**Java web**

Servlet tomcat jsp

Javaee:13种技术规范

动态网页开发 jsp php asp.net

常见的web服务器：

Tomcat：Apache组织，免费、开源，支持Servlet、jsp规范，轻量级服务器

Weblogic：Bean公司，收费的，重量级，支持Javaee的全部规范，重量级的服务器

Websphere：IBM公司，收费的，重量级，支持Javaee的全部规范，重量级的服务器

Jboss: 收费的，重量级，支持Javaee的全部规范，重量级的服务器

**Tomcat服务器的安装**

1. 配置JAVA\_HOME：D:\jdk1.7
2. CATALINA\_HOME：指定你的tomcat的根目录

**Tomcat服务器的目录**

**Web应用的目录**

**URL地址：统一资源定位**

http://localhost:8080/myweb/myfirstServlet

http://==〉表示使用的协议

localhost==〉解析主机地址，默认会先到

C:\Windows\System32\drivers\etc\hosts.txt去找有没有对应的ip地址，如果没有才到运营服务商的DNS服务器上去找

Myweb：web应用的名称

/myfirstServlet：资源的名称

**虚拟目录映射**

1、在Server.xml文件中配置如下，但需要重启服务器：不推荐使用

<Context path="/myHello" docBase="E:\myweb" />

1. conf/catalina/localhost/ 建一个xml文件

文件名就是对外访问路径

<?xml version='1.0' encoding='utf-8'?>

<Context docBase="E:\myweb" />

**虚拟主机映射**

<Host name="www.sina.com" appBase="e:\sina"

unpackWARs="true" autoDeploy="true"

xmlValidation="false" xmlNamespaceAware="false" />

**默认的web应用和默认的端口**

conf/catalina/localhost/ 建一个ROOT.xml文件,里面内容如下：

<Context docBase="E:\myweb" />

**HTTP协议 1.0 1.1**

**请求的方式**

GET ：默认的方式 ，响应很快，不安全，对发送的数据量有限制的

POST：

<form action=”” method=”post”>

Accept: text/html,image/\*

Accept-Charset: ISO-8859-1

Accept-Encoding: gzip,compress

Accept-Language: en-us,zh-cn

Host: www.xinhua.com:80

If-Modified-Since: Tue, 11 Jul 2016 18:23:51 GMT

Referer: http://www.xinhua.com/index.jsp

User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)

Cookie

Connection: close/Keep-Alive

Date: Tue, 11 Jul 2016 18:23:51 GMT

**请求消息行**

GET /web01/1.html HTTP1.1

URI：/web01/1.html 统一资源标识符 可以表示互联网和本地的资源

URL：<http://localhost:8080/web01/1.html> 统一资源定位符

只可以表示互联网的资源

**请求消息头**

Accept:告知服务器，客户端可以接受的数据类型（MIME类型）

文件系统：通过文件的扩展名区分不同的文件的。

MIME类型：大类型/小类型。

txt

对应关系：Tomcat\conf\web.xml)

Accept-Encoding：告知服务器，客户端可以接受的压缩编码。比如gzip

Accept-Language：告知服务器，客户端支持的语言。

Referer：告知服务器，从哪个页面过来的。

防盗链、统计广告效果

Content-Type：告知服务器，请求正文的MIME类型；如果你是get请求，没有请求正文，必须是POST才会带这个类型过去

默认类型：application/x-www-form-urlencoded(表单enctype属性的默认取值)

具体体现：username=abc&password=123

其他类型：multipart/form-data(文件上传时用的)

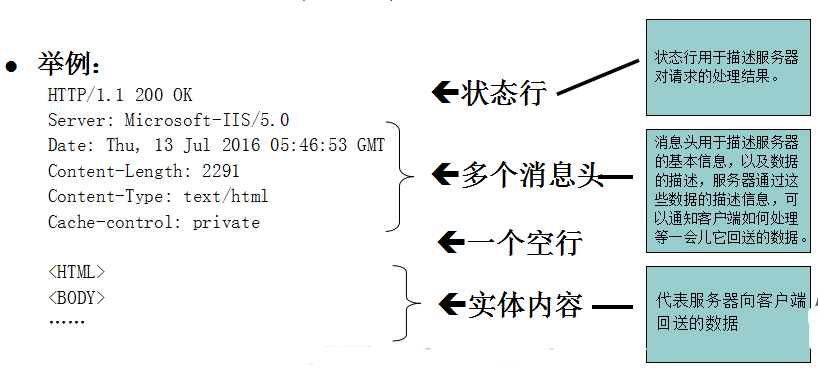
If-Modified-Since：告知服务器，当前访问的资源，缓存中的文件的最后修改时间。

User-Agent:告知服务器，浏览器的类型

Content-Length：请求正文的数据长度

数据能提交到服务器，表单中的输入域必须有name属性值

**HTTP响应**



如果只有普通的html代码，则只会发出一次请求，但是如果里面含有img、css、js，则浏览器看到这些元素的时候，则会自动发出一次或多次请求

HTTP请求中的常用响应头

Location: http://www.xinhua.com/index.jsp

Server:apache tomcat

Content-Encoding: gzip

Content-Length: 80

Content-Language: zh-cn

Content-Type: text/html; charset=GB2312

Last-Modified: Tue, 11 Jul 2016 18:23:51 GMT

Refresh: 1;url=http://www.xinhua.com

Content-Disposition: attachment; filename=aaa.zip

Transfer-Encoding: chunked

Set-Cookie:SS=Q0=5Lb\_nQ; path=/search

Expires: -1

Cache-Control: no-cache

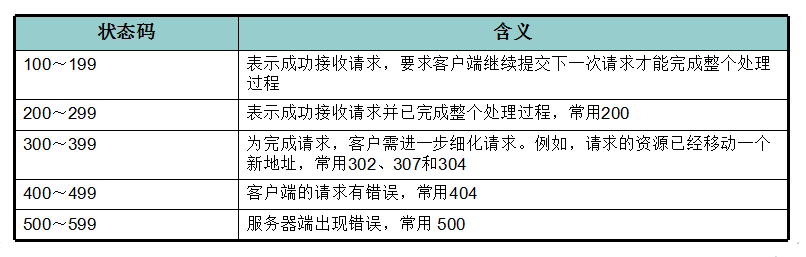
Pragma: no-cache

Connection: close/Keep-Alive

Date: Tue, 11 Jul 2016 18:23:51 GMT

**响应消息行**

响应协议版本 响应码 响应结果描述



**响应消息头**

Location：告知客户端，你去访问的地址。

和302/307实现请求重定向

Content-Encoding：告知客户端，响应正文使用的压缩编码（gzip）

Content-Length:告知客户端，响应正文的长度

Content-Type：告知客户端，响应文正的MIME类型。默认text/html

Refresh:告知客户端，定时刷新

Content-Disposition：告知客户端，用下载的方式打开

attachment;filename=123.jpg

Expires: -1 控制时间的

Cache-Control: no-cache

Pragma: no-cache

三头一块用，用于告知浏览器，不要缓存。

**Servlet概述**

A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

**Servlet的编码步骤**

init

 Called by the servlet container to indicate to a servlet that the servlet is being placed into service.

Called by the servlet container to allow the servlet to respond to a request.

**Servlet的执行过程**

**Servlet的编写方式：**

1、一个Servlet可以被映射到多个地址上

2、可以使用地址通配符\*

写法一：\*.do结尾 。必须以\*开头 比如\*.do

写法二(优先级略高)：以/开头，必须以\*结尾

3、默认的Servlet

默认的Servlet的映射路径是<url-pattern>/</url-pattern>。默认的Servlet负责处理用户的请求URL找不到匹配的处理工作。

4、应用在启动时就完成Servlet的实例化和初始化

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

**Servlet的生命周期**

诞生：用户第一次访问时，由容器创建他的实例。

活着：一旦创建就驻留内存（单例）。每次用户的访问，容器都会调用sevice方法发出响应（多线程）

死亡：应用被卸载或者Tomcat关闭了

**Servlet的线程安全**

**ServletConfig**

**ServletContext**

代表当前的web应用,是一个域对象，容器对象，是所有servlet共享的，而且这个域对象是最大的

response**中文乱码处理**

//控制response以utf-8将数据些出去

response.setCharacterEncoding("utf-8");

//告诉浏览器以什么码表打开

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

结论：以后只需要设置response.setContentType即可

注意：

1. 字符流和字节流不能同时使用
2. 使用完相关的输出流，是不需要关闭的，会自动关闭

Request中文乱码处理

**客户端技术：Cookie**

目的：共享多次请求中产生的数据

HTTP协议的消息头

请求消息头：Cookie 客户端向服务器端传递信息

响应消息头：Set-Cookie 服务器端向客户端传递信息

属性：

name：Cookie的名称，必要属性

value：Cookie的取值（不能为中文），必要属性

可选属性

path：Cookie的路径

默认值就是写cookie的那个资源的访问路径

如：[http://localhost:8080/cookie/servlet/Cookie1](http://localhost:8080/day09_00_cookie/servlet/CookieDemo1) path就是/cookie/servlet/

注意：

如果一个cookie的路径是/cookie/servlet/

当访问http://localhost:8080/cookie/Cookie1时，浏览器不会带Cookie给服务器。浏览器比对的是cookie的路径和当前访问的资源的路径。

浏览器满足一下条件就会带cookie给服务器：

当前访问的地址的路径.startWith(已存cookie的路径)。

即：如果一个Cookie的路径设置为了当前应用，说明访问该网站的任何资源时浏览器都带该cookie给服务器。

maxAge：Cookie的最大生存时间。默认是在浏览器的内存中。

domain：Cookie的域名。默认就是写cookie的那个资源所属的网站。

http://localhost:8080/cookie/servlet/Cookie1域名就是localhost

version：版本号

comment:注释

把Cookie写给客户端：

HttpServletResponse.addCookie(Cookie c):实际上就是向客户端发送了一个响应消息头。

注意：客户端只能保存一个网站最多20个Cookie数据，总共最多300个。每个Cookie的大小不能超过4kb。

服务器端获取客户端带来的Cookie：

Cookie [] HttpServletRequest.getCookies();

注意：domain+path+name唯一确定一个Cookie。

1、显示用户最近的访问时间

2、记住登录的用户名

**JSP**

也是SUN公司推出的开发动态web资源的技术，属于JavaEE技术之一。本质上是Servlet。

Jsp=Html+java

**JSP的原理**

**JSP的语法**

模板元素:就是jsp的html部分

Java程序片段

语法：<%java程序%>

Java表达式

语法：<%=表达式%>

JSP声明

语法：<%! %>

作用：定义JSP对应Serlvet的成员（变量和方法、静态方法）

JSP注释

语法：<%--这是注释--%>

注释掉的代码，JSP引擎不会翻译到Servlet源码中，要注意和HTML注释的区别

**JSP指令：**

指令是为JSP引擎设计的。而只是告诉引擎如何处理JSP页面中的其余部分（非指令部分）；

基本语法格式：<%@指令 属性1=”值1” ....%>.

