Eclipse 注释模板

Types:

/\*\*

\* **@ClassName**: ${type\_name}

\* **@Description**: ${todo}(这里用一句话描述这个类的作用)

\* **@author**: 章征武【orlando】

\* **@date**: ${date} ${time}

\* **@tel**: 17520490925

\* **@email**: zhangzw368319@163.com

\* ${tags}

\*/

Methods

/\*\*

\* **@Title**: ${enclosing\_method}

\* **@Description**: ${todo}(这里用一句话描述这个方法的作用)

\* **@param** ${tags} 参数

\* **@return** ${return\_type} 返回类型

\* **@author**: 章征武【orlando】

\* **@date**: ${date} ${time}

\* **@tel**: 17520490925

\* **@email**: zhangzw368319@163.com

\* **@throws**

\*/

String path1=request.getServletContext().getRealPath("/");

// D:\Prosay\eclipse\_workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\20180915WebProject\

String path2 = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+request.getContextPath()+"/";

// http://localhost:28080/20180919WebProject/

String path3 = request.getServletContext().getContextPath();

// /20180919WebProject

String path4 = request.getRequestURI();

// /20180919WebProject/index.jsp

String path5 = request.getRequestURI().substring(request.getContextPath().length());

// /index.jsp

String path6 = request.getContextPath();

// /20180919WebProject

封装的思路：

封装的时候需要考虑的问题可以分成三个部分：

第一部分：不变的代码。（需要封装，通用代码）

第二部分：不确定的代码。（无需封装）

第三部分：有规律变化的代码。（找到规律，完成封装）

考虑一个方法尽量的完成最小的一个工作！

有规律变化的数据，找到规律，相同的部分封装，不同的部分采用参数传递。

参数的传递，通常采用如下两种方式：

1、构造函数，不会发生改变的参数的传递用构造函数

2、set方法，很有可能发生改变的参数的传递用set方法；参数的数据类型都一致导致无法重载的时候用set方法

如果你的MySQL无法通过IP地址访问的话，是由于MySQL默认情况下只接收本地访问方式。

grant all privileges on \*.\* to 'root'@'%' identified by '';

//FIXME 命名函数和匿名函数的封装

DAO的封装：

构建查询语句

执行查询操作（通过JDBC完成，调用DbUtil完成，将来可能会见到DBCP DabaBaseConnectionPool、DataSource）

封装数据

技术人员的四个阶段？

1、觉得什么都会

2、觉得什么都不会

3、不知道自己会了

4、知道自己会了

JSP:Java Server Page 服务器上的页面

B/S : Blower/Server

C/S : Client/Server

客户端：向服务器提交请求，并且接收服务器响应回来的结果的程序。

服务器：完成向客户端提供数据、功能的请求的处理，并将处理的结果响应回对应的客户端的程序。

浏览器：网页的查看工具。

Tomcat

安装格式：exe Windows

压缩格式：zip(tar.gz/rpm) Linux

目录：

bin 程序文件和批处理文件

conf 配置文件

lib jar包

logs 日志

temp tommcat运行过程中的临时文件

webapps Web应用程序

work Web应用程序运行过程中产生的临时文件

本地访问：file://

远程访问：http://

jsp不支持本地访问

端口号:8080

startup

shutdown

查ip

ipconfig /all

查端口

netstat -an

netstat -abn --带进程

高逼格启动

cmd

d:

cd mysql\bin

mysqld --console

高逼格启动查询

cmd

d:

cd mysql

--查系统的服务  sc query ""

netstat -an   (all network) --查询mysql服务是否启动的操作

Web应用程序在Tomcat的webapps文件中，每一个Web应用程序是一个文件夹。

每一个Web应用程序都有一个关键的子文件夹WEB-INF。是一个隐藏文件夹，通常无法通过浏览器直接访问。

在WEB-INF文件夹中有一个重要文件web.xml，是Web应用程序的配置文件。

URL 统一资源定位符

协议://计算机名:端口号/Web应用程序名/资源路径/资源名称

临时设置 path 环境：一般用于在别人电脑上启动类似tomcat软件又不影响电脑原有文件。

cmd 环境下，进到jdk 的安装目录下

set JAVA\_HOME = %CD% CD表示当下路径

set PATH = %JAVA\_HOME%\bin;%PAHT%

<%-- 导入java 类 --%>

<%@ page import=*"com.orlando.action.TestAction"* %>

<%--JSP注释 --%>

<%--JSP文件保存时使用的字符编码 --%>

<%@ page pageEncoding=*"UTF-8"* %>

<%--JSP页面在编译的时候的字符编码，是给java的编译器看的 --%>

<%@ page contentType=*"text/html; charset=UTF-8"* %>

<!DOCTYPE HTML>

<html>

<head>

<title>20180822WebProJect</title>

<!-- HTML的注释 -->

<!-- 网页的字符编码设置，是给浏览器看的 -->

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

</head>

<body>

<%

String str = TestAction.run();

