## 一xml

**1.xml概述**

1.1xml：xml一种数据存储格式，这种数据存储格式在存储数据内容的同时，还能够保存数据之间的关系

1.2xml保存数据的方法：xml利用标签来保存数据的内容，利用标签之间的嵌套关系来保存数据之间的关系。

1.3xml的应用场景：

1.3.1利用xml跨平台的特性，用来在不同的操作系统不同的开发语言之间传输数据。如果说java是一门跨平台的语言，那xml就是跨平台的数据。

1.3.2利用xml可以保存具有关系的数据的特性，还常常被用来做为配置文件使用。

1.4xml文件：把xml格式的数据保存到文件中，这样的文件通常起后缀名为.xml，这样的文件就叫做xml文件，xml文件是xml数据最常见的存在形式，但是，这不是xml的唯一存在形式（在内存中或在网络中也可以存在），不要把xml狭隘的理解成xml文件。

1.5xml校验：浏览器除了内置html解析引擎外还内置了xml解析器，利用浏览器打开xml格式的数据，就可以进行xml校验。

**2.xml语法**

2.1文档声明：一个格式良好的xml必须包含也只能包含一个文档声明，并且文档声明必须出现在xml文档第一行，其前不能有其他任何内容。

2.1.1最简单的写法：<?xml version="1.0" ?>其中的version代表当前xml所遵循的规范版本。

2.1.2使用encoding属性指定文档所使用的字符集编码：<?xml version="1.0" encoding="gb2312" ?>

注意：encoding属性指定的编码集和xml真正使用的编码应该一致，如果不一致就会有乱码问题。encoding属性默认值为老外喜欢的iso8859-1。这是中文乱码的原因所在。

2.1.3使用standalone属性指定当前xml文档是否是一个独立文档：<?xml version="1.0" standalone="no" ?>,standalone默认值为yes表示是一个独立文档。

注意：很多的解析器会忽略这个属性，但是学习知识要按标准去学，所以这个属性也要掌握。

2.2元素

2.2.1元素分为开始标签和结束标签，在开始标签和结束标签之间的文本称为标签体，如果一个标签即不含标签体也不包含其他标签，那这样的标签可以把开始标签和结束标签进行合并，这样的标签叫自闭标签。

<a>xxxxx</a> <a/>

2.2.2一个元素也可以包含若干子元素，但是要注意所有的标签都要进行合理嵌套。

2.2.3一个格式良好的xml文档应该具有并且只能有一个根标签，其他标签都应该是这个根标签的子孙标签。

2.2.4元素的命名规范：

区分大小写，例如，<P>和<p>是两个不同的标记。

不能以数字或标点符号或"\_"开头。

不能以xml(或XML、或Xml 等)开头。

不能包含空格。

名称中间不能包含冒号（:）

2.3属性

一个元素可以包含多个属性，属性的值要用单引号或双引号括起来。如果属性的之中包含双引号，就要用单引号了。

属性的命名规范，参照元素的命名规范。

2.4注释

<!--这是一段注释-->

注意：注释不能出现在文档声明之前。实验：把注释写到文档声明之前，用ie打开是没问题，但是用chrome打开是报错的。这就看出来了不同的解析器有不同的处理，我们学习的时候还是按标准去学。

注释不能嵌套注释

2.5CDATA区、转义字符

都可以用来转义特殊字符。

2.5.1CDATA区<![CDATA[这是要转义的内容]]>

被CDATA区扩起来的内容，将会被浏览器当作文本来处理。

2.5.2转义字符

& --> &amp;

< --> &lt;

>--> &gt;

" --> &quot;

' --> &apos;

2.5.3CDATA区和转义字符的区别

(1)CDATA区可以成段的进行转义，而转义字符一次只能转义一个字符

(2)CDATA区转义的字符可以保存数据本来的格式只是通知解析器按文本去处理。转义字符改变了数据本身的内容，利用其他字符替代了转义字符。请思考，如果要转义的内容是一段js程序的话，如果用转义字符合适不合适？不合适,一是内容太多,用转义字符会很繁琐,二是转义到html中js已经无法再使用了.

2.6处理指令:一段指令通知解析器以何种方式解析xml

<?xml-stylesheet type="text/css" href="1.css" ?>指定解析器使用1.css去渲染当前的xml数据

其实文档声明就是一个最常见的处理指令。

**3.DTD技术**

3.1DTD是一门xml约束技术，用来约束xml写法。

3.2如何在xml中引入一个DTD

3.2.1外部引入：dtd约束文件存在在一个外部文件中，我们在xml中引入该约束。

(1)本地文件引入：该dtd文件存在在本地硬盘中

<!DOCTYPE 根元素的名称 SYSTEM "文件所在的路径">

(2)公共位置文件引入:dtd约束文件存在在一个公共网络上，我们在xml引入该约束

<!DOCTYPE 根元素的名称 PUBLIC "dtd名称" "dtd所在的URL">

3.2.2在xml内部写dtd约束

在文档声明下<!DOCTYPE 根元素名称 [dtd约束的内容]>

3.3利用dtd约束xml中的元素

<!ELEMENT 元素名称 元素约束>

3.3.1元素约束(可以是存放类型或元素内容)

(1)存放类型

ANY:当前声明的元素可以包含任意子元素

EMPTY:当前声明的元素不能包含任何元素

(2)存放内容：利用小括号括起来的元素的名称，用来表示该元素中可以存放哪些内容

<!ELEMENT "元素名" (可以包含的元素的名称)>

小括号中的内容，可以有多个子元素的名称

如果用“,”分割这些子元素就表明这些子元素必须按指定的顺序出现

如果用“|”分割这些内容就表明这些子元素只能出现其中之一

使用“+”来表明内容可以出现一次或多次

使用“\*”来表明内容可以出现零次或多次

使用“?”来表明内容可以出现零次或一次

#PCDATA表明该元素可以包含标签体，任意的字符串

可以利用()进行组操作：

<!ELEMENT MYFILE ((TITLE\*, AUTHOR?, EMAIL)\* | COMMENT)>

3.4利用dtd约束xml中的属性

<!ATTLIST 元素名

属性名 属性类型 属性约束

。。。。>

3.4.1属性类型

（1）CDATA：表明该属性的值是一个普通的文本值。

（2）ENUMERATED：表明该属性的值只能取指定范围内的其中之一

（3）ID:表明该属性值在整个文档中必须唯一，注意ID类型的属性的值必须以字母下划线开头，并且不能以数字开头，不能包含空白字符

（4）实体类型ENTITY

3.4.2属性约束

（1）#REQUIRED 来表明当前这个属性是必须存在的属性

（2）#IMPLIED 来表明当前这个属性是可选的属性

（3）#FIXED "固定值" 来表明当前这个属性具有一个默认的值，可以不明确指定该属性，解析器会帮你加上，如果你硬是指定了一个其他的值，会出错。

（4）"默认值" 来表明当前属性具有一个默认的值，如果给这个属性指定一个值就用指定的值，如果不指定呢，就使用默认值。

3.5实体：可以理解为对一段内容的引用，如果有一段内容到处在被使用，可以将其设计为一个实体

3.5.1引用实体：用在xml中的实体

声明实体：<!ENTITY 实体名称 "实体内容">

引用引用实体：&实体名称;

3.5.2参数实体：用在DTD文件中的实体

声明实体:<!ENTITY % 实体名称 "实体内容">

引用参数实体: %实体名称;

4.xml编程

4.1xml编程概念一个XML文件除了我们人去读写以外，我们希望可以通过程序去读写，利用程序去增删改查XML的过程就是XML编程。 crud（create/read/update/delete）

4.2xml两种解析方式——dom和sax

两种解析方式的比较（区别）根本上是解析思想不同

Dom解析：优点是方便对节点进行增删改查的操作，只需解析一次可以多次获取数据。缺点是占用内存，第一次解析时费时间。当解析超大xml文件时慎用。

Sax解析：优点是不占内存，解析较快，可以解析超大xml文件。缺点是每次都要重新解析，只能进行查找数据的操作，不能进行增删改。

4.3xml解析开发包(API)——Jaxp(sun,j2se)、dom4j（dom4j）需要实际编程练习。

与dom4j配合使用的Xpath

5.xml schema

5.1schema与dtd的区别比较

XML Schema符合XML语法结构。

DOM、SAX等XML API很容易解析出XML Schema文档中的内容。

XML Schema对名称空间支持得非常好。

XML Schema比XML DTD支持更多的数据类型，并支持用户自定义新的数据类型。

XML Schema定义约束的能力非常强大，可以对XML实例文档作出细致的语义限制。

XML Schema不能像DTD一样定义实体，比DTD更复杂，但Xml Schema现在已是w3c组织的标准，它正逐步取代DTD。

5.2快速入门

5.2.1后缀是.xsd，xml schema文档称之为模式文档（约束文档），被约束的xml文档称之为实例文档。

5.2.2根节点只有一个且必须是schema。

5.2.3名称空间uri——把XML Schema文档声明的元素绑定到一个名称空间上，以后XML文件就可以通过这个URI（即名称空间）来告诉解析引擎，xml文档中编写的元素来自哪里，被谁约束。在XML Schema中，每个约束模式文档都可以被赋以一个唯一的名称空间，名称空间用一个唯一的URI（Uniform Resource Identifier，统一资源标识符）表示。

5.2.4对schema的要求是能看懂就行，可以根据schema约束写对应的xml文件就行。

## 二web概述

1．web概述

静态web资源：内容是静态的，不同的人在不同的时间来访问时都是相同的内容。HTML、CSS、JS

动态web资源：内容是由程序生成的，不同的人在不同的时间访问的内容很可能是不同的。

常见的动态web资源开发技术： ASP、PHP、JSP/Servlet

C/S B/S之争

云、移动互联网、html5、物联网

C/S与B/S的区别

C/S：优点是页面展示能力很强（大型3d游戏），由于大量的资源已经保存在了本地，所以运行时不过分占用网络资源。缺点是第一次使用的时候需要下载客户端。当程序需要更新的时候需要所有的客户端都进行更新，这在某些情况下是不可接受的。

B/S：优点是不需要下载客户端，只要一个浏览器就可以使用应用。更新的操作只需要服务器做一次，所有的浏览器访问时都是已经更新过的内容。缺点是展示能力目前还不够强大，特别的占用网络资源。

2、TOMCAT服务器的安装与配置

2.1.常见服务器：WebLogic（BEA）、webSphere（IBM）、Tomcat（Apache）

2.2.Tomcat 的下载与安装

下载地址：http://tomcat.apache.org/

安装目录不能包含中文和空格

JAVA\_HOME环境变量指定Tomcat运行时所要用的jdk所在的位置，注意，配到目录就行了，不用指定到bin

端口占用问题：netstat -ano命令查看端口占用信息

Catalina\_Home环境变量：startup.bat启动哪个tomcat由此环境变量指定，如果不配置则启动当前tomcat，推荐不要配置此环境变量

2.3.Tomcat的目录结构

bin--存放tomcat启动关闭所用的批处理文件

conf--tomcat的配置文件，最终要的是server.xml

\*实验:修改servlet.xml,更改tomcat运行所在的端口号，从8080改为80

lib--tomcat运行所需jar包

logs--tomcat运行时产生的日志文件

temp--tomcat运行时使用的临时目录，不需要我们关注

webapps--web应用所应存放的目录

work--tomcat工作目录，后面学jsp用到

2.4.虚拟主机（一个真实主机可以运行多个网站，对于浏览器来说访问这些网站感觉起来就像这些网站都运行在自己的独立主机中一样，所以，我们可以说这里的每一个网站都运行在一个虚拟主机上，一个网站就是一个虚拟主机）

2.4.1配置虚拟主机

在server.xml中<Engine>标签下配置<Host>,其中name属性指定虚拟主机名，appBase指定虚拟主机所在的目录

只在servlet.xml中配置Hosts，还不能是其他人通过虚拟主机名访问网站，还需要在DNS服务器上注册一把，我们可以使用hosts文件模拟这个过程。Hosts文件所在目录为C:\Windows\System32\drivers\etc

默认虚拟主机：在配置多个虚拟主机的情况下，如果浏览器使用ip地址直接访问网站时，该使用哪个虚拟主机响应呢？可以在<Engine>标签上设置defaultHost来指定

2.5.web应用（web资源不能直接交给虚拟主机，需要按照功能组织用目录成一个web应用再交给虚拟主机管理）

2.5.1web应用的目录结构

WEB-INF目录下的文件不能通过浏览器直接访问。

web应用目录

|

-html、css、js、jsp

|

-WEB-INF

|

-classes

|

-lib

|

-web.xml

2.5.2web.xml文件的作用：

某个web资源配置为web应用首页

将servlet程序映射到某个url地址上

为web应用配置监听器

为web应用配置过滤器

但凡涉及到对web资源进行配置，都需要通过web.xml文件

\*实验：配置一个web应用的主页

2.5.3web应用的虚拟目录映射——三种方式。

（1）在server.xml的<Host>标签下配置<Context path="虚拟路径" docBase="真实路径">如果path=""则这个web应用就被配置为了这个虚拟主机的默认web应用

（2）在tomcat/conf/引擎名/虚拟主机名 之下建立一个.xml文件，其中文件名用来指定虚拟路径，如果是多级的用#代替/表示，文件中配置<Context docBase="真实目录">，如果文件名起为ROOT.xml则此web应用为默认web应用

（3）直接将web应用放置到虚拟主机对应的目录下，如果目录名起为ROOT则此web应用为默认web应用

~如果三处都配置默认web应用则server.xml > config/.../xx.xml > webapps

2.5.4杂项

(1)打war包：方式一：jar -cvf news.war \* 方式二：直接用压缩工具压缩为zip包，该后缀为.war

(2)通用context和通用web.xml，所有的<Context>都继承自conf/context.xml,所有的web.xml都继承自conf/web.xml

(3)reloadable让tomcat自动加载更新后的web应用，当java程序修改后不用重启，服务器自动从新加载，开发时设为true方便开发，发布时设为false，提高性能

(4)Tomcat管理平台，可以在conf/tomcat-users.xml下配置用户名密码及权限

2.6.作业

配置一个www.google.com这样一台虚拟主机，其中包含news应用和mail应用，news应用中有一个new.html。实现在浏览器中输入www.google.com直接看到new.html中的芙蓉姐姐。

3、HTTP协议

3.1.HTTP协议概述

HTTP协议用于定义客户端与web服务器通迅的格式。

HTTP是hypertext transfer protocol（超文本传输协议）的简写，它是TCP/IP协议的一个应用层协议

HTTP使用请求-响应的方式进行传输，一个请求对应一个响应，并且请求只能是由客户端发起的。

HTTP协议的版本：HTTP/1.0、HTTP/1.1，其中1.0一次请求发起一次连接，响应过后连接自动断开。1.1里每次请求响应后连接将保持一段时间，这段时间内可以再执行请求响应。

3.2.HTTP请求

3.2 .1请求行

GET /books/java.html HTTP/1.1

请求方式 请求的资源名 所遵循的协议

3.2.1.1请求方式：GET、POST，区别

其中GET方式在请求资源的URL后跟“？参数名=参数值&参数名=。。。”方式传递参数，传输的数据内容最大为1K

其中POST方式在请求实体中传输数据，数据内容无限制。

除了用Form表单明确用method指定用post方式提交数据以外，其他的方式都是GET提交方式

3.2.2请求头

Accept: text/html,image/\* 客户端可以接受的数据类型

Accept-Charset: ISO-8859-1 客户端接受数据需要使用的字符集编码 中文乱码

Accept-Encoding: gzip,compress 客户端可以接受的数据压缩格式

Accept-Language: en-us,zh-cn 可接受的语言环境——国际化

Host: www.it315.org:80 想要访问的虚拟主机名

If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT 这是和缓存相关的一个头，带着缓存资源的最后获取时间

Referer: http://www.it315.org/index.jsp 这个头表示当前的请求来自哪个链接，这个头和防盗链的功能相关

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0) 客户端的一些基本信息

Cookie 会在后面讲会话技术的时候单讲

Connection: close/Keep-Alive 指定是否继续保持连接

Date: Tue, 11 Jul 2000 18:23:51 GMT 当前时间

3.2.3实体内容

3.3.HTTP响应

3.3.1状态行

HTTP/1.1 200 OK

格式： HTTP版本号　状态码　原因叙述<CRLF>

状态码：

200：请求处理成功

302：请求重定向+location

304、307：服务器通知浏览器使用缓存

404：资源未找到

500：服务器端错误

3.3.2若干响应头

Location: http://www.it315.org/index.jsp 配合302实现请求重定向

Server:apache tomcat 服务器的基本信息

Content-Encoding: gzip 服务器发送数据时使用的压缩格式

Content-Length: 80 发送数据的大小

Content-Language: zh-cn 发送的数据使用的语言环境

Content-Type: text/html; charset=GB2312 当前所发送的数据的基本信息，（数据的类型，所使用的编码）中文乱码

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT 缓存相关的头

Refresh: 1;url=http://www.it315.org 通知浏览器进行定时刷新，此值可以是一个数字指定多长时间以后刷新当前页面，这个数字之后也可以接一个分号后跟一个URL地址指定多长时间后刷新到哪个URL