一般把指令放在JSP页面的顶部（不是必须的）

指令的属性可以写在一个指令的中，也可以分开写：比如：

<%@指令 属性1=”值1” 属性2=”值2”%>.

等价于：

<%@指令 属性1=”值1” %>.

<%@指令 属性2=”值2” %>.

**page：**

属性：

language：告知引擎，脚本用的是java。默认java，支持java。不写也行。

extends：告知引擎，JSP对应的Servlet的父类是哪个。不需要写，也不要改。

import：告知引擎，导入哪些包。

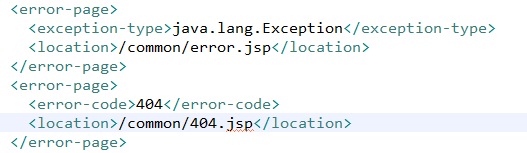
注意：引擎自动导入：java.lang.\*;javax.servlet.\*;javax.servlet.http.\*;javax.servlet.jsp.\*;

session:告知引擎是否产生HttpSession对象，即是否在代码中调用request.getSession()。默认值true。

buffer：JspWriter用于输出JSP的内容到页面上。告知引擎，设定他的缓存大小。默认8Kb。

errorPage：告知引擎，当前页面出现了异常，应该转发到哪个页面上（路径写法：/代表着当前应用）

配置全局错误页面：web.xml



isErrorPage:告知引擎，是否抓住异常。如果该属性为true，页面中就可以使用exception隐含对象，打印异常的详细信息。<%=exception.getMessage() %>

contentType：告知引擎，响应正文的MIME类型。contentType=”text/html;charset=UTF-8”

相当于response.setContentType(“”);

pageEncoding:告知引擎，翻译JSP时（要从磁盘上读JSP文件）所用的码表。

pageEncoding=”UTF-8”:相当于，告知了引擎用UTF-8读jsp，还response.setContentType(“text/html;charset=UTF-8”);

isELIgnored:告知引擎，是否忽略EL表达式。默认值是false，不忽略。

**include：**

包含属性：file，以”/”开头，就代表着当前应用。

**taglib:**

引入外部的标签。

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

uri：外部的标签的uri地址，好比名称空间

prefix:前缀。

**JSP中的转发和包含**

转发：

<jsp:forward page="/02.jsp"></jsp:forward>

2、参数：不能单独使用

向目标传递请求参数



3、包含：

静态包含：<%@include%>

动态包含：其余的

<jsp:include page=””/>

**JSP的九个隐含对象**

Request

Response

Session

pageContext

out

application

page

exception

**JSP中的异常处理**

**PageContext抽象类**

1、本身是一个域对象，还能操作其他3个域对象中的属性

void setAttribute(String name,Object value);

void removeAttribute(String name);

Object getAttribute(String name);

void setAttribute(String name,Object value,int scope);

void removeAttribute(String name,int scope);

Object getAttribute(String name,int scope);

2、在四个范围中依次查找指定名称对象的方法：

PAGE(页面) REQUEST（请求）SESSION（会话）APPLICATION（应用）

3、获取其他八个JSP隐含对象(自定义标签时用)

4、提供转发和包含

pageContext.forward();

pageContext.include();

**四大域对象**

PageContext:页面范围,最小,开发中很少用。

ServletRequest:请求范围。不同的资源，只有用转发。

HttpSession:会话范围。多次请求共享数据，但不同的客户端不能共享。

ServletContext:应用范围，最大。尽量少用。

**JSP开发的最佳实践**

Sevlet:专门用来处理业务逻辑

JSP:专门用来显示数据

**JavaBean的概念**

JavaBean又称之为POJO（Plain Old Java Object）、VO（Value Object）、DO（Data Object）

必须是public的

提供默认的构造方法

字段都是私有的： private String username;

提供公有的getter或setter方法：属性。

getUsername():读属性.

setUsername(String username):写属性。

一般需要实现java.io.Serializable接口。

结论：JavaBean在开发中，主要担当封装数据用的。

Jsp+javaBean model1模式

**JSP常见标签**

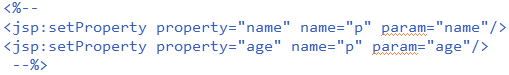
<jsp:userBean>

<jsp:getProperty>:如果属性是null，那么界面就显示null。

<jsp:setProperty>:注意类型转换



可以使用请求参数设置JavaBean的属性



**EL简介**

不是一种语言，只是一个表达式而已，是JSP中专门用于显示数据用的，SinceJSP2.0

主要作用：获取数据并显示到页面上。

**EL表达式语法**

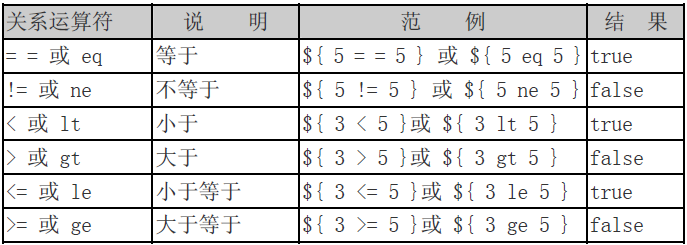
${表达式}

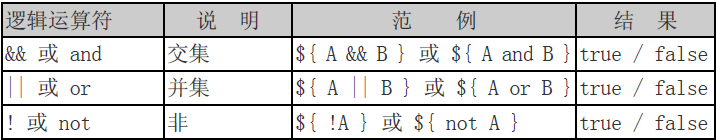
EL表达式，目的是从四大域对象中获取数据。

EL表达式中没有空指针异常，没有数组越界，没有字符串连接。

**EL获取数据方式**

**EL运算符**





empty运算符：

判断一个字符串是否为””或对象是否为null，集合中是否有元素。

三元运算符：

<input type="radio" name="sex" ${sex=='man'?"checked='checked'":'' } />男

**EL中的隐含对象**

pageContext ${pageContext.request.contextPath }

pageScope

requestScope ${applicationScope.p }

sessionScope

applicationScope

param

paramValues ${paramValues.username[1] }

header

headerValues ${headerValues['accept-language'][0] }

initParam ${initParam.user }

Cookie ${cookie.JSESSIONID.value }

**定义和使用EL函数**

${myfn:toUpperCase(user)} ${myfn:pinjie(user,password)}

JSTL中的函数

fn:contains函数：用于判断在源字符串中是否包含目标字符串。

fn:containsIgnoreCase函数：用于判断在源字符串中是否包含目标字符串，并且在判断时忽略大小写。

fn:startsWith函数：用于判断源字符串是否以指定的目标字符串开头。

fn: endsWith函数：用于判断源字符串是否以指定的目标字符串结尾。

fn:indexOf函数：用于在源字符串中查找目标字符串，并返回源字符串中最先与目标字符串匹配的第一个字符的索引。

fn:replace函数：用于把源字符串中的一部分替换为另外的字符串，并返回替换后的字符串。

fn:substring函数：用于获取源字符串中的特定子字符串。

fn:substringBefore函数：用于获取源字符串中指定子字符串之前的子字符串。

fn: substringAfter函数：用于获取源字符串中指定子字符串之后的子字符串

fn:split函数：用于将源字符串拆分为一个字符串数组。

${fn:split(fruit,"-")[1]}

fn:join函数：用于将源字符串数组中的所有字符串连接为一个字符串。

fn:toLowerCase函数：用于将源字符串中的所有字符改为小写。

fn: toUpperCase函数：用于将源字符串中的所有字符改为大写。

fn:trim函数：用于将源字符串中的开头和末尾的空格删除。

fn:escapeXml函数：用于将源字符串中的字符“<”、“>”、“””和“&”等转换为转义字符。

${fn:escapeXml(fruit)}

fn:length函数：用于返回字符串中的字符的个数，或者集合和数组的元素的个数

**自定义标签：简单标签**

void doTag():遇到自定义标签时由服务器调用执行。

JspTag getParent():得到父标签的引用。

void setJspBody(JspFragment jsp):由服务器调用，传出JSP的片段

void setJspContext(JspContext jc):由服务器调用，传入当前页面的PageContext

void setParent(JspTag parent):由服务器调用，传入他的父标签

1、编写一个类，实现javax.servlet.jsp.tagext.SimpleTag，或者继承javax.servlet.jsp.tagext.SimpleTagSupport

