servlet对象的service方法来处理，且可以被执行多次。

2）HttpServlet已经实现了service方法，该方法会依据请求类型（get/post）分别调用doGet或doPost方法。所以我们在写一个Servlet时，有两种选择：

选择一：覆盖 HttpServlet的doGet，doPost方法

选择二：覆盖 HttpServlet的service方法

## 8.6销毁

销毁指的是Servlet容器在销毁Servlet对象之前，会调用destroy方法，且只会执行一次。

## 8.7 Servlet生命周期图

服务器加载Servlet

服务器创建Servlet实例

调用Servlet实例的init()方法

Servlet

收到请求

调用service()方法

Service()方法处理请求并将输出结果返回客户端

Servlet等待下一个请求或由服务器卸载

Servlet在服务器调用destroy()方法后被卸载

①请求

ServletConfig

Servlet

Response

Request

容器

② new

③ new

④ new

⑤ new（实例化）

⑥ init（初始化）

⑦ service（就绪）

⑧ destroy（销毁）

## 8.8 Servlet生命周期相关的几个接口与类

1）Servlet接口： ①init(ServletConfig config)//有参的init方法

②service(ServletRequest req,ServletResponse res) ③destroy()

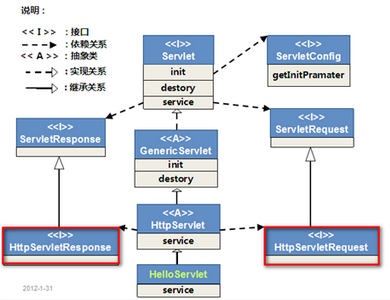
2）GenericServlet抽象类：实现了Servlet接口中的init，destroy方法

3）HttpServlet抽象类：继承了GenericServlet抽象类，实现了service方法

4）ServletConfig接口：String getInitParameter(String paraName)

5）ServletRequest接口是HttpServletRequest的父接口

6）ServletResponse接口是HttpServletResponse的父接口



# JSP（简要介绍，详细内容见JSP笔记）

## 9.1什么是JSP

JSP（Java Server Page）是Java服务器端动态页面技术。是sun公司制订的一种服务器端的动态页面生成技术规范。

## 9.2为什么要使用JSP

因为直接使用Servlet，虽然也可以生成动态页面。但是，编写繁琐（需要使用out.println来输出），并且维护困难（如果页面发生了改变，需要修改Java代码），所以sun指定了JSP规范。

## 9.3 JSP与Servlet的关系

JSP其实是一个以.jsp为后缀的文件，容器会自动将.jsp文件转换成一个.java文件（其实就是一个Servlet），然后调用该Servlet。所以，从本质上讲，JSP其实就是一个Servlet。

## 9.4如何写一个JSP文件

step1：创建一个以“.jsp”为后缀的文件

step2：在该文件里面，可以添加如下的内容

1）HTML（CSS、JS）：直接写即可

2）Java代码：形式一：Java代码片段：<% Java代码 %>

形式二：JSP表达式：<%= Java表达式 %>

形式三：JSP声明：<%! %>

3）指令

## 9.5 JSP是如何运行的

step1：容器依据.jsp文件生成.java文件（也就是先转换成一个Servlet）

1）HTML（CSS、JS）放到service方法里，使用out.write输出

2）<% %>也放到service方法里，照搬，不改动。

3）<%= %>也会放到service方法里，使用out.print输出。

4）<%! %>给Servlet添加新的属性或者新的方法。

这样就把一个JSP变成了一个Servlet容器。

* 注意事项：out.writer方法只能输出简单的字符串，对象是没法输出的。优点是把null自动转换成空字符串输出。如：<% out.println(new Date()); %>不能用writer

step2：容器接下来就会调用Servlet来处理请求了（会将之前生成的.java文件进行编译、然后实例化、初始化、调用相应的方法处理请求）

step3：隐含对象

1）所谓隐含对象（共9个，详细见JSP总结），指的是在.jsp文件里面直接可以使用的对象，比如out、request、response、session、application（ServletContext上下文）等。

2）之所以能直接使用这些对象，是因为容器会自动添加创建这些对象的代码（JSP仅仅是个草稿，最终会变为一个Servlet）。

## 9.6指令

1）指令是什么

通知容器，在将.jsp文件转换成.java文件时，作一些额外的处理，比如导包。

2）指令的语法

<%@指令名称 属性名=属性值 %>

3）page指令（更多属性，见JSP笔记）

①import属性：导包

例如：<%@page import="java.util.\*"%><!-- 注意：没有分号！-->

<%@page import="java.util.\*,java.text.\*"%>

<!-- 多个包以逗号隔开！都在一个双引号里-->

eg：页面显示系统时间

current time:<% out.println(new Date()); %><br />

current time:<%=new Date() %>

* 注意事项：两个是等价，new Date()会被放入out.print();里

②contenType属性：设置response.setContentType的内容

例如：<%@page import="java.util.\*" contentType="text/html;charset=utf-8" %>

<!-- 属性之间用空格隔开 -->

③pageEncoding属性：告诉容器.jsp的文件的编码格式，这样，容器杂在获取jsp文件的内容（即解码）时，不会出现乱码。最好加入，有些容器默认以ISO 8859-1编码。

例如：<%@page contentType="text/html;charset=utf-8" pageEncoding="utf-8" %>

* 注意事项：页面最好添加：<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">，会影响到CSS样式，甚至JS。如IE有个混杂模式，即自动降级，为了兼容老的页面。

4）include指令

对于页面的公共部分，我们可以使用相同的jsp文件，并使用include指令导入，如此实现代码的优化。

告诉容器，在将.jsp文件转换成.java文件时，在指令所在的位置插入相应的文件的“内容“。插入的页面并未运行，而是机械的将内容插入。

比如：<%@include file="head.jsp" %>

## 9.7案例：创建emplist.jsp页面，将表示逻辑交给JSP处理

<%@page contentType="text/html;charset=utf-8" pageEncoding="utf-8"

import="java.util.\*,dao.\*,util.\*,entity.\*,dao.impl.\*" %>

<head>

<style>

.row1{background-color:silver;}

.row2{background-color:yellow;}

</style>

</head>

<body style="font-size:20px">

<table cellpadding="0" cellspacing="0" border="1" width="60%">

<tr><td>ID</td><td>姓名</td><td>薪水</td><td>年龄</td></tr>

<% //使用DAO来访问数据库

EmployeeDAO dao=(EmployeeDAO)Factory.getInstance("EmployeeDAO");

List<Employee> employees=dao.findAll();

for(int i=0;i<employees.size();i++){

Employee e=employees.get(i);

%>

<tr class="row<%=i%2+1 %>">

<td><%=e.getId()%></td>

<td><%=e.getName()%></td>

<td><%=e.getSalary()%></td>

<td><%=e.getAge()%></td>

</tr>

<%

}

%>

</table>

</body>

# 请求转发

## 10.1什么是转发

一个Web组件（Servlet/JSP）将未完成的处理通过容器转交给另外一个Web组件继续完成。常见的情况是：一个Servlet将数据处理完毕之后，转交给一个JSP去展现。