out.print(str); /\* 通过out.print在网页上输出 \*/

%>

</body>

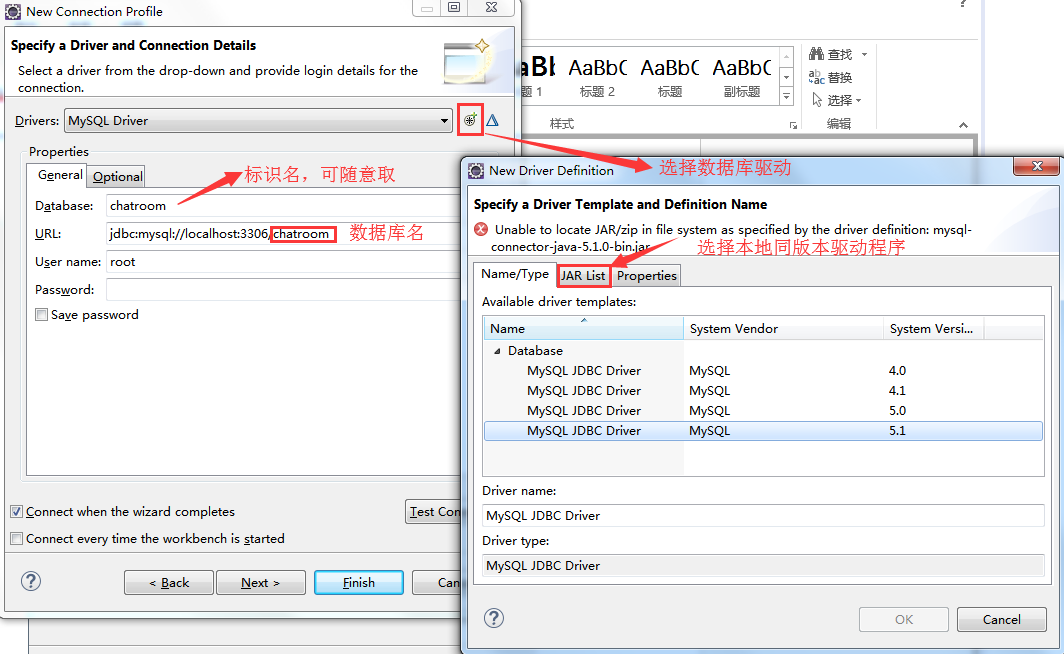
<footer></footer>

</html>

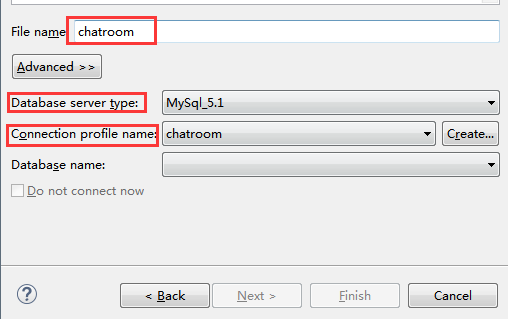
如何在eclipse中设置数据库操作？

Window 🡪 Show View 🡪 Data Source Explorer 🡪 Database Connections 右键 🡪 New...

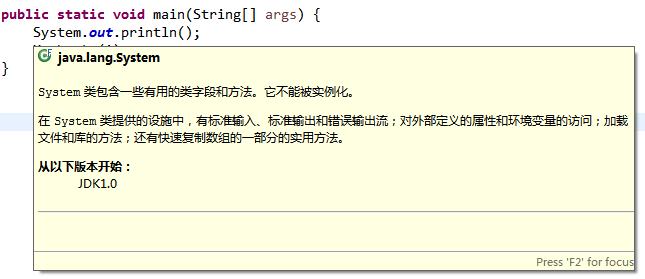
🡪选择需要连接的数据库🡪取个易辨识的名字🡪next🡪 选择对应数据库的连接驱动，输入连接信息🡪finish。