Content-Disposition: attachment;filename=aaa.zip 与下载相关的头

Transfer-Encoding: chunked 传输类型，如果是此值是一个chunked说明当前的数据是一块一块传输的

Set-Cookie:SS=Q0=5Lb\_nQ; path=/search 和cookie相关的头，后面课程单讲

ETag: W/"83794-1208174400000" 和缓存机制相关的头

Expires: -1 指定资源缓存的时间，如果取值为0或-1浏览就不缓存资源

Cache-Control: no-cache 缓存相关的头，如果为no-cache则通知浏览器不缓存

Pragma: no-cache 缓存相关的头，如果为no-cache则不缓存

以上三个头都是用来控制缓存的，是因为历史原因造成的，不同的浏览器认识不同的头，我们通常三个一起使用保证通用性。

Connection: close/Keep-Alive 是否保持连接

Date: Tue, 11 Jul 2000 18:23:51 GMT 当前时间

3.3.3实体内容

\*额外作业：重写学生信息管理系统的DAO，用Dom4j去实现

## 三、servlet

1、Servlet概述

1.1.sun公司提供的动态web资源开发技术。本质是上一段java小程序，要求这个小程序必须实现Servlet接口，以便服务器能够调用。

1.2.开发Servlet的两个步骤

\*实验：Servlet的快速入门

（1）步骤一：写一个java程序实现Servlet接口（此处直接继承了默认实现类GenericServlet）

package cn.itheima;

import java.io.\*;

import javax.servlet.\*;

public class FirstServlet extends GenericServlet{

public void service(ServletRequest req, ServletResponse res) throws ServletException, java.io.IOException{

res.getOutputStream().write("My FirstServlet!".getBytes());

}

}

（2）将编译好的带包的.class放到WEB-INF/classes下以外，还要配置web应用的 web.xml注册Servlet

<servlet>

<servlet-name>FirstServlet</servlet-name>

<servlet-class>cn.itheima.FirstServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>FirstServlet</servlet-name>

<url-pattern>/FirstServlet</url-pattern>

</servlet-mapping>

1.3.利用MyEclipse开发Servlet

2、Servlet的详述

2.1.生命周期：一件事物什么时候生，什么时候死，在生存期间必然会做的事情，所有这些放在一起就是该事物的声明周期。

2.2Servlet的生命周期：通常情况下，servlet第一次被访问的时候在内存中创建对象，在创建后立即调用init()方法进行初始化。对于每一次请求都调用service(req,resp)方法处理请求，此时会用Request对象封装请求信息，并用Response对象（最初是空的）代表响应消息，传入到service方法里供使用。当service方法处理完成后，返回服务器服务器根据Response中的信息组织成响应消息返回给浏览器。响应结束后servlet并不销毁，一直驻留在内存中等待下一次请求。直到服务器关闭或web应用被移除出虚拟主机，servlet对象销毁并在销毁前调用destroy()方法做一些善后的事情。

2.3Servlet接口的继承结构

Servlet接口：定义了一个servlet应该具有的方法，所有的Servlet都应该直接或间接实现此接口

|

|----GenericServlet：对Servlet接口的默认实现，通用Servlet，这是一个抽象类，其中的大部分方法都做了默认实现，只有service方法是一个抽象方法需要继承者自己实现

|

|----HttpServlet:对HTTP协议进行了优化的Servlet，继承自GenericServlet类，并且实现了其中的service抽象方法，默认的实现中判断了请求的请求方式，并根据请求方式的不同分别调用不同的doXXX()方法。通常我们直接继承HttpServlet即可

2.4.web.xml注册Servlet的注意事项

2.4.1利用<servlet><servlet-mapping>标签注册一个Servlet

<servlet>

<servlet-name>FirstServlet</servlet-name>

<servlet-class>cn.itheima.FirstServlet</servlet-class> 注意：此处要的是一个Servlet的完整类名，不是包含.java或.class扩展的文件路径

</servlet>

<servlet-mapping>

<servlet-name>FirstServlet</servlet-name>

<url-pattern>/FirstServlet</url-pattern>

</servlet-mapping>

2.4.2一个<servlet>可以对应多个<servlet-mapping>

2.4.3可以用\*匹配符配置<serlvet-mapping>,但是要注意，必须是\*.do或者/开头的以/\*结束的路径。

~由于匹配符的引入有可能一个虚拟路径会对应多个servlet-mapping，此时哪个最像找哪个servlet，并且\*.do级别最低。

2.4.4可以为<servlet>配置<load-on-startup>子标签，指定servlet随着服务器的启动而加载，其中配置的数值指定启动的顺序

<servlet>

<servlet-name>invoker</servlet-name>

<servlet-class>

org.apache.catalina.servlets.InvokerServlet

</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

2.4.5缺省servlet：如果一个servlet的对外访问路径被设置为/，则该servlet就是一个缺省servlet，其他servlet不处理的请求都由它来处理

~在conf/web.xml中配置了缺省servlet，对静态资源的访问和错误页面的输出就是由这个缺省servlet来处理的。如果我们自己写一个缺省servlet把爸爸web.xml中的缺省servlet覆盖的话，会导致静态web资源无法访问。所以不推荐配置。

2.4.6servlet的线程安全问题

2.4.6.1由于通常情况下，一个servlet在内存只有一个实例处理请求，当多个请求发送过来的时候就会有多个线程操作该servlet对象，此时可能导致线程安全问题。

(1)serlvet的成员变量可能存在线程安全问题

\*实验：定义一个成员变量 int i = 0；在doXXX()方法中进行i++操作并输出i值到客户端，此时由于延迟可能导致线程安全问题

(2)serlvet操作资源文件时，多个线程操作同一文件引发线程安全问题

\*实验：请求带着一个参数过来，servlet将请求参数写入到一个文件，再读取该文件，将读取到的值打印到客户端上，有可能有线程安全问题

2.4.6.2解决方法

(1)利用同步代码块解决问题。缺陷是，同一时间同步代码块只能处理一个 请求，效率很低下，所以同步代码块中尽量只包含核心的导致线程安全问题的代码。

(2)为该servlet实现SingleThreadModel接口，此为一个标记接口，被标记的servlet将会在内存中保存一个servlet池，如果一个线程来了而池中没有servlet对象处理，则创建一个新的。如果池中有空闲的servlet则直接使用。这并不能真的解决线程安全问题。此接口已经被废弃。

(3)两种解决方案都不够完美，所以尽量不要在servlet中出现成员变量。

3、ServletConfig

3.1.代表servlet配置的对象，可以在web.xml中<servlet>中配置

<servlet>

<servlet-name>Demo5Servlet</servlet-name>

<servlet-class>cn.itheima.Demo5Servlet</servlet-class>

<init-param>

<param-name>data1</param-name>

<param-value>value1</param-value>

</init-param>

</servlet>

然后在servlet中利用this.getServletConfig()获取ServletConfig对象，该对象提供了getInitParameter()和getInitParameterNames()方法，可以遍历出配置中的配置项。

不想在servlet中写死的内容可以配置到此处。

4、ServletContext

4.1.代表当前web应用的对象。

4.2.作为域对象使用，在不同servlet之间传递数据,作用范围是整个web应用

~域：一个域就理解为一个框，这里面可以放置数据，一个域既然称作域，他就有一个可以被看见的范围，这个范围内都可以对这个域中的数据进行操作，那这样的对象就叫做域对象。

4.3.在web.xml可以配置整个web应用的初始化参数，利用ServletContext去获得

<context-param>

<param-name>param1</param-name>

<param-value>pvalue1</param-value>

</context-param>

this.getServletContext().getInitParameter("param1")

this.getServletContext().getInitParameterNames()

4.4.在不同servlet之间进行转发 this.getServletContext().getRequestDispatcher("/servlet/Demo10Servlet").forward(request, response);

方法执行结束，service就会返回到服务器，再有服务器去调用目标servlet，其中request会重新创建，并将之前的request的数据拷贝进去。

4.5.读取资源文件

4.5.1由于相对路径默认相对的是java虚拟机启动的目录，所以我们直接写相对路径将会是相对于tomcat/bin目录，所以是拿不到资源的。如果写成绝对路径，当项目发布到其他环境时，绝对路径就错了。传统方式：如果写相对路径，则相对于jdk启动目录，在web环境下找到了tomcat的启动目录tomcat/bin.如果写绝对路径，则写死在程序里，在本机中可以，发布到其他主机的tomcat下时，路径很可能就是错的。

4.5.2为了解决这个问题ServletContext提供了this.getServletContext().getRealPath("/1.properties")，给进一个资源的虚拟路径，将会返回该资源在当前环境下的真实路径。this.getServletContext().getResourceAsStream("/1.properties")，给一个资源的虚拟路径返回到该资源真实路径的流。

4.5.3当在非servlet下获取资源文件时，就没有ServletContext对象用了，此时只能用类加载器

classLoader.getResourceAsStream("../../1.properties")，此方法利用类加载器直接将资源加载到内存中，有更新延迟的问题，以及如果文件太大，占用内存过大。

classLoader.getResource("../1.properties").getPath()，直接返回资源的真实路径，没有更新延迟的问题。

5.其它总结

HTML-----w3c组织提供，浏览器页面语言

HTTP----美国的一个标准化组织，浏览器和服务器之间通信的规范，协议

servlet规范----sun公司提供，定了servlet相关的内容，api

servlet容器---servlet运行的环境，tomcat--apache

web容器---所有能运行web应用的容器。

Servlet继承结构

Servlet接口:定义一个Servlet应该具有的最基本的方法

|

|--GenericServlet：抽象类，实现了Servlet接口，提供了对Servlet接口的默认实现，其中大部分的方法都做了默认实现，只有service方法没有实现是一个抽象方法。

|

|--HttpServlet：抽象类，此类在GenericServlet类的基础上提供了针对HTTP协议的增强的方法，其中service方法中判断请求的方式再去调用对应的doXXX方法。自己不要去覆盖service方法，而将代码卸载doXXX方法中，如果是get方式提交数据写在doGet中，post方式提交的数据卸载doPost中

生命周期：

Servlet：服务器创建，第一次访问时创建，创建出来调用init，每次请求调用service，在服务器停止或web应用移除容器时销毁，销毁前调用destory。

ServletConfig:服务器创建，Servlet实例时，调用init方法时作为参数使用传入，在Servlet中保存了对它的引用。随着Servlet的销毁而销毁。

ServletContext:服务器创建，服务器启动，web应用加载后立即创建。服务器关闭或者web应用被移除出web容器时销毁。

域：一个可以被看见的范围

域对象：如果一个对象具有了一个可以被看见的范围则这个对象就叫做一个域的对象。

底层是map对象存储的键值对，可以在所有的servlet范围内共享

## 四、Response和Request

1、Response

1.Resonse的继承结构：ServletResponse<--HttpServletResponse

2.Response代表响应，于是响应消息中的 状态码、响应头、实体内容都可以由它进行操作,由此引伸出如下实验：

3.利用Response输出数据到客户端

response.getOutputStream（）.write("中文".getBytes())输出数据，这是一个字节流，是什么字节输出什么字节，而浏览器默认用平台字节码打开服务器发送的数据，如果服务器端使用了非平台码去输出字符的字节数据就需要明确的指定浏览器编码时所用的码表，以防止乱码问题。response.addHeader("Content-type","text/html;charset=gb2312")

response.getWriter().write(“中文”);输出数据，这是一个字符流，response会将此字符进行转码操作后输出到浏览器，这个过程默认使用ISO8859-1码表，而ISO8859-1中没有中文，于是转码过程中用?代替了中文，导致乱码问题。可以指定response在转码过程中使用的目标码表，防止乱码。response.setCharcterEncoding("gb2312");

其实response还提供了setContentType("text/html;charset=gb2312")方法，此方法会设置content-type响应头，通知浏览器打开的码表，同时设置response的转码用码表，从而一行代码解决乱码。

4.利用Response 设置 content-disposition头实现文件下载

设置响应头content-disposition为“attachment;filename=xxx.xxx”

利用流将文件读取进来，再利用Response获取响应流输出

如果文件名为中文，一定要进行URL编码，编码所用的码表一定要是utf-8

5.refresh头控制定时刷新

设置响应头Refresh为一个数值，指定多少秒后刷新当前页面

设置响应头Refresh为 3;url=/Day05/index.jsp,指定多少秒后刷新到哪个页面

可以用来实现注册后“注册成功，3秒后跳转到主页”的功能

在HTML可以利用<meta http-equiv= "" content="">标签模拟响应头的功能。

6.利用response设置expires、Cache-Control、Pragma实现浏览器是否缓存资源，这三个头都可以实现，但是由于历史原因，不同浏览器实现不同，所以一般配合这三个头使用

6.1控制浏览器不要缓存（验证码图片不缓存）设置expires为0或-1设置Cache-Control为no-cache、Pragma为no-cache

6.2控制浏览器缓存资源。即使不明确指定浏览器也会缓存资源，这种缓存没有截至日期。当在地址栏重新输入地址时会用缓存，但是当刷新或重新开浏览器访问时会重新获得资源。

如果明确指定缓存时间，浏览器缓存是，会有一个截至日期，在截至日期到期之前，当在地址栏重新输入地址或重新开浏览器访问时都会用缓存，而当刷新时会重新获得资源。

response.setDateHeader("Expires", System.currentTimeMillis()+1000l\*3600l\*24l\*30l);

7.Response实现请求重定向

7.1古老方法：response.setStatus(302);response.addHeader("Location","URL");

7.2快捷方式：response.sendRedirect("URL");

\*8.输出验证码图片

9.Response注意的内容：

9.1对于一次请求，Response的getOutputStream方法和getWriter方法是互斥，只能调用其一，特别注意forward后也不要违反这一规则。

9.2利用Response输出数据的时候，并不是直接将数据写给浏览器，而是写到了Response的缓冲区中，等到整个service方法返回后，由服务器拿出response中的信息组成响应消息返回给浏览器。

9.3service方法返回后，服务器会自己检查Response获取的OutputStream或者Writer是否关闭，如果没有关闭，服务器自动帮你关闭，一般情况下不要自己关闭这两个流。

2、Request：Request代表请求对象，其中封装了对请求中具有请求行、请求头、实体内容的操作的方法

1.获取客户机信息

getRequestURL方法返回客户端发出请求完整URL

getRequestURI方法返回请求行中的资源名部分,在权限控制中常用

getQueryString 方法返回请求行中的参数部分

getRemoteAddr方法返回发出请求的客户机的IP地址

getMethod得到客户机请求方式

getContextPath 获得当前web应用虚拟目录名称，特别重要！！！，工程中所有的路径请不要写死，其中的web应用名要以此方法去获得。

2.获取请求头信息

getHeader(name)方法 --- String ，获取指定名称的请求头的值

getHeaders(String name)方法 --- Enumeration<String> ，获取指定名称的请求头的值的集合，因为可能出现多个重名的请求头

getHeaderNames方法 --- Enumeration<String> ，获取所有请求头名称组成的集合

getIntHeader(name)方法 --- int ，获取int类型的请求头的值

getDateHeader(name)方法 --- long(日期对应毫秒) ，获取一个日期型的请求头的值，返回的是一个long值，从1970年1月1日0时开始的毫秒值

\*实验：通过referer信息防盗链

String ref = request.getHeader("Referer");

if (ref == null || "".equals(ref) || !ref.startsWith("http://localhost")) {

response.sendRedirect(request.getContextPath() + "/homePage.html");

} else { this.getServletContext().getRequestDispatcher("/WEB-INF/fengjie.html").forward(request, response);

}

3.获取请求参数

getParameter(name) --- String 通过name获得值

getParameterValues（name） --- String[ ] 通过name获得多值 checkbox

getParameterNames --- Enumeration<String> 获得所有请求参数名称组成的枚举

getParameterMap --- Map<String,String[ ]> 获取所有请求参数的组成的Map集合，注意，其中的键为String，值为String[] BeanUtils使用的时候可以直接传给它，把请求参数封装到javabean中

获取请求参数时乱码问题：

浏览器发送的请求参数使用什么编码呢？当初浏览器打开网页时使用什么编码，发送就用什么编码。

服务器端获取到发过来的请求参数默认使用ISO8859-1进行解码操作，中文一定有乱码问题

对于Post方式提交的数据，可以设置request.setCharacterEncoding("gb2312");来明确指定获取请求参数时使用编码。但是此种方式只对Post方式提交有效。

对于Get方式提交的数据，就只能手动解决乱码：String newName = new String(name.getBytes("ISO8859-1"),"gb2312");此种方法对Post方式同样有效。

在tomcat的server.xml中可以配置http连接器的URIEncoding可以指定服务器在获取请求参数时默认使用的编码，从而一劳永逸的决绝获取请求参数时的乱码问题。也可以指定useBodyEncodingForURI参数，令request.setCharacterEncoding也对GET方式的请求起作用，但是这俩属性都不推荐使用，因为发布环境往往不允许修改此属性。

4.利用请求域传递对象

生命周期：在service方法调用之前由服务器创建，传入service方法。整个请求结束，request生命结束。

作用范围：整个请求链。

作用：在整个请求链中共享数据，最常用的：在Servlet中处理好的数据要交给Jsp显示，此时参数就可以放置在Request域中带过去。

5.request实现请求转发

ServletContext可以实现请求转发，request也可以。

在forward之前输入到response缓冲区中的数据，如果已经被发送到了客户端，forward将失败，抛出异常

在forward之前输入到response缓冲区中的数据，但是还没有发送到客户端，forward可以执行，但是缓冲区将被清空，之前的数据丢失。注意丢失的只是请求体中的内容，头内容仍然有效。