## 10.2如何转发

step1：绑定数据到requset：request里有个HashMap。

request.setAttribute(String name,Object obj);//name：绑定名。obj：绑定值。

另一个方法获取绑定值：

Object request.getAttribute(String name);//如果绑定对象的值不存在，会返回null

step2：获得一个转发器：url：要转发给哪一个Web组件

RequestDispatcher rd=request.getRequsetDispatcher(String url);

step3：转发

rd.forward(request,response);//JSP和Servlet会共享相同的请求和响应对象

step4：在转发的目的地，可以使用request.getAttribute方法获得绑定的数据，然后进行处理。

## 10.3编程需要注意的两个问题

1）转发之前，先清空response对象中缓存的数据。

out.println("hello");//看不到结果

rd.forward(request, response);//能看到结果

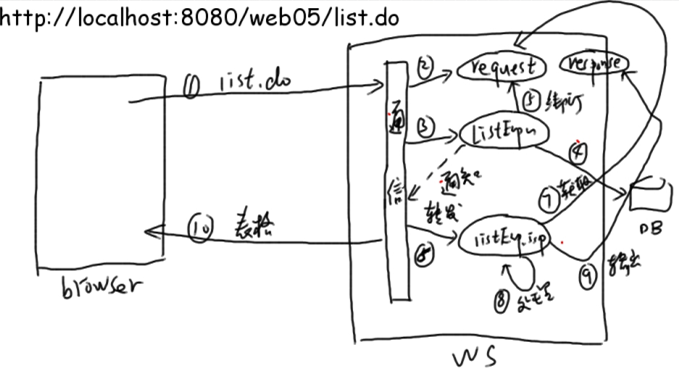
2）转发之前，不能调用out.close()或着out.flush只能同时处理一个响应

out.println("hello");//看到结果

out.close();

rd.forward(request, response);//看不到结果

## 10.4转发原理图



## 10.5转发的特点

1）转发的目的地只能够是同一个应用内部的某个组件的地址。

2）转发之后，浏览器地址栏的地址不变。

3）转发所涉及的各个Web组件可以共享同一个request对象和response对象

## 10.6转发和重定向的区别

1）转发的目的地只能是同一个应用内部某个组件的地址，而重定向的目的地是任意的。

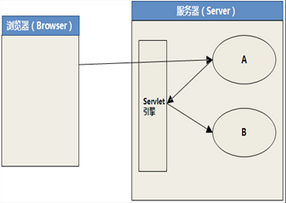
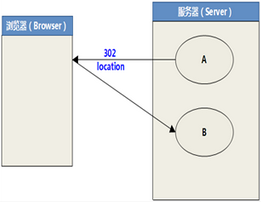
2）转发之后，浏览器地址栏的地址不变，而重定向会变。

3）转发所涉及的各个Web组件可以共享request对象，而重定向不可以。

4）转发是一件事情未做完，而重定向是一件事情已经做完。

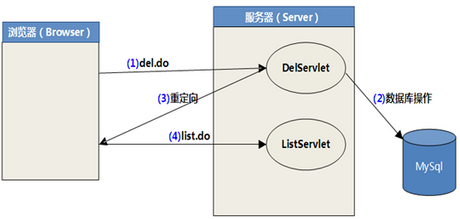
转发

重定向



## 10.7何时用重定向

1）比如用户作删除操作时，删除操作已做完，重定向访问list.do



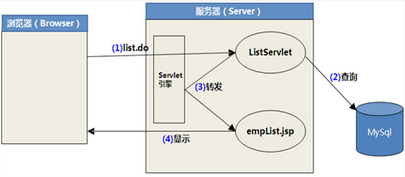
## 10.8何时用转发

1）用户调用list.do

2）有ListServlet到数据库查询数据

3）ListServlet将查询到的结果通过Servlet引擎（通信模块）转发给负责显示的empList.jsp

4）empList.jsp将数据通过友好的界面显示给用户，比如用户作删除操作时，删除操作已做完，重定向访问list.do



## 10.9案例：修改6.4案例中step7中的ListEmpServlet.java

step1：删除try块里面的输出流out.prinltn();即将表示逻辑删除。使用转发，将表示逻辑交给emplist.jsp展示

try{ //使用工厂来访问数据库

EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

List<Employee> employees=dao.findAll();

request.setAttribute("employees", employees);//转发 step1:绑定数据

RequestDispatcher rd=request.getRequestDispatcher("emplist.jsp");//step2:获得转发器

rd.forward(request, response);//step3:转发 }

step2：将9.7案例中的emplist.jsp修该为接收绑定的数据，注意导包

<tr><td>ID</td><td>姓名</td><td>薪水</td><td>年龄</td></tr>

<% List<Employee> employees=

(List<Employee>)request.getAttribute("employees");//接收绑定的数据

for(int i=0;i<employees.size();i++){

Employee e=employees.get(i);

%>

<tr class="row<%=i%2+1 %>">

<td><%=e.getId()%></td>

<td><%=e.getName()%></td>

<td><%=e.getSalary()%></td>

<td><%=e.getAge()%></td>

<td> <a href="del.do?id=<%=e.getId() %>"

onclick="return confirm('确定删除<%=e.getName()%>吗？');">删除员工</a>&nbsp;

<a href="load.do?id=<%=e.getId() %>">修改员工</a>

</td></tr>

<%

}

%>

step3：同理，将其他Servlet中有表示逻辑的都使用转发，并创建相应的.jsp页面，无表示逻辑的使用重定向。将几种Servlet合并到一个Servlet中，命名为ActionServet.java，其中的service方法内容为：

request.setCharacterEncoding("utf-8");

response.setContentType("text/html;charset=utf-8");

PrintWriter out=response.getWriter();

String uri=request.getRequestURI();

String action=uri.substring(uri.lastIndexOf("/"),uri.lastIndexOf("."));//截取从“/”开始到"."结束，不包括“.”的字符串

if(action.equals("/list")){ //使用工厂来访问数据库

try { EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

List<Employee> employees=dao.findAll();

request.setAttribute("employees", employees);//转发 step1:绑定数据

RequestDispatcher rd=request.getRequestDispatcher("emplist.jsp");//step2:获得转发器

rd.forward(request, response);//step3:转发

}catch(Exception e) { e.printStackTrace(); out.println("系统繁忙！稍后再试"); }

}else if(action.equals("/add")){

String name=request.getParameter("name");

Double salary=Double.parseDouble(request.getParameter("salary"));

int age=Integer.parseInt(request.getParameter("age"));

try {//将员工信息插入到数据库

EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=new Employee(); e.setName(name); e.setSalary(salary);

e.setAge(age); dao.add(e); response.sendRedirect("list.do");

} catch (Exception e) { e.printStackTrace(); out.println("添加失败！稍后再试"); }

}else if(action.equals("/del")){

long id=Long.parseLong(request.getParameter("id"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

dao.delete(id); response.sendRedirect("list.do");

}catch(Exception e){ e.printStackTrace(); out.println("删除失败！稍后再试"); }

}else if(action.equals("/load")){

long id = Long.parseLong(request.getParameter("id"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e = dao.findById(id); request.setAttribute("e\_modify", e);//转发

request.getRequestDispatcher("modifyEmp.jsp").forward(request,response);

}catch(Exception e){ e.printStackTrace(); out.println("读取失败！稍后再试"); }

}else if(action.equals("/modify")){

long id = Long.parseLong(request.getParameter("id"));

String name=request.getParameter("name");

double salary=Double.parseDouble(request.getParameter("salary"));

int age=Integer.parseInt(request.getParameter("age"));

try{ EmployeeDAO dao=(EmployeeDAO) Factory.getInstance("EmployeeDAO");

Employee e=new Employee(); e.setId(id); e.setName(name);

e.setSalary(salary); e.setAge(age); dao.modify(e);

response.sendRedirect("list.do");

}catch(Exception e){ e.printStackTrace(); out.println("修改失败！稍后再试！"); }

}

step4：web.xml配置如下

<servlet>

<servlet-name>ActionServlet</servlet-name>

<servlet-class>web.ActionServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ActionServlet</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

# 异常的处理

## 11.1用输出流out

catch(Exception e) { e.printStackTrace(); out.println("系统繁忙，稍后再试！"); }

## 11.2用转发的方式

不用输出流out了。

step1：Servlet中catch块

catch(Exception e) {

request.setAttribute("syserror", "系统繁忙，稍后重试！");

request.getRequestDispatcher("error.jsp").forward(request, response); }

step2：error.jsp页面

<body style="font-size:30px;font-style:italic;">

<% String msg = (String)request.getAttribute("syserror"); %>

<%=msg%>

</body>

## 11.3让容器处理系统异常

step1：将异常抛给容器

不用输出流out了。

catch(Exception e) {

e.printStackTrace();//写throw e;报错，子类异常个数不能大于父类

throw new ServletException(e);//这里将异常转化 }

step2：编写一个错误处理页面，比如system\_error.jsp

<body style="font-size:30px;font-style:italic;color:red;">系统繁忙，稍后重试。</body>

step3：配置错误处理页面（让容器知道，当捕获到了相应的异常后，应该调用哪一个页面）

<!-- 配置错误处理页面 -->

<error-page>

<exception-type>javax.servlet.ServletException</exception-type>

<location>/system\_error.jsp</location>

</error-page>

## 11.4案例：将10.9案例中的step3中的所有catch块修改

修改为将异常抛给容器，具体步骤见11.3节。

# 路径问题

即链接地址、表单提交地址、重定向、转发这四种情况下，如何写正确的路径（地址）。

<a href="list.do"></a>

<form action="add.do">

response.sendRedirect("list.do");

request.getRequestDispatcher("emplist.jsp");