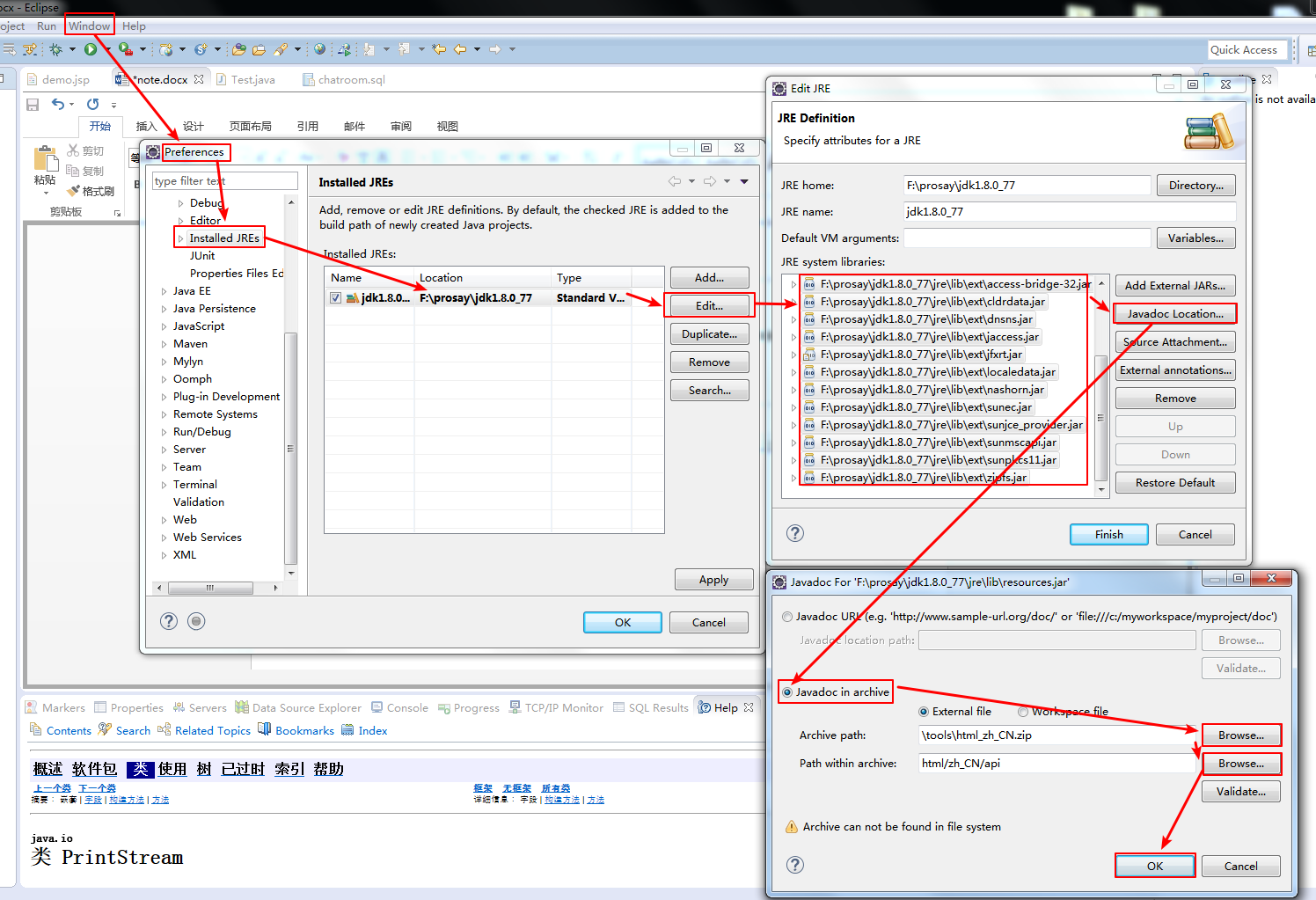
在非自动加载区新建sql文件，填写信息后点击 Finish，即可。



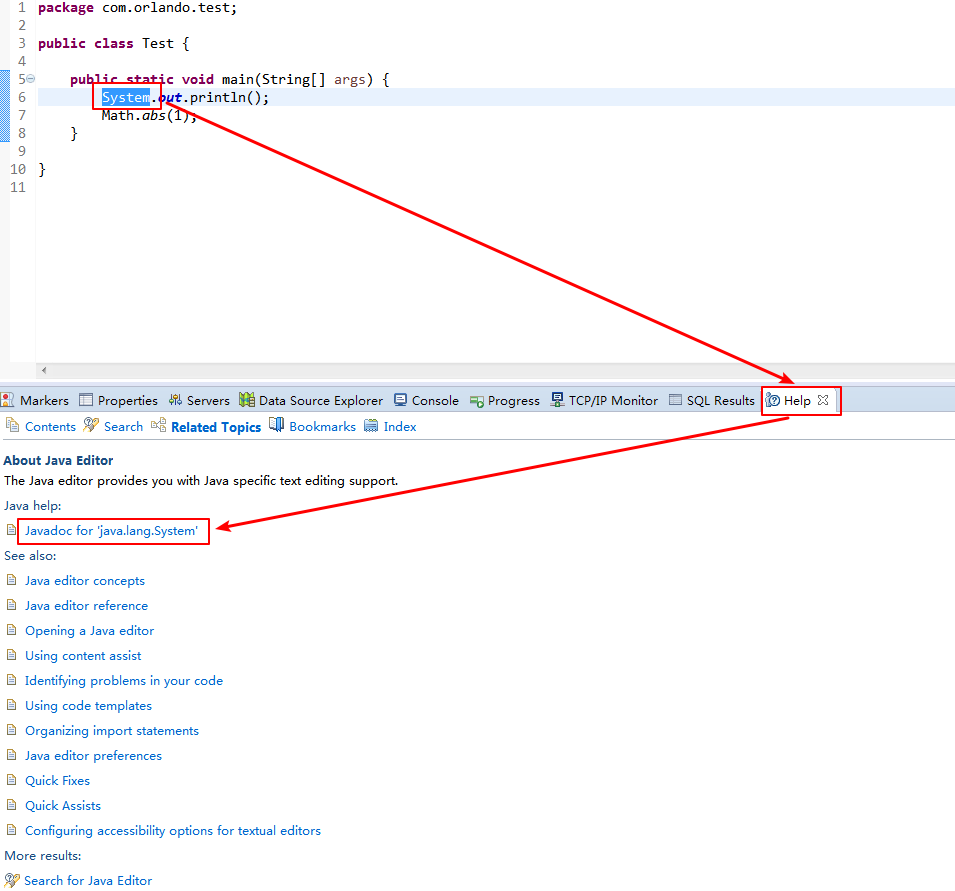
如何在eclipse中显示中文解释？



如下图：



也可以选中你需要查看的某个类，按F1键盘进行查看。



JSP的七大组成部分

1、HTML

标准的HTML语言的标签，包括CSS和JavaScript，统称为HTML。

2、注释

&lt; <

&gt; >

HTML的注释和JSP的注释

HTML: <!-- -->

JSP: <%-- --%>

区别：

所有的注释都会被忽略，区别在于：

HTML注释是浏览器会忽略。

JSP注释Web容器就已经忽略了。

通俗的来说，HTML注释可以在网页的源代码中看到，JSP注释则看不到。

<!-- 这个是HTML的注释 -->

<%-- 这个是JSP的注释 --%>

<%--

多行注释也可以

--%>

3、JSP代码段

<% Java代码; %>

JSP的代码段，Java的大多数语句可以放在代码段中，语法规则按照标准的Java语法规则。

什么语句不能放？

定义语句不能放。不管是定义什么都不行（类，方法，属性统统不行）。

举例：

<%

A a=**new** A();

**int** number=10;//局部变量，在方法中定义的变量，仅仅在方法中有效

a.sayHello();

System.out.println("userName="+a.getUserName());

System.out.println("number="+number);

%>

4、JSP表达式

<%= Java代码 %>

JSP表达式，可以放属性和方法的调用语句。

JSP表达式有一个特别的地方。Java语句的结尾，不可以有“;”，也不可以有多句语句！！！

JSP表达式的功能，其实就是输出语句。输出属性的值或者是方法的返回值。输出的位置是页面。

举例：

<%="number="+number%>

<%="userName is :"+a.getUserName()%>

5、JSP声明

<%! Java代码; %>

JSP的声明，可以放的仅仅只能是定义语句，语法规则按照标准的Java语法规则。

除了定义语句，其他的语句一概不能放。

JSP当中不可以定义抽象的东西！！！

举例：

<%!

**public** **class** A {//定义类

**private** String userName="zhangsan";//定义属性，也可以叫做成员变量

//尽量避免使用首字母仅有一个小写字母的标识符uName，u\_Name，这种变量名在EL表达式中可能会有问题。

**public** **void** sayHello() {//定义方法

System.out.println(**this**.userName+"说：你好，JSP!");

}

**public** String getUserName() {

**return** **this**.userName;

}

}

//System.out.println("你好，JSP!");//错误的写法，非定义语句不能出现在JSP声明中！

%>

6、JSP指令

完成给JSP下一些命令的功能。指令必须放在JSP开头。同一个指令可以采用多行书写。也可以一行书写，一行书写的时候，多个属性使用空格隔开。

常见的指令：

page指令，主要完成JSP的一些基本设置工作。比如说，可以指定JSP的语言language属性，页面的字符编码pageEncoding属性，页面内容的字符编码contentType属性，导入类import属性，EL表达式支持的属性，指定错误页面的属性。