在一个Servlet中进行多次forward也是不行的，因为第一次forward结束，response已经被提交了，没有机会再forward了

总之，一条原则,一次请求只能有一次响应，响应提交走后，就再没有机会输出数据给浏览器了。

6.RequestDispatcher进行include操作

forward没有办法将多个servlet的输出组成一个输出，因此RequestDispatcher提供了include方法，可以将多个Servlet的输出组成一个输出返回个浏览器

request.getRequestDispatcher("/servlet/Demo17Servlet").include(request, response);

response.getWriter().write("from Demo16");

request.getRequestDispatcher("/servlet/Demo18Servlet").include(request, response);

常用在页面的固定部分单独写入一个文件，在多个页面中include进来简化代码量。

3、地址的写法：

绝对路径（以斜线开头的路径，代表相对与当前web应用）：

如果地址是给服务器用的，/代表web应用名称，web应用的名称可以省略。如果地址是给客户端（浏览器）用的，/代表虚拟主机，必须写上web应用名

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

request.getRequestDispatcher("/index.jsp").include(request, response);

response.setHeader("Location","/Day05/index.jsp");

response.sendRedirect("/Day05/index.jsp");

this.getServletContext().getRealPath("/index.jsp");

this.getServletContext().getResourceAsStream("/index.jsp");

<a href="/Day05/index.jsp">

<from action="/Day05/index.jsp">

Request是用来获取参数的，是给服务器用的。Response是发送到客户端的，是给客户端用的。Html/jsp页面上的内容都是给客户端用的。

//类加载器的路径相对于.class所在的根目录，在web开发下，WEB-INF/classes,在普通java工程下是bin目录下

ClassLoader.getResource("/1.properties");----WEB-INF/classes/1.properties

ClassLoader.getResourceAsStream("/1.properties");

//普通路径是相对于jdk启动目录的，在web环境下是相对于tomcat/bin目录，普通java工程下，相对于当前工程

FileInputStream("...");

例子

String path = XMLDaoUtil.class.getClassLoader().getResource("user.xml").getPath();

XMLWriter writer = new XMLWriter(new FileOutputStream(path),format);

writer.write(doc);

相对路径（不以斜杠开头的路径，要参考当前所在的路径去拼新的路径）---除了在必须使用的情况外，都不要使用相对路径：

如果直接写相对路径或写./相对路径的话，相对路径替换当前路径最后一级

如果写../相对路径，则替换当前路径的最后一级路径的上一级路径。

如果想替换更高层，则写多个../

4、URL编码

1.由于HTTP协议规定URL路径中只能存在ASCII码中的字符，所以如果URL中存在中文或特殊字符需要进行URL编码。

2.编码原理：

将空格转换为加号（+）

对0-9,a-z,A-Z之间的字符保持不变

对于所有其他的字符，用这个字符的当前字符集编码在内存中的十六进制格式表示，并在每个字节前加上一个百分号（%）。如字符“+”用%2B表示，字符“=”用%3D表示，字符“&”用%26表示，每个中文字符在内存中占两个字节，字符“中”用%D6%D0表示，字符“国”用%B9%FA表示调对于空格也可以直接使用其十六进制编码方式，即用%20表示，而不是将它转换成加号（+）

说明：

如果确信URL串的特殊字符没有引起使用上的岐义或冲突你也可以对这些字符不进行编码，而是直接传递给服务器。例如，http://www.it315.org/dealregister.html?name=中国&password=123

如果URL串中的特殊字符可能会产生岐义或冲突，则必须对这些特殊字符进行URL编码。例如，服务器会将不编码的“中+国”当作“中国”处理。还例如，当name参数值为“中&国”时，如果不对其中的“&”编码，URL字符串将有如下形式：http://www.it315.org/dealregister.html?name=中&国&password=123，应编码为：http://www.it315.org/dealregister.html?name=中%26国&password=123

http://www.it315.org/example/index.html#section2可改写成http://www.it315.org/example%2Findex.html%23section2

3.在java中进行URL编码和解码

URLencoder.encode("xxxx","utf-8");

URLDecoder.decode(str,"utf-8");

5、请求重定向和请求转发的区别

1.区别

RequestDispatcher.forward方法只能将请求转发给同一个WEB应用中的组件；而HttpServletResponse.sendRedirect 方法还可以重定向到同一个站点上的其他应用程序中的资源，甚至是使用绝对URL重定向到其他站点的资源。

如果传递给HttpServletResponse.sendRedirect 方法的相对URL以“/”开头，它是相对于服务器的根目录；如果创建RequestDispatcher对象时指定的相对URL以“/”开头，它是相对于当前WEB应用程序的根目录。

调用HttpServletResponse.sendRedirect方法重定向的访问过程结束后，浏览器地址栏中显示的URL会发生改变，由初始的URL地址变成重定向的目标URL；调用RequestDispatcher.forward 方法的请求转发过程结束后，浏览器地址栏保持初始的URL地址不变。

HttpServletResponse.sendRedirect方法对浏览器的请求直接作出响应，响应的结果就是告诉浏览器去重新发出对另外一个URL的访问请求；RequestDispatcher.forward方法在服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果，并不知道在服务器程序内部发生了转发行为。

RequestDispatcher.forward方法的调用者与被调用者之间共享相同的request对象和response对象，它们属于同一个访问请求和响应过程；而HttpServletResponse.sendRedirect方法调用者与被调用者使用各自的request对象和response对象，它们属于两个独立的访问请求和响应过程。

2.应用场景（参照图想）

通常情况下都用请求转发，减少服务器压力

当需要更新地址栏时用请求重定向，如注册成功后跳转到主页。

当需要刷新更新操作时用请求重定向，如购物车付款的操作。

## 五、会话技术

1.会话技术：从浏览器开始访问服务器，到关闭浏览器，这期间发生了许多次请求和响应，这个过程就叫做一次会话。

2.问题：如何在一次会话中保存会话相关的数据。

3.Cookie：将会话相关的数据保存到浏览器中，并且在每次访问服务器时都带过去。

3.1javax.servlet.http.Cookie，可以直接利用此类的构造方法创建一个Cookie，创建出来的Cookie需要设置一个名称和值

3.2response身上具有addCookie的方法，可以将创建出来的组织成响应消息中的set-cookie头，通知浏览器保存该cookie

3.3request身上具有getCookies方法，可以获取浏览器带过来的所有Cookie

3.4Cookie方法：注意,浏览器是根据cookie的名称加上cookie的path来区分是否是同一个cookie的，如果需要覆盖之前的cookie，除了保证名称相同外还要保证path也相同。

public Cookie(String name,String value)利用构造方法创建一个Cookie对象，在创建的时候就要指定该Cookie的名和值

setValue与getValue方法 设置或者获取Cookie的值

setMaxAge与getMaxAge方法 如果不设置cookie的MaxAge（或将其值设置为负值），则默认情况下浏览器会将cookie保存在浏览器的内存中，会随着浏览器关闭而消失。如果设置为一个正值，则代表该Cookie要保存的以秒为单位的时间值，如此，该cookie将会被浏览器保存到硬盘中去。如果将MaxAge设置为0，则是通知浏览器去删除该Cookie。

setPath与getPath方法 用来指定访问哪个ULR及其子URL时带上此cookie，如果不设置此值，则浏览器默认会将发送该cookie的servlet所在的路径作为path使用。

例如：setPath("/Day06")则/Day06/.../...的路径都会带上该Cookie

如果发送该Cookie的Servlet是 /Day06/servlet/Demo1Servlet，并且未设置setPat,则浏览器在访问/Day06/servlet/..时会带上该cookie

setDomain与getDomain方法 设置cookie对应的域名，此方法一旦调用，则浏览器会认为该cookie是一个第三方cookie而拒收

getName方法 获取该cookie的名字，注意没有setName方法，一个Cookie一旦创建出来就不能修改名字了

3.5浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。

实验： 利用cookie实现显示上次访问时间

!~~利用cookie显示最近浏览的商品

4.HttpSession:在服务器中，为浏览器创建独一无二的内存空间，在其中保存会话相关的信息。

4.1session作为域使用：他是j2ee中四大域对象之一，作用范围为整个会话。

4.2session的生命周期：在第一次调用reqeust.getSession()方法的时候，服务器会检查是已经有对应的session，如果没有就在内存中创建一个session并返回。

当一段时间内session没有被使用，一般为30分钟(此值可以在web.xml中配置<session-config>来配置，也可以使用TomcatManager进行配置)，则服务器会销毁该session

当服务器关闭时，没有到期的session也会跟着销毁。

如果调用session提供的invalidate(),可以立即销毁session。

4.3session的原理：在服务器第一次调用request.getSession()方法的时候，会在内存中创建一个session对象，此对象具有一个独一无二的id值，此id值将会以cookie（JSESSIONID）的形式发送给浏览器，浏览器以后每次访问都会带着此cookie，服务器就利用此cookie区分浏览器找到对应的session空间。

4.4同一电脑内的不同浏览器使用同一session：JSESSIONID这个cookie默认是保存在浏览器内存中的，我们可以自己创建一个同名同path的Cookie，并设置maxage值，使其被保存在硬盘中，从而实现统一电脑中不同浏览器公用一个JSESSIONID从而使用同一个session。

4.5使禁用Cookie的浏览器也可以使用session:由于session是基于cookie运行的，如果禁用了cookie则会导致session不可用，我们可以将提供给这种浏览器的所有的URL进行重写，在所有的URL后跟上JSEESIONID，从而保证即使禁用了Cookie也能以URL的形式带回JSESSIONID，从而可以使用session。要重写所有的URL是一项成本很高的工作，一般我们不会这么做。

response. encodeRedirectURL(java.lang.String?url)如果此url是作为重定向操作的地址时使用此方法

response. encodeURL(java.lang.String?url)如果此url是普通连接则使用此方法

实验：利用session实现简单的购物功能，并提供对同一台电脑上的多个浏览器共享session的支持以及对禁用cookie浏览器的支持。

实验：使用Session完成用户登陆：当用户登录时在session中保存用户名，在其他页面就可以检查session中是否存在用户名，如果存在则认为已经登录过。注销的过程就是将session杀死的过程。

实验：使用session完成防止表单重复提交：当提供表单页面时，在表单中隐藏一个随机数值，并且将该随机数保存到session中，当表单提交时，检查提交上来到随机数与session中的随机数是否相同，如果相同则允许注册，注册后立即删除session中的随机数，如果不同则认为是表单的重复提交。

？？作业：利用session实现一次性验证码.

\*request.getSession()和request.getSession(false);的不同之处：前面的方法一调用，就会去检查是否有对应的session，没有就创建，有就取回。后面的方法只会去检查，如果有就取回，如果没有也不创建。

5.ServeltContext 、reqeust、session域的比较

servletContext 的作用域是整个web应用，随着服务器启动而创建，如果应用被移除出主机或服务器关闭则销毁。

request 的作用域是整个请求链，每一次请求都会创建一个request，当请求结束时request销毁。

session 的作用于是整个会话，第一次调用reqeust.getSession时创建，当一段时间没有使用或服务器关闭或调用session.invalidate方法时销毁

5.2什么时候用ServeltContext什么时候用reqeust什么时候用session？

如果一个数据只是用来显示的话就用request域

如果一个数据除了用来显示以外我一会还要用，这时候用session

如果一个数据除了用来显示以外还要给别人用，这时候用ServletContext域

## 六、Jsp和标签技术

一、jsp基础

1.JSP概念

Servlet是j2ee提供的动态资源开发技术，是以java的程序的形式进行开发，在java中书写HTML标签是一件十分头疼的事情，所以人们开发出了JSP，看起来像是HTML一样，但是通过服务器的编译最终可以生成Servlet

2.JSP的组成部分

2.1模版元素

直接书写在JSP中的HTML内容，看起来就像写HTML一样的方便，但是最终会在被翻译成Servlet的过程中 out.write()直接输出

2.2脚本表达式

<%= 表达式%> 接受的是一段java表达式，在JSP翻译到Servlet的过程中，将会计算表达式的值，利用out.write()输出出去

2.3脚本片段

<% %>直接可以在脚本片段中书写java源代码，其中的源代码将会直接拷贝到翻译过来的servlet中的响应位置上。

2.4JSP声明

<%! %>在其中可以写java代码，其中的源代码会被拷贝到servlet中的service方法之外，可以利用它来为servlet增加成员方法、成员变量、静态代码块

2.5JSP注释

<%-- --%>被jsp注释包围起来的内容将不会被服务器翻译到servlet之中，要注意区分html注释和java注释的区别

jsp注释不会被翻译到servlet中，会在翻译时遗弃

java注释不会被编译到class文件中，会在编译时遗弃

html注释将会当作模版元素，直接输出到浏览器中，浏览器将不会显示html注释中的内容

2.6JSP指令

2.6.1page指令

用来通知翻译引擎，如何翻译当前的JSP

[ language="java" ] 当前JSP使用的开发语言

[ extends="package.class" ] 当前jsp翻译成servlet后要继承的类，注意此值必须是一个servlet的子类，一般情况下不要改

[ import="{package.class | package.\*}, ..." ] 导入需要使用到的包 java.lang.\*;javax.servlet.\*;javax.servlet.jsp.\*;javax.servlet.http.\*;注意要么分开写,要么写一块,但只能用逗号分隔

[ session="true | false" ] 用来指定当前页面是否使用session，默认值为true.如果设置为true，则翻译过来的servlet中将会有对session对象的引用，于是可以直接在jsp中使用session隐式对象。但是这将导致一旦访问jsp就会调用request.getSession()方法，可能导致不必要的空间浪费。如果确定jsp中不需要session可以设为false

[ buffer="none | 8kb | sizekb" ] out隐式对象所使用的缓冲区的大小

[ autoFlush="true | false" ] out隐式对象是否自动刷新缓冲区，默认为true，不需要更改

[ isThreadSafe="true | false" ] 翻译过来的servlet是否实现SingleThreadModel

[ errorPage="relative\_url" ] 如果页面出错，将要跳转到的页面，除了在jsp中使用此属性指定错误页面外也可以在web.xml中配置整个web应用的错误页面，如果两个都设置则jsp中的此属性起作用

<%@ page errorPage="/errorpage.jsp" %> IE9不起作用？？？

[ isErrorPage="true | false" ] 如果设置此属性为true,翻译过来的servlet中将含有Exception隐式对象,其中封装的就是上一个页面中抛出的异常对象

[ contentType="mimeType [ ;charset=characterSet ]" | "text/html ; charset=ISO-8859-1" ] 和jsp乱码相关的指令,用来指定jsp输出时,设置的Content-Type响应头用来指定浏览器打开的编码

[ pageEncoding="characterSet | ISO-8859-1" ] 服务器翻译jsp时使用的编码集.如果向防止jsp乱码,应该保证文件的保存编码和jsp翻译成servlet用的编码以及输出到浏览器后浏览器打开的编码一致.此属性一旦设置好,翻译引擎会间接帮我们设置content-type属性.

[ isELIgnored="true | false" ] 当前页面是否使用el表达式,设置为false时表示启用el,j2ee4.0开始默认支持,j2ee4.0一下做开发时,如果要使用el表达式,需将此属性设置为fals

2.6.2include指令

<%@ incluede file=""%> 静态引入其他页面的内容

静态引入和动态引入的区别

\*静态引入：在源文件级别进行合并，多个jsp生成一个servlet，最终由这一个servlet生成响应。推荐使用。案例是<%@include %>

\*动态引入：在运行时将多个输出进行合并，多个jsp分别生成多个servlet，最终由这多个servlet生成响应，组成一个输出流，提供响应。执行效率没有静态引入高。案例是servletContext、request、pageContext、<jsp:include> jstl<c:impport>

2.6.3taglib指令

<%@ taglib uri="" prefix=""%>用来引入标签库。

uri指定被引入.tld文件的名称空间

prefix 对该名称空间的一个缩写

2.7JSP中的九大隐式对象

JSP引擎在调用JSP对应的\_jspServlet时，会传递或创建9个与web开发相关的对象供\_jspServlet使用。

config

application

request

response

session

out

page（this）

pageContext

Exception

2.7.1out对象

可以将他理解成response.getWriter()获得的PrintWriter.