2、对标签处理类进行配置，在WEB-INF目录下建立一个扩展名为tld的xml文件

empty：没有主体内容。

scriptless：说明主体内容是非脚本。

tagdependent：把主体内容的EL表达式作为普通字符串

3、在jsp中引入

<%@ taglib uri="http://www.xinhua.com/jsp/tags" prefix="mytag"%>

## 标签应用

控制主体内容是否显示。

控制结束标签后的JSP内容不执行。

控制主体内容重复执行

改变主体内容后再输出

## JSTL标签库

JSP Standard Tag Libary：JSP中标准的标签库

Core：核心标签库。通用逻辑处理

Fmt：国际化。

Functions：EL函数

SQL：操作数据库。

XML：操作XML。

<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

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| Value | true | Object | 指定要输出的内容 |
| escapeXml | true | Boolean | 指定是否将>、<、&、'、" 等特殊字符进行HTML编码转换后再进行输出。默认值为true |
| default | true | Object | 指定如果value属性的值为null时所输出的默认值 |

<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

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| value | true | Object | 用于指定属性值 |
| var | false | String | 用于指定要设置的Web域属性的名称 |
| scope | false | String | 用于指定属性所在的Web域 |
| target | true | Object | 用于指定要设置属性的对象，这个对象必须是JavaBean对象或java.util.Map对象 |
| property | true | string | 用于指定当前要为对象设置的属性名称 |

<c:remove>标签用于删除各种Web域中的属性。

其语法格式如下：

<c:remove var="varName"

[scope="{page|request|session|application}"] />

<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>

var属性用于标识<c:catch>标签捕获的异常对象，它将保存在page这个Web域中

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

<%@ page contentType="text/html;charset=gb2312" %>

<c:catch var="myex“ scope=“page”>

<%

10/0;

%>

</c:catch>

异常：<c:out value="${myex}" /><br />

异常 myex.getMessage：<c:out value="${myex.message}" /><br />

异常 myex.getCause：<c:out value="${myex.cause}" /><br />

异常 myex.getStackTrace：<c:out value="${myex.stackTrace}" />

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

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| test | true | boolean | 决定是否处理标签体中的内容的条件表达式 |
| var | false | String | 用于指定将test属性的执行结果保存到某个Web域中的某个属性的名称 |
| scope | false | String | 指定将test属性的执行结果保存到哪个Web域中 |

<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。

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

<%@ page contentType="text/html;charset=gb2312" %>

<c:set value="${param.count}" var="count“ /> pageContext(count,2)

<c:choose>

<c:when test="${count == 0}">

对不起，没有符合您要求的记录。

</c:when>

<c:otherwise>

符合您要求的记录共有${count}条.

</c:otherwise>

</c:choose>

<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。

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| var | false | String | 指定将当前迭代到的元素保存到page这个Web域中的属性名称 |
| items | true | 任何支持的类型 | 将要迭代的集合对象 |
| begin | true | int | 如果指定items属性，就从集合中的第begin个元素开始进行迭代，begin的索引值从0开始编号；如果没有指定items属性，就从begin指定的值开始迭代，直到end值时结束迭代 |
| end | true | int | 参看begin属性的描述 |
| step | true | int | 指定迭代的步长，即迭代因子的迭代增量 |

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:param>标签在为一个URL地址附加参数时，将自动对参数值进行URL编码，例如，如果传递的参数值为“中国”，则将其转换为“%d6%d0%b9%fa”后再附加到URL地址后面，这也就是使用<c:param>标签的最大好处。

<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

|  |  |  |  |
| --- | --- | --- | --- |
| **属性名** | **是否支持EL** | **属性类型** | **属 性 描 述** |
| value | true | String | 指定要构造的URL |
| var | false | String | 指定将构造出的URL结果保存到Web域中的属性名称 |
| scope | false | String | 指定将构造出的URL结果保存到哪个Web域中 |

<c:redirect>标签用于实现请求重定向

## MySQL的安装与配置

1、验证安装是否成功

C:/>mysql -u root -p

1. DDL：数据定义语言

window下MySQL不区分大小写，其他系统严格区分大小写  
创建一个名称为mydb的数据库。

create database mydb;

查看当前有多少库。

show databases;

查看数据库的创建细节

show create database mydb;

创建一个使用gbk字符集的mydb2数据库。

create database mydb2 character set gbk;

创建一个使用gbk字符集，并带校对规则的mydb3数据库。

create database mydb3 character set gbk collate gbk\_chinese\_ci;

删除前面创建的mydb3数据库

drop database mydb3;

查看服务器中的数据库，并把mydb2的字符集修改为utf8;

alter database mydb2 character set utf8;

查看当前选择的数据库

select database();

选择一个数据库

use mydb

创建一个员工表

create table emp(

id int,

name varchar(100) unique,

sex varchar(10),

birthday date,

job varchar(100),

salary float(6,2)

);

查看表结构

desc emp;

在上面员工表的基本上增加一个image列。

alter table emp add (image longblob);

修改job列，使其长度为60。

alter table emp modify job varchar(60);

删除image列。

alter table emp drop image;

表名改为user。

rename table emp to user;

查看当前库中的所有表

show tables;

修改表的字符集为utf8

alter table USER character set utf8;

列名name修改为username

alter table user change name username varchar(100);

## DML：数据操作语言

作用：用于向数据库表中插入、删除、修改数据

查看数据用到的编码情况：

show variables like 'character\_set%';

编码问题：

告知服务器，客户端使用的编码是什么？

set character\_set\_client=gbk;

告知服务器，返回的结果用什么编码？

set character\_set\_results=gbk;

删除表中所有记录。

delete from USER; (逐条删除)

使用truncate删除表中记录

truncate table USER; (摧毁表格，让后重建表结构)

|  |  |  |
| --- | --- | --- |
| 分类 | 数据类型 | 说明 |
| 数值类型 | BIT(M)  TINYINT [UNSIGNED] [ZEROFILL]  BOOL，BOOLEAN  SMALLINT [UNSIGNED] [ZEROFILL]  INT [UNSIGNED] [ZEROFILL]  BIGINT [UNSIGNED] [ZEROFILL]  FLOAT[(*M*,*D*)] [UNSIGNED] [ZEROFILL]  DOUBLE[(*M*,*D*)] [UNSIGNED] [ZEROFILL] | 位类型。M指定位数，默认值1，范围1-64  带符号的范围是-128到127。无符号0到255。  使用0或1表示真或假  2的16次方  2的32次方  2的64次方  M指定显示长度，d指定小数位数  表示比float精度更大的小数 |
| 文本、二进制类型 | CHAR(size) char(20)  VARCHAR(size)  varchar(20)  BLOB   LONGBLOB  TEXT(clob)          LONGTEXT(longclob) | 固定长度字符串  可变长度字符串  二进制数据  大文本 |
| 时间日期 | DATE/DATETIME/TimeStamp | 日期类型(YYYY-MM-DD)  (YYYY-MM-DD HH:MM:SS)，TimeStamp表示时间戳，它可用于自动记录insert、update操作的时间 |

**Mysql中文乱码**

mysql有六处使用了字符集，分别为：client 、connection、database、results、server 、system。

client是客户端使用的字符集。

connection是连接数据库的字符集设置类型，如果程序没有指明连接数据库使用的字符集类型就按照服务器端默认的字符集设置。

database是数据库服务器中某个库使用的字符集设定，如果建库时没有指明，将使用服务器安装时指定的字符集设置。

results是数据库给客户端返回时使用的字符集设定，如果没有指明，使用服务器默认的字符集。

server是服务器安装时指定的默认字符集设定。

system是数据库系统使用的字符集设定

## 实体（记录）完整性

规定表的一行（即每一条记录）在表中是唯一的实体(Entity)。实体完整性通过表的主键来实现。

主键：不能为null；唯一

三种方式：

alter table 表名 add primary key(ID);

注意：

逻辑主键（建议）：除了唯一标识一条记录外，没有别的意义。一般取名ID

业务主键：还有一定的业务意义

## 域（字段）完整性

指数据库表的列（即字段）必须符合某种特定的数据类型或约束

删除主键：alter table tablename drop primary key ;

定义主键自动增长

auto\_increment

定义唯一约束

unique

定义非空约束

not null

create table person(

pid int primary key auto\_increment,

name varchar(50) unique,

age int not null

)

Create table test(

tid int,

tpid int foreign key references person(pid)

)

## 参照完整性（多表的设计）：定义外键

定义外键约束

Create table emp(

empno int,

enmae varchar(50),

deptno int not null,

primary key(empno),

constraint DEPTNO\_FK foreign key(deptno) references dept(dept\_no)

)

**表与表之间的关系**

一对多的关系

客户customers 订单 orders

表的结构对应javaBean的类的定义

表的一行对应javaBean的一个对象

表的一个列对应的javaBean的一个属性

表的创建

create table customers(

cid int primary key,

name varchar(50)

)

Create table `order`(

oid int primary key,

orderNum varchar(100) unique not null,

customer\_id int,

constraint CUSTOMER\_ID\_FK foreign key(customer\_id) references customers (cid)

)

javaBean表示一对多的关系：

class Customers{

private Integer cid;

privater String name;

private List<Orders> list;

}

Class Orders{

Private Integer oid;

Private String orderNum;

Private Customers cus;

}

多对多关系：

students course

create table students(

sno int primary key auto\_increment,

sname varchar(50)

);

create table course(

cno int primary key auto\_increment,

cname varchar(50) not null,

grade char(2)

);

create table stu\_course(

sno int,

cno int,

primary key(sno,cno),

constraint SNO\_FK foreign key(sno) references students (sno),

constraint CNO\_FK foreign key(cno) references course (cno)

);

一对一关系：

（1）通过外键的方式

Person IdCard

Create table person(

Pid int primary key,

Name varchar(50) not null,

Address varchar(100) not null

);