include指令，包含页面。在一个JSP页面中，可以引入另外一个JSP。

// TODO 会话跟踪时候补充include指令

taglib指令，标签库指令。在JSP页面当中，引用标签库。

// FIXME JSTL标准标签库时候补充taglib指令

7、JSP动作

<jsp:include page=*""*></jsp:include> 包含

<jsp:forward page=*""*></jsp:forward> 转发

// TODO 会话跟踪时候补充redirect、forward、include动作。

JSP标准动作，看起来有点像HTML的标签，实际上也有点像JSTL的标签库。动作实际上是标签

JSP的工作原理：

编辑源码->部署至服务器->启动服务器->通过浏览器访问

其实JSP的工作原理中有很大一部分工作程序员并未参与。而是由Web容器自动完成的。

Tomcat(Web容器在JSP的工作中完成很大一部分工作)，具体如下：

1、查询并加载JSP页面，找不到资源直接报404的错误代码。

2、转译:翻译（将JSP文件翻译成了java文件）。

3、编译:（将java文件编译成了class文件）。

4、执行:（运行class文件，文件执行的结果就是一个完整HTML页面）

（所谓的动态网页，我们在JavaWeb课程中，实际上是指的内容动态，在这里Java程序可以从数据库中获取数据，所以页面的内容会根据数据库的情况发生改变）

5、响应:（将生成的结果HTML页面传输到浏览器）

接下来的工作由浏览器接班，负责解析HTML页面，并采用图形化的方式显示出来。

// TODO 将要完成的工作，主要是帮助我们记住什么地方的工作未完成

// FIXME 修复我，主要是保住我们记住什么地方有可能有问题未处理，FIXME优先级高一点，会排在前面。

在eclipse中配置的tomcat，运行的时候配置文件存放在哪里？

String path = request.getServletContext().getRealPath("/");

eclipse的工作空间：F:\prosay\eclipse\_workspace

eclipse插件的配置信息：\.metadata\.plugins

Web容器的插件的配置信息：\org.eclipse.wst.server.core

对于不同的Web项目的服务器配置信息：\tmpX

相当于tomcat的webapps子文件夹：\wtpwebapps

项目名称：\20180825WebProject\

index.jsp 🡪 index\_jsp.java 🡪 index\_jsp.class

HTML的代码，在容器转译过程中，相当于把HTML代码变成了字符串，使用out对象的write方法输出，在前面加上了out.write("，在后面加上了");

<body>

out.write("<body>\r\n");

JSP注释，在容器转译过程中，就已经被Web容器的转译组件忽略了。所以在java源代码中找不到。

<%-- JSP 注释 --%>

HTML注释，在容器转译过程中，当作HTML代码处理。

<!-- HTML注释 -->

out.write("\t<!-- HTML注释 -->\r\n");

JSP代码段中的Java代码，在容器的转译过程中，直接复制到了转译以后的java文件的\_jspservice方法中。所以也就解释了，代码段中Java代码为什么不能有定义语句。

<%

path = request.getServletContext().getRealPath("/");

%>

path = request.getServletContext().getRealPath("/");

JSP表达式，在容器转译过程中，相当于首先完成表达式的计算，其次将表达式计算的结果，使用out对象，直接输出。前面加上了out.print(，后面加上了);

<%=path%>

out.print(path);

所以，这个时候应该可以理解，为什么JSP表达式的结尾不能有;，也不可以有多于语句。其实也可以通俗的理解，JSP表达式中的内容，作为out.print()方法的参数。

JSP声明中的Java代码，在容器转译过程中，直接将代码转译到了类中。所以也就解释了，声明中的Java代码为什么不能有非定义语句（方法调用的语句）。

<%!

String path = **null**;

%>

String path = null;

会话跟踪

什么是会话？(session对象)

用专业的说法来说，浏览器与服务器之间，在某一个时间段内的，多次请求和响应的过程，我们叫做会话。

所谓的会话实际上可以理解为一组请求/响应的过程。

在某一个时间段内如何解释呢？

指浏览器第一次向服务器发送消息开始，这个会话开启，会话终止的时间，是指服务器完成最后一次响应后延迟XX时间以后，终止会话。

会话的实例化/销毁(开启和终止)，是由Web容器(Tomcat)完成的。

什么是请求/响应？（request对象和response对象）

简单的理解请求和响应的机制，就是服务器和浏览器之间的消息传递。以方向来区分请求和响应。

浏览器向服务器发送的消息，叫做请求。服务器向浏览器发送的消息，叫做响应。

提交数据的两种方式

查询字符串

在浏览器的地址栏上输入的URL后面加上一个 ?userName=张三 可以给JSP页面传递参数。

查询字符串：?字符串

查询字符串是一个标准的字符串。可以传递中文，但建议不要传递中文。因为URL的编码与否是由客户端浏览器决定的。

查询字符串的规则：参数名称=参数值构成的。

多个参数使用 & 符号隔开。

通过表单提交

通过表单提交的时候，如果采用get方式提交，其实就是以查询字符串的方式提交。

如果采用post方式提交，可以传递大量文字信息，并且信息不会显示在浏览器地址栏上。安全性较高，并且没有字符数的限制。

但是需要注意一点，采用post方式提交的数据，需要做字符编码处理。

<%

// 处理字符编码问题

request.setCharacterEncoding("utf-8");

response.setCharacterEncoding("utf-8");

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

%>

页面数据传递？

通过requset传递

<jsp:forward page="" ></jsp:forward> 转发

request.setAttribute("username",uname);

request.getAttribute("username");

JSP/Servlet 中的页面跳转

通过Java代码完成转发

RequestDispatcher rd = request.getRequestDispatcher(".jsp");

rd.forward("请求对象","响应对象");

rd.forward(request,response);