## 12.1什么是相对路径

不以"/"开头的路径。

* 注意事项：
* <a href="../a2.jsp">点击访问a2.jsp</a> <!-- ../ 的用法为：向上跳一级 -->
* 相对路径较易出错，在实际开发中建议使用绝对路径。

## 12.2什么是绝对路径

以“/”开头的路径。

## 12.3如何写绝对路径

链接地址、表单提交地址、重定向的绝对路径，应该从“应用名”开始写，而转发应该从“应用名之后”开始写。可看案例12.5。

## 12.4如何防止硬编码

所谓硬编码就是固定、写死在某个文件中，如JSP页面中。如果Web应用的应用名被更改，则应用会因为路径问题而无法正常运行。

解决此问题需要使用String request.getContextPath()方法来即时获取应用名。

## 12.5案例：相对、绝对路径对比

应用名web05\_path中的WebRoot文件夹下有a2.jsp和app01文件夹，而app01文件夹中有a1.jsp。

a1.jsp页面：

<body style="font-size:30px;color:red;"><!-- a1.jsp -->

a1的内容……<br />

使用相对路径<br /><a href="../a2.jsp">访问a2.jsp</a><br />

使用绝对路径：硬编码<br /><a href="/web05\_path/a2.jsp">访问a2.jsp</a><br />

使用绝对路径：方法<br />

<a href="<%=request.getContextPath() %>/a2.jsp">访问a2.jsp</a> </body>

a2.jsp页面

<body style="font-size:30px;color:red;"><!-- a2.jsp -->

a2的内容……<br />

使用相对路径<br /><a href="app01/a1.jsp">访问a1.jsp</a><br />

使用绝对路径：硬编码<br /><a href="/web05\_path/app01/a1.jsp">访问a1.jsp</a><br/>

使用绝对路径：方法<br />

<a href="<%=request.getContextPath() %>/app01/a1.jsp">访问a1.jsp</a> </body>

## 12.6四种情况下，正确的绝对路径写法

1. 链接地址：从“应用名”开始写

<a href="<%=request.getContextPath() %>/app01/a1.jsp">点击，访问a1.jsp</a>

1. 表单提交地址：从“应用名”开始写

<form method="post" action="<%=request.getContextPath() %>/add">……</form>

1. 重定向而：从“应用名”开始写

response.sendRedirect(request.getContextPath()+"/app01/a1.jsp");

1. 转发：从“应用名之后”开始写

request.getRequestDispatcher("app01/a1.jsp").forward(request, response);

# 状态管理

## 13.1什么是状态管理

将浏览器与Web服务器之间多次交互当作一个整体来看待，并且将多次交互所涉及的数据保存下来。

## 13.2如何进行状态管理

第一大类：客户端状态管理技术，即将状态（也就是多次交互所涉及的数据）保存在客户端（浏览器）。

第二大类：服务器端状态管理技术，即将状态保存在Web服务器端。

## 13.3 cookie

什么是cookie：①是一种客户端的状态管理技术。②当浏览器访问服务器的时候，服务器可以将少量的数据以set-cookie消息头的方式发送给浏览器，浏览器会将这些数据保存下来。当浏览器再次访问服务器时，会将之前保存的这些数据以cookie消息头的方式发送给服务器。

* 注意事项：有几个cookie，就有几个set-cookie消息头。

## 13.4如何创建一个cookie

step1：创建Cookie

Cookie c=new Cookie(String name,String value);//name：cookie的名称。value：cookie的值

step2：添加Cookie：response.addCookie(c);//调用响应对象的addCookie方法

例如：AddCookieServlet类中的service方法中写

response.setContentType("text/html;charset=utf-8");

PrintWriter out=response.getWriter();

Cookie c1=new Cookie("username","tom"); response.addCookie(c1);

Cookie c2=new Cookie("city","beijing");

response.addCookie(c2);//消息头中将会有两个set-cookie

out.println("<h1>添加Cookie成功</h1>"); out.close();

## 13.5查询cookie

1）方法：Cookie[] request.getCookies();//如果没有任何的cookie，则返回null

String cookie.getName();//返回cookie的名称

String cookie.getValue();//返回cookie的值

2）案例：FindCookieServlet类中的service方法中写

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter(); Cookie[] cookies=request.getCookies();

if(cookies!=null){ for(int i=0;i<cookies.length;i++){

Cookie c=cookies[i]; String name=c.getName();

String value=c.getValue();

out.println("name："+name+"，value："+value+"<br />"); }

}else{ out.println("没有cookie"); }

## 13.6编码问题

cookie的值或者名称只允许合法的ASCII码字符串，如果是中文，需要将中文转换成ASCII码字符串。

1）需要用到的两个方法

String URLEncoder.encode(String str,String code);

String URLDecoder.decode(String str,String code);

2）例如：上例代码修改如下

Cookie c1=new Cookie("username",URLEncoder.encode("常","utf-8"));

out.println("name："+name+"，value："+URLDecoder.decode(value,"utf-8")+"<br />");

## 13.7 cookie的生存时间

默认情况下，浏览器会将cookie保存在内存里，只要浏览器不关闭，cookie会一直存在。

1）启动FireFox后，操作系统会为浏览器开辟一块内存空间

①浏览器（FireFox）访问服务器。

②服务器生成消息头setCookie发回给浏览器。

③浏览器将服务器生成的消息头保存到内存区域。

④当浏览器关闭，系统回收为浏览器分配的内存，cookie也相应消失。

2）设置生存时间：使用cookie.setMaxAge(int seconds)方法。

* 注意事项：①单位是秒。②当seconds>0时，浏览器会将cookie保存在硬盘上，当cookie保存的时间超过了seconds，则cookie会被浏览器删除。③当seconds<0时，缺省值（浏览器会将cookie保存在内存里）。④当seconds=0时，删除cookie。所以Cookie中无删除方法，设置为0即可。而修改，则为覆盖，用相同的name。

3）案例

eg1：删除名称为userId的cookie

Cookie c=new Cookie("userId","");//如果已有个叫userId的，则先覆盖

c.setMaxAge(0);//然后删除！即自杀～

response.addCookie(c);

eg2：读取名为name，值为zs的cookie，若不存在则创建。service方法如下

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter(); Cookie[] cookies=request.getCookies();

if(cookies!=null){ boolean flag=false;

for(int i=0;i<cookies.length;i++){ Cookie c=cookies[i];

if(c.getName().equals("name")){

out.println(c.getValue()); flag=true; } }

if(!flag){ Cookie c=new Cookie("name","zs");

response.addCookie(c); }

}else{ Cookie c=new Cookie("name","zs"); response.addCookie(c); }

## 13.8 cookie的路径问题

1）路径问题指的是：浏览器在向服务器上的某个地址发请求时，会比较请求地址与cookie的路径是否匹配，只有匹配的cookie才会发送。

2）cookie有一个默认的路径，值等于创建该cookie的组件的路径，比如：/web06\_Cookie/

app01/addCookie.jsp创建了一个cookie，则该cookie默认的路径为：/web06\_Cookie/app01。

3）匹配规则：只有当访问的地址是cookie的路径或着是其子路径时，浏览器才会将这个cookie进行发送。

4）案例：若cookie的默认路径是“/web\_Cookie/app01”，则访问。

step1：addCookie.jsp页面，且放在web06\_Cookie/app01目录中，并执行

<% Cookie c=new Cookie("userid","abc"); response.addCookie(c); %>

add cookie success.

step2：findCookie1.jsp页面（2、3相同），1放在web06\_Cookie目录中，2放在web06\_Cookie/app01目录中，3放在web06\_Cookie/app01/sub目录中

<% Cookie[] cookies=request.getCookies();

if(cookies!=null){ for(int i=0;i<cookies.length;i++){

Cookie c=cookies[i];

out.println(c.getName()+""+c.getValue()+"<br />"); }

}else{ out.println("no cookie"); } %>

step3：分别执行各个jsp

web06\_Cookie/findCookie1.jsp //是不能访问，不是no cookie提示（页面内容为空白，忽略JSESSIONID）

web06\_Cookie/app01/findCookie2.jsp //可以访问

web06\_Cookie/app01/sub/findCookie3.jsp //可以访问

5）修改cookie的默认路径

可以调用cookie.setPath(String path)方法，一般我们会设置cookie.setPath("/appname")，这样，应用内部的某个组件所添加的cookie可以被该应用内部的其他组件访问到。