**Create table idcard(**

**Id int primary key,**

**Idno char(18),**

**Person\_id int unique,**

**constraint PERSON\_ID\_FK foreign key(Person\_id) references person (pid),**

**)**

**（2）通过引用主表主键的方式（了解）**

**运算符**

|  |  |  |
| --- | --- | --- |
| 比较运算符 | >   <   <=   >=   =    <> | 大于、小于、大于(小于)等于、不等于 |
| **BETWEEN  ...AND...** | 显示在某一区间的值 |
| **IN(set)** | 显示在in列表中的值，例：in(100,200) |
| **LIKE ‘张pattern’** | 模糊查询 |
| **IS NULL** | 判断是否为空 |
| 逻辑运算符 | and | 多个条件同时成立 |
| or | 多个条件任一成立 |
| not | 不成立，例：where not(salary>100); |

## 相关函数 sum count max min avg

Select deptno ,sum(sal) from emp group by deptno;

|  |  |
| --- | --- |
| **ADDTIME (date2 ,time\_interval )** | **将time\_interval加到date2** |
| **CURRENT\_DATE (  )** | **当前日期** |
| **CURRENT\_TIME (  )** | **当前时间** |
| **CURRENT\_TIMESTAMP (  )** | **当前时间戳** |
| **DATE (datetime )** | **返回datetime的日期部分** |
| **DATE\_ADD (date2 , INTERVAL d\_value d\_type )** | **在date2中加上日期或时间** |
| **DATE\_SUB (date2 , INTERVAL d\_value d\_type )** | **在date2上减去一个时间** |
| **DATEDIFF (date1 ,date2 )** | **两个日期差** |
| **NOW (  )** | **当前时间** |
| **YEAR|Month|DATE (datetime )** | **年月日** |

|  |  |  |
| --- | --- | --- |
| **CHARSET(str)** | **返回字串字符集** | |
| **CONCAT (string2  [,... ])** | **连接字串** | |
| **INSTR (string ,substring )** | **返回substring在string中出现的位置,没有返回0** | |
| **UCASE (string2 )** | **转换成大写** | |
| **LCASE (string2 )** | **转换成小写** | |
| **LEFT (string2 ,length )** | **从string2中的左边起取length个字符** | |
| **LENGTH (string )** | **string长度** | |
| **REPLACE (str ,search\_str ,replace\_str )** | **在str中用replace\_str替换search\_str** | |
| **STRCMP (string1 ,string2 )** | **逐字符比较两字串大小,** | |
| **SUBSTRING (str , position  [,length ])** | **从str的position开始,取length个字符** | |
| **LTRIM (string2 ) RTRIM (string2 )  trim** | **去除前端空格或后端空格** | |
| **ABS (number2 )** | | **绝对值** | |
| **BIN (decimal\_number )** | | **十进制转二进制** | |
| **CEILING (number2 )** | | **向上取整** | |
| **CONV(number2,from\_base,to\_base)** | | **进制转换** | |
| **FLOOR (number2 )** | | **向下取整** | |
| **FORMAT (number,decimal\_places )** | | **保留小数位数** | |
| **HEX (DecimalNumber )** | | **转十六进制** | |
| **LEAST (number , number2  [,..])** | | **求最小值** | |
| **MOD (numerator ,denominator )** | | **求余** | |
| **RAND([seed])** | | **RAND([seed])** | |

**MySQL的备份与数据恢复**

mysqldump -h localhost -p3306 -u root -p mydb>d:/mydb.sql

方式一:

source d:/mydb.sql;

方式二:

mysql -u root -p test<d:/mydb.sql

create table person(

id int primary key auto\_increment,

userName varchar(50),

password varchar(6)

);

Insert into person (userName,password) values('zhangsan ', '123456 ');

Insert into person (userName,password) values('lisi ', '45678 ');

Insert into person (userName,password) values('wangwu ', '111111 ');

复制表中的数据

Insert into person (name) select name from person;

Mysql的分页语句

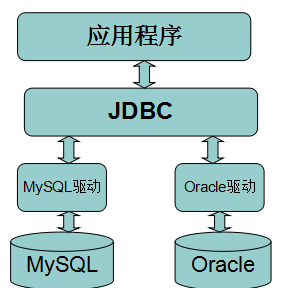
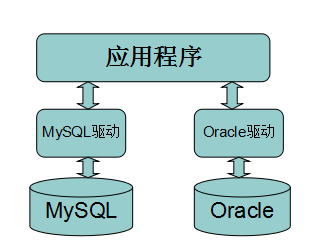
开始索引号=(当前页-1)\* 每页显示条数

Select \* from person limit 开始索引号,每页显示条数;

**JDBC简介**

数据库驱动

SUN公司为了简化、统一对数据库的操作，定义了一套Java操作数据库的规范，称之为JDBC。



DBC全称为：Java Data Base Connectivity（java数据库连接），它主要由接口组成。

组成JDBC的２个包：

　java.sql

　javax.sql

开发JDBC应用需要以上2个包的支持外，还需要导入相应JDBC的数据库实现(即数据库驱动)

编写一个程序，这个程序从person表中读取数据，并打印在命令行窗口中。

一、搭建实验环境 ：

1、在mysql中创建一个库，并创建person表和插入表的数据。

2、新建一个Java工程，并导入数据驱动。

二、编写程序，在程序中加载数据库驱动

DriverManager. registerDriver(Driver driver)

三、建立连接(Connection)

Connection conn = DriverManager.getConnection(url,user,pass);

四、创建用于向数据库发送SQL的Statement对象，并发送sql

Statement st = conn.createStatement();

ResultSet rs = st.excuteQuery(sql);

五、从代表结果集的ResultSet中取出数据，打印到命令行窗口

六、断开与数据库的连接，并释放相关资源

**DriverManager**

Jdbc程序中的DriverManager用于加载驱动，并创建与数据库的链接，这个API的常用方法：

DriverManager.registerDriver(new Driver())

DriverManager.getConnection(url, user, password)，

注意：在实际开发中并不推荐采用registerDriver方法注册驱动。原因有二：

一、查看Driver的源代码可以看到，如果采用此种方式，会导致驱动程序注册两次，也就是在内存中会有两个Driver对象。

二、程序依赖mysql的api，脱离mysql的jar包，程序将无法编译，将来程序切换底层数据库将会非常麻烦。

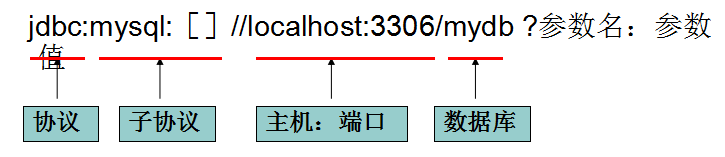
推荐方式：Class.forName(“com.mysql.jdbc.Driver”);

采用此种方式不会导致驱动对象在内存中重复出现，并且采用此种方式，程序仅仅只需要一个字符串，不需要依赖具体的驱动，使程序的灵活性更高。

同样，在开发中也不建议采用具体的驱动类型指向getConnection方法返回的connection对象。

**数据库URL**

URL用于标识数据库的位置，程序员通过URL地址告诉JDBC程序连接哪个数据库，URL的写法为：



常用数据库URL地址的写法：

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

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

MySql—jdbc:mysql://localhost:3306/sid

Mysql的url地址的简写形式： jdbc:mysql:///sid

常用属性：useUnicode=true&characterEncoding=UTF-8

**Connection**

Jdbc程序中的Connection，它用于代表数据库的链接，Collection是数据库编程中最重要的一个对象，客户端与数据库所有交互都是通过connection对象完成的，这个对象的常用方法：

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。

prepareCall(sql)：创建执行存储过程的callableStatement对象。

setAutoCommit(boolean autoCommit)：设置事务是否自动提交。

commit() ：在链接上提交事务。

rollback() ：在此链接上回滚事务。

**Statement**

Jdbc程序中的Statement对象用于向数据库发送SQL语句， Statement对象常用方法：

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

addBatch(String sql) ：把多条sql语句放到一个批处理中。

executeBatch()：向数据库发送一批sql语句执行。

**ResultSet**

Jdbc程序中的ResultSet用于代表Sql语句的执行结果。Resultset封装执行结果时，采用的类似于表格的方式。ResultSet 对象维护了一个指向表格数据行的游标，初始的时候，游标在第一行之前，调用ResultSet.next() 方法，可以使游标指向具体的数据行，进行调用方法获取该行的数据。

ResultSet既然用于封装执行结果的，所以该对象提供的都是用于获取数据的get方法：

获取任意类型的数据

getObject(int index)

getObject(string columnName)

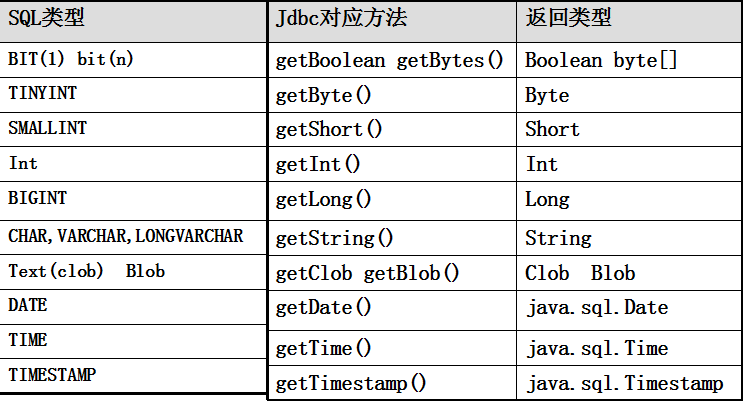
获取指定类型的数据，例如：

getString(int index)

getString(String columnName)

提问：数据库中列的类型是varchar，获取该列的数据调用什么方法？Int类型呢？bigInt类型呢？Boolean类型？

**常用数据类型转换表**



ResultSet还提供了对结果集进行滚动的方法：

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

**释放资源**

Jdbc程序运行完后，切记要释放程序在运行过程中，创建的那些与数据库进行交互的对象，这些对象通常是ResultSet, Statement和Connection对象。