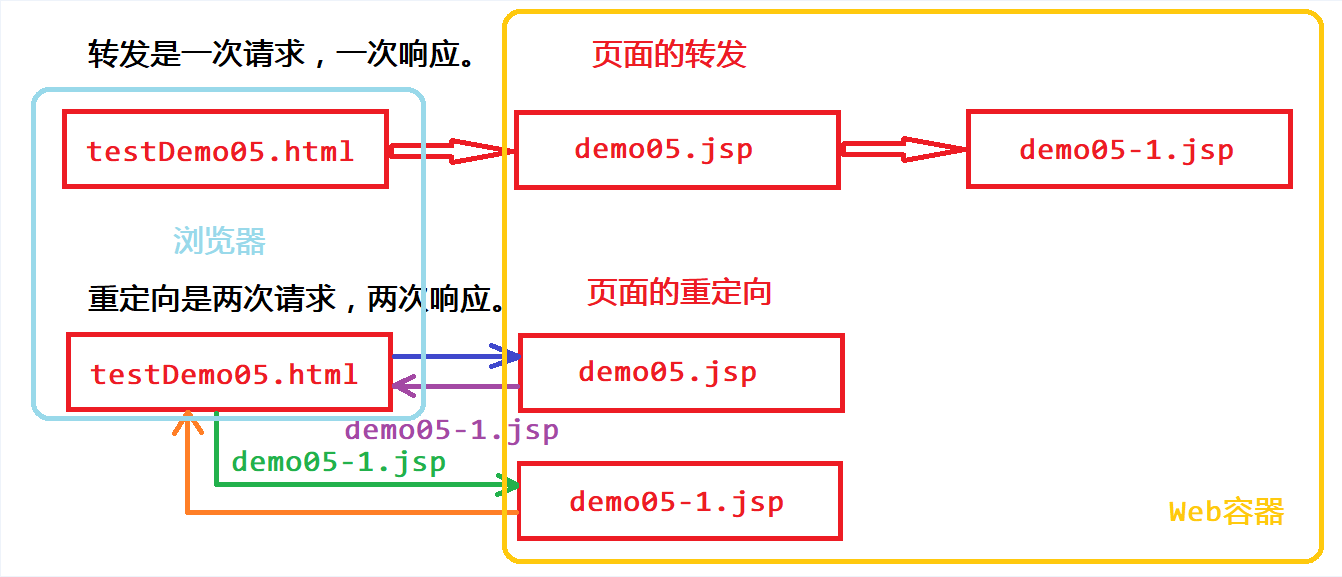
通过Java代码完成重定向

response.sendRedirect(".jsp");

使用 session 来辅助完成数据的传递

session.setAttribute("username",uname);

session.getAttribute("username");



通过查询字符串传递

response.sendRedirect(".jsp?xxx="+xxx);

request.getParameter("xxx");

servlet默认编码： iso8859-1

String str = ".jsp?xxx="+xxx;

URLEncoder.encode(Str,"UTF-8"); --编码

URLDecoder.decode(Str,"UTF-8"); --解码

JSP中的四大作用域

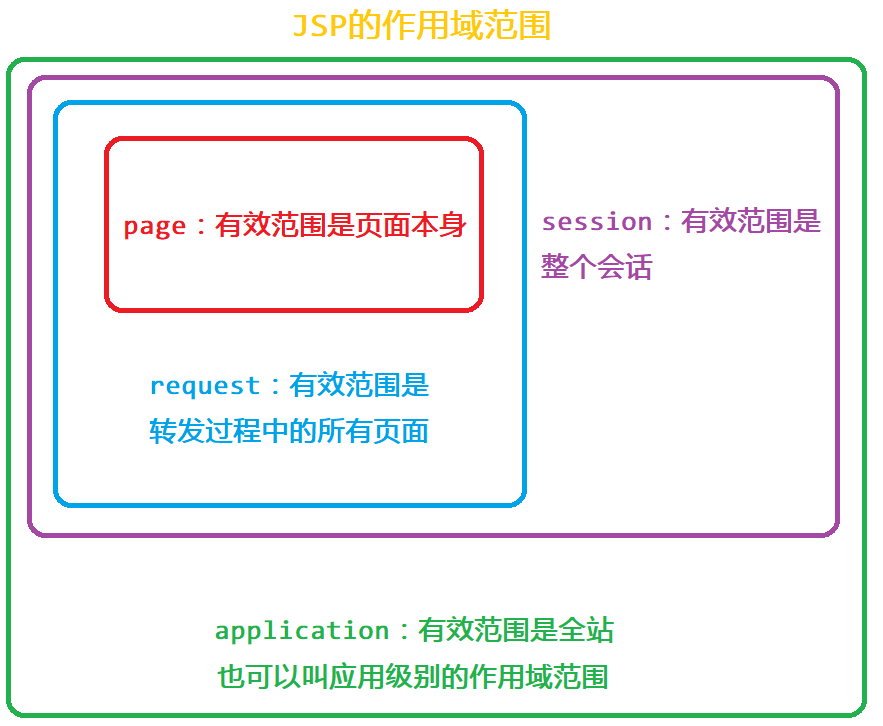
page(页面级别)、request(请求级别)、session(会话)、application(应用)

page:有效范围是页面的本身

request:有效范围是转发过程中的所有页面

session:会话的有效范围是整个会话

application:有效的范围是全站，也叫做应用级别的作用域范围



作用域的原则：能小则小

能用小作用域范围解决的问题，不用大作用域范围。

能不用作用域解决的问题，就不用作用域。

转发和重定向

相同点：

都是完成页面跳转的功能。

结果都是后面的目标页面完成的功能产生的结果。

不同点：

转发是 一次请求，一次响应。

重定向是两次请求，两次响应。

转发以后，在页面的地址栏上显示的是前一个页面地址。

重定向以后，在页面地址栏上，显示的是后面一个地址。

转发只能转发同一个网站的页面。

重定向可以重定向到其他网站。

JSP的隐式对象

page request session application request out

pageContext

config

Exception

不需要通过实例化(new)的操作既可以获取到的对象，叫做隐式对象。

隐式对象都是由 web容器负责实例化的。所以session 的创建和销毁都是由容器来做的。

但是在Servlet 中有一点小问题，Servlet中sesson的创建和销毁容器是不负责维护的。

session.invalidate

Cookie

保存在客户端浏览器的临时文件夹中,不占用服务器内存

Cookie可以被客户端关闭

Cookie不安全

Cookie保存的是文本信息，无法保存过大的内容，也无法保存二进制的内容

JSP的分类：四大类

输入输出有关： request(HttpServletRequest),response(HttpServletResponse),out(PrintWriter)

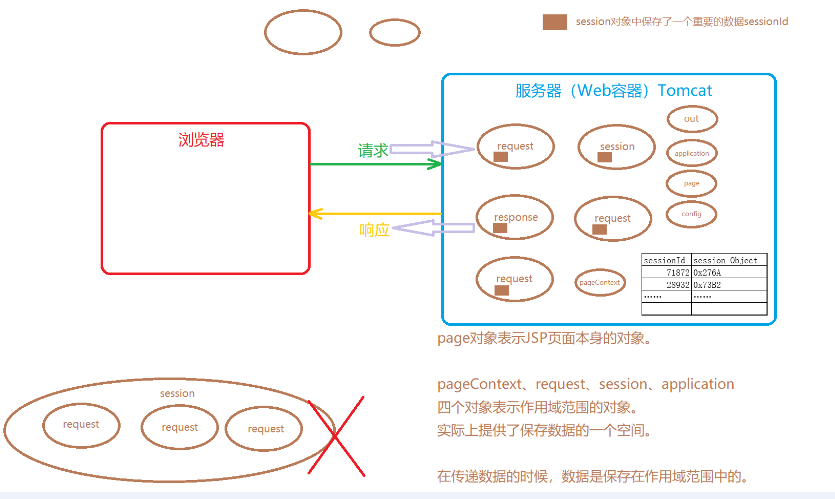
作用域有关：pageContext(上下文),request,session(HttpSession),application(ServletContext)