例如：上例修改如下，则三种地址都可访问。

<% Cookie c=new Cookie("userid","abc"); c.setPath("/web06\_Cookie");

response.addCookie(c); %>

## 13.9 cookie的限制

1）cookie可以被用户禁止。

2）因为cookie保存在浏览器端，所以cookie不安全。对于一些敏感的数据，需要加密处理。

3）cookie只能够保存少量的数据（大约是4K左右）。

4）cookie的个数也有限制（浏览器只能够保存大约300个左右的cookie，另外对于某个服务器也有cookie个数的限制，大约是20个）。

5）cookie只能够保存字符串，对象、集合、数组都不能保存。

## 13.10案例：写一个CookieUtil

step1：创建CookieUtil类

//appname是应用名，在使用该工具类时，将应用名改为实际部署时的应用名

private static String appname="/web06\_Cookie";

//添加一个cookie需要考虑编码问题

public static void addCookie(String name,String value

,HttpServletResponse response,int age) throws UnsupportedEncodingException{

Cookie c=new Cookie(name,URLEncoder.encode(value,"utf-8"));

c.setMaxAge(age); c.setPath(appname); response.addCookie(c); }

//依据cookie名称返回cookie的值，找不到则返回null

public static String findCookie(String name,HttpServletRequest request){

String value=null; Cookie[] cooikes=request.getCookies();

if(cooikes!=null){ for(int i=0;i<cooikes.length;i++){

Cookie c=cooikes[i];

if(c.getName().equals(name)){ value=c.getValue(); } }

}

return value; }

//删除一个cookie

public static void deleteCookie(String name,HttpServletResponse response){

Cookie c=new Cookie(name,""); c.setMaxAge(0); response.addCookie(c); }

step2：测试，创建FindCookieServlet类，其中的service方法为：

response.setContentType("text/html"); PrintWriter out = response.getWriter();

String value=CookieUtil.findCookie("name", request);

if(value==null){ CookieUtil.addCookie("name", "zs", response, 3600);

}else{ out.println(value); } out.close();

## 13.11 session（会话）

什么是session：①是服务器端的状态管理技术。②当浏览器访问服务器时，服务器会创建一个session对象（该对象有一个唯一的id号，称之为sessionId）。接下来，服务器在默认情况下，会使用set-cookie消息头将这个sessionId发送浏览器，浏览器会将这个sessionId保存下来（内存中，因为指定生存时间）。当浏览器再次访问服务器时，会将sessionId使用cookie消息头发送给服务器，服务器依据这个sessionId就可以找到之前创建的session对象。

③用户与服务器之间的多次交互叫一次会话。

## 13.12如何创建一个session对象

1）方式一：HttpSession s=request.getSession(boolean flag);//HttpSession是一个接口，返回一个符合要求的对象（工厂）。

当flag=true时：

服务器会先检查请求当中是否有sessionId，如果没有则创建一个session对象。

如果有sessionId，则服务器会依据sessionId查找对应的session对象，如果找到了，则返回session对象。根据sessionId找不到，则服务器会创建一个新的session对象。

当flag＝false时：

服务器会先检查请求当中是否有sessionId，如果没有则返回null。

如果有sessionId，则服务器会依据sessionId查找对应的session对象，如果找到了，则返回session对象。根据sessionId找不到，则服务器返回null。

2）方式二：HttpSession s=request.getSession();

等价于HttpSession s=request.getSession(true);

## 13.13 HttpSession接口中提供的常用方法

1）String getId()：获得sessionId

2）setAttribute(String name,Object obj)：绑定一个对象到session对象上

3）getAttribute(String name)：获得绑定对象，如果不存在则返回null

4）removeAttribute(String name)：解除绑定（request也有）

## 13.14 session的超时

1）服务器会将空闲时间过长的session对象删除。大部分服务器默认的session超时限制是30分钟。

2）修改超时限制：可以修改这个默认的超时限制。

例如：可以修改Tomcat的web.xml（影响Tomcat服务器下的所有应用），修改之后，需要重新启动Tomcat服务器。也可以修改某个应用的web.xml。

<session-config>

<session-timeout>30</session-timeout>

</session-config>

也可以通过编程的方式来修改超时的限制：setMaxInactiveInterval(int seconds);

3）立即删除session：invalidate();//实际是清空session对象的内容，然后可以给其他人继续使用该session对象

## 13.15用户禁止cookie以后，如何继续使用session

1）如果用户禁止cookie，服务器仍然会将sessionId以cookie的方式发送给浏览器，但是，浏览器不再保存这个cookie（即sessionId）。

2）如果想要继续使用session，需要采取其他方式来实现sessionId的跟踪，如可以使用url重写来实现sessionId的跟踪。

## 13.16 url重写

1）什么是url重写

浏览器在访问服务器上的某个地址时，不能够直接写这个组件（Servlet/JSP）的地址，而应该使用服务器生成的包含有sessionId的地址。比如以13.19案例中访问次数为例：

当一个新页面有个链接时<a href="SomeServlet">someServlet</a>是错误的！点刷新是不会计数的，因为cookie被禁止了。

应该是<a href="<%=response.encodeURL("SomeServlet")%>">someServlet</a>此时去点击刷新count会自动加1的。

* 注意事项：当用户禁止cookie时，encodeURL方法会在“SomeServlet”后面添加sessionId（不禁止不会有）。例如：

http://localhost:8080/web08/SomeServlet;jsessionid=CD20CBCCC38B4AA3137084D9966D2874

2）如何进行url重写

①response.encodeURL(String utl);//encodeURL方法用在链接地址、表单提交地址。

例如：<form action="<%=response.encodeURL("SomeServlet")%>">

* 注意事项：还一个encodeUrl()是个老方法，内部实现和encodeURL一模一样！

②response.encodeRedirectURL(String url);//用于重定向

例如：response.sendRedirect(response.encodeRedirectURL("list.do"));

③转发不用考虑！是服务器内部，不是浏览器访问服务器。

## 13.17 session的优点

1）session比较安全（相对于cookie）因为将所有状态都写在服务器端。

2）session能够保存的数据类型更加丰富（cookie只能保存字符串）。

3）session能够保存的数据大小更大（cookie只能保存大约4K左右的数据），理论上

session是没有限制的。

4）session没有被禁止的问题，也就不需要url重写（cookie可以被用户禁止）。

## 13.18 session的缺点

1）session会将数据放在服务器端，所以，对服务器的资源的占用比较大，而cookie会将数据保存在浏览器端，对服务器资源的占用没有。

可以将session存到数据库或文件中（钝化），采取最近最少使用原则。当再次使用session时，从数据库或文件中激活

2）session默认情况下，会将sessionId以cookie的方式发送给浏览器，浏览器会将session保存到内存中，如果浏览器关闭，浏览器发请求时就没有sessionId，服务器端的session对象就找不到了。

## 13.19案例：session验证和访问次数

1）session验证：session验证经常用于保护一些需要登录之后才能访问的资源。比如：只有登录成功以后，才能访问main.jsp

step1：登录成功以后，绑定一些数据到session对象上。

例如：session.setAttribute("user",user);

step2：对于需要保护的资源，添加session验证的代码。例如：为main.jsp设置session验证

<% Object obj=session.getAttribute("user");

if(obj==null){

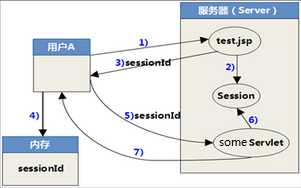
//没有登录成功，或者因为session超时，服务器已经删除了之前的session对象

response.sendRedirect("login.jsp"); return; }

%>

* 注意事项：若不写return，则该页面中body标签中的内容，虽然看不到，但还是会执行！

2）访问次数



step1：SomeServlet中service方法

response.setContentType("text/html;charset=utf-8"); PrintWriter out = response.getWriter();

HttpSession session=request.getSession();//获得一个session对象

//设置超时限制，注意：若不到10秒刷新则又重新开始计时

//session.setMaxInactiveInterval(10);