它自带一个缓冲区，其大小收page指令中的buffer的设定限制。当缓冲区满或缓冲区被关闭或当前jsp页面结束，则此缓冲区中的内容将被刷新到response.getWriter（）的缓冲区中。

2.7.2PageContext对象

（1）获取其它八大隐式对象，可以认为是一个入口对象。

（2）获取其所有域中的数据

pageContext操作所有域中属性的方法

\*\*可以指定域的范围去获得指定范围的域中的内容

public java.lang.Object getAttribute(java.lang.String?name,int?scope)

public void setAttribute(java.lang.String?name, java.lang.Object?value,int?scope)

public void?removeAttribute(java.lang.String?name,int?scope)

pageContext中代表域的常量

PageContext.APPLICATION\_SCOPE

PageContext.SESSION\_SCOPE

PageContext.REQUEST\_SCOPE

PageContext.PAGE\_SCOPE

findAttribute方法,在四大域中搜寻属性，搜寻的顺序是page域、request域、session域、application域，从小域到大域开始搜索，如果搜索到就直接获取该值，如果所有域中都找不到，返回一个null(与el表达式不同，此处返回null，对网页是不友好的)

（3）作为域对象使用

作用的范围：整个jsp页面，是四大作用域中最小的一个。

声明周期：当对jsp的请求开始时生成，当响应结束时销毁。

（4）跳转到其他资源

其身上提供了forward和include方法，简化转发和包含的操作

pageContext.forward("");

pageContext.include("");

\*8.为jsp配置对外访问路径

<servlet>

<servlet-name>forward</servlet-name>

<jsp-file>/jsp\_Tag/forward.jsp</jsp-file>

</servlet>

<servlet-mapping>

<servlet-name>forward</servlet-name>

<url-pattern>/forward/\*</url-pattern>

</servlet-mapping>

9.域的总结

如果一个数据只是在当前jsp页面中使用使用page域

如果一个数据只是当前请求处理后用来展示放在request域

如果一个数据除了用来当前请求使用，我过一会还要用，放在session域中

如果一个数据除了当前请求使用，过一会别人还要用，放在application域中。

===============================================================================

二、标签：JSP既可以用来生成HTML页面，也可以直接书写java源码处理逻辑，这就导致了很多开发者在JSP出现初期，只用JSP做开发，这个JSP页面十分庞大、充满了java源码和HTML标签、许多百分号，逻辑结构混乱，不宜调试程序和页面美化。于是人们希望将java源码从JSP页面中尽量抽离，但是把所有java源码都抽走是不现实的，最基本的获取属性、简单的页面逻辑还是需要的，于是，sun公司就提供了JSP中的标签开发技术，以一个标签代表一种功能的java代码，是整个jsp看起来更像一个HTML，并且不丢失JSP进行逻辑处理的功能。

1.JSP标签：由sun公司提供，属于jsp规范中的内容，不需要引入第三方标签库

<jsp:include>用来替代request.getRequestDispatcher().include()

<jsp:forward>用来替代request.getRequestDispatcher().forward()

<jsp:param>配合前两个标签使用，可以在包含或转发时，带一些参数过去

<jsp:include page="/jsp/head.jsp"></jsp:include>---动态引入

<jsp:forward page="relativeURL | <%=expression%>" /> ---- 请求转发

<jsp:param>当使用<jsp:include>和<jsp:forward>标签引入或将请求转发给其它资源时，可以使用<jsp:param>标签向这个资源传递参数。使用request.getParameter（）去获取，注意乱码问题，在传递参数之前设置request.setCharacterEncoding（试验是通过get方式提交过去的）

<jsp:useBean id="beanName" class="package.class" scope="page|request|session|application"/>在指定域中搜寻名字为id设置值的bean，如果没有就在该域中创建一个

<jsp:setProperty name="beanName"

{

property="propertyName" value="{string | <%= expression %>}" |

property="propertyName" [ param="parameterName" ] |

property= "\*"

}/> 用于为bean的属性赋值name指定bean的名字，property指定要设定的属性的名字，value指定设定的属性的值，param用来指定使用哪个请求参数设定该属性，property可以设置为\*,用来将所有的请求参数自动设置懂啊bean对应的属性上

<jsp:getProperty name="beanInstanceName" property="PropertyName" />用于获取属性的值输出到输出流中，其中name指定bean的名字，property指定bean上属性的名字

2.el表达式：替代<%= %>脚本表达式，在j2ee1.4以前默认是不支持el，如果需要需要指定page指令[isELIgnored="true | false" ]为false，j2ee4.0后默认支持el。若想正确解析EL表达式，需使用支持Servlet2.4/JSP2.0技术的WEB服务器

* **注意：有些Tomcat服务器如不能使用EL表达式**

（1）升级成tomcat6

（2）在JSP中加入<%@ page isELIgnored="false" %>

（0）只能用来获取域中的属性，不能设置属性，也只能从域中获取属性。

（1）用中括号的地方都可以用点替代，除了中括号中是数字或者包含特殊字符的情况下不能用点替代

（2）用中括号括起来的内容，如果用双引号引起来则表示使用的是一个键，如果没有用双引号引起来则表示是一个变量，先到域中去搜，搜出来的结果作为键

2.1获得域中的属性——当获取不到值时输出空字符串””

${propName}在四个域中搜寻proName属性，输出该值到输出流中

${pageScope/requestScope/sessionScope/applicationScope.proName}获取指定域中的属性

${attr[0]}获取域中的集合的指定元素

${list[0]}获取list中的指定位置元素

${map.keyName}获取map中指定键的值

${bean.propName}获取javaBean的属性，可以认为是调用了javaBean的getXXX方法，

~最重要的一个应用场景：在写路径的时候最好不要把web应用名称写死，java中应该用request.getContextPath去获取。jsp中就用el获取：${pageContext.request.contextPath}

~.和[]区别使用点的地方都可以用中括号，如果属性名是数字或包含特殊符号（.-）就必须使用中括号。例子：${map["first.name"]}

2.2进行简单运算

（1）算数运算：所有参与元算的元素都会被转成数字，如果不能转就报错，空元素参与运算当作没参与。

（2）关系运算：

（3）逻辑运算:

（4）empty/not empty判断一个对象或集合或数组是否为空或长度为0

（5）三元表达式 name == null ? "张三" : name;

2.3获取web开发常用对象——EL表达式的11个隐含对象

EL表达式语言中定义了11个隐含对象，使用这些隐含对象可以很方便地获取web开发中的一些常见对象，并读取这些对象的数据。

pageContext:代表pageContext对象，注意和pageScope进行区分

pageScope：代表page域，可以用来获取page域中保存属性的map对象

reqeustScope：代表reqeust域，可以用来获取reqeust域中的属性map对象

sessionScope：代表session域，可以用来获取session域中的属性map对象

applicationScope：代表application域，可以用来获取application域中的属性

param 代表请求参数组成的map集合${param.userName}

paramValues 代表请求参数组成的map集合，但是此集合的value是String[]，用来获取一组多值的param

header 获取请求头组成的map

headerValues 获取请求头组成的map但是value是一个String[]，用来获取一名多值的head

cookie 获取cookie组成的map对象，此map的值是一个cookie对象${cookie.cookieName.cookieValue}

initParam 以map封装的web.xml中配置的整个web应用的初始化参数

2.4调用java方法

在EL表达式中调用的只能是Java类的静态方法。

这个Java类的静态方法需要在TLD文件中描述，才可以被EL表达式调用。

EL自定义函数用于扩展EL表达式的功能，可以让EL表达式完成普通Java程序代码所能完成的功能。

el表达式可以调用java中的静态方法,分如下步骤完成：

(1)编写一个类，其中应该包含要使用el调用的静态方法

(2)编写一个tld文件，描述该方法的调用，在创建tld文件时应选用2.0版本的jsp配置，指定名称空间uri和缩写prefix

(3)在tld文件中配置方法信息

<function>

<name>encodeURL</name>el在调用时所使用的方法名

<function-class>cn.itheima.util.EncodeURL</function-class>静态方法所在的类全路径名

<function-signature>

java.lang.String EncodURL( java.lang.String )//对该方法的描述：返回值类型 方法名(参数类型)

</function-signature>

</function>

(4)在jsp中使用<%@ taglib uri="" prefix="ppp"%>引入tld文件

(5)在jsp中使用${ppp:encodeURL("xxxx")}调用

2.5SUN提供的EL函数库，基本上都是对字符串进行操作的函数，请参考张老师写的《第8章 标准标签库\_0519.doc》

## 七、jstl和javaee开发模式

3.JSTL标签库，在javaee4.0需要导入JSTL相关的jar包，在javaee5.0开始，默认已经包含了此jar包。还要需要用<%@ taglib%>指令引入标签库

3.1介绍

JavaServer Pages Standard Tag Library

由JCP（Java Community Process）指定标准

提供给 Java Web 开发人员一个标准通用的标签函数库

和 EL 配合来取代传统直接在页面上嵌入 Java 程序（Scripting）的做法，以提高程序可读性、维护性和方便性

一般我们使用JSTL1.1以上的版本，应为从这个版本开始支持EL表达式

JSTL1.0默认不支持el表达式，不建议使用

3.2JSTL标签库

\*\*\*\*核心标签库 (core) --- c

国际化标签 fmt

数据库标签 sql --Servlet

XML标签 xml

JSTL函数(EL函数) el

3.3JSTL核心标签库

<c:out> 标签用于输出一段文本内容到pageContext对象当前保存的“out”对象中。

<c:set>标签用于把某一个对象存在指定的域范围内，或者设置Web域中的java.util.Map类型的属性对象或JavaBean类型的属性对象的 属性。

<c:remove>标签用于删除各种Web域中的属性

<c:catch>标签用于捕获嵌套在标签体中的内容抛出的异常，其语法格式如下：<c:catch [var="varName"]>nested actions</c:catch>

<c:if test=“”>标签可以构造简单的“if-then”结构的条件表达式

<c:choose>标签用于指定多个条件选择的组合边界，它必须与<c:when>和<c:otherwise>标签一起使用。使用<c:choose>，<c:when>和<c:otherwise>三个标签，可以构造类似 “if-else if-else” 的复杂条件判断结构。

<c:forEach>标签用于对一个集合对象中的元素进行循环迭代操作，或者按指定的次数重复迭代执行标签体中的内容。

<c:forTokens>用来浏览一字符串中所有的成员，其成员是由定义符号所分隔的

<c:param>标签 在JSP页面进行URL的相关操作时，经常要在URL地址后面附加一些参数。<c:param>标签可以嵌套在<c:import>、<c:url>或<c:redirect>标签内，为这些标签所使用的URL地址附加参数。

<c:import> 标签,实现include操作

<c:url>标签用于在JSP页面中构造一个URL地址，其主要目的是实现URL重写。URL重写就是将会话标识号以参数形式附加在URL地址后面

<c:url value="/index.jsp"context="/day04" var="myurl2" scope="page">

<c:param name="username" value="张三"></c:param>

</c:url>

<c:redirect>标签用于实现请求重定向

## 九、自定义标签和国际化

4. 自定义标签

1.自定义标签技术出现的原因：虽然有第三方组织提供了很多标签，但是这些都是一些通用标签，开发中常常需要根据业务需求使用jsp页面，这个时候通用的标签就不够用了，我们需要自己去开发标签库。

自定义标签开发的两个步骤：写一个类实现Tag接口，写一个tld文件对自定义标签进行描述。

自定义标签的生命周期：在第一次被使用时创建，每次使用都调用setPageContext、setParent、doStartTag、doEndTag方法，使用过后并不销毁，一直驻留在内存中为下一次自定义标签使用而服务，直到服务器停止或web应用被移除出容器，自定义标签处理类被销毁，销毁之前调用release方法

2.标签处理类生命周期

2.1传统标签处理类的生命周期：传统标签处理类在标签被第一次使用时创建实例，从此驻留内存为后续请求提供服务，其中每次标签执行都会依次执行setPageContext(PageContext pc) 、setParent(Tag t) 、doStartTag() 、doEndTag() 、~在服务器停止之前调用release()

2.2简单标签的生命周期：setJspContext(JspContext pc) 、setParent(JspTag parent) 如果没有父标签就不调用、如果有属性就调用对应的setXXX方法如果有el表达式，则计算结果后传入、如果有标签体调用setJspBody(JspFragment jspBody) 、doTag() 执行标签体处理.每次访问该标签时都创建简单标签实例，每次使用后销毁该实例。

3.传统标签的继承结构：

JspTag

（1）传统标签

Tag接口-----定义了一个标签处理类应具有的最基本的方法和属性（EVAL\_BODY\_INCLUDE dostart方法返回表示执行标签体，SKIP\_BODY dostart方法用，跳过标签体。EVAL\_PAGE

用在doendtag里通知后续页面继续执行，SKIP\_PAGE doendtag里通知后续页面不再执行）

|

|----IterationTag接口 (提供了doAfterBody() 在标签体执行过后立即执行，并提供EVAL\_BODY\_AGAIN/EVAL\_BODY\_INCLUDE 供doafterbody方法返回表示要重新执行标签体)

|

|----TagSupport类(提供了对pageContext的引用)

|

|--BodyTag接口（EVAL\_BODY\_BUFFERED 在doStartTag方法中返回通知标签处理器去缓存标签体bodyContent）

|

|----BodyTagSupprot类 （getBodyContent() 获取缓存对象bodyContent）

（2）简单标签（简单标签的标签体不能包含脚本内容，所以tld文件中配置body-content时不能配置为JSP）

SimpleTag接口

|

|---SimpleTagSupport实现类（getJspContext() 获取代表jsp页面的pageContext，getJspBody() 代表标签体内容的JspFragment对象）

JspFragment：invoke(Writer out) 此方法一经调用会将标签体输出到对应的流中。如果你直接给一个null默认输出到pageContext.getOut()中

如果doTag（）方法抛出SkipPageException异常，标签后的jsp页面就不再执行。

（3）自定义标签声明属性：在标签处理类中提供setXXX方法，并在tld文件中通过<attribute>标签声明属性的特性即可。

4.自定义标签应该具有的最基本的功能

实验1：控制jsp页面某一部分内容是否执行。<c:if>

实验2：控制整个jsp页面是否执行。

实验3：控制jsp页面内容重复执行。<c:forEach>

实验4：修改jsp页面内容输出。<c:out> HTML转义

实验5：获取属性

作业：模拟c标签中choose when otherwise的功能

5.tld文件配置自定义标签

配置时候，如果myeclipse不进行提示可以，更改taglib的声明信息，复制xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee 拼接到后面的文件名就可以提示了。

<tag>

<name>demo10</name>-----定义标签的名字

<tag-class>cn.itheima.simple.Demo10</tag-class>----标签处理类

<body-content>scriptless</body-content>----标签体内容JSP表示任意jsp内容，empty代表只能为空、scriptless代表不能包含脚本内容、tagdependent代表其中的内容是用来给服务器用的，不是输出到浏览器的内容。比如标签体为一段sql语句

<attribute>

<name>times</name>---属性的名字(名字必须有)

<required>true</required>---是否是一个必须的属性

<rtexprvalue>true</rtexprvalue>---是否支持el表达式

<type>int</type>---属性的类型

</attribute>

</tag>

5.(1)是否执行标签体（2）是否执行标签之后的内容(3)循环执行标签体（4）修改标签体内容

(5)具有属性的标签

6.简单标签：

SimpleTag--setJspContext(JspContext pc) setParent(JspTag parent) setJspBody(JspFragment jspBody) doTag()

|

|--SimpleTagSupport

生命周期：每一此使用简单标签时都会重新创建一个简单标签对象，每次使用调用setJspContext方法将pageContext传入、setParent方法将父标签传入（如果没有父标签不调用）、如果有属性调用setXXX将属性设置进来、调用setJSPBody方法将标签体缓存对象传入、调用doTag方法处理标签

（1）是否执行标签体：如果不做任何操作那么缓存在JspFragment对象中的标签体内容没有进行过输出，自然页面也就没有输出。如果想进行输出就调用 fragment.invoke(wirter)将缓存的内容发送到给定的流中，此时可以给out流，将标签体输出到浏览器。如果invoke方法直接给一个null作为参数默认就输出到out流中

（2）是否执行标签之后的内容：默认就执行标签之后的内容，如果想控制不执行标签之后的内容，抛出一个javax.servlet.jsp.SkipPageException

（3）循环执行标签体：多次执行invoke方法即可

（4）修改标签体内容：将JspFragment中的内容调用invoke方法输出到StringWtirer，再StringWtirer获取字符串修改输出

（5）具有属性的标签：和传统标签相同

<tag>

<name>it2</name>标签的名称

<tag-class>cn.itheima.tag.Iterationtag2</tag-class>标签处理类

<body-content>JSP</body-content>(简单标签中不可以配置JSP，因为从jsp2.0标准开始就不再推荐在jsp页面中写java代码了)标签体可以包含的类型，JSP：可以包含任意JSP内容 Empty：标签体必须为空 Scriptless：可以是除了java源码以外的JSP内容 tagdependent：声明标签体是给服务器用的不要输出至页面，比如标签体为一段sql语句

<attribute>

<name>times</name>属性的名称

<required>true</required>属性是否必须存在

<rtexprvalue>true</rtexprvalue>属性是否支持el表达式

<type>int</type>属性的类型，可以是java基本类型或全路径名的java类

</attribute>

</tag>

一：国际化

1.国际化的概念：一款软件希望不同的国家和地区的使用者都可以使用，这个时候软件中的一些内容和数据需要根据用户地区信息不同而展示成不同的样子。

2.国际化的组成部分：

2.1页面中静态内容的国际化（导航栏、提示信息、错误信息等等。。）

利用javase提供 ResourceBundle类管理资源包的功能，实现web应用中根据浏览器不同语言环境提供不同信息的过程

\*ResourceBundle在寻找资源文件时按如下顺序寻找：如果有指定的找指定的,没有指定的找近似的，如果近似的没有找当前平台环境的，如果还没有就用默认的。

\*<fmt:setBundle>这种方式，先找指定的，找近似的，找默认的

\*资源文件由若干properties文件组成，这些文件都必须有相同的基名，然后根据保存的地域化信息的不同再跟上不同的后缀。注意properties文件不能出现ascii编码文字以外的内容，中文要进行unicode编码表示

\*资源包必须处在src目录之下

\*J2SE中的实现

ResourceBundle bundle = ResourceBundle.getBundle("myProp");

String value = bundle.getString("prop1");

System.out.println(value)

\*JSP源码方式实现：

<h1>java方式实现国际化</h1>

<%

ResourceBundle bundle = ResourceBundle.getBundle("config", request.getLocale());

%>

<form action="#">

<%=bundle.getString("name")%>:

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

<%=bundle.getString("psw")%>:

<input type="password" name="password" />

<input type="submit" value="提交" />

</form>

<h1>JSTL提供的国际化标签实现国际化</h1>

<fmt:setLocale value="zh\_CN"/>//这里可以指定用那种语言环境，如果不设置此行则<fmt:setBundle>将根据客户机传入的Locale生成

<fmt:setBundle basename="config" scope="page" var="bundle" />

<form action="#">

<fmt:message bundle="${bundle}" key="name" />:

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

<fmt:message bundle="${bundle}" key="psw" />:

<input type="password" name="password" />

<input type="submit" value="提交" />

</form>

2.2程序产生的日期、货币、时间等等跟语言地域相关的数据的国际化

2.2.1日期

(1)SimpleDateFormat

(2)DateFormat

DateFormat.getDateInstance(DateFormat.LONG, Locale.CANADA);

DateFormat.getTimeInstance(DateFormat.SHORT,Locale.CANADA);

DateFormat.getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT,Locale.CHINA);

2.2.2NumberFormat

(1)货币转换：

NumberFormat.getCurrencyInstance(Locale.CHINA);

(2)百分号转换：

NumberFormat.getPercentInstance(Locale.CANADA)

2.2.3MessageFormat:如果一段字符串中有多个对地区敏感的信息，可以使用此类进行批量国际化设置

（1）MessageFormat类允许开发人员用占位符{0}{1}{2}…替换掉字符串中的敏感数据（即国际化相关的数据）。

（2）MessageFormat类在格式化输出包含占位符的文本时，messageFormat类可以接收一个参数数组，以替换文本中的每一个占位符。

（3）占位符有三种方式书写方式：

{argumentIndex}: 0-9 之间的数字，表示要格式化对象数据在参数数组中的索引号

{argumentIndex,formatType}: 参数的格式化类型

{argumentIndex,formatType,FormatStyle}: 格式化的样式，它的值必须是与格式化类型相匹配的合法模式、或表示合法模式的字符串。

formatType:

number

date

time

choice

fomatStyle

short

medium

long

full

integer

currency

percent

subformatpattern

（4）例子

//模式字符串

//At 12:30 pm on jul 3,1998, a hurricance destroyed 99 houses and caused $1000000 of damage

@Test

public void demo6(){

String str = "On {0}, a hurricance destroyed {1} houses and caused {2} of damage.";

Calendar calendar = Calendar.getInstance();

calendar.set(1998, 6, 3, 12, 30, 0);

Date date = calendar.getTime();

DateFormat format = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.SHORT, Locale.CHINA);

String dateStr = format.format(date);

Object [] objs = {dateStr,99,"$1000000"};

String result = MessageFormat.format(str,objs);

System.out.println(result);

}

@Test

public void test(){

//At 12:30 pm on jul 3,1998, a hurricance destroyed 99 houses and caused $1000000 of damage

String msg = "At {0,date,FULL}, a hurricance destroyed {1,number} houses and caused {2,number,currency} of damage";

MessageFormat format = new MessageFormat(msg,Locale.CHINA);

Calendar c = Calendar.getInstance();

c.set(1998, 6, 3);

Date date = c.getTime();

Object [] objs = {date,99,1000000};

String str = format.format(objs);

System.out.println(str);

}

3.练习：

请创建一个date对象，并把date对象中表示日期部分的时间值，以及表示时间部分的时间值，分别以short、long模式进行格式化输出（国家设置为中国）。

请将时间值：09-11-28 上午10时25分39秒 CST，反向解析成一个date对象。

请将整数198，输出为货币形式：$198，并将$198反向解析成整数198。

请将0.78654321，输出百分比格式，保留两位小数

At 12:30 pm on jul 3,1998, a hurricance destroyed 99 houses and caused $1000000 of damage

以上字符串中包含了时间、数字、货币等多个与国际化相关的数据，对于这种字符串，可以使用MessageFormat类对其国际化相关的数据进行批量处理。

## 八、第八天项目总结

一、为什么：要分层

使软件具有结构性，便于开发、维护和管理。

将不同功能模块独立，在需要替换某一模块时不需要改动其他模块，方便代码的复用、替换

二、层与层耦合的概念，利用工厂类解耦

在分层结构中，我们希望将各个功能约束在各自的模块（层）当中的，而当属于某一层的对象、方法“入侵”到了其他层，如将web层的ServletContext对象传入service层，或service层调用XMLDao独有的方法，就会导致层与层之间的关系过于“紧密”，当需要修改某一层时不可避免的要修改其他关联的层，这和我们软件分层最初的设想-----层与层分离，一个层尽量不依赖其他层存在，当修改一层时无需修改另一层的设想是违背的。这种“入侵”造成的“紧密”关系就早做层与层之间发生的“耦合”，而去掉这种耦合性的过程就叫做层与层之间“解耦”

利用工厂类可以实现解耦的功能

三、如何判断一项功能到底属于哪一层

某一项功能属于哪一层，往往是不能明确确定出来的，这时可以参考如下标准进行判断：

此项功能在业务逻辑上更贴近与哪一层,放在哪一层更能较少耦合

此项功能是否必须使用某一层特有的对象

如果放在哪一层都可以，那么放在哪一层更方便技术上的实现，及方便代码的编写和维护

四、异常的处理

如果一个异常抛给上一层会增加程序的耦合性，请当场解决:如将xml解析错误抛给service层，那么当换成mysqldao时，还需要修改service去掉xml解析异常的处理

如果上一层明确需要此异常进行代码的流转，请抛出：如当查找一个用户信息而用户找不到时，可以抛出一个用户找不到异常，明确要求上一层处理

如果这一层和上一层都能解决尽量在这一层解决掉

如果这一层不能解决，而上一层能解决抛给上一层

如果所有层都不能解决，则应抛出给虚拟机使线程停止，但是如果直接抛出这个异常，则还需要调用者一级一级继续往上抛出最后才能抛给虚拟机，所以还不如在出现异常的位置直接trycatch住后转换为RuntimeException抛出。：如读取配置文件出错，任何层都不能解决，转为RuntimeException抛出，停止线程。

五、其它

分层

层次感，条理分明，独立

降低耦合性，关联度降低

service层中使用接口，

UserDao dao = new XMLUserDao();---工厂设计模式、单例设计模式

如下工厂模式解耦案例

public class DaoFactory {

private static Properties prop ;

private static DaoFactory factory = new DaoFactory();

private DaoFactory(){}

static{

try{

ClassLoader loader = DaoFactory.class.getClassLoader();

String path = loader.getResource("config.properties").getPath();

FileInputStream in = new FileInputStream(path);

prop = new Properties();

prop.load(in);

}catch (Exception e) {

e.printStackTrace();

throw new RuntimeException();

}

}

public static DaoFactory newInstance(){

return factory;

}

public UserDao newDaoInstance(){

try{

String dao = prop.getProperty("dao");

Class clazz = Class.forName(dao);

return (UserDao)clazz.newInstance();

}catch (Exception e) {

e.printStackTrace();

throw new RuntimeException();

}

}

}

处理异常的原则

（1）如果异常是和当前层严密绑定的话，应该当场解决掉，防止抛到上一次，增加层与层直接的耦合性 （xmldao解析xml出错）

（2）如果异常当前层能够解决就要当场解决掉，

（3）如果异常当前层不能解决，但是明确知道上一层可以解决，抛给上一层（用户找不到）

（4）如果当前层和上一层都不能解决，数据库连接失败（将异常获取trycatch，转换为一个runtimeException抛出去）不要求上一层必须捕获

//1.保存错误日志

2.抛出一个runtimeException，上一层不接的话，直接抛给虚拟机，停止当前线程

junit测试时候查看xml文件应该看工程目录下的xml文件

将解析器和更新到xml写到工具类中。

设计异常最重要的是名字，默认包含无参构造函数，一般会覆盖写个有参构造函数

什么是异常

从下向上传递消息（返回值只能是一个，可能被占用）

上层可以实现代码的流转

工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，

通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，

该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，

开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

## 十天SQL

一.数据库

1.创建数据库

create database [if not exists] db\_name [character set xxx] [collate xxx]

\*创建一个名称为mydb1的数据库。

create database mydb1;

\*创建一个使用utf8字符集的mydb2数据库。

create database mydb2 character set utf8;

\*创建一个使用utf8字符集，并带校对规则的mydb3数据库。

create database mydb3 character set utf8 collate utf8\_bin ;

2.查看数据库

show databases;查看所有数据库

show create database db\_name; 查看数据库的创建方式

3.修改数据库

alter database db\_name [character set xxx] [collate xxxx]

4.删除数据库

drop database [if exists] db\_name;

5.使用数据库

切换数据库 use db\_name;

查看当前使用的数据库 select database();

--------------------------------------------------------------------------------------------------

二、表

1.创建表

create table tab\_name(

field1 type,

field2 type,

...

fieldn type

)[character set xxx][collate xxx];

\*\*\*\*java和mysql的数据类型比较

String ---------------------- char(n) varchar(n) 255 65535

byte short int long float double -------------- tinyint smallint int bigint float double

boolean ------------------ bit 0/1

Date ------------------ Date、Time、DateTime、timestamp

FileInputStream FileReader ------------------------------ Blob Text

\*创建一个员工表employee

create table employee(

id int primary key auto\_increment ,

name varchar(20),

gender bit default 1,

birthday date,

entry\_date date,

job varchar(20),

salary double,

resume text

);

约束:

primary key

unique

not null

auto\_increment 主键字段必须是数字类型。

外键约束

2.查看表信息

desc tab\_name 查看表结构

show tables 查看当前数据库中的所有的表

show create table tab\_name 查看当前数据库表建表语句

3.修改表结构

(1)增加一列

alter table tab\_name add [column] 列名 类型;

(2)修改一列类型

alter table tab\_name modify 列名 类型;

(3)修改列名

alter table tab\_name change [column] 列名 新列名 类型;

(4)删除一列

alter table tab\_name drop [column] 列名;

(5)修改表名

rename table 表名 to 新表名;

(6)修该表所用的字符集

alter table student character set utf8;

4.删除表

drop table tab\_name;

--------------------------------------------------------------------------------------------------

三、表记录

1.增加一条记录insert

insert into tab\_name (field1,filed2,.......) values (value1,value2,.......);

\*插入的数据应与字段的数据类型相同。

\*数据的大小应在列的规定范围内，例如：不能将一个长度为80的字符串加入到长度为40的列中。

\*在values中列出的数据位置必须与被加入的列的排列位置相对应。

\*字符和日期型数据应包含在单引号中'zhang' '2013-04-20'

\*插入空值：不指定某列的值或insert into table value(null)，则该列取空值。

\*如果要插入所有字段可以省写列列表，直接按表中字段顺序写值列表insert into tab\_name values(value1,value2,......);

\*练习：使用insert语句向表中插入三个员工的信息。

insert into emp (name,birthday,entry\_date,job,salary) values ('张飞','1990-09-09','2000-01-01','打手',999);

insert into emp (name,birthday,entry\_date,job,salary) values ('关羽','1989-08-08','2000-01-01','财神',9999);

insert into emp (name,birthday,entry\_date,job,salary) values ('赵云','1991-07-07','2000-01-02','保安',888);

insert into emp values (7,'黄忠',1,'1970-05-05','2001-01-01','战神',1000,null);

2.修改表记录

update tab\_name set field1=value1,field2=value2,......[where 语句]

\*UPDATE语法可以用新值更新原有表行中的各列。

\*SET子句指示要修改哪些列和要给予哪些值。

\*WHERE子句指定应更新哪些行。如没有WHERE子句，则更新所有的行。

\*实验：

将所有员工薪水修改为5000元。

update emp set salary=5000;

将姓名为’zs’的员工薪水修改为3000元。

update emp set salary=3000 where name='zs';

将姓名为’ls’的员工薪水修改为4000元,job改为ccc。

update emp set salary=4000,job='ccc' where name='zs';

将wu的薪水在原有基础上增加1000元。

update emp set salar=salary+4000 where name='wu';

3.删除表操作

delete from tab\_name [where ....]

\*如果不跟where语句则删除整张表中的数据

\*delete只能用来删除一行记录，不能值删除一行记录中的某一列值（这是update操作）。

\*delete语句只能删除表中的内容，不能删除表本身，想要删除表，用drop

\*同insert和update一样，从一个表中删除记录将引起其它表的参照完整性问题，在修改数据库数据时，头脑中应该始终不要忘记这个潜在的问题。

\*TRUNCATE TABLE也可以删除表中的所有数据，词语句首先摧毁表，再新建表。此种方式删除的数据不能在事务中恢复。

\*实验：

删除表中名称为’zs’的记录。

delete from emp where name='黄忠';

删除表中所有记录。

delete from emp;

使用truncate删除表中记录。

truncate table emp;

4.select操作

（1）select [distinct] \*|field1，field2，...... from tab\_name 其中from指定从哪张表筛选，\*表示查找所有列，也可以指定一个列列表明确指定要查找的列，distinct用来剔除重复行。

\*实验：

查询表中所有学生的信息。

select \* from exam;

查询表中所有学生的姓名和对应的英语成绩。

select name,english from exam;

过滤表中重复数据。

select distinct english from exam;

（2）select 也可以使用表达式，并且可以使用 as 别名

在所有学生分数上加10分特长分显示。注意:这里只是查询出来的结果集加上10,而数据库表里面的数据是不会变化的.

select name,english+10,chinese+10,math+10 from exam;

统计每个学生的总分。

select name,english+chinese+math from exam;

使用别名表示学生总分。

select name,english+chinese+math as 总成绩 from exam;

select name,english+chinese+math 总成绩 from exam;

select name english from exam;

（3）使用where子句，进行过滤查询。

\*练习：

查询姓名为XXX的学生成绩

select \* from exam where name='张飞';

查询英语成绩大于90分的同学

select name,english from exam where english>90;

查询总分大于200分的所有同学

select name,math+english+chinese as 总成绩 from exam where math+english+chinese>200 ;

\*where字句中可以使用：

\*比较运算符： > < >= <= <>

between 10 and 20 值在10到20之间

in(10,20,3)值是10或20或30

like '张pattern' pattern可以是%或者\_，如果是%则表示任意多字符，此例中张三丰 张飞 张abcd ，如果是\_则表示一个字符张\_，张飞符合。张

Is null

\*逻辑运算符——在多个条件直接可以使用逻辑运算符 and or not

\*实验

查询英语分数在 80－100之间的同学。

select name ,english from exam where english between 80 and 100;

查询数学分数为75,76,77的同学。

select name ,math from exam where math in (75,76,77);

查询所有姓张的学生成绩。

select \* from exam where name like '张%';

查询数学分>70，语文分>80的同学。

select name from exam where math>70 and chinese >80;

查找缺考数学的学生的姓名

select name from exam where math is null;

（4）Order by 指定排序的列，排序的列即可是表中的列名，也可以是select 语句后指定的别名。

Asc 升序、Desc 降序，其中asc为默认值

ORDER BY 子句应位于SELECT语句的结尾。

\*练习：

对数学成绩排序后输出。

select \* from exam order by math;

对总分排序按从高到低的顺序输出

select name ,(ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0)) 总成绩 from exam order by 总成绩 desc;

对姓张的学生成绩排序输出

select name ,(ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0)) 总成绩 from exam where name like '张%' order by 总成绩 desc;

（5）聚合函数：技巧,先不要管聚合函数要干嘛，先把要求的内容查出来再包上聚合函数即可。

count(列名)

统计一个班级共有多少学生？先查出所有的学生，再用count包上

select count(\*) from exam;

统计数学成绩大于70的学生有多少个？

select count(math) from exam where math>70;

统计总分大于250的人数有多少？

select count(name) from exam where (ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0))>250;

sum(列名)

统计一个班级数学总成绩？先查出所有的数学成绩，再用sum包上

select sum(math) from exam;

统计一个班级语文、英语、数学各科的总成绩

select sum(math),sum(english),sum(chinese) from exam;

统计一个班级语文、英语、数学的成绩总和

select sum(ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0)) as 总成绩 from exam;

统计一个班级语文成绩平均分

select sum(chinese)/count(\*) from exam ;

注意：sum仅对数值起作用，否则会报错。

AVG(列名)：

求一个班级数学平均分？先查出所有的数学分，然后用avg包上。

select avg(ifnull(math,0)) from exam;

求一个班级总分平均分

select avg((ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0))) from exam ;

Max、Min

求班级最高分和最低分（数值范围在统计中特别有用）

select Max((ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0))) from exam;

select Min((ifnull(math,0)+ifnull(chinese,0)+ifnull(english,0))) from exam;

（6）group by字句，其后可以接多个列名，也可以跟having子句对group by 的结果进行筛选。

练习：对订单表中商品归类后，显示每一类商品的总价

select product，sum(price) from orders group by product;

练习：查询购买了几类商品，并且每类总价大于100的商品

select product，sum(price) from orders group by product having sum(price)>100;

!~having 和 where 的差别： where语句用在分组之前的筛选，having用在分组之后的筛选，having中可以用合计函数，where中就不行。使用where的地方可以用having替代。

练习:查询商品列表中除了橘子以外的商品，每一类商品的总价格大于500元的商品的名字

select product,sum(price) from orders where product<>'桔子'group by product having sum(price)>500;