特别是Connection对象，它是非常稀有的资源，用完后必须马上释放，如果Connection不能及时、正确的关闭，极易导致系统宕机。Connection的使用原则是尽量晚创建，尽量早的释放。

为确保资源释放代码能运行，资源释放代码也一定要放在finally语句中。

**JDBC进行批处理**

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的批处理机制，以提升执行效率。

实现批处理有两种方式，第一种方式：

Statement.addBatch(sql) list

执行批处理SQL语句

executeBatch()方法：执行批处理命令

clearBatch()方法：清除批处理命令

采用Statement.addBatch(sql)方式实现批处理：

优点：可以向数据库发送多条不同的ＳＱＬ语句。

缺点：

Connection conn = null;

Statement st = null;

ResultSet rs = null;

try {

conn = JdbcUtil.getConnection();

String sql1 = "insert into user(name,password)

values('kkk','123')";

String sql2 = "update person set password='123456' where id=3";

st = conn.createStatement();

st.addBatch(sql1); //把SQL语句加入到批命令中

st.addBatch(sql2); //把SQL语句加入到批命令中

st.executeBatch();

} finally{

JdbcUtil.free(conn, st, rs);

}

SQL语句没有预编译。

当向数据库发送多条语句相同，但仅参数不同的SQL语句时，需重复写上很多条SQL语句。例如：

Insert into user(name,password) values(‘aa’,’111’);

Insert into user(name,password) values(‘bb’,’222’);

Insert into user(name,password) values(‘cc’,’333’);

Insert into user(name,password) values(‘dd’,’444’);

实现批处理的第二种方式：

PreparedStatement.addBatch()

conn = JdbcUtil.getConnection();

String sql = "insert into person(name,password) values(?,?)";

st = conn.prepareStatement(sql);

for(int i=0;i<100000;i++){

st.setString(1, "aaa" + i);

st.setString(2, "123" + i);

st.addBatch();

if(i%1000==0){

st.executeBatch();

st.clearBatch();

}

}

st.executeBatch();

采用PreparedStatement.addBatch()实现批处理

优点：发送的是预编译后的SQL语句，执行效率高。

缺点：只能应用在SQL语句相同，但参数不同的批处理中。因此此种形式的批处理经常用于在同一个表中批量插入数据，或批量更新表的数据。

**获得数据库自动生成的主键**

Connection conn = JdbcUtil.getConnection();

String sql = "insert into person(name,password,email,birthday)

values('abc','123')";

PreparedStatement st = conn.

prepareStatement(sql,Statement.RETURN\_GENERATED\_KEYS );

st.executeUpdate();

ResultSet rs = st.getGeneratedKeys(); //得到插入行的主键

if(rs.next())

System.out.println(rs.getObject(1));

此参数仅对insert操作有效

**使用JDBC处理大数据**

在实际开发中，程序需要把大文本或二进制数据保存到数据库。

基本概念：大数据也称之为LOB(Large Objects)，LOB又分为：

clob和blob

clob用于存储大文本。

blob用于存储二进制数据，例如图像、声音、二进制文等。

对MySQL而言只有blob，而没有clob，mysql存储大文本采用的是Text，Text和blob分别又分为：

TINYTEXT、TEXT、MEDIUMTEXT和LONGTEXT

TINYBLOB、BLOB、MEDIUMBLOB和LONGBLOB

对于MySQL中的Text类型，可调用如下方法设置：

PreparedStatement.setCharacterStream(index, reader, length);

//注意length长度须设置，并且设置为int型

对MySQL中的Text类型，可调用如下方法获取:

reader = resultSet. getCharacterStream(i);

reader = resultSet.getClob(i).getCharacterStream();

string s = resultSet.getString(i);

**JDBC处理二进制数据**

对于MySQL中的BLOB类型，可调用如下方法设置：

PreparedStatement. setBinaryStream(i, inputStream, length);

可调用如下方法获取：

InputStream in = resultSet.getBinaryStream(i);

InputStream in = resultSet.getBlob(i).getBinaryStream();

**JDBC调用存储过程**

得到CallableStatement，并调用存储过程：

CallableStatement cStmt = conn.prepareCall("{call demoSp(?, ?)}");

设置参数，注册返回值，得到输出

cStmt.registerOutParameter(2, Types.VARCHAR);

cStmt.setString(1, "abcdefg");

cStmt.execute();

System.out.println(cStmt.getString(2));

**存储过程案例：**

delimiter $$

CREATE PROCEDURE mypro(IN inputParam VARCHAR(255), INOUT inOutParam varchar(255))

BEGIN

SELECT CONCAT('hello', inputParam) into inOutParam;

END $$

delimiter ;

**事务**

事务的概念

事务指逻辑上的一组操作，组成这组操作的各个单元，要不全部成功，要不全部不成功。

例如：A——B转帐，对应于如下两条sql语句

update from account set money=money+100 where name=‘b’;

update from account set money=money-100 where name=‘a’;

数据库开启事务命令

start transaction 开启事务

Rollback 回滚事务

Commit 提交事务

**使用事务**

当Jdbc程序向数据库获得一个Connection对象时，默认情况下这个Connection对象会自动向数据库提交在它上面发送的SQL语句。若想关闭这种默认提交方式，让多条SQL在一个事务中执行，可使用下列语句：

JDBC控制事务语句

Connection.setAutoCommit(false); start transaction

Connection.rollback(); rollback

Connection.commit(); commit

**事务的特性(ACID)**

原子性（Atomicity）  
原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性（Consistency）  
事务必须使数据库从一个一致性状态变换到另外一个一致性状态。

隔离性（Isolation）  
事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

持久性（Durability）  
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，接下来即使数据库发生故障也不应该对其有任何影响

**事务的隔离级别**

多个线程开启各自事务操作数据库中数据时，数据库系统要负责隔离操作，以保证各个线程在获取数据时的准确性。

**脏读**：

指一个事务读取了另外一个事务未提交的数据。

这是非常危险的，假设Ａ向Ｂ转帐１００元，对应sql语句如下所示

1.update account set money=money+100 while name=‘b’;

2.update account set money=money-100 while name=‘a’;

当第1条sql执行完，第2条还没执行(A未提交时)，如果此时Ｂ查询自己的帐户，就会发现自己多了100元钱。如果A等B走后再回滚，B就会损失100元

**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。

例如银行想查询A帐户余额，第一次查询A帐户为200元，此时A向帐户内存了100元并提交了，银行接着又进行了一次查询，此时A帐户为300元了。银行两次查询不一致，可能就会很困惑，不知道哪次查询是准的。

和脏读的区别是，脏读是读取前一事务未提交的脏数据，不可重复读是重新读取了前一事务已提交的数据。

很多人认为这种情况就对了，无须困惑，当然是后面的为准。我们可以考虑这样一种情况，比如银行程序需要将查询结果分别输出到电脑屏幕和写到文件中，结果在一个事务中针对输出的目的地，进行的两次查询不一致，导致文件和屏幕中的结果不一致，银行工作人员就不知道以哪个为准了。

**虚读(幻读)**

是指在一个事务内读取到了别的事务插入的数据，导致前后读取不一致。

如丙存款100元未提交，这时银行做报表统计account表中所有用户的总额为500元，然后丙提交了，这时银行再统计发现帐户为600元了，造成虚读同样会使银行不知所措，到底以哪个为准。

数据库共定义了四种隔离级别：

Serializable：可避免脏读、不可重复读、虚读情况的发生。（串行化）

Repeatable read：可避免脏读、不可重复读情况的发生。（可重复读）

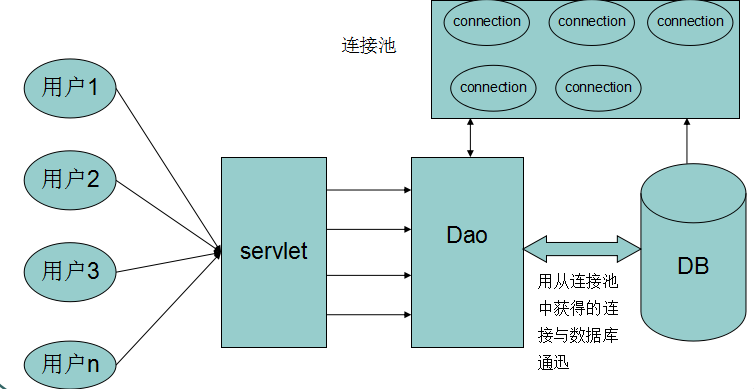
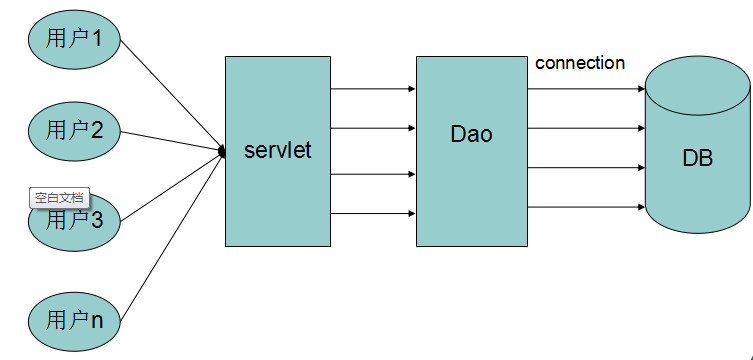
Read committed：可避免脏读情况发生（读已提交）。

Read uncommitted：最低级别，以上情况均无法保证。(读未提交)

set transaction isolation level 设置事务隔离级别

select @@tx\_isolation 查询当前事务隔离级别

**数据库连接池**



**编写数据库连接池**

编写连接池需实现java.sql.DataSource接口。DataSource接口中定义了两个重载的getConnection方法：

Connection getConnection()