String sessionId=session.getId();//获得sessionId

System.out.println("sessionId："+sessionId);

Integer count=(Integer)session.getAttribute("count");//绑定必须是个对象，所以用包装类

if(count==null){ count=1;//是第一次访问，则设置count=1

}else{ count++;//不是第一次访问，则原有值基础上加1 }

session.setAttribute("count", count);

out.println("<h1>你是第："+count+"次访问</h1>");

//session.invalidate();//立即删除session对象

out.close();

step2：web.xml配置

<servlet>

<servlet-name>SomeServlet</servlet-name>

<servlet-class>web.SomeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>SomeServlet</servlet-name>

<url-pattern>/SomeServlet</url-pattern>

</servlet-mapping>

## 13.20案例：验证码

step1：CheckcodeServlet中service方法：生成验证码图片

/\*\* step1：绘图 \*/

//1 创建一个内存映像对象（类似一个画布）

BufferedImage image=

new BufferedImage(WIDTH,HEIGHT,BufferedImage.TYPE\_INT\_RGB);

Graphics g=image.getGraphics();//2 获得画布

Random r=new Random();//3 给笔上色

g.setColor(new Color(r.nextInt(255),r.nextInt(255),r.nextInt(255)));

g.fillRect(0,0,WIDTH,HEIGHT);//4 给画布设置一个背景颜色

String number=getNumber(5);//5 绘图

//或String number=r.nextInt(99999)+"";//""作用是把整数转成字符串

//6 绑定number到session对象上

HttpSession session=request.getSession(); session.setAttribute("number", number);

g.setColor(new Color(0,0,0));

//7 字体类型、风格、高度

g.setFont(new Font(null,Font.BOLD|Font.ITALIC,18));//null表示缺省字体

g.drawString(number,10,20);//第一个字符的左下角的点离左的距离和字高度

//8 加一些干扰线

for(int i=0;i<5;i++){ g.drawLine(r.nextInt(80),r.nextInt(30), r.nextInt(80),r.nextInt(30)); }

/\*\* step2：将图片压缩，然后输出到客户端（浏览器） \*/

/\*\* 设置content-type消息头，告诉浏览器返回的数据是一张jpeg格式的图片 \*/

response.setContentType("image/jpeg");

OutputStream os=response.getOutputStream();//获得字节输出流（要输出的是图片）

/\*\* 将原始图片image按照指定的压缩格式jpeg进行压缩，然后，将压缩之后生成的数据通过os输出到response对象上，服务器会从response对象上取出数据，打包发送给浏览器 \*/

javax.imageio.ImageIO.write(image, "jpeg", os);

step2：CheckcodeServlet中getNumber方法

private String getNumber(int size) {

String str="ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

String rs=""; Random r=new Random();

for(int i=0;i<size;i++){ rs +=str.charAt(r.nextInt(str.length())); }

return rs; }

step3：ActionServlet中service方法：对登录的判断

request.setCharacterEncoding("utf-8");

response.setContentType("text/html;charset=utf-8"); PrintWriter out=response.getWriter();

String uri=request.getRequestURI();

String action=uri.substring(uri.lastIndexOf("/"),uri.lastIndexOf("."));

if(action.equals("/login")){//先看验证码是否正确，比较两个验证码

String number1=request.getParameter("number");//number1：表单提交时用户填写的验证码

HttpSession session=request.getSession();//number2：获得绑定的number

String number2=(String)session.getAttribute("number");

if(!number1.equalsIgnoreCase(number2)){//如果验证码错误，则提示用户验证码错误

request.setAttribute("number\_error","验证码错误");

request.getRequestDispatcher("login.jsp").forward(request, response);

return;//没有必要往下执行了 }

String username=request.getParameter("username");String pwd=request.getParameter("pwd");

UserDAO dao=new UserDAO();

try { User user=dao.findByUsername(username);

//登录成功，跳转到一个只有登录成功后才能访问的页面

if(user!=null && user.getPwd().equalsIgnoreCase(pwd)){

//添加session验证的代码：先绑定一些数据到session对象上，用于session验证

session.setAttribute("user", user); response.sendRedirect("main.jsp");

}else{ request.setAttribute("login\_error", "用户名或密码错误！");

request.getRequestDispatcher("login.jsp").forward(request, response); }

} catch (Exception e) { e.printStackTrace(); throw new ServletException(e); }

step4：web.xml配置

<servlet>

<servlet-name>CheckcodeServlet</servlet-name>

<servlet-class>web.CheckcodeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CheckcodeServlet</servlet-name>

<url-pattern>/checkcode</url-pattern>

</servlet-mapping>

step5：login.jsp页面

<tr><td valign="middle" align="right">验证码：</td>

<td valign="middle" align="left">

<input type="text" name="number" />

<% String msg2=(String)request.getAttribute("number\_error"); %>

<span style="color:red;"><%=(msg2==null?" ":msg2)%></span></td></tr>

<tr><td></td><td valign="middle" align="left">

<img src="checkcode" id="img1" onclick="this.src='checkcode?'+Math.random();" />

<a href="javascript:;" onclick="document.getElementById('img1').src=

'checkcode?'+Math.random();">看不清，换一个</a>

</td></tr>

* 注意事项：Math.random();获得随机数，(new Date()).getTime();获得毫秒数，加在问号“？”后面都可以起到欺骗服务器作用，以达到刷新验证码的作用！

## 13.21案例：购物车

step1：建表

create table chang\_product(

id bigint primary key auto\_increment, model varchar(20), pic varchar(50),

prodDesc text, price double);

insert into chang\_product(model,pic,prodDesc,price) values('x200','x200.jpg','还算便宜',2000);

insert into chang\_product(model,pic,prodDesc,price) values('x500','x500.jpg','性价比最好',4000);

insert into chang\_product(model,pic,prodDesc,price) values('x600','x600.jpg','性能不错',6000);

step2：Product实体类

private int id; private String model; private String pic; private String prodDesc;

private double price; ……//相应的get/set方法

step3：DBUtil工具类

public static Connection getConnection() throws Exception{

Connection conn=null;

try { Class.forName("com.mysql.jdbc.Driver");

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/db\_chang?" +

"useUnicode=true&characterEncoding=utf8","root","");

} catch (Exception e) { e.printStackTrace(); throw e; }

return conn; }

public static void close(Connection conn){

if(conn!=null){ try { conn.close(); } catch (SQLException e) {

e.printStackTrace(); } } }

step4：ProductDAO类

public List<Product> findAll() throws Exception{

List<Product> products=new ArrayList<Product>(); Connection conn=null;

try { conn=DBUtil.getConnection(); Statement stat=conn.createStatement();

ResultSet rs=stat.executeQuery("select \* from chang\_product");

while(rs.next()){ Product p=new Product(); p.setId(rs.getInt("id"));

p.setModel(rs.getString("model")); p.setPic(rs.getString("pic"));

p.setProdDesc(rs.getString("prodDesc")); p.setPrice(rs.getDouble("price"));

products.add(p); }

} catch (Exception e) { e.printStackTrace();

}finally{ DBUtil.close(conn); } return products; }

public Product findById(int id) { Product p=null; Connection conn=null;

try { conn=DBUtil.getConnection(); Statement stat=conn.createStatement();

ResultSet rs=stat.executeQuery("select \* from chang\_product where id="+id);

while(rs.next()){ p=new Product(); p.setId(rs.getInt("id"));

p.setModel(rs.getString("model")); p.setPic(rs.getString("pic"));

p.setProdDesc(rs.getString("prodDesc")); p.setPrice(rs.getDouble("price")); }

} catch (Exception e) { e.printStackTrace();

}finally{ DBUtil.close(conn); } return p; }

step5：新建CartItem.java商品条目类，作用是为了方便实现Cart类的操作而设计的类。

private Product p; private int qty; ……//get/set方法

step6：新建Cart.java类，原因：用户发送请求（buy）购买，服务器创建Session对象，不需要再直接访问Session的方法，而是Session绑定一个Cart对象，然后用户购买商品时，只调用Cart的add方法进行操作，而不再直接操作Session，删除（修改）时也一样。如此，我们以后只要操作Cart类的方法就OK了，不再直接操作Session。这是一个经典的设计。

//items：用来保存用户所购买的所有商品

private List<CartItem> items=new ArrayList<CartItem>();