（7）重点：Select from where group by having order by

Mysql在执行sql语句时的执行顺序：from where select group by having order by

\*分析：

select math+english+chinese as 总成绩 from exam where 总成绩 >250; ---- 不成功

select math+english+chinese as 总成绩 from exam having 总成绩 >250; --- 成功

select math+english+chinese as 总成绩 from exam group by 总成绩 having 总成绩 >250; ----成功

select math+english+chinese as 总成绩 from exam order by 总成绩;----成功

select \* from exam as 成绩 where 成绩.math>85; ---- 成功

-------------------------------------------------------------------------------

四、约束

1.创建表时指定约束：

create table tb(

id int primary key auto\_increment,

name varchar(20) unique not null,

ref\_id int,

foreign key(ref\_id) references tb2(id)

);

create table tb2(

id int primary key auto\_increment

);

2.外键约束：

（1）增加外键：

可以明确指定外键的名称，如果不指定外键的名称,mysql会自动为你创建一个外键名称。

RESTRICT : 只要本表格里面有指向主表的数据， 在主表里面就无法删除相关记录。

CASCADE : 如果在foreign key 所指向的那个表里面删除一条记录，那么在此表里面的跟那个key一样的所有记录都会一同删掉。

alter table book add [constraint FK\_BOOK] foreign key(pubid) references pub\_com(id) [on delete restrict] [on update restrict];

（2）删除外键

alter table 表名 drop foreign key 外键（区分大小写，外键名可以desc 表名查看）;

3.主键约束：

（1）增加主键（自动增长，只有主键可以自动增长）

Alter table tb add primary key(id) [auto\_increment];

（2）删除主键

alter table 表名 drop primary key

（3）增加自动增长

Alter table employee modify id int auto\_increment;

（4）删除自动增长

Alter table tb modify id int;

--------------------------------------------------------------------------------------------------

五、多表设计

一对一（311教室和20130405班级，两方都是一）:在任意一方保存另一方的主键

一对多、多对一（班级和学生，其中班级为1，学生为多）：在多的一方保存一的一方的主键

多对多（教师和学生，两方都是多）：使用中间表，保存对应关系

--------------------------------------------------------------------------------------------------

六、多表查询

create table tb (id int primary key,name varchar(20) );

create table ta (

id int primary key,

name varchar(20),

tb\_id int

);

insert into tb values(1,'xxx');

insert into tb values(2,'yyy');

insert into tb values(3,'zzz');

insert into ta values (1,'aaa',1);

insert into ta values (2,'bbb',2);

insert into ta values (3,'ccc',3);

mysql> select \* from ta;

+----+------+-------+

| id | name | tb\_id |

+----+------+-------+

| 1 | aaa | 1 |

| 2 | bbb | 2 |

| 3 | bbb | 4 |

+----+------+-------+

mysql> select \* from tb;

+----+------+

| id | name |

+----+------+

| 1 | xxx |

| 2 | yyy |

| 3 | yyy |

+----+------+

1.笛卡尔积查询：两张表中一条一条对应的记录，m条记录和n条记录查询，最后得到m\*n条记录，其中很多错误数据

select \* from ta ,tb;

mysql> select \* from ta ,tb;

+----+------+-------+----+------+

| id | name | tb\_id | id | name |

+----+------+-------+----+------+

| 1 | aaa | 1 | 1 | xxx |

| 2 | bbb | 2 | 1 | xxx |

| 3 | bbb | 4 | 1 | xxx |

| 1 | aaa | 1 | 2 | yyy |

| 2 | bbb | 2 | 2 | yyy |

| 3 | bbb | 4 | 2 | yyy |

| 1 | aaa | 1 | 3 | yyy |

| 2 | bbb | 2 | 3 | yyy |

| 3 | bbb | 4 | 3 | yyy |

+----+------+-------+----+------+

2.内连接：查询两张表中都有的关联数据,相当于利用条件从笛卡尔积结果中筛选出了正确的结果。

select \* from ta ,tb where ta.tb\_id = tb.id;

select \* from ta inner join tb on ta.tb\_id = tb.id;

mysql> select \* from ta inner join tb on ta.tb\_id = tb.id;

+----+------+-------+----+------+

| id | name | tb\_id | id | name |

+----+------+-------+----+------+

| 1 | aaa | 1 | 1 | xxx |

| 2 | bbb | 2 | 2 | yyy |

+----+------+-------+----+------+

3.外连接

（1）左外连接：在内连接的基础上增加左边有右边没有的结果

select \* from ta left join tb on ta.tb\_id = tb.id;

mysql> select \* from ta left join tb on ta.tb\_id = tb.id;

+----+------+-------+------+------+

| id | name | tb\_id | id | name |

+----+------+-------+------+------+

| 1 | aaa | 1 | 1 | xxx |

| 2 | bbb | 2 | 2 | yyy |

| 3 | bbb | 4 | NULL | NULL |

+----+------+-------+------+------+

（2）右外连接：在内连接的基础上增加右边有左边没有的结果

select \* from ta right join tb on ta.tb\_id = tb.id;

mysql> select \* from ta right join tb on ta.tb\_id = tb.id;

+------+------+-------+----+------+

| id | name | tb\_id | id | name |

+------+------+-------+----+------+

| 1 | aaa | 1 | 1 | xxx |

| 2 | bbb | 2 | 2 | yyy |

| NULL | NULL | NULL | 3 | yyy |

+------+------+-------+----+------+

（3）全外连接：在内连接的基础上增加左边有右边没有的和右边有左边没有的结果

select \* from ta full join tb on ta.tb\_id = tb.id; --mysql不支持全外连接

select \* from ta left join tb on ta.tb\_id = tb.id

union

select \* from ta right join tb on ta.tb\_id = tb.id;

mysql> select \* from ta left join tb on ta.tb\_id = tb.id

-> union

-> select \* from ta right join tb on ta.tb\_id = tb.id; --mysql可以使用此种方式间接实现全外连接

+------+------+-------+------+------+

| id | name | tb\_id | id | name |

+------+------+-------+------+------+

| 1 | aaa | 1 | 1 | xxx |

| 2 | bbb | 2 | 2 | yyy |

| 3 | bbb | 4 | NULL | NULL |

| NULL | NULL | NULL | 3 | yyy |

+------+------+-------+------+------+

## 十一、jdbc

一、数据库驱动的概念、JDBC

数据库厂商提供的用来操作数据库用的jar包就是数据库驱动。各个厂商如果提供各自的数据库驱动的话会导致开发人员学习成本太高，所以sun公司提供了一套数据库驱动应该遵循的接口规范，这套规范就叫做JDBC，本质上是很多的接口。

由于所有的数据库驱动都遵循JDBC规范，我们在学习和使用数据库时只要学习JDBC中的接口就可以了。

二、JDBC快速入门

\*在数据库中建立好表

\*在程序中导入数据库驱动包

1.注册数据库驱动

DriverManager.registerDriver(new Driver());//缺点一：观察mysqlDriver源码发现此方法导致了数据库驱动被注册了两次。缺点二：整个程序与mysql数据库驱动绑定增加了耦合性

Class.forName(“com.mysql.jdbc.Driver”);

2.获取连接

DriverManager.getConnection(url, user, password);

~url的写法：

Oracle写法：jdbc:oracle:thin:@localhost:1521:sid

SqlServer—jdbc:microsoft:sqlserver://localhost:1433; DatabaseName=sid

MySql—jdbc:mysql://localhost:3306/sid

~url可以接的参数

user、password

useUnicode=true&characterEncoding=UTF-8

3.获取传输器

createStatement()：创建向数据库发送sql的statement对象。

prepareStatement(sql) ：创建向数据库发送预编译sql的PrepareSatement对象。

4.利用传输器执行sql语句获取结果集

executeQuery(String sql) ：用于向数据发送查询语句。

executeUpdate(String sql)：用于向数据库发送insert、update或delete语句

execute(String sql)：用于向数据库发送任意sql语句

5.遍历结果集取出结构

ResultSet以表的样式在内存中保存了查询结果，其中还维护了一个游标，最开始的时候游标在第一行之前，每调用一次next()方法就试图下移一行，如果移动成功返回true；

ResultSet还提供了很多个Get方法，用来获取查询结果中的不同类型的数据

除了next方法，还有以下方法可以用来遍历结果集：

next()：移动到下一行

Previous()：移动到前一行

absolute(int row)：移动到指定行

beforeFirst()：移动resultSet的最前面。

afterLast() ：移动到resultSet的最后面。

6.释放资源

conn是一个有限的资源，用完立即要释放表

stat占用内存，所以使用完后也要释放

rs占用内存，所以使用完后也要释放

释放时后创建的先释放

if(rs != null){

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

} finally{

rs = null;

}

}

if(stat != null){

try {

stat.close();

} catch (SQLException e) {

e.printStackTrace();

} finally{

stat = null;

}

}

if(conn != null){

try {

conn.close();

} catch (SQLException e) {

e.printStackTrace();

} finally{

conn = null;

}

}

三、PreparedStatement

1.Sql注入：由于jdbc程序在执行的过程中sql语句在拼装时使用了由页面传入参数，如果用户恶意传入一些sql中的特殊关键字，会导致sql语句意义发生变化，这种攻击方式就叫做sql注入，参考用户注册登录案例。

2.PreparedStatement是Statement的孩子，不同的是，PreparedStatement使用预编译机制，在创建PreparedStatement对象时就需要将sql语句传入，传入的过程中参数要用?替代，这个过程回导致传入的sql被进行预编译，然后再调用PreparedStatement的setXXX将参数设置上去，由于sql语句已经经过了预编译，再传入特殊值也不会起作用了。

3.PreparedStatement使用了预编译机制，sql语句在执行的过程中效率比Statement要高。

四、大数据

1.mysql数据库也可以直至在数据库中保存大文本和大二进制数据，

Text

TINYTEXT(255)、TEXT(64k)、MEDIUMTEXT(16M)和LONGTEXT(4G)

Blob

TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB

2.JDBC去操作大文本：

~插入大文本：

ps = conn.prepareStatement("insert into Demo2Text values(null,?,?)");

ps.setString(1, "钢铁是怎样练成");

File file = new File("1.txt");

ps.setCharacterStream(2, new FileReader(file), (int) file.length());

//1.Exception in thread "main" java.lang.AbstractMethodError: com.mysql.jdbc.PreparedStatement.setCharacterStream(ILjava/io/Reader;J)V

//ps.setCharacterStream(2, new FileReader(file), file.length());第三个参数是long型的是从1.6才开始支持的，驱动里还没有开始支持。

//解决方案：ps.setCharacterStream(2, new FileReader(file), (int)file.length());

//2.Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

//文件大小过大，导致PreparedStatement中数据多大占用内存，内存溢出

//-Xms256M-Xmx256M

//3.com.mysql.jdbc.PacketTooBigException: Packet for query is too large (10886466 > 1048576). You can change this value on the server by setting the max\_allowed\_packet' variable.

//数据库连接传输用的包不够大，传输大文本时报此错误

//在my.ini中配置max\_allowed\_packet指定包的大小

~查询大文本：

Reader rd = rs.getCharacterStream("content");

3.JDBC操作大二进制

~插入：

ps = conn.prepareStatement("insert into Demo3Blob values(null,?,?)");

ps.setString(1, "梦想的力量");

File file = new File("1.mp3");

ps.setBinaryStream(2, new FileInputStream(file), (int) file.length());

~查询

InputStream in = rs.getBinaryStream("content");

五、批处理

业务场景：当需要向数据库发送一批sql语句执行时，应避免向数据库一条条的发送执行，而应采用jdbc的处理机制，以提升执行效率。

实现批处理的两种方式：

1. 使用Statement.addBatch(sql)执行批处理sql语句，executeBatch()用来执行批处理语句，clearBatch()用来清空statement的sql缓存集。
2. PreparedStatement是Statement的子类，所以也可以调用它身上的方法来执行批处理sql语句。

两种方式的比较：

Statement优点是可以向数据库发送多种不同类型的sql语句。缺点是sql语句没有进行预编译，当向数据库发送多条语句相同，但参数不同的sql语句时，需要重复写上多条sql语句。

PreparedStatement优点是发送的是预编译后的sql语句，执行效率高。缺点是只能应用在sql语句相同，但参数不同的批处理中。

## 十二、事务

一、事务

1.事务的概念：事务是指逻辑上的一组操作，这组操作要么同时完成要么同时不完成。参考转账操作。

2.如果你自己不去控制事务，数据库默认一条sql语句就处在自己单独的事务当中。

3.也可以使用命令去开启一个事务：

start transaction;--开启事务，这条语句之后的sql语句将处在一个事务当中，这些sql语句并不会立即执行

Commit--提交事务，一旦提交事务，事务中的所有sql语句才会执行。

Rollback -- 回滚事务，将之前所有的sql取消。

conn.setAutoCommit(false);

conn.commit();

conn.rollback();

conn.setSavePoint();

conn.rollback(sp);

4.事务的四大特性ACID

~原子性：事务的一组操作是原子的不可再分割的，这组操作要么同时完成要么同时不完成。

~一致性: 事务在执行前后数据的完整性保持不变。数据库在某个状态下符合所有的完整性约束的状态叫做数据库具有完整性。在解散一个部门时应该同时处理员工表中的员工保证这个事务结束后，仍然保证所有的员工能找到对应的部门，满足外键约束。

~隔离性：当多个事务同时操作一个数据库时，可能存在并发问题，此时应保证各个事务要进行隔离，事务之间不能互相干扰。

~持久性：持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，不能再回滚。

5.事务的隔离性导致的问题（所有的问题都是在某些情况下才会导致问题）

~脏读：一个事务读取到了另一个事务未提交的数据。

1 | a | 1000

2 | b | 1000

b--->a

start transaction;

update account set money=money-100 where name='b';

update account set money=money+100 where name='a';

rollback;

select \* from account where name = 'a';1000 1000

~不可重复读：在一个事务内读取表中的某一行数据，多次读取结果不同.

在一个事务内读取表中的某一行数据，多次读取结果不同.

start transaction:

活期存款：1000

定期存款：1000

固定资产: 2000

--------------

开启事务

取走获取存款1000

提交事务

--------------

总资产：3000

~幻读(虚读)：一个事务读取到了另一个事务插入的数据（已提交）

a 2000

b 2000

c 2000

start transaction；

select sum(money) from account;6000

--------------

开启事务

创建一个账户并存入1000块钱

提交了事务

--------------

select count(\*)from account;4

avgMoney = allMoney/count;6000/4=1500

6.数据库的隔离级别

~Read uncommitted：如果将数据库设定为此隔离级别，数据库将会有脏读、不可重复度、幻读的问题。

~Read committed：如果将数据库设定为此隔离级别，数据库可以防止脏读，但有不可重复度、幻读的问题。

~Repeatable read: 如果将数据库设定为此隔离级别，数据库可以防止脏读、不可重复度，但是不能防止幻读。

~Serializable：将数据库串行化,可以避免脏读、不可重复读、幻读。

安全性来说：Serializable>Repeatable read>Read committed>Read uncommitted

效率来说：Serializable<Repeatable read<Read committed<Read uncommitted

通常来说，一般的应用都会选择Repeatable read或Read committed作为数据库隔离级别来使用。

mysql默认的数据库隔离级别为：REPEATABLE-READ

如何查询当前数据库的隔离级别？select @@tx\_isolation;

如何设置当前数据库的隔离级别？set [global/session] transaction isolation level ...;

~此种方式设置的隔离级别只对当前连接起作用。

set transaction isolation level read uncommitted;

set session transaction isolation level read uncommitted;

~此种方式设置的隔离级别是设置数据库默认的隔离级别

set global transaction isolation level read uncommitted;

7.做实验，演示脏读、不可重复度、幻读

~演示脏读

~演示不可重复度

~演示幻读：幻读有可能发生有可能不发生，但是一旦发生就可能造成危害。

~演示Serializable:基于锁机制运行

8.锁机制：

共享锁：共享锁和共享锁可以共存。

排他锁：排他锁和所有锁都不能共存。

在非串行化下，所有的查询都不加锁，所有的修改操作都会加排他锁。

在串行化下，所有的查询都加共享锁，所有的修改都加排他锁。

死锁

9.更新丢失

如果多个线程操作，基于同一个查询结构对表中的记录进行修改，那么后修改的记录将会覆盖前面修改的记录，前面的修改就丢失掉了，这就叫做更新丢失。

Serializable可以防止更新丢失问题的发生。其他的三个隔离级别都有可能发生更新丢失问题。

Serializable虽然可以防止更新丢失，但是效率太低，通常数据库不会用这个隔离级别，所以我们需要其他的机制来防止更新丢失:

乐观锁和悲观锁不是数据库中真正存在的锁，只是人们在解决更新丢失时的不同的解决方案，体现的是人们看待事务的态度。

悲观锁：（ Pessimistic Locking ）

*select \* from table lock in share mode（读锁、共享锁）*

*select \* from table for update （写锁、排它锁）*

隔离级别不设置为Serializable，防止效率过低。

在查询时手动加上排他锁。

如果数据库中的数据查询比较多而更新比较少的话，悲观锁将会导致效率低下。

乐观锁：（ Optimistic Locking ）通过时间戳字段

在表中增加一个version字段，在更新数据库记录时将version加一，从而在修改数据时通过检查版本号是否改变判断出当前更新基于的查询是否已经是过时的版本。

如果数据库中数据的修改比较多，更新失败的次数会比较多，程序需要多次重复执行更新操作。

二、数据库连接池

1.连接池概念：

2.自己写一个连接池：

使自己的连接池在conn调用close方法是可以将连接直接返回池中:

修改一个类中某个方法的功能：

继承

装饰设计模式模式

动态代理

3.DBCP开源连接池——apache组织提供的开源连接池

两个jar包

——commons-dbcp.jar 连接池的实现

——commons-pool.jar 连接池实现的依赖库

数据库连接池既可以与应用服务器整合使用，也可由应用程序独立使用。

BasicDataSource dataSource = new BasicDataSource();

dataSource.setDriverClassName("com.mysql.jdbc.Driver");

dataSource.setUrl("jdbc:mysql:///day12");

dataSource.setUsername("root");

dataSource.setPassword("root");

String path = this.getClass().getClassLoader().getResource("dbcp.properties").getPath();

FileInputStream inStream = new FileInputStream(path);

Properties prop = new Properties();

prop.load(inStream);

DataSource dataSource = BasicDataSourceFactory.createDataSource(prop);

dbcp.properties

driverClassName=com.mysql.jdbc.Driver

url=jdbc:mysql:///day12

username=root

password=root

4.c3p0

ComboPooledDataSource dataSource = new ComboPooledDataSource();

dataSource.setDriverClass("com.mysql.jdbc.Driver");

dataSource.setJdbcUrl("jdbc:mysql:///day12");

dataSource.setUser("root");

dataSource.setPassword("root");

ComboPooledDataSource dataSource = new ComboPooledDataSource("mysource");

c3p0-config.xml

<?xml version="1.0" ?>

<c3p0-config>

<default-config>

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="jdbcUrl">jdbc:mysql:///day12</property>

<property name="user">root</property>

<property name="password">root</property>

</default-config> <!-- This app is massive! -->

<named-config name="mysource">

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="jdbcUrl">jdbc:mysql:///day12</property>

<property name="user">root</property>

<property name="password">root</property>

</named-config>

</c3p0-config>

5.配置tomcat数据源

tomcat中具体配置文件的写法可以在运行起来tomcat后，进tomcat管理界面——>Tomcat Documentation——>Configuration中查找详细的配置信息。

在Tomcat Documentation界面中还可以找到JNDI Resources，JDBC DataSources的配置用法。

tomcat数据源的思想是

1.在tomcat的context中配置（一般是放到web应用的context文件中）

<Context>

<Resource name="jdbc/EmployeeDB"

auth="Container"

type="javax.sql.DataSource"

username="dbusername"

password="dbpassword"

driverClassName="org.hsql.jdbcDriver"

url="jdbc:HypersonicSQL:database"

maxActive="8"

maxIdle="4"/>

</Context>

2.在tomcat的lib目录中添加数据库驱动包，这样，在数据库启动时候把context中的配置信息，配置到jndi中。

3.接下来就可以在应用中调用了

Context initCtx = new InitialContext();

Context envCtx = (Context) initCtx.lookup("java:comp/env");

dataSource = (DataSource)envCtx.lookup("jdbc/datasource");

由于jndi提供了InitialContext的入口类，利用入口类找到tomcat中最初始的名为"java:comp/env"的jndi，然后在调用lookup方法找到对应的jndi就是注册的数据源。

JNDI技术简介

JNDI(Java Naming and Directory Interface)，Java命名和目录接口，它对应于J2SE中的javax.naming包，

这套API的主要作用在于：它可以把Java对象放在一个容器中（支持JNDI容器 Tomcat），并为容器中的java对象取一个名称，以后程序想获得Java对象，只需通过名称检索即可。

其核心API为Context，它代表JNDI容器，其lookup方法为检索容器中对应名称的对象。

## 十三DBUTILS

一、JDBC元数据

二、DBUTILS

1.QueryRunner的API

QueryRunner() //由于创建时没有指定数据源，其下的所有操作都要明确的传入Connection，由于是自己传入的Connection对象，可以在传入的Connection对象上进行事务管理

update(Connection conn, String sql)

update(Connection conn, String sql, Object... params)

update(Connection conn, String sql, Object param)

query(Connection conn, String sql, ResultSetHandler<T> rsh)

query(Connection conn, String sql, ResultSetHandler<T> rsh, Object... params)

QueryRunner(DataSource ds) //由于创建是指定了数据源，其下的所有操作都不需要传入Connection，由于是由DBUtils帮住我们管理连接，所以所有的语句都在各自的事务当中，不能手动控制事务

update(String sql)

update(String sql, Object... params)

update(String sql, Object param)

query(String sql, ResultSetHandler<T> rsh)

query(String sql, ResultSetHandler<T> rsh, Object... params)

2.ResultSetHandler的实现类

//1.ArrayHandler 将查询结果的第一行转换为一个数组对象返回

Object[] objs = runner.query("select \* from account where name=?",new ArrayHandler() , "c");

System.out.println(objs);

//2.ArrayListHandler 将查询结果的第一行转换为一个数组对象返回

List<Object[]> list = runner.query("select \* from account",new ArrayListHandler() );

System.out.println(list);

//3.BeanHandler,将查询结果的第一行转换为一个JavaBean对象返回

Account acc = runner.query("select \* from account where name=?",new BeanHandler<Account>(Account.class) , "c");

System.out.println(acc);

//4.BeanListHandler：将结果集中的每一行数据都封装到一个对应的JavaBean实例中，存放到List里。

List<Account> acclist = runner.query("select \* from account",new BeanListHandler<Account>(Account.class) );

System.out.println(acclist);

//5.MapHandler：将结果集中的第一行数据封装到一个Map里，key是列名，value就是对应的值。

Map map = runner.query("select \* from account",new MapHandler() );

System.out.println(map);

//6.MapListHandler：将结果集中的每一行数据都封装到一个Map里，然后再存放到List

List<Map<String, Object>> maplist = runner.query("select \* from account",new MapListHandler() );

System.out.println(maplist);

//7.ColumnListHandler：将结果集中某一列的数据存放到List中。

List<Object> columnList = runner.query("select \* from account",new ColumnListHandler(2) );

System.out.println(columnList);

//8.KeyedHandler(name)：将结果集中的每一行数据都封装到一个Map里(List<Map>)，再把这些map再存到一个map里，其key为指定的列。

Map<Object, Map<String, Object>> keymap = runner.query("select \* from account",new KeyedHandler("id") );

System.out.println(keymap);

//9.ScalarHandler: 单值查询

//select count(\*) from account;

Long count = (Long)runner.query("select count(\*) from account",new ScalarHandler(1) );

System.out.println(count);

1. DBUtils工具类，提供如关闭连接、装载JDBC驱动程序等常规工作的工具类，里面的所有方法都是静态的。

## 十四、客户信息管理系统

1.创建数据库表

create database day14;

use day14;

create table customer(

id int primary key auto\_increment,

name varchar(20),

gender varchar(10),

birthday date,

cellphone varchar(20),

email varchar(40),

preference varchar(100),

type varchar(40),

description varchar(255)

);

2.创建工程，导入相关开发包

mysql数据驱动

c3p0连接池相关包、配置文件

dbutils相关开发包

BeanUtils开发包

JSTL开发包

3.根据软件分层的思想创建包

web

service

dao

util

domain

factory

4.开发一些工具类、工厂类

public class PubFactory {

private static PubFactory factory = new PubFactory();

private static Properties prop = null;

static{

try{

//通过配置文件来获得具体的要构建的工厂

String path = PubFactory.class.getClassLoader().getResource("config.properties").getPath();

FileInputStream in = new FileInputStream(path);

/\*\*\*竟然忘了，一定要记得\*\*\*\*\*竟然忘了，一定要记得\*\*\*\*\* \*\*\*\*/

prop = new Properties();

/\*\*\*\*竟然忘了，一定要记得\*\*\*\*\*\*\*竟然忘了，一定要记得\*\*\* \*\*\*\*/

prop.load(in);

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException();

}

}

private PubFactory(){

}

public static PubFactory newInstance(){

return factory;

}

public <T> T createFactory(String name,Class<T> T){

try {

// return (T) this.getClass().forName(name);

Class clazz = Class.forName(prop.getProperty(name));

return (T)clazz.newInstance();

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException();

}

}

}

5.开发

主页

增加用户、查找所有用户、删除用户、批量删除用户、修改用户、条件查 询用户、分页查询用户

(1)创建接口

(2)创建JavaBean

~增加用户

add.jsp AddUser addUser addUser

~查找所有用户

AllUser all.jsp

~删除单条用户

DelUser

~多条删除

DelBatchUser

~修改用户

FindUser update.jsp UpdateUser

~条件查询

FindUserByCondition

~分页查询

index.jsp pageUser page.jsp?page=1

~分页条

一共xx条记录 一共x页 首页 上一页 1 2 3 4 5 6 7 8 9 10 11 下一页 末页

一共多少条记录

一共有多少页

当前是多少页

分析：

if(如果总条数不足5条，有多少条显示多少条）

else(总条数大于5条)｛

if(如果当前的页码小于等于3，显示1到5)

else(如果当前页码大于3)｛

if(当前页码小于总页数-2)｛

显示当前页码-2到当前页码+2的范围

｝else{

显示后五个页面

}

｝

｝

## 十五、过滤器

Servlet技术规范 描述三种技术 ： Servlet(服务器小程序) 、Filter（过滤器） 、Listener（监听器）

Filter运行在服务器端，对服务器端web资源的访问 进行拦截，起到过滤的作用

Servlet API中 定义接口 Filter，用户只需要编写程序实现Filter接口，完成过滤器编写

Filter快速入门

1、编写类 实现 Filter接口

2、在服务器端注册 Filter （配置拦截哪个web资源） ----- web.xml

<!-- 注册过滤器 -->

<filter>

<filter-name>Filter1</filter-name>

<filter-class>cn.itcast.filter.Filter1</filter-class>

</filter>

<!-- 配置过滤器去拦截哪个资源 -->

<filter-mapping>

<filter-name>Filter1</filter-name>

<url-pattern>/hello.jsp</url-pattern>

</filter-mapping>

3、客户端访问被拦截目标资源之前，服务器调用Filter的doFilter方法 ，执行过滤

4、Filter的doFilter方法中传入 FilterChain, 如果调用FilterChain的doFilter 就会执行目标资源，否则目标资源不会执行

chain.doFilter(request, response);

FilterChain

在客户端访问服务器web资源时，服务器端为一个web资源，配置多个过滤器拦截 ，这多个过滤器，就会组成过滤器链 FilterChain, 调用FilterChain的doFilter 表示要执行过滤器链下一个资源，如果当前过滤器已经是链上最后一个过滤器，就会执行目标资源

\* web服务器根据Filter在web.xml文件中的注册顺序<mapping>，决定先调用哪个Filter

Filter生命周期 init(FilterConfig) doFilter(request,response,filterChain) destroy()

1、Filter对象在tomcat服务器启动时 创建，调用init方法 （只会创建一个对象，init方法执行一次）

2、doFilter 每次拦截目标资源时，执行

3、destroy 服务器关闭时执行

FilterConfig 作用和 ServletConfig 类似，用来在Filter初始化阶段，将参数传递给过滤器

1、通过 String getInitParameter(String name) 获得过滤器初始化参数

2、通过 ServletContext getServletContext() 获得ServletContext对象

\* FilterConfig 提供参数，是Filter类私有参数，Filter2的初始化参数，不能在Filter1 中进行获取

\* 配置全局参数，<context-param> 进行配置，通过ServletContext 获得

<filter-mapping> 过滤器拦截配置

1、如果连接目标资源是一个Servlet，可以选择url和servlet名称两种配置方式

<!-- 拦截/hello是Servlet 路径 -->

<url-pattern>/hello</url-pattern>

<!-- 拦截Servlet 还可以通过Servlet 名称进行拦截 -->

<servlet-name>HelloServlet</servlet-name>

2、url-pattern 和 Servlet中路径写法一样，有三种 ： 完全匹配、目录匹配、扩展名匹配

3、<dispatcher>指定过滤器所拦截的资源被 Servlet 容器调用的方式

容器调用服务器端资源 有四种方式

REQUEST、FORWARD、INCLUDE、ERROR

REQUEST:拦截对资源的请求

FORWARD:拦截转发对资源的请求

INCLUDE：拦截包含对资源的请求

ERROR：拦截错误处理对资源的请求

=========================================================================================================================================

Filter应用

应用一：统一全站字符编码过滤器

案例：编写jsp 输入用户名，在Servlet中获取用户名，将用户名输出到浏览器上

处理请求post乱码代码

request.setCharacterEncoding("utf-8");

设置响应编码集代码

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

经常会使用，而过滤器可以在目标资源之前执行，将很多程序中处理乱码公共代码，提取到过滤器中 ，以后程序中不需要处理编码问题了

应用二：禁止浏览器缓存动态页面的过滤器

因为动态页面数据，是由程序生成的，所以如果有缓存，就会发生，客户端查看数据不是最新数据情况 ，对于动态程序生成页面，设置浏览器端禁止缓存页面内容

response.setDateHeader("Expires",-1);

response.setHeader("Cache-Control","no-cache");

response.setHeader("Pragma","no-cache");

将禁用缓存代码，提起到过滤器中，通过url配置，禁用所有JSP页面的缓存

应用三：控制浏览器缓存静态web资源

Tomcat缓存策略

对于服务器端经常不变化文件，设置客户端缓存时间，在客户端资源缓存时间到期之前，就不会去访问服务器获取该资源 -------- 比tomcat内置缓存策略更优手段

\* 减少服务器请求次数，提升性能

设置静态资源缓存时间，需要设置 Expires 过期时间 ，在客户端资源没有过期之前，不会产生对该资源的请求的

\* 设置Expires 通常使用 response.setDateHeader 进行设置 设置毫秒值

===============================================================================================================================================

应用四：自动登陆过滤器

在访问一个站点，登陆时勾选自动登陆（三个月内不用登陆），操作系统后，关闭浏览器；过几天再次访问该站点时，直接进行登陆后状态

在数据库中创建 user表

create table user (

id int primary key auto\_increment,

username varchar(20),

password varchar(40),

role varchar(10)

);

insert into user values(null,'admin','123','admin');

insert into user values(null,'aaa','123','user');

insert into user values(null,'bbb','123','user');

自动登陆 ：未登录、存在自动登陆信息、自动登陆信息正确

在用户完成登陆后，勾选自动登陆复选框，服务器端将用户名和密码 以Cookie形式，保存在客户端 。当用户下次访问该站点，AutoLoginFilter 过滤器从Cookie中获取 自动登陆信息

1、判断用户是否已经登陆，如果已经登陆，没有自动登陆的必要

2、判断Cookie中是否含有自动登陆信息 ，如果没有，无法完成自动登陆

3、使用cookie用户名和密码 完成自动登陆

如果将用户密码保存在cookie文件中，非常不安全的 ，通常情况下密码需要加密后才能保存到客户端

\* 使用md5算法对密码进行加密

\* md5 加密算法是一个单向加密算法 ，支持明文---密文 不支持密文解密

MySQL数据库中提供md5 函数，可以完成md5 加密

mysql> select md5('123');

+----------------------------------+

| md5('123') |

+----------------------------------+

| 202cb962ac59075b964b07152d234b70 |

+----------------------------------+

解密后结果是32位数字 16进制表示

Java中提供类 MessageDigest 完成MD5加密

------------------------------------------------------------------

将数据表中所有密码 变为密文 update user set password = md5(password) ;

在Demo4Servlet 登陆逻辑中，对密码进行md5 加密

在AutoLoginFilter 因为从Cookie中获得就是加密后密码，所以登陆时无需再次加密

------------------------------------------------------------------

MD5 在2004 年被王小云破解，md5算法是多对一加密算法，出现两个加密后相同密文的明文很难发现 ，王小云并没有研究出md5 解密算法，研究出一种提高碰撞概率的算法

-------------------------------------------------------------------

应用五：过滤器实现URL级别权限认证

系统中存在很多资源，将需要进行权限控制的资源，放入特殊路径中，编写过滤器管理访问特殊路径的请求，如果没有相应身份和权限，控制无法访问

认证：who are you ? 用户身份的识别 ------------ 登陆功能

权限：以认证为基础 what can you do ? 您能做什么？ 必须先登陆，才有身份，有了身份，才能确定可以执行哪些操作

=====================================================================================================================================================

Filter高级应用

Decorator模式

1、包装类需要和被包装对象 实现相同接口，或者继承相同父类

2、包装类需要持有 被包装对象的引用

在包装类中定义成员变量，通过包装类构造方法，传入被包装对象

3、在包装类中，可以控制原来那些方法需要加强

不需要加强 ，调用被包装对象的方法

需要加强，编写增强代码逻辑

ServletRequestWrapper 和 HttpServletRequestWrapper 提供对request对象进行包装的方法，但是默认情况下每个方法都是调用原来request对象的方法，也就是说包装类并没有对request进行增强

在这两个包装类基础上，继承HttpServletRequestWrapper ，覆盖需要增强的方法即可

应用六：完全解决get和post乱码的过滤器

在Filter中，对request对象进行包装，增强获得参数的方法

getParameter

getParameterValues

getParameterMap

ServletResponseWrapper 和 HttpServletResponseWrapper 提供了对response 对象包装，继承 HttpServletResponseWrapper ，覆盖需要增强response的方法

应用七：增强Response对象，对响应数据进行压缩

复习：Tomcat服务器内，提供对响应压缩 配置实现

在conf/server.xml 中

<Connector port="80" protocol="HTTP/1.1" connectionTimeout="20000"

redirectPort="8443"/> 添加 compressableMimeType 和 compression

没有压缩 ： 00:00:00.000 0.063 7553 GET 200 text/html http://localhost/

<Connector port="80" protocol="HTTP/1.1" connectionTimeout="20000"

redirectPort="8443" compressableMimeType="text/html,text/xml,text/plain" compression="on"/>