Connection getConnection(String username, String password)

实现DataSource接口，并实现连接池功能的步骤：

在DataSource构造函数中批量创建与数据库的连接，并把创建的连接加入LinkedList对象中。

实现getConnection方法，让getConnection方法每次调用时，从LinkedList中取一个Connection返回给用户。

当用户使用完Connection，调用Connection.close()方法时，Collection对象应保证将自己返回到LinkedList中,而不要把conn还给数据库。

**包装设计模式**

**使用动态代理技术构建连接池中的connection**

proxyConn = (Connection) Proxy.newProxyInstance(this.getClass()

.getClassLoader(), conn.getClass().getInterfaces(),

new InvocationHandler() {

//此处为内部类，当close方法被调用时将conn还回池中,其它方法直接执行

public Object invoke(Object proxy, Method method,

Object[] args) throws Throwable {

if (method.getName().equals("close")) {

pool.addLast(conn);

return null;

}

return method.invoke(conn, args);

}

});

**CGLIB代理技术构建连接池中的connection**

final Roler role = new Roler ();

Roler proxy = (Roler) Enhancer.create(Roler.class,new MethodInterceptor(){

public Object intercept(Object proxy, Method method, Object[] args,MethodProxy arg3) throws Throwable {

}

});

**开源数据库连接池**

现在很多WEB服务器(Weblogic, WebSphere, Tomcat)都提供了DataSoruce的实现，即连接池的实现。通常我们把DataSource的实现，按其英文含义称之为数据源，数据源中都包含了数据库连接池的实现。

也有一些开源组织提供了数据源的独立实现：

DBCP 数据库连接池

C3P0 数据库连接池

实际应用时不需要编写连接数据库代码，直接从数据源获得数据库的连接。程序员编程时也应尽量使用这些数据源的实现，以提升程序的数据库访问性能。

**DBCP数据源**

DBCP 是 Apache 软件基金组织下的开源连接池实现，使用DBCP数据源，应用程序应在系统中增加如下两个 jar 文件：

Commons-dbcp.jar：连接池的实现

Commons-pool.jar：连接池实现的依赖库

Tomcat 的连接池正是采用该连接池来实现的。该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。

public class DBCPUtil {

private static DataSource dataSource;

static{

try {

InputStream in = DBCPUtil.class.getClassLoader().getResourceAsStream("dbcpconfig.properties");

Properties props = new Properties();

props.load(in);

dataSource = BasicDataSourceFactory.createDataSource(props);

} catch (Exception e) {

throw new ExceptionInInitializerError(e);

}

}

public static DataSource getDataSource(){

return dataSource;

}

public static Connection getConnection(){

try {

return dataSource.getConnection();

} catch (SQLException e) {

throw new RuntimeException(e);

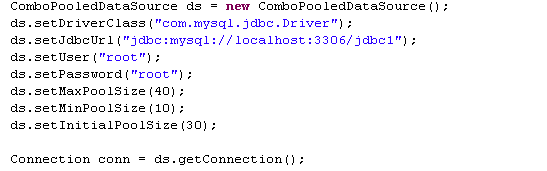
}

}

}

**C3P0 数据源**

**（1）硬编码方式**

**（2）配置文件方式**

public class C3P0Util {

private static ComboPooledDataSource dataSource = new ComboPooledDataSource();

public static DataSource getDataSource(){

return dataSource;

}

public static Connection getConnection(){

try {

return dataSource.getConnection();

} catch (SQLException e) {

throw new RuntimeException(e);

}

}

}

JNDI容器

**元数据－ DataBaseMetaData**

元数据：数据库、表、列的定义信息。

Connection.getDatabaseMetaData()

DataBaseMetaData对象

getURL()：返回一个String类对象，代表数据库的URL。

getUserName()：返回连接当前数据库管理系统的用户名。

getDatabaseProductName()：返回数据库的产品名称。

getDatabaseProductVersion()：返回数据库的版本号。

getDriverName()：返回驱动驱动程序的名称。

getDriverVersion()：返回驱动程序的版本号。

isReadOnly()：返回一个boolean值，指示数据库是否只允许读操作。

**元数据－ ParameterMetaData**

PreparedStatement . getParameterMetaData()

获得代表PreparedStatement元数据的ParameterMetaData对象。

Select \* from user where name=? And password=?

ParameterMetaData对象

getParameterCount()

获得指定参数的个数

getParameterType(int param)

获得指定参数的sql类型

**元数据－ ResultSetMetaData**

ResultSet. getMetaData()

获得代表ResultSet对象元数据的ResultSetMetaData对象。

ResultSetMetaData对象

getColumnCount()

返回resultset对象的列数

getColumnName(int column)

获得指定列的名称

 getColumnTypeName(int column)

获得指定列的类型

**使用元数据封装代码**

所有实体的CUD操作代码基本相同，仅仅发送给数据库的SQL语句不同而已，因此可以把CUD操作的所有相同代码抽取到工具类的一个update方法中，并定义参数接收变化的SQL语句。

实体的R操作，除SQL语句不同之外，根据操作的实体不同，对ResultSet的映射也各不相同，因此可义一个query方法，除以参数形式接收变化的SQL语句外，可以使用策略模式由qurey方法的调用者决定如何把ResultSet中的数据映射到实体对象中。

**DBUtils框架**

commons-dbutils 是 Apache 组织提供的一个开源 JDBC工具类库，它是对JDBC的简单封装，学习成本极低，并且使用dbutils能极大简化jdbc编码的工作量，同时也不会影响程序的性能。因此dbutils成为很多不喜欢hibernate的公司的首选。

API介绍：

org.apache.commons.dbutils.QueryRunner

org.apache.commons.dbutils.ResultSetHandler

DbUtils ：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。主要方法如下：

public static void close() throws java.sql.SQLException：　DbUtils类提供了三个重载的关闭方法。这些方法检查所提供的参数是不是NULL，如果不是的话，它们就关闭Connection、Statement和ResultSet。

public static void closeQuietly(): 这一类方法不仅能在Connection、Statement和ResultSet为NULL情况下避免关闭，还能隐藏一些在程序中抛出的SQLEeception。

public static void commitAndCloseQuietly(Connection conn)： 用来提交连接，然后关闭连接，并且在关闭连接时不抛出SQL异常。

public static boolean loadDriver(java.lang.String driverClassName)：这一方装载并注册JDBC驱动程序，如果成功就返回true。使用该方法，你不需要捕捉这个异常ClassNotFoundException。

QueryRunner类提供了两个构造方法：

默认的构造方法

需要一个 javax.sql.DataSource 来作参数的构造方法

public Object query(Connection conn, String sql, Object[] params, ResultSetHandler rsh) throws SQLException：执行一个查询操作，在这个查询中，对象数组中的每个元素值被用来作为查询语句的置换参数。该方法会自行处理 PreparedStatement 和 ResultSet 的创建和关闭。

public Object query(String sql, Object[] params, ResultSetHandler rsh) throws SQLException:　几乎与第一种方法一样；唯一的不同在于它不将数据库连接提供给方法，并且它是从提供给构造方法的数据源(DataSource) 或使用的setDataSource 方法中重新获得 Connection。

public Object query(Connection conn, String sql, ResultSetHandler rsh) throws SQLException : 执行一个不需要置换参数的查询操作。

public int update(Connection conn, String sql, Object[] params) throws SQLException:用来执行一个更新（插入、更新或删除）操作。

public int update(Connection conn, String sql) throws SQLException：用来执行一个不需要置换参数的更新操作。

**ResultSetHandler接口**

该接口用于处理 java.sql.ResultSet，将数据按要求转换为另一种形式。

ResultSetHandler 接口提供了一个单独的方法：Object handle (java.sql.ResultSet .rs)

ArrayHandler：把结果集中的第一行数据转成对象数组。

ArrayListHandler：把结果集中的每一行数据都转成一个数组，再存放到List中。

BeanHandler：将结果集中的第一行数据封装到一个对应的JavaBean实例中。

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。

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

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

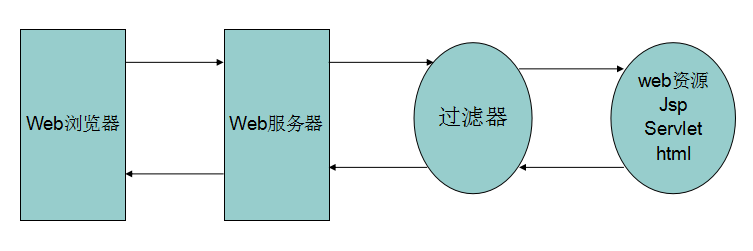
MapHandler：将结果集中的第一行数据封装到一个Map里，key是列名，value就是对应的值。

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

**Filter简介**

Filter也称之为过滤器，它是Servlet技术中最激动人心的技术，WEB开发人员通过Filter技术，对web服务器管理的所有web资源：例如Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现URL级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。

Servlet API中提供了一个Filter接口，开发web应用时，如果编写的Java类实现了这个接口，则把这个java类称之为过滤器Filter。通过Filter技术，开发人员可以实现用户在访问某个目标资源之前，对访问的请求和响应进行拦截，如下所示：



**Filter拦截原理**

Filter接口中有一个doFilter方法，当开发人员编写好Filter，并配置对哪个web资源进行拦截后，WEB服务器每次在调用web资源的service方法之前，都会先调用一下filter的doFilter方法，因此，在该方法内编写代码可达到如下目的：

调用目标资源之前，让一段代码执行

是否调用目标资源（即是否让用户访问web资源）。

web服务器在调用doFilter方法时，会传递一个filterChain对象进来，filterChain对象是filter接口中最重要的一个对象，它也提供了一个doFilter方法，开发人员可以根据需求决定是否调用此方法，调用该方法，则web服务器就会调用web资源的service方法，即web资源就会被访问，否则web资源不会被访问。