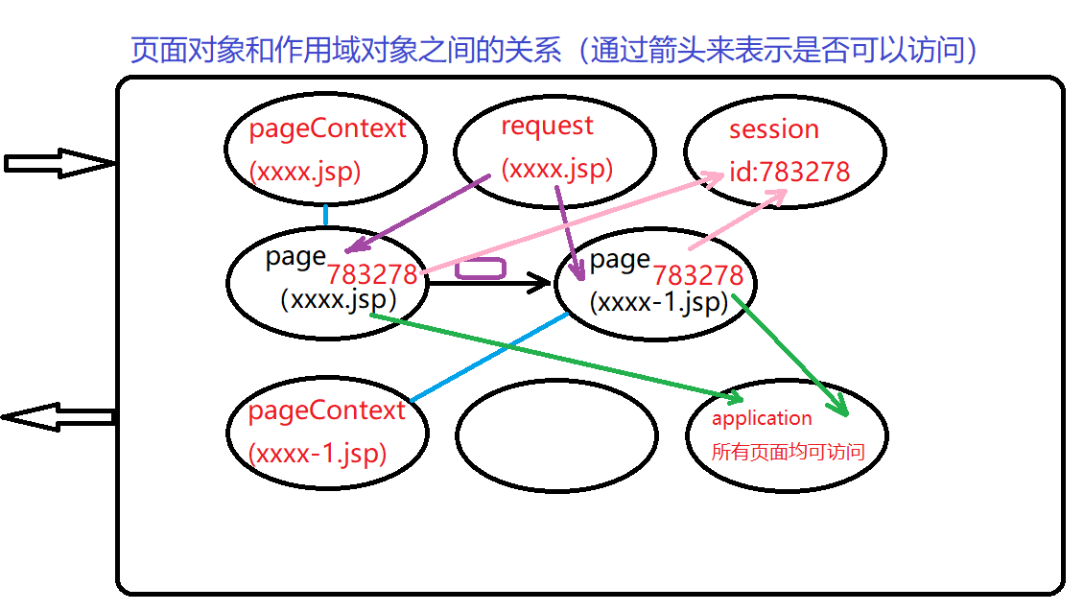
配置有关：page(表示JSP页面本身的对象),config(ServletConfig)(表示页面本身Servlet的配置信息)

异常有关：exception(必须页面指令为 isErrorPage=”true”的时候才有效)

Exception.printStackTrace()







JSTL: JavaServer Pages Standard Tag Library/JSP Standard Tag Library/JSP标准标签库

JSTL的分类

核心标签库：c core

国际化标签库：功能比较小 fmt

SQL标签库：违背了mvc设计模式和三层体系结构，抛弃 sql

XML标签库：JSON取代 xml

//FIXME 自定义标签库

标签库的使用

1、导入 jar 包

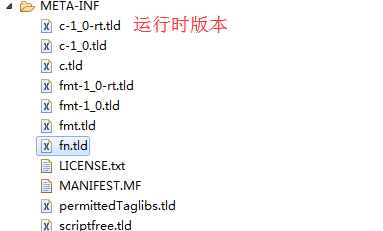
2、标签库的引入： taglib 指令 uri http://java.sun.com/jstl/core\_rt

prefix 前缀 c

<%@ taglib uri= prefix= %>

标签库的组成：由若干个标签处理程序.class 和 标签描述文件 .tld 组成

每一个标准标签库都有三个版本的tld 描述文件



3、使用 <c:标签名> </c:标签名>

标签库获取数据通常是通过作用域来传递

JSTL将JAVA代码转变成了HTML代码

1、<c:forEach></c:forEach>

<%

pageContext.setAttribute(“mymsg”,msg);

%>

<c:forEach items=”${pageScope.mymsg}” var=li >

${li}

</c:forEach>

2、<c:if></c:if>

<c:if test=”{}” >

</c:if>

<c:if test=”{not …}” >

</c:if>

test 后面只能跟 boolean 值，不支持else，所以要另外写一个 c:if 标签

相等 eq and or empty not ! --EL表达式详述

3、<c:choose></c:choose>

<c:choose>

<c:when test >

</c:when>

<c:when test >

</c:when>

<c:otherwise>

</c:otherwise>

</c:choose>

4、<c:forTokens></c:forTokens> 相当于 String 类型的 split()

<c:forTokens items=”” delims=”|” var >

</c:forTokens>

5、<c:redirect></c:redirect> 重定向

<c:redirect url=””>

</c:redirect>

6、<c:param></c:param> 从查询字符串中获取数据(?xxx=xxx&xxx=xxx)

EL表达式

Expression Language(描述性语言)

作用：用来取代JSP表达式<%= %>

原因：JSP表达式容易造成语法规则的混乱

处理null

对页面非常友好，表达式处理过程有问题也不会抛出异常而导致页面报错，而是返回空字符串

语法格式：${}

功能：完成运算和输出的功能

【注意】EL表达式不能控制程序流程，也就是没有循环结构，流程结构，用JSTL配合完成

使用：通过“.” 或者 “[]” 来完成对象的属性的调用，数组和集合中元素的调用

EL内置对象

与请求参数相关：从查询字符串后面获取信息

param paramValues

与作用域相关

pageScope requestScope sessionScope applicationScope

与请求头相关

header headerValues

与初始化参数相关

initParam

与cookie相关

Cookie

与pageContext相关

pageContext(他的功能是可以获取到JSP中的隐式对象)

EL运算符号

算术运算符

+、-、\*、/、%(mod)

比较运算符

== (eq)、!= (ne)、< (lt)、> (gt)、 <= (le)、 >= (ge)

逻辑运算符

&& (and)、 || (or)、 ! (not)

其他运算符

empty(空值的判断)

三元运算符

() 提升运算优先级

通过EL表达式获取对象的属性

<%

UserInfo userInfo=new UserInfo();

userInfo.setUserName("zhangsan");

userInfo.setUserPwd("password");

userInfo.setUserNick("张三丰");

pageContext.setAttribute("user", userInfo);

%>

<table>

<tr>

<th>账号：</th>

<td>${pageScope.user.userName}</td>

<%-- 相当于调用userName属性对应的get方法，也就是调用：getUserName() --%>

</tr>

<tr>

<th>密码：</th>

<td>${pageScope.user.userPwd}</td>

</tr>

<tr>

<th>昵称：</th>

<td>${pageScope.user["userNick"]}</td>

</tr>

</table>

通过EL表达式获取数组的元素

<%

String[] array=new String[10];

array[0]="Hello";

array[1]="Java";

array[2]="JavaScript";

array[3]="JSP";

array[4]="MySQL";

array[5]="HTML";

array[6]="CSS";

array[7]="Hibernate";

array[8]="Spring";

array[9]="MyBatis";

pageContext.setAttribute("myArray", array);

%>

<p style="color:red;">${pageScope.myArray[0]}</p>

<%

UserInfo[] allStu=new UserInfo[3];

UserInfo stu1=new UserInfo();

stu1.setUserName("张三");

stu1.setUserPwd("111111");

stu1.setUserNick("张三丰");

UserInfo stu2=new UserInfo();

stu2.setUserName("李四");

stu2.setUserPwd("222222");

stu2.setUserNick("菊花哥");

UserInfo stu3=new UserInfo();

stu3.setUserName("王五");

stu3.setUserPwd("333333");

stu3.setUserNick("隔壁老王");

allStu[0]=stu1;

allStu[1]=stu2;

allStu[2]=stu3;

pageContext.setAttribute("stuArray", allStu);

%>

<p style="color:blue;">

${stuArray[0].userNick}

${stuArray[1].userNick}

${stuArray[2]["userNick"]}

</p>

通过EL表达式获取集合的元素

<%

List<UserInfo> stuList=new ArrayList<UserInfo>();

stuList.add(stu1);

stuList.add(stu2);

stuList.add(stu3);

UserInfo stu4=new UserInfo();

stu4.setUserName("bingbing");

stu4.setUserPwd("444444");

stu4.setUserNick("微软小冰");

stuList.add(stu4);

request.setAttribute("stuArray", stuList);

%>

<p style="color:yellowgreen;">${requestScope.stuArray[3]["userNick"]}</p>

<table>

<tr>

<th>账号</th>

<th>密码</th>

<th>昵称</th>

</tr>

<c:forEach items="${requestScope.stuArray}" var="stuObj">

<tr>

<td>${stuObj.userName}</td>

<td>${stuObj.userPwd}</td>

<td>${stuObj.userNick}</td>

</tr>

</c:forEach>

</table>

在EL表达式中，作用域范围可以省略不写，如果省略的话，EL将默认自动从最小的作用域范围中查找属性。

先找pageScope，如果没找到接下来找requestScope，再没找到找sessionScope，最后到applicationScope中查找。

如果全部没有找到，返回空字符串。

EL 获取属性时会自动调用 private 属性的getXxx方法，原理是 将属性的首字母大写，前面自动添加get

EL中的param对象和paramValues对象

<a href="demo08.jsp?id=你好中国&type=动作&type=灾难&type=喜剧">

查询字符串中的属性id的值是：${param.id}

查询字符串中的属性type的值是：

<c:forEach items="${paramValues.type}" var="typeStr">

${typeStr}

</c:forEach>

Servlet

运行在服务器上的一个处理请求和响应机制的动态页面的Java程序。

extends GenericServlet service() 不分请求类型

extends HttpServlet doGet() doPost()

init()

init(ServletConfig config)

destroy()

Servlet的生命周期

实例化：Tomcat

初始化

服务

销毁

不可用：JVM，属于垃圾回收的机制

Servlet生命周期的方法

1、Servlet对象是由Web容器创建的！

2、Web容器在启动的时候并没有创建Servlet的对象！（什么时候创建的？？）

3、Web容易在首次接收到访问该Servlet的请求的时候，创建Servlet对象的。

4、Servlet对象创建完成以后，将会自动调用init()方法完成初始化操作。

5、完成初始化以后，通过service()方法处理请求。

6、处理完请求以后，该对象不会销毁，处于等待请求状态，等待下一次的请求。

下一次请求的时候，直接调用service()方法处理请求。

7、当Web容器资源紧张的时候，即将垃圾回收时，

会根据Servlet对象有多长时间没有接收请求来判断，

较长时间没有访问的Servlet对象将被垃圾回收。

或者是Web容器shutdown的时候，销毁Servlet对象。

8、可以通过配置的方式，让Servlet在容器启动的时候及完成对象的创建。Servlet的预加载。

9、生命周期的初始化参数。通过注解或者xml配置文件配置。

通过ServletConfig的对象调用getInitParameter()获取已经配置好的初始化参数。

Servlet 配置方式一：

Xml

<servlet>

<servlet-name></servlet-name>

<servlet-class></servlet-class>

</servlet>

<servlet-mapping>

<servlet-name></servlet-name>

<url-pattern></url-pattern> --虚拟路径/虚拟文件

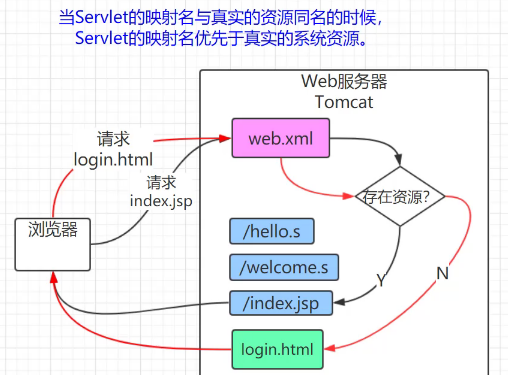
</servlet-mapping>

通配符 <url-pattern>

虚拟文件夹 /servlet.\*

虚拟路径 \*.do -- 注意前面不加 /

当虚拟servlet和真实文件同时存在时，访问后会以Servlet优先



Servlet 配置方式二：注解

注解，泛型，自动装箱拆箱，反射 JDK 1.5 版本之后出现

功能：告知编译器，接下来的类、方法、属性的作用

@WebServlet(name="testservlet",urlPatterns="/test" )

**public** **class** TestServlet **extends** GenericServlet {

//FIXME

}

Servlet 声明周期的方法

当web容器启动时并没有创建Servlet的对象。

第一次请求

构造方法

init()

service()

第二次请求

service()

destroy()

当web容器资源紧张，即将垃圾回收时，会根据Servlet对象有多长时间没有接收请求来判断。或者web容器shutdown时候。

Servlet 预加载：可以通过配置的方式，让Servlet在容器启动时就创建Servlet对象

loadOnStartup = 1

注解方式

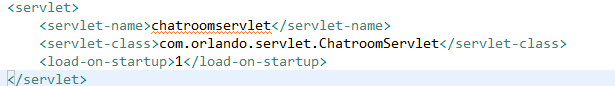
@WebServlet(name="testservlet",urlPatterns="/test",loadOnStartup = 1 )

**public** **class** TestServlet **implements** Servlet {

//FIXME

}

Xml方式



注解初始化参数

@WebServlet(name="testservlet",urlPatterns="/test",loadOnStartup = 1,

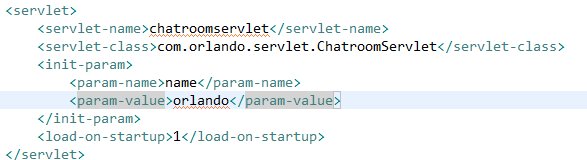
initParams = { @WebInitParam(name="name",value="orlando"),@WebInitParam(name="nick",value="orlando") } )

**public** **class** TestServlet **implements** Servlet {

//FIXME

}

Xml初始化参数



获取初始化参数

@Override

**public** **void** init(ServletConfig config) **throws** ServletException {

System.***out***.println("init...");

String name = config.getInitParameter("name");

System.***out***.println(name);

}

传值的方式：

1、表单传值（页面提交数据）

2、URL重写（传递较少的非中文信息）

3、作用域传值（Servlet转发）

在Servlet中获取Session对象

request.getSession();

request.getSession(boolean);

为true的时候：

如果第一次访问，获取不到session对象，容器会重新创建一个session对象。

如果不是第一次访问，就将之前创建好的session对象获取到。

为false的时候：

如果第一次访问，获取不到session对象，容器将返回null。

如果不是第一次访问，就将之前创建好的session对象获取到。

不带参数的方法，相当于getSession(true);

session对象的类型：HttpSession

Servlet中完成的工作：

1、获取用户提交的数据。（某些情况下可能缺失）

2、调用业务方法，完成业务操作。（暂时还没有业务类和业务方法，所以就直接调用数据访问类来完成数据的查询）

3、将结果数据保存在作用域范围内。（某些情况下可能缺失）至于作用域范围的选择，就小原则。

4、完成页面跳转。

标题 <caption></caption>

下角标<tfoot></tfoot>

访问WEB-INF,只能通过转发方式

url重写 ?xxx=xxx 通过 param 获取

API的学习（五个什么）

1. 什么类（接口）提供的
2. 方法做什么工作
3. 需要什么参数
4. 参数什么类型
5. 返回什么结果

<%@ include file=”” %> 指令包含，静态包含

将include包含的文件完整的拷贝到主页面，在一并执行

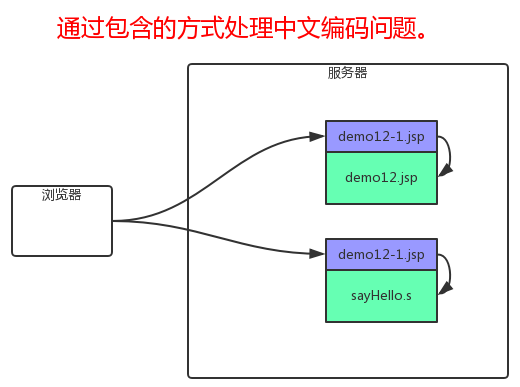
<jsp:include page=”” ></jsp:include> 动作包含，动态包含

将include包含的文件先执行，执行结果返回到主页面

Servlet中的包含

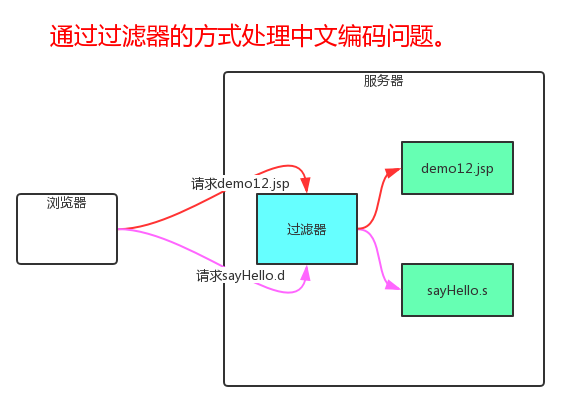
Request.getRequestDispatcher(“”).include(request,response); 动作包含

Servlet中没有静态包含



过滤器

其实是在我们访问的资源前面加一个收费站。



写一个过滤器的步骤：

1、就是一个普通的Java类，但是需要实现一个接口。

2、需要通过web.xml配置。五个标签以及标签的嵌套关系。

3、也可以通过注解的方式配置。

Listener

Servlet常见的监听器 8 个

监听作用域的事件

上下文

ServletContextListener

ServletContextAttributeListener

请求

ServletRequestListener

ServletRequestAttributeListener

会话

HttpSessionListener

HttpSessionAttributeListener

HttpSessionBindingListener 监听请求绑定会话的事件

HttpSessionActivationListener 会话激活相关的事件

Servlet监听器的事件 6 个

Listener监听

Event事件 事件源 EventSource

handler处理

delegate委托

上下文

ServletContextEvent

ServletContextAttributeEvent

请求

ServletRequestEvent

ServletRequestAttributeEvent

会话

HttpSessionEvent

HttpSessionBindingEvent

监听/事件/事件源/处理/（委托）

Listener/Event/EventSource/Handler/（Delegate）

测试结论：

1、Listener 运行早于 Filter

2、HttpSessionBindingListener valueBound()

HttpSessionBindingListener必须实例化后放入某一个session中，才可以进行监听。从监听范围上比较，HttpSessionListener