压缩后 ： 00:00:00.000 0.171 2715 GET 200 text/html http://localhost/

Content-Encoding: gzip

Content-Length : 2715

实际开发中，很多情况下，没有权限配置server.xml ,无法通过tomcat配置开启gzip 压缩

编写过滤器对响应数据进行gzip压缩

flush 方法

只有没有缓冲区字节流，FileOutputStream 不需要flush

而字节数组ByteArrayOutputStream、字节包装流、字符流 需要flush ----- 这些流在调用close方法时都会自动flush

==================================================================================================================

过滤器

1、过滤器编写步骤

2、全局编码、禁用缓存、设置过期时间 过滤器

3、自动登陆/URL级别权限控制 --- 必须掌握 \*\*\*\*\*

4、通用get/post乱码过滤器和响应压缩过滤器 ------------ 理解实现过程 ，保存起来会使用

客户信息系统增删改查必备

掌握分页查询原理

## 十六、监听器

监听器：监听器就是一个java程序，功能是监听另一个java对象变化（方法调用、属性变更）

监听器监听过程：事件源、事件对象、监听器对象 、操作事件源

1、存在被监听对象（事件源）

2、存在监听器对象

3、在事件源中注册监听器

4、操作事件源，使事件源发生改变 ，产生事件对象

\* 事件对象 就是 事件源的改变

5、事件对象会被传递给监听器，触发监听器相应行为

监听器技术主要应用于图形界面编程 ---------- swing中监听器的使用

自定义监听器 ---- 监听人的行为（监听器案例）

1、创建事件源对象 Person

2、创建监听器对象 （通常定义为接口） PersonListener

3、注册监听器

4、触发事件源改变 --- 产生事件对象

思考：事件对象能用来做什么 ？？

事件对象传递事件源的数据给监听器 ，通过事件对象获得事件源对象

\* 监听器可以监听到事件源的数据变化

============================================================================================

今天的学习目标：Servlet提供8个监听器对象

Servlet提供8个监听器，监听事件源主要是三个对象 ： ServletRequest, HttpSession, ServletContext ------Servlet的三种数据范围对象

Servlet的8个监听器，分为三类：

第一类：监听三个数据范围对象（request、session、ServletContext）的创建和销毁监听器

第二类：监听三个数据范围对象中属性变更(增加、替换、删除)的监听器

第三类：监听HttpSession中对象状态改变（被绑定、解除绑定、钝化、活化）的监听器

一、三个域对象创建和销毁监听器

1、ServletContextListener 监听 ServletContext 对象的创建和销毁事件

void contextInitialized(ServletContextEvent sce) ---- 监听Context对象创建

void contextDestroyed(ServletContextEvent sce) ----- 监听Context对象的销毁

ServletContext 全局唯一对象，每个工程创建唯一Context对象（配置全局初始化参数、保存全局共享数据、读取web资源文件）

在服务器启动时创建ServletContext对象，在服务器关闭时销毁ServletContext对象

编写监听器步骤

1) 编写类 ，实现特定监听器接口

2) Servlet监听器，不是注册在事件源上，而是注册在web.xml 中，由容器tomcat完成监听器注册

<!-- 注册监听器 , tomcat将监听器注册给事件源， 事件源操作后，会自动监听器执行 -->

<!-- 和Servlet Filter不同，不需要配置url -->

<listener>

<listener-class>cn.itcast.web.listener.MyServletContextListener</listener-class>

</listener>

应用：

1) 保存全局范围对象，因为监听ServletContext对象，监听器都可以通过事件对象获得事件源

// 获得被监听事件源对象

ServletContext context = sce.getServletContext();

2) 读取框架配置文件 例如: spring框架 org.springframework.web.context.ContextLoaderListener

3) 在ServletContextListener 定义一些定时器程序 （任务调度程序）

最简单java中任务调度 ----- 定时器Timer 和 TimerTask的使用

Timer启动定时器任务

void schedule(TimerTask task, Date firstTime, long period) ------- 指定启动任务第一次时间，通过period参数指定任务重复执行

void schedule(TimerTask task, long delay, long period) ----- 指定任务距离当前时间delay 多久开始启动，通过period指定任务重复执行

终止定时器任务执行 timer.cancel();

2、HttpSessionListener 监听HttpSession对象的创建和销毁

void sessionCreated(HttpSessionEvent se) ----- 监听Session对象创建

void sessionDestroyed(HttpSessionEvent se) ---- 监听Session对象销毁

Session对象何时创建 ： request.getSession(); 第一次执行时 创建Session对象

\* 访问JSP时，因为其内置对象session，所以一定会创建Session对象的

Session对象何时销毁 ： 1) 不正常关闭服务器 2) Session对象过期 3) invalidate方法调用

\* 正常关闭服务器时，Session的数据保存tomcat/work目录下 --- 产生 SESSIONS.ser

\* session的过期时间在web.xml 进行配置

<session-config>

<!-- 单位是分钟，连续30分钟没有使用该Session对象，就会销毁对象 -->

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

</session-config>

3、ServletRequestListener 监听Request对象的创建和销毁

void requestDestroyed(ServletRequestEvent sre) ----- 监听request对象销毁的

void requestInitialized(ServletRequestEvent sre) ---- 监听request对象创建的

每次客户端发起一次新的请求 产生request对象，当response响应结束后，request对象进行销毁

forward不会产生新的request对象，sendRedirect产生新的request对象

监听器案例：

案例一： 统计当前在线人数

分析：统计Session 的个数 ，存在一个Session，意味着一个浏览器在访问

案例二： 自定义session定时扫描器（销毁session对象）

编写定时器程序，定时去扫描系统中所有Session对象，发现如果一个Session 1分钟没有使用了，就销毁该Session对象

=============================================================================================================================

二、ServletRequest、HttpSession、ServletContext 三个数据范围中 数据变更监听器

ServletContextAttributeListener, HttpSessionAttributeListener ServletRequestAttributeListener

这三个接口中都定义了三个方法来处理被监听对象中的属性的增加，删除和替换的事件

1、attributeAdded 属性添加方法

public void attributeAdded(ServletContextAttributeEvent scae)

public void attributeAdded (HttpSessionBindingEvent hsbe)

public void attributeAdded(ServletRequestAttributeEvent srae)

2、attributeRemoved 属性移除方法

public void attributeRemoved(ServletContextAttributeEvent scae)

public void attributeRemoved (HttpSessionBindingEvent hsbe)

public void attributeRemoved (ServletRequestAttributeEvent srae)

3、attributeReplaced 属性替换方法

public void attributeReplaced(ServletContextAttributeEvent scae)

public void attributeReplaced (HttpSessionBindingEvent hsbe)

public void attributeReplaced (ServletRequestAttributeEvent srae)

何时调用attributeAdded 、attributeRemoved 、attributeReplaced ？

以HttpSessionAttributeListener为例

// 向session中添加了一个属性

session.setAttribute("name","张三"); // 因为name属性还不存在 --- attributeAdded

// 向Session保存属性name的值李四，因为name属性已经存在，替换效果

session.setAttribute("name","李四"); // 因为name属性以及存在 --- attributeReplaced

// 移除session的属性name

session.removeAttribute("name"); // 移除name属性 ---- attributeRemoved

\*\* 如果执行session.invalidate(); 销毁了Session 导致Session中所有属性被移除

===========================================================================================================================

三、Session中对象状态改变 监听器

Session中对象共有四种状态

1、绑定 ----- 对象被添加到Session中

2、解除绑定 ----- 对象从Session中移除

3、钝化 ----- Session中数据被序列化到硬盘上

4、活化 ----- Session序列化数据 从硬盘被加载回内存

使JavaBean 了解到自己在Session中状态的变化

HttpSessionBindingListener 感知到绑定以及解除绑定

HttpSessionActivationListener 感知钝化和活化 状态改变

不需要 web.xml 文件中进行注册 （这两类监听器，不是由容器进行管理的，由HttpSession对象管理 ）

HttpSessionBindingListener

void valueBound(HttpSessionBindingEvent event) 绑定

void valueUnbound(HttpSessionBindingEvent event) 解除绑定

编写javabean实现 HttpSessionBindingListener接口，该javabean对象感知到自己被绑定到Session或者从Session解除绑定

valueBound ， 当对象被加入session 就会执行

valueUnbound， 当对象从Session 移除时 就会执行

HttpSessionActivationListener 监听对象被钝化和活化

void sessionDidActivate(HttpSessionEvent se) 活化

void sessionWillPassivate(HttpSessionEvent se) 钝化 （java对象被序列化到硬盘上 ）

如果tomcat正常关闭，Session中对象会被序列化到硬盘上 ---- java对象如果能被序列化必须实现Serializable接口

异常 ：IOException while loading persisted sessions: java.io.WriteAbortedException: writing aborted; java.io.NotSerializableException: cn.itcast.bean.Bean2

原因 ：恢复Session.ser文件时，因为需要对象无法从文件中加载，发生异常 （删除Session.ser就可以了 ）

\* 保存到Session中对象，应该被序列化

在Session中数据对象，保存了一段时间后没有使用，不想删除对象中数据，（在不关闭服务器情况下）可以让对象数据进行钝化 ---- 序列化到硬盘上

下次再访问数据时，从钝化的文件中读取序列化数据 ----- 对数据进行活化

\* 由tomcat完成钝化和活化，配置<Context> 标签

配置Context标签有三个常用位置

1) tomcat/conf/context.xml 所有虚拟主机、所有web应用都可以使用配置

2) tomcat/conf/Catalina/localhost/context.xml 当前虚拟主机 所有web应用都可以使用配置

3) 当前工程/META-INF/context.xml 只对当前工程有效

配置钝化目录在tomcat/work 下 ---- 和Session默认保存位置一样

=================================================================================================================================================

综合练习：实现在线用户列表 和 踢人功能

在线用户列表： 显示有哪些人在线

踢人功能 ： 管理员将进行非法操作用户踢下线

创建day19kick web工程，创建day19数据库

create table users(

id int primary key auto\_increment,

username varchar(20),

password varchar(40),

nickname varchar(20),

role varchar(10)

);

insert into users values(null,'aaa','111','小丽','user');

insert into users values(null,'bbb','111','老张','user');

insert into users values(null,'ccc','111','小王','user');

insert into users values(null,'ddd','111','老李','user');

insert into users values(null,'admin','111','超级管理员','admin');

如果 Map的key是 自定义对象，对于HashMap 重写equals和hashcode方法 ---- 排重

踢人的必须是管理员，被踢的不能是管理员

## 十七、文件上传和下载

一、文件上传

1.文件上传的两个步骤

（1）在web页面中添加上传输入项

<1>上传输入项应该具有一个名字

<2>form的enctype属值设为multipart/form-data

如果不设置这个属性，这个属性的默认值为enctype="application/x-www-form-urlencoded"，这种提交方式下，文件并不会真的上传，只会将文件的路径作为参数值带走。

info=xxxx&upload=D%3A%5CDocuments+and+Settings%5Cpark%5C%E6%A1%8C%E9%9D%A2%5C1.txt对应的文件路径D:\Documents and Settings\park\桌面\1.txt

<3>表单的提交方式要是post:

get方式在提交数据的时候是将数据防止到url后，get提交方式传输参数有大小的限制，最大只能传输1kb的数据。

post方式将参数放置到实体内容中，没有大小限制。

在进行文件上传时，只能使用post方式。

POST /Day17/servlet/Update HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, \*/\*

Referer: http://localhost/Day17/upload.jsp

Accept-Language: zh-cn

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

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2)

Host: localhost

Content-Length: 82

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: JSESSIONID=C00FBE69708623E527DDA8A0F77CE335

info=xxxx&upload=D%3A%5CDocuments+and+Settings%5Cpark%5C%E6%A1%8C%E9%9D%A2%5C1.txt

~~~~~~~~~~~~~~~~~

POST /Day17/servlet/Update HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, \*/\*

Referer: http://localhost/Day17/upload.jsp

Accept-Language: zh-cn

Content-Type: multipart/form-data; boundary=---------------------------7dd242107099e

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2)

Host: localhost

Content-Length: 534

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: JSESSIONID=C00FBE69708623E527DDA8A0F77CE335

-----------------------------7dd242107099e

Content-Disposition: form-data; name="info"

xxxx

-----------------------------7dd242107099e

Content-Disposition: form-data; name="upload"; filename="D:\Documents and Settings\park\妗岄潰\1.txt"

Content-Type: text/plain

在web应用加载时利用servletcontextListener初始化Map到servletcontext域

当增加用户时在在servletcontext中的map<user,session>保存

当用户session失效时 移除

kickservlet，获取要踢下线的id，从map中获取session，销毁。

-----------------------------7dd242107099e--

（2）在servlet中读取上传文件的数据，并保存到服务器硬盘中

Commons-fileupload ---Apache组织提供文件上传的开发包

JAVAEE6开始，sun公司默认提供了上传组件。

二、Common-fileupload

（1）DiskFileItemFactory

public DiskFileItemFactory(in sizeThreshold, java.io.File repository)

构造函数

public void setSizeThreshold(int sizeThreshold)

设置内存缓冲区的大小，默认值为10K。当进行文件上传时，文件的内容默认会缓存到内存当中，此方法用来设定这个缓存区的大小。

public void setRepository(java.io.File repository)

指定临时文件目录，如果文件的大小超过了内存中的缓冲区的大小，将被缓存到缓存文件夹中。，默认值为System.getProperty("java.io.tmpdir").

(2)ServletFileupLoad

boolean isMultipartContent(HttpServletRequest?request)

判断上传表单是否为multipart/form-data类型

List parseRequest(HttpServletRequest?request)

解析request对象，并把表单中的每一个输入项包装成一个fileItem 对象，并返回一个保存了所有FileItem的list集合。

setFileSizeMax(long?fileSizeMax)

设置单个上传文件的最大值

setSizeMax(long?sizeMax)

设置上传文件总量的最大值

setHeaderEncoding(java.lang.String?encoding)

设置编码格式，解决上传文件名乱码问题

setProgressListener(ProgressListener?pListener)

实时监听文件上传状态

(3)FileItem

request.getParameter获取请参数只能获取"application/x-www-form-urlencoded方式提价的参数，在文件上传时，用的是multipart/form-data方式，所以此处那个是不能使用。

同时request.setCharacterEncoding 也不能解决请求参数中的乱码问题了。

如果FileItem中封装的是一个普通的请求参数信息：getFieldName获取参数名称 getString获取参数的值

如果FileItem中封装的是文件内容：getInputStream获取流 getName获取文件的名称

boolean isFormField() 判断FileItem是一个文件上传对象还是普通表单对象，如果是返回true，通常在遍历parseRequest获得的List时进行类型判断

\*如果是一个普通的表单对象需要用如下方法获得表单对象的信息：

String getFieldName() 获得普通表单对象的name属性

String getString();

String getString(String encoding) 获得普通表单对象的value属性，可以用encoding进行编码设置

\*如果是一个文件上传可以调用如下的api：

String getName() 获得上传文件的文件名（有些浏览器会携带客户端路径）

InputStream getInputStream() 获得上传文件的输入流

delete() 在关闭FileItem输入流后，删除临时文件

三、细节

（1）大部分的浏览器在上传数据时传输的文件是当前文件的名称，而ie浏览器上传时传输的文件名是该文件带硬盘路径的全路径名，导致在服务器端拼接路径时报出异常：

java.io.FileNotFoundException: E:\workspace\tomcat6\webapps\Day17\update\D:\Documents and Settings\park\桌面\music\海阔天空-信乐团.mp3 (文件名、目录名或卷标语法不正确。

（2）如果上传同名的文件，后上传的文件将会覆盖之前的文件，这会导致之前的文件丢失。

利用UUID改造文件名

（3）上传时不宜将所有的上传文件都保存在一个文件夹中，应该设计算法，将文件分目录保存。

如果一个用户上传很多个文件，有多个用户：一个用户一个文件

如果有许多用户但是每个用户上传的文件并不多，但是每天都有很多人在上传：按时间保存，一天一个文件夹

当一个文件夹存够300个文件时在创建另一个文件夹

（4）如果上传的文件不希望其他人能够直接访问，可以将上传目录指定为WEB-INF目录或其子目录，或不受WEB服务器管理的目录。

四、上传监控: request发送服务器，在获取上传文件时拿到是一个到上传的流，慢慢的传输，服务根据request中的头信息Content-Length获知上传的文件有多大。

在ServletFileUpload可以设置监听器ProgressListener，监听上传时的数据变化

上传时的总字节数contentLength

当前上传的字节数bytesRead

当前上传的进度：bytesRead/contentLength

上传的速度：" 上传速度为(KB/s)"+bytesRead/(time2-time)

剩余时间：(contentLength-bytesRead)/(bytesRead/(time2-time))

===============================

一、文件下载

1.使用超链接下载

2.使用程序下载

下载时要设置的头：

(1)Content-Type

response.addHeader("Content-Type","image/jpeg");

response.setContentType(this.getServletContext().getMimeType("美女.jpg"));

(2) Content-Disposition,中文文件名要进行URL编码

attachment;filename="+URLEncoder.encode("美女.jpg","utf-8");

(3) ie6浏览器在URL编码时特殊字符转换有问题，推荐不要在超链接中将中文直接发给浏览器，而是先进行URL编码，将编码好的内容发给浏览器，从而能防止ie6自己进行URL编码时的错漏。