调用目标资源之后，让一段代码执行

**Filter开发步骤**

Filter开发分为二个步骤：

编写java类实现Filter接口，并实现其doFilter方法。

在 web.xml 文件中使用<filter>和<filter-mapping>元素对编写的filter类进行注册，并设置它所能拦截的资源。

**Filter链**

在一个web应用中，可以开发编写多个Filter，这些Filter组合起来称之为一个Filter链。

web服务器根据Filter在web.xml文件中的注册顺序，决定先调用哪个Filter，当第一个Filter的doFilter方法被调用时，web服务器会创建一个代表Filter链的FilterChain对象传递给该方法。在doFilter方法中，开发人员如果调用了FilterChain对象的doFilter方法，则web服务器会检查FilterChain对象中是否还有filter，如果有，则调用第2个filter，如果没有，则调用目标资源。

**Filter的生命周期**

init(FilterConfig filterConfig)throws ServletException：

和我们编写的Servlet程序一样，Filter的创建和销毁由WEB服务器负责。 web 应用程序启动时，web 服务器将创建Filter 的实例对象，并调用其init方法，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（注：filter对象只会创建一次，init方法也只会执行一次。示例 ）

开发人员通过init方法的参数，可获得代表当前filter配置信息的FilterConfig对象。

destroy()：

在Web容器卸载 Filter 对象之前被调用。该方法在Filter的生命周期中仅执行一次。在这个方法中，可以释放过滤器使用的资源。

**FilterConfig接口**

用户在配置filter时，可以使用<init-param>为filter配置一些初始化参数，当web容器实例化Filter对象，调用其init方法时，会把封装了filter初始化参数的filterConfig对象传递进来。因此开发人员在编写filter时，通过filterConfig对象的方法，就可获得：

String getFilterName()：得到filter的名称。

String getInitParameter(String name)： 返回在部署描述中指定名称的初始化参数的值。如果不存在返回null.

Enumeration getInitParameterNames()：返回过滤器的所有初始化参数的名字的枚举集合。

public ServletContext getServletContext()：返回Servlet上下文对象的引用。

**应用1：统一全站字符编码的过滤器**

**应用2：禁止浏览器缓存动态页面的过滤器**

有 3 个 HTTP 响应头字段都可以禁止浏览器缓存当前页面，它们在 Servlet 中的示例代码如下：

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

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

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

并不是所有的浏览器都能完全支持上面的三个响应头，因此最好是同时使用上面的三个响应头。

Expires数据头：值为GMT时间值，为-1指浏览器不要缓存页面

Cache-Control响应头有两个常用值：

no-cache指浏览器不要缓存当前页面。

max-age:xxx指浏览器缓存页面xxx秒

**应用3：控制浏览器缓存静态资源的过滤器**

**文件上传**

实现web开发中的文件上传功能，需完成如下二步操作：

在web页面中添加上传输入项

在servlet中读取上传文件的数据，并保存到本地硬盘中。

如何在web页面中添加上传输入项?

<input type=“file”>标签用于在web页面中添加文件上传输入项，设置文件上传输入项时须注意：

1、必须要设置input输入项的name属性，否则浏览器将不会发送上传文件的数据。

２、必须把form的enctype属值设为multipart/form-data.设置该值后，浏览器在上传文件时，将把文件数据附带在http请求消息体中，并使用ＭＩＭＥ协议对上传的文件进行描述，以方便接收方对上传数据进行解析和处理。

如何在Servlet中读取文件上传数据，并保存到本地硬盘中?

Request对象提供了一个getInputStream方法，通过这个方法可以读取到客户端提交过来的数据。但由于用户可能会同时上传多个文件，在servlet端编程直接读取上传数据，并分别解析出相应的文件数据是一项非常麻烦的工作，示例。

为方便用户处理文件上传数据，Apache 开源组织提供了一个用来处理表单文件上传的一个开源组件（ Commons-fileupload ），该组件性能优异，并且其API使用极其简单，可以让开发人员轻松实现web文件上传功能，因此在web开发中实现文件上传功能，通常使用Commons-fileupload组件实现。

使用Commons-fileupload组件实现文件上传，需要导入该组件相应的支撑jar包：Commons-fileupload和commons-io。commons-io 不属于文件上传组件的开发jar文件，但Commons-fileupload 组件从1.1 版本开始，它工作时需要commons-io包的支持

**DiskFileItemFactory**

DiskFileItemFactory 是创建 FileItem 对象的工厂，这个工厂类常用方法：

public void setSizeThreshold(int sizeThreshold)

设置内存缓冲区的大小，默认值为10K。当上传文件大于缓冲区大小时， fileupload组件将使用临时文件缓存上传文件。

public void setRepository(java.io.File repository)

指定临时文件目录，默认值为System.getProperty("java.io.tmpdir").

public DiskFileItemFactory(int sizeThreshold, java.io.File repository)

构造函数

ServletFileUpload 负责处理上传的文件数据，并将表单中每个输入项封装成一个 FileItem 对象中。常用方法有：

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

setProgressListener(ProgressListener pListener)   
实现步骤

１、创建DiskFileItemFactory对象，设置缓冲区大小和临时文件目录

２、使用DiskFileItemFactory 对象创建ServletFileUpload对象，并设置上传文件的大小限制。

３、调用ServletFileUpload.parseRequest方法解析request对象，得到一个保存了所有上传内容的List对象。

４、对list进行迭代，每迭代一个FileItem对象，调用其isFormField方法判断是否是上传文件

为普通表单字段，则调用getFieldName、getString方法得到字段名和字段值

为上传文件，则调用getInputStream方法得到数据输入流，从而读取上传数据。

**中文文件乱码问题**

文件名中文乱码问题，可调用ServletUpLoader的setHeaderEncoding方法，或者设置request的setCharacterEncoding属性

临时文件的删除问题

由于文件大小超出DiskFileItemFactory.setSizeThreshold方法设置的内存缓冲区的大小时，Commons-fileupload组件将使用临时文件保存上传数据，因此在程序结束时，务必调用FileItem.delete**方法删除临时文件**

Delete方法的调用必须位于流关闭之后，否则会出现文件占用，而导致删除失败的情况。

**文件存放位置**

为保证服务器安全，上传文件应保存在应用程序的WEB-INF目录下，或者不受WEB服务器管理的目录。

为防止多用户上传相同文件名的文件，而导致文件覆盖的情况发生，文件上传程序应保证上传文件具有唯一文件名。

为防止单个目录下文件过多，影响文件读写速度，处理上传文件的程序应根据可能的文件上传总量，选择合适的目录结构生成算法，将上传文件分散存储。

**ProgressListener显示上传进度**

ProgressListener progressListener = new ProgressListener() {

public void update(long pBytesRead, long pContentLength, int pItems) {

System.out.println("到现在为止, " + pBytesRead + " 字节已上传，总大小为 "

+ pContentLength);

}

};

upload.setProgressListener(progressListener);

以KB为单位显示上传进度

long temp = -1; //temp注意设置为类变量

long ctemp = pBytesRead /1024;

if (mBytes == ctemp)

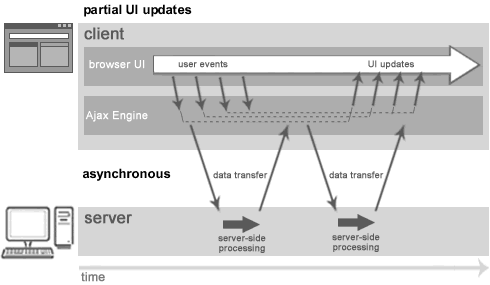
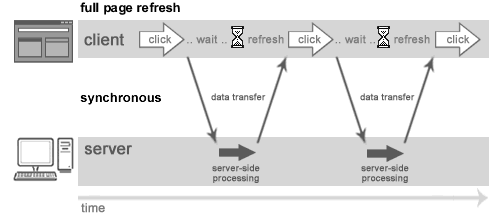
return;

temp = mBytes;

**AJAX**

Asynchronous JavaScript And XML指异步 JavaScript 及 XML，不是一种新的编程语言，而是一种用于创建更好更快以及交互性更强的 Web 应用程序的技术

**同步和异步**



**AJAX编码步骤**

1）、创建XmlHttpRequest对象

2）、注册状态监控回调函数

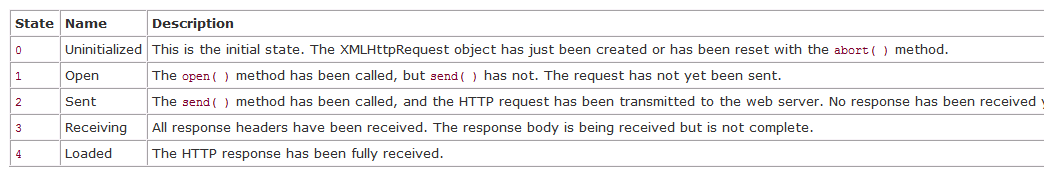
3）、建立与服务器的异步连接

4）、发出异步请求

**XmlHttpRequest对象**

属性

readyState：类型short



XmlHttpRequest对象被创建时，值为0

刚建立与服务器建立了链接，但是还没有发请求，值为1

调用了send方法发出请求，但是服务器没有任何响应，值为2

接收到服务器发出的响应消息头时，值为3

接收到服务器发送的响应正文，响应结束，值为4

responseText：类型String

responseXML：类型Document

status：类型short

statusText：类型String

方法：

abort()

getAllResponseHeaders()

getResponseHeader()

open()

open(method,url, asynch)

其中method表示HTTP调用方法，一般使用"GET"，"POST"

url表示调用的服务器的地址

asynch表示是否采用异步方式，true表示异步，一般这个参数不写

示例:

xmlHttp.open("POST", "url");

xmlHttp.open("GET", "url?name=zhangsan&passwd=123");

send()

GET方式提交

数据在URL上

xmlHttp.send(null);

POST方式提交

xmlHttp.setRequestHeader("CONTENT-TYPE","application/x-www-form-urlencoded");

xmlHttp.send("name=zhangsan&passwd=123");

setRequestHeader()

事件处理器

onreadystatechange

function 回调函数(){

if(xmlHttp.readyState == 4) { //如果响应完成

if(xmlHttp.status == 200) {//如果返回成功

}

}

}

**JSON**

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。 易于人阅读和编写。同时也易于机器解析和生成。JSON采用完全独立于语言的文本格式，JSON是理想的数据交换语言。

JSON结构：

1）“名称/值”对的集合。在Java语言中，我们可以将它理解成HashMap.