/\*\* 添加商品：需要对items进行遍历，如果已经购买过该商品，则不再添加并且返回false，否则，将商品添加到items然后返回true \*/

public boolean add(CartItem item) {

for(int i=0;i<items.size();i++){ CartItem curr=items.get(i);

if(curr.getP().getId()==item.getP().getId()){ //已经购买过

return false; } }

items.add(item);//没有买过 return true; }

/\*\* 计价 \*/

public double cost(){ double total=0;

for(int i=0;i<items.size();i++){ CartItem curr=items.get(i);

total +=curr.getQty()\*curr.getP().getPrice(); }

return total; }

/\*\* 返回用户所购买的所有商品 \*/

public List<CartItem> list() { return items; }

/\*\* 清空购物车 \*/

public void clear() { if(items.size()>0){ items.clear(); } }

/\*\* 删除购物车中某项 \*/

public void delete(int id){

for(int i=0;i<items.size();i++){ CartItem curr=items.get(i);

if(curr.getP().getId()==id){ items.remove(curr); } } }

/\*\* 修改商品数量 \*/

public void modify(int id,int qty){

for(int i=0;i<items.size();i++){ CartItem curr=items.get(i);

if(curr.getP().getId()==id){ curr.setQty(qty); } } }

step7：ActionServlet类中的service方法

String uri = request.getRequestURI();

String action = uri.substring(uri.lastIndexOf("/"),uri.lastIndexOf("."));

if(action.equals("/list")){ ProductDAO dao = new ProductDAO();

try { List<Product> products = dao.findAll();

request.setAttribute("products", products);

request.getRequestDispatcher("productList.jsp").forward(request, response);

} catch (Exception e) { e.printStackTrace(); throw new ServletException(e); }

}else if(action.equals("/buy")){ HttpSession session=request.getSession();

Cart cart=(Cart)session.getAttribute("cart");

if(cart==null){ cart=new Cart(); session.setAttribute("cart", cart); }//第一次购买

int id=Integer.parseInt(request.getParameter("id"));

ProductDAO dao=new ProductDAO();

try { Product p=dao.findById(id);//将Product封装成一个CartItem对象

CartItem item=new CartItem(); item.setP(p); item.setQty(1);

boolean flag=cart.add(item);

if(flag){//如果返回值为true，则提示用户购买成功

request.setAttribute("buy\_msg\_"+id, "购买成功");

}else{ request.setAttribute("buy\_msg\_"+id, "已经购买过"); }

request.getRequestDispatcher("list.do").forward(request, response);

} catch (Exception e) { e.printStackTrace(); throw new ServletException(e); }

}else if(action.equals("/delete")){ int id=Integer.parseInt(request.getParameter("id"));

HttpSession session=request.getSession(); Cart cart=(Cart)session.getAttribute("cart");

cart.delete(id); response.sendRedirect("cart.jsp");

}else if(action.equals("/clear")){ HttpSession session=request.getSession();

Cart cart=(Cart)session.getAttribute("cart"); cart.clear();

response.sendRedirect("cart.jsp");

}else if(action.equals("/modify")){ int id=Integer.parseInt(request.getParameter("id"));

int qty=Integer.parseInt(request.getParameter("qty"));

HttpSession session=request.getSession(); Cart cart=(Cart)session.getAttribute("cart");

cart.modify(id,qty); response.sendRedirect("cart.jsp"); }

step8：productList.jsp页面动态部分

<% List<Product> products = (List<Product>)request.getAttribute("products");

for(int i=0;i<products.size();i++){ Product p = products.get(i); %>

<tr> <td class="altbg2">&nbsp;&nbsp;<%=p.getModel()%></td>

<td class="altbg2"><img src="img/d007/<%=p.getPic()%>" width="150"

height="90" /></td>

<td class="altbg2"><%=p.getProdDesc() %></td>

<td class="altbg2">￥<%=p.getPrice() %></td>

<td class="altbg2">

<% String msg=(String)request.getAttribute("buy\_msg\_"+p.getId()); %>

<a href="buy.do?id=<%=p.getId() %>">购买</a>

<span style="color:red;"><%=(msg==null?"":msg)%></span></td></tr>

<% } %>

<center>

<input class="button" type="button" value="查看购物车" name="settingsubmit"

onclick="location = 'cart.jsp';">

</center>

step9：cart.jsp页面动态部分

<% //从session当中，取出cart对象

Cart cart=(Cart)session.getAttribute("cart");

if(cart!=null&&cart.list().size()>0){ List<CartItem> items=cart.list();

for(int i=0;i<items.size();i++){ CartItem item=items.get(i); %>

<tr> <td class="altbg2"><%=item.getP().getModel() %></td>

<td class="altbg2"><%=item.getP().getPrice() %></td>

<td class="altbg2"><%=item.getQty() %></td>

<td class="altbg2"><input type="text" size="3" value=""

id="num\_<%=item.getP().getId() %>" /></td><!-- 动态生成id -->

<td class="altbg2">

<a href="javascript:;" onclick="location='modify.do?

id=<%=item.getP().getId()%>&qty='+document.getElementById

('num\_<%=item.getP().getId()%>').value;">

更改数量</a></td><!-- 注意！location中的地址不能用回车换行！ -->

<td class="altbg2"><a href="delete.do?id=<%=item.getP().getId()%>">删除</a></td>

</tr>

<% } %>

<tr><td class="altbg1" colspan="6"><b>总价格：￥<%=cart.cost() %></b></td></tr>

<%

}else{

%>

<tr><td class="altbg2" colspan="6"><b>还没有选购商品</b></td></tr>

<%

}

%>

<center>

<input class="button" type="button" value="返回商品列表" name="settingsubmit"

onclick="location = 'list.do';">

<input class="button" type="button" value="清空购物车" name="settingsubmit"

onclick="location = 'clear.do';">

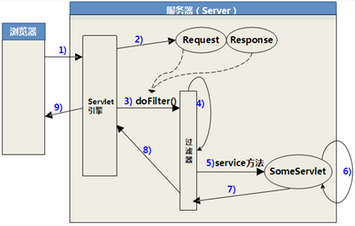
</center>

step10：测试，输入http://localhost:8080/shoppingcart/list.do

# 过滤器

## 14.1什么是过滤器

Servlet规范当中定义的一种特殊的组件，可以拦截Servlet容器的调用过程并进行相应的处理。某个过滤器只有一个实例，即单例模式。



## 14.2如何写一个过滤器

step1：写一个Java类，实现Filter接口

step2：在doFilter方法里面，编写处理逻辑

step3：配置过滤器

* 注意事项：Filter的xml配置在Servlet的xml配置前！

## 14.3案例：敏感字过滤器

step1：创建comment.jsp页面

<form action="comment" method="post">

请输入你的评论：<input name="content" /><input type="submit" value="发表"/>

</form>

step2：创建CommentFilter1过滤器

public class CommentFilter1 implements Filter{//实现Filter接口

private FilterConfig config;

public CommentFilter1(){ System.out.println("Filter1's constructor..."); }

/\*\* 容器在销毁过滤器之前，会调用destroy方法。该方法只会执行一次。 \*/

public void destroy() { System.out.println("Filter1's destroy ..."); }

/\*\* 容器会调用doFilter方法来处理请求，类似于容器调用Servlet的service方法一样。如果调用了FilterChain(过滤器链)的doFilter方法，容器会继续调用后续的过滤器或者Servlet，如果没有调用FilterChain的doFilter方法，则容器不再向后调用。 \*/

public void doFilter(ServletRequest arg0,ServletResponse arg1,

FilterChain arg2) throws IOException, ServletException {

System.out.println("Filter1's doFilter begin...");

HttpServletRequest request =

(HttpServletRequest)arg0;//强转为子接口，是为了调用更多方法

HttpServletResponse response =(HttpServletResponse)arg1;

request.setCharacterEncoding("utf-8");

String content = request.getParameter("content");

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter();

//通过FilterConfig对象读取初始化参数

String illegalStr = config.getInitParameter("illegalStr");//初始化参数详见14.5

if(content.indexOf(illegalStr) != -1){//有敏感字，提示用户评论当中有敏感字

out.println("<h1>评论当中有敏感字</h1>");

}else{ arg2.doFilter(arg0, arg1); }//无敏感字，继续向后调用

System.out.println("Filter1's doFilter end."); }

/\*\* 容器在启动时，会创建过滤器实例，接下来，会立即调用init方法完成初始化操作。该方法只会执行一次。容器会事先创建好一个符合FilterConfig接口要求的对象。可以通过FilterConfig.getInitParameter方法来访问过滤器的初始化参数。 \*/

public void init(FilterConfig arg0) throws ServletException {

System.out.println("Filter1's init...");

/\*\* 因为init方法只执行一次，方法执行完之后，arg0变量就会被销毁，所以需要将arg0的值赋给config属性，这样，就可以在doFilter方法里面访问到FilterConfig对象了。 \*/

config = arg0; } }

step3：配置web.xml文件

<filter>

<filter-name>filter1</filter-name>

<filter-class>web.CommentFilter1</filter-class>

<init-param><!-- 过滤器的初始化参数 详见14.5-->

<param-name>illegalStr</param-name>

<param-value>cat</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>filter1</filter-name>

<url-pattern>/comment</url-pattern>

</filter-mapping>

<servlet>

<servlet-name>CommentServlet</servlet-name>

<servlet-class>web.CommentServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CommentServlet</servlet-name>

<url-pattern>/comment</url-pattern>

</servlet-mapping>

## 14.4过滤器的优先级

如果有多个过滤器都满足过滤的条件，则容器会依据<filter-mapping>的先后顺序来调用过滤器。比如：

step1：14.3案例中再添加一个CommentFilter2过滤器，用于判断长度，部分代码修改如下

int length =Integer.parseInt(config.getInitParameter("length"));//初始化参数详见14.5

if(content.length() > length){

out.println("<h1>评论长度不能超过"+length+"个字符</h1>");

}else{ arg2.doFilter(arg0, arg1); }//长度符合要求，继续向后调用

step2：web.xml配置

<filter>

<filter-name>filter2</filter-name>

<filter-class>web.CommentFilter2</filter-class>

<init-param><!-- 初始化参数详见14.5 -->

<param-name>length</param-name>

<param-value>20</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>filter2</filter-name>

<url-pattern>/comment</url-pattern>

</filter-mapping>

## 14.5初始化参数

step1：使用<init-param>配置初始化参数

step2：调用FilterConfig.getInitParameter(String parameter);返回一个字符串

## 14.6优点

1）可以实现代码的“可插拔性”（增加或减少某个模块，不会影响程序的正常运行）。

2）可以将多个组件相同的处理逻辑集中写在过滤器里面，方便代码的维护。如：四个Web组件都需要session验证，把验证写在一个过滤器即可。配置中写“/\*”，拦截所有请求。

# 监听器

## 15.1什么是监听器

Servlet规范当中定义的一种特殊的组件，用来监听容器产生的事件并进行处理。

## 15.2容器会产生两大类事件

1）生命周期相关的事件：容器在创建或者销毁Requset对象、Session对象、ServletContext对象（Servlet上下文）时产生的事件。

2）绑定相关的事件：容器调用了Request对象、Session对象、ServletContext的setAttribute、removeAttribute时产生的事件。

## 15.3如何写一个监听器

step1：写一个Java类，实现监听器接口（依据监听的事件类型来选择对应的接口，8种，如继承HttpSessionListener）

step2：在监听器接口定义的方法里面，编写处理逻辑

step3：配置监听器

## 15.4 ServletContext（Servlet上下文）

容器在启动的时候，会为每一个应用创建唯一的一个符合ServletContext接口要求的对象（Servlet上下文），该对象一直存在，只有非容器关闭时对象被销毁。

## 15.5如何获得Servlet上下文

1）GenericServlet.getServletContext();//通过GenericServlet抽象类获得上下文（其实也是调用了ServletConfig的getServletContext方法）

2）HttpSession.getServletContext();//通过Session获得上下文

3）ServletConfig.getServletContext();//通过ServletConfig接口获得上下文

4）FilterConfig.getServletContext();//通过过滤器获得上下文

## 15.6 Servlet上下文的作用

1）绑定数据：setAttribute、removeAttribute、getAttribute

Request、Session、ServletContext都提供了绑定数据的相关的三个方法，如果都满足使用的条件，应该优先使用生命周期短的（Request<Session<ServletContext）。

①Request对象上绑定的数据只有同一个请求所涉及的各个Web组件可以共享，例如：一个Servlet将数据绑定到Request，然后转发到一个jsp。请求先交给过滤器来处理，然后调用Servlet。

②Session对象上绑定的数据是用一个会话所涉及的各个Web组件可以共享。

③ServletContext对象绑定的数据是公开的，谁都可以访问，而且随时可访问。

2）访问全局的初始化参数

即使用<context-param>配置的初始化参数，调用String getInitParameter(String paramName)方法，可以被同一个应用中的所有的Servlet、Filter共享。例如：

<context-param><!-- 全局初始化参数，放在servlet前面 -->

<param-name>company</param-name>

<param-value>chang全局</param-value>

</context-param>

<servlet><!-- 局部初始化参数，放在某个servlet里面 -->

<servlet-name>A</servlet-name>

<servlet-class>web.A</servlet-class>

<init-param>

<param-name>company</param-name>

<param-value>chang局部</param-value>

</init-param>

</servlet>

3）依据逻辑路径（path）获得实际部署时的物理路径

String getRealPath(String path);//详情见16.2案例step3

## 15.7案例：统计在线人数

step1：index.jsp页面

在线人数：<%=application.getAttribute("count") %><br />

<a href="logout.jsp">退出系统</a>

step2：创建CountListener监听类，并实现HttpSessionListener接口

public class CountListener implements HttpSessionListener {

private int count=0;//计数器

/\*\* session对象创建之后，容器会产生HttpSessionEvent事件，然后调用sessionCreated方法 \*/

public void sessionCreated(HttpSessionEvent arg0) {

System.out.println("sessionCreated...");

//通过事件对象（HttpSessionEvent）找到session

HttpSession session=arg0.getSession();

ServletContext sctx=session.getServletContext();

//将人数绑定到servletContext，这样可以随时访问

count++; sctx.setAttribute("count", count); }

/\*\* 容器在销毁session对象时候，会调用sessionDestroyed方法 \*/

public void sessionDestroyed(HttpSessionEvent arg0) {

System.out.println("sessionDestroyed...");

HttpSession session=arg0.getSession();

ServletContext sctx=session.getServletContext();

count--; sctx.setAttribute("count", count); } }

step3：logout.jsp页面

<%session.invalidate();%>

step4：web.xml配置

<listener>

<listener-class>web.CountListener</listener-class><!-- 直接写包名.类名即可 -->

</listener>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

* 注意事项：
* 因为多个火狐浏览器在一台计算机共享一块内存，所以不论启动多少个窗口都显示是一个用户。同时使用火狐浏览器和其它浏览器打开，因为使用的不是同一块内存，所以用户数会增加。
* web.xml配置文件中配置信息的顺序如下：

<!-- 全局配置参数 -->

<!-- 过滤器 -->

<!-- 监听器 -->

<!-- servlet -->

<!-- 错误配置页面 -->

<!-- 首页 http://ip:port/appname访问时写到应用名就能访问了-->

# 上传文件

## 16.1如何上传文件

step1：给表单设置enctype="multipart/form-data"（http协议的要求），并且，表单只能使用post方式来提交。

step2：在服务器端，不能使用request.getParameter方法来获得参数值。此时，需要调用request.getInputStream获得一个InputStream（字节流），然后分析这个流来获得数据。一般，使用一些工具来分析这个流（比如：apache提供的commons-fileupload-1.2.1.jar）

## 16.2案例：上传文件

step1：将commons-fileupload-1.2.1.jar和commons-io-1.4.jar包导入

step2：创建form.jsp页面

<form action="fileupload" method="post" enctype="multipart/form-data">

<fiedset><legend>上传文件</legend>

用户名：<input name="username" /><br />照片：<input type="file" name="file1"/>

<input type="submit" value="确定" /></fieldset>

</form>

step3：FileUploadServlet类中service方法

/\*\* step1 创建一个DiskFileItemFactory工厂类对象，该对象为解析器提供了解析时的缺省配置 \*/

DiskFileItemFactory dfif = new DiskFileItemFactory();

ServletFileUpload sfu = new ServletFileUpload(dfif);/\*\* step2 创建解析器 \*/

/\*\* step3 使用解析器来解析，解析器会调用request.getInputStream获得一个流，然后分析这个流，并且将分析的结果封装到FileItem对象里面。一个FileItem对象封装了一个表单域中的所有数据。 \*/

try { List<FileItem> items = sfu.parseRequest(request);

//只需要遍历items集合就可以访问表单中的每一个表单域的数据。

for(int i=0;i<items.size();i++){ FileItem curr = items.get(i);

if(curr.isFormField()){//如果是普通表单域

String fieldname = curr.getFieldName();

System.out.println("fieldname:" + fieldname);

String username = curr.getString();

System.out.println("username:" + username);

}else{//如果是上传文件域

ServletContext sctx = getServletContext();

//依据逻辑路径获得实际部署时的物理路径

String path = sctx.getRealPath("upload");

System.out.println("path:" + path);

String filename = curr.getName();//获得文件名

File file = new File(path + "//" + filename);

curr.write(file);//注意，windows用"\\"，Linux用“//” } }

} catch (Exception e) { e.printStackTrace(); }

step4：web.xml配置

<servlet>

<servlet-name>FileUploadServlet</servlet-name>

<servlet-class>web.FileUploadServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>FileUploadServlet</servlet-name>

<url-pattern>/fileupload</url-pattern>

</servlet-mapping>

# Servlet线程安全问题

可使用jmater压力测试工具。

## 17.1为何Servlet会有线程安全问题

当容器收到一个请求之后，会启动一个线程来处理该请求，如果有多个请求到达容器，并且这多个请求要访问的是同一个Servlet，则会发生多个线程调用同一个Servlet的情况，此时，就需要考虑线程安全问题了，比如，这多个线程都要修改Servlet的属性值。

## 17.2如何处理线程安全问题

1）加锁：使用synchronized关键字对方法或者代码块加锁。加锁会影响一些性能

2）让一个Servlet实现SingleThreadMode接口：容器会为这样的Servlet（实现

SingleThreadMode接口 的）创建多个实例（一个线程一个实例）。因为有可能会产生过多的Servlet实例，所以，在比较大型的应用当中，尽量少用。

# Servlet小结

## 18.1 Servlet基础

1）什么是Servlet 2）如何写一个Servlet

3）Servlet容器是什么 4）Servlet依不依赖某个特定的Servlet容器？（不依赖）

5）HTTP协议 6）get请求和post请求

7）表单的处理：①如何获得表单中的参数值 ②中文参数值如何获得

8）Servlet如何输出中文

## 18.2 Servlet核心

1）重定向和转发 2）如何获得请求参数值

3）容器如何处理请求资源路径 4）Servlet的生命周期

5）线程安全问题 6）Servlet上下文

## 18.3状态管理

1）什么是状态管理

2）Cookie：

①Cookie是什么 ②如何创建一个Cookie ③Cookie的编码问题

④Cookie的生成时间 ⑤Cookie的路径问题 ⑥Cookie的限制

1. Session：

①Session是什么 ②如何获得一个Session ③Session的常用方法

④Session的超时 ⑤删除Session

⑥如果用户禁止Cookie，如何继续使用Session

⑦Session的优缺点（和Cookie比较）

## 18.4数据库访问

1）如何使用JDBC访问数据库 2）DAO

## 18.5过滤器和监听器

1）过滤器：

①什么是过滤器 ②如何写一个过滤器 ③过滤器的优先级

④初始化参数 ⑤优点

2）监听器：

①什么是监听器 ②如何写一个监听器

## 18.6典型案例和扩展

1）员工管理 2）Session验证 3）验证码 4）购物车

5）上传文件

# 其他注意事项

## 19.1连接数据库的工具

1. toad：3星 2）sql-front：2星 3）eclipse自带的

## 19.2知名网站

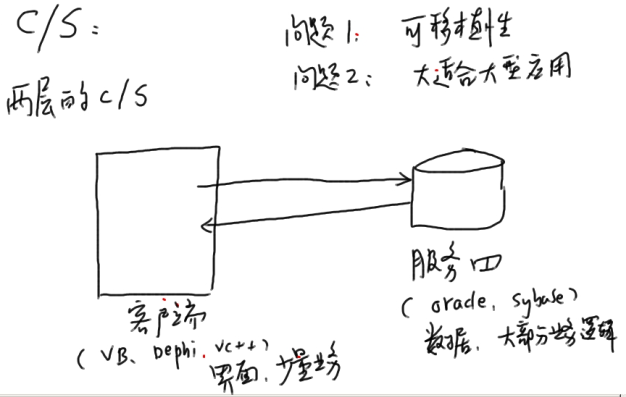
www.iteye.com

## 19.3 C/S架构：Client/Server

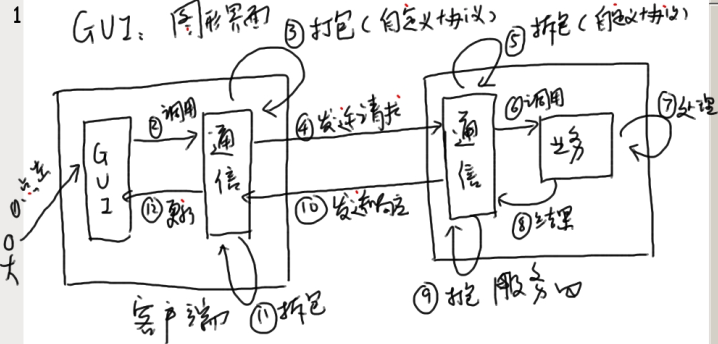
1. 两层的C/S架构：

优点：发开效率高。

缺点：①对数据库而言可移植性差。②不适合大型应用（服务器提供的连接个数是有限的）。



2）三层的C/S架构：



客服端可用VB、VC、Java、Dephi编写。

3）C/S架构一定有自定义协议。

4）TCP/IP协议有两个问题：①拆包问题。②粘包问题。

5）三层的C/S架构执行过程：

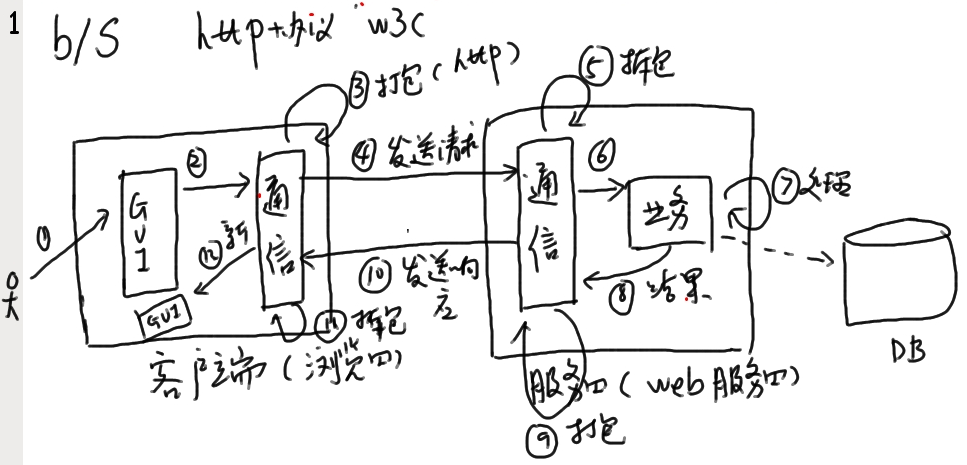
用户点击GUI（图形界面），GUI调用通信，通信打包（自定义协议），发送请求给服务器通信，通信拆包（自定义协议），调用业务，业务处理，结果发给通信，通信打包，发送响应给客户端通信，通信拆包，然后更新GUI。

6）C/S架构客户端需要单独安装。

## 19.4 B/S架构：Browser/Server

和C/S三层架构相同，只是客户端不用写，服务器不用写，通信也不用写，浏览器里都有，采用http协议，是w3c标准，不需要自定义协议了。

只需要写界面（负责输入和输出）和业务逻辑（用Java写Servlet、Jsp）。



用户点击GUI（图形界面），GUI调用通信，通信打包（http协议），发送请求给web服务器通信，通信拆包（http协议），调用业务，业务处理，结果发给通信，通信打包，发送响应给客户端通信，通信拆包，然后重新生成一个GUI。