一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间使用“,”（逗号）分隔。

var obj= {“name”:”zhangsan”,”age”:20,”sex”:”男”};

1. 值的有序列表

值（value）的有序集合。以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。

var json = [“zhangsan”,”lisi”,”wangwu”,true,10];

**Json-lib**

JSON-lib 是一个第三方java类库

转换 maps,javabeans, arrays,collections,XML 成为json 格式数据

**使用JsonConfig过虑属性：适用于JavaBean/Map**

JsonConfig jc=new JsonConfig();

jc.setExcludes(new String[]{});

JSONObject.fromObject(javaBean,jc);

转换json格式数据成为 javabean对象

将json串转换成Array

JSONArray js=JSONArray.fromObject(“[\“zhangsan\”,\”lisi\”,\”wangwu\”]”);

Object obj=JSONArray.toArray(js);

将json串转成JavaBean/Map

JSONObject jo=JSONObject.fromObject(“{\“name\”:\”zhangsan\”,\”age\”:20,\”sex\”:\”男\”}”);

JSONObject.toBean(jo,Map.class);

或者

JSONObject.toBean(jo,javaBean.class);

**Xstream**

Xstream是一个简单类库用来转换java对象成为XML和转换XML成为 java对象

创建Xstream对象：

**Xstream xs=new Xstream();**

**xs.toXML();**

别名设置1：

Xs.alias(“province”,”Province.class”);

@XStreamAlias(别名) 对类和变量设置别名

@XStreamAsAttribute 设置变量生成属性

@XStreamOmitField 设置变量不生成到XML

@XStreamImplicit(itemFieldName=””)

设置集合类型变量别名

在Xstream生成XML时使用注解

xStream.autodetectAnnotations(true);

**struts2框架**

**1）struts2入门**

框架含义及作用：框架是实现了部分功能的半成品，使用框架简化企业级软件开发 ,目的是提高开发效率。

Struts2 是一个非常优秀的MVC框架，基于Model2（model2=servlet+jsp+javaBean）设计模型，由传统Struts1和WebWork两个经典框架发展而来，使用struts2框架，可以简化我们的web开发，并且降低程序的耦合度

Struts 2以WebWork为核心，XWork是webwork核心

1. **Struts2的安装**

struts2目录结构

apps 该文件夹包含了基于struts2 的示例应用，这些示例应用对于学习者是非常有用的

docs 该文件夹下包含了struts2 相关文档，包括struts2 快速入门、struts2的文档以及API文档等

lib 该文件夹下包含了Struts2框架和核心类库，以及struts2第三方插件类库

src 该文件夹下包含了Struts2框架的全部源代码：

core是struts2的源代码

xwork-core是xwork的源代码

struts2开发最少导入Jar包：到struts2-blank示例中复制

1. **Struts2第一个程序**

web.xml文件中配置核心控制器

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

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

</filter-mapping>

创建一个struts.xml配置文件 ,这个是struts2框架核心配置文件

<package name="default" namespace="/" extends="struts-default">

<action name="hello" class="cn.xinhua.action.HelloAction"

method="fun">

<result >/index.jsp</result>

</action>

</package>

1. **模拟程序**
2. **struts2配置文件加载顺序**

struts2框架要能执行，必须先加载StrutsPrepareAndExecuteFilter.

init\_DefaultProperties(); // [1] ---------- **org/apache/struts2/default.properties :**定义了struts2框架中所有常量

init\_TraditionalXmlConfigurations(); // [2] ---

**struts-default.xml:**配置了interceptor,result

**struts-plugin.xml:**struts2框架中插件的配置文件

**struts.xml**

init\_LegacyStrutsProperties(); // [3] ---

**自定义struts.properties**

init\_CustomConfigurationProviders(); // [5] ----- 自定义配置提供

init\_FilterInitParameters() ; // [6] ----- web.xml

注意：后加载文件中的配置会将先加载文件中的配置覆盖 一般会在 **struts.xml完成配置**

Action配置

struts2结果类型

**在struts2中获取请求参数**

属性驱动

1 、直接将action做一个model,就可以得到请求参数

1. 在action中声明一个model。

原理：通过struts2中的interceptor进行了数据封装.

<interceptor name="params" class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>

模型驱动

1.让action类实现ModelDriven

2.重写getModel方法

3.在action中实例化一个model对象，让getModel方法返回

这个对象。

原理：<interceptor name="modelDriven" class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>

List集合和Map集合的封装

action类:

private List<User> list;

get/set方法

用户名1:<input type="text" name="list[0].username"><br>

密码1:<input type="password" name="list[0].password"><br>

用户名2:<input type="text" name="list[1].username"><br>

密码2:<input type="password" name="list[1].password"><br>

action类:

private Map<String, User> map;

提供get/set

用户名1:<input type="text" name="map[‘a’].username"><br>

密码1:<input type="password" name="map[‘a’].password"><br>

用户名2:<input type="text" name="map[‘b’].username"><br>

密码2:<input type="password" name="map[‘b’].password"><br>

**struts2类型转换**

struts2 内部提供大量类型转换器，用来完成数据类型转换问题.

struts2中的类型转换器根接口是：com.opensymphony.xwork2.conversion.TypeConverter

**自定义类型转换器**

1、创建一个类继承DefaultTypeConverter类的一个子类StrutsTypeConverter

2、重写接口中方法，实现类型转换操作.

public abstract Object convertFromString(Map context, String[] values, Class toClass);

public abstract String convertToString(Map context, Object o);

3、注册类型转换器.

1）针对action

配置文件所在位置以及名称: 在Action类所在包，创建 Action类名-conversion.properties ,

格式 ： 属性名称=类型转换器的全类名

2）针对model

配置文件所在位置以及名称: 在model类所在包，创建 model类名-conversion.properties ,

格式 ： 属性名称=类型转换器的全类名

3）、全局

配置文件所在位置以及名称:在src下创建一个xwork-conversion.properties

格式: 要转换的类型全名=类型转换器的全类名

重点：

如果是自定义类型转换器，要跳转到input视图，在类型转换器中，必须抛出异常才可以，类型转换出现问题，会向input视图跳转

<action name="regist" class="cn.xinhua.web.RegistAction">

<result name="input">/error.jsp</result>

</action>

原理：

<interceptor name="conversionError" class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>：向action中记录出现的问题

<interceptor name="workflow" class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>：用于得到问题，向input视图跳转

可以通过 <s:fieldError/> 标签在页面中得到问题

**struts2的校验**

required (必填校验器,要求被校验的属性值不能为null)

requiredstring (必填字符串校验器,要求被校验的属性值不能为null，并且长度大于0,默认情况下会对字符串去前后空格)

stringlength (字符串长度校验器，要求被校验的属性值必须在指定的范围内，否则校验失败,minLength参数指定最小长度，maxLength参数指定最大长度，trim参数指定校验field之前是否去除字符串前后的空格)

regex (正则表达式校验器，检查被校验的属性值是否匹配一个正则表达式，expression参数指定正则表达式，caseSensitive参数指定进行正则表达式匹配时，是否区分大小写,默认值为true)

int(整数校验器，要求field的整数值必须在指定范围内，min指定最小值，max指定最大值)

double(双精度浮点数校验器,要求field的双精度浮点数必须在指定范围内,min指定最小值,max指定最大值)

fieldexpression (字段OGNL表达式校验器,要求field满足一个ognl表达式，expression参数指定ognl表达式,该逻辑表达式基于ValueStack进行求值,返回true时校验通过，否则不通过)

email(邮件地址校验器，要求如果被校验的属性值非空，则必须是合法的邮件地址)

url(网址校验器,要求如果被校验的属性值非空,则必须是合法的url地址)

date(日期校验器,要求field的日期值必须在指定范围内,min指定最小值,max指定最大值)

注意：如果只对action类中某个方法校验。

action类名-action名称-validation.xml.

**struts2国际化**

1、全局配置方式

需要通过一个常量来声明.

struts.custom.i18n.resources=testmessages,testmessages2

**message.properties在src下**

**<constant name="struts.custom.i18n.resources" value="message">**

**message.properties在cn.xinhua.resource包下**

**<constant name="struts.custom.i18n.resources" value="cn.xinhua.resource.message">**

2、局部配置方式

1）对于特定的action类

与action类在同一个包下创建Action名称.properties，

这个配置文件只对当前action有效。

2）对某个包下所有的action

在指定的包下创建名称:package.properties

3）jsp页面使用某一个properties文件.

<s:i18n name="cn.xinhua.package"></s:i18n>

3、国际化的使用

1）在action类中使用

action类必须要继承ActionSupport类。通过getText(String name)方法就可以获取配置文件中对应名称的值。

2）在校验文件中使用

<message key="名称"/>

3）在jsp页面上使用

如果要从某一个配置文件中获取，通过name属性来指定， 包名.配置文件名称

<s:i18n name="cn.xinhua.package">

<s:text name="名称">

</s:i18n>

如果没有使用<s:i18n name="">来指定，会从全局配置文件中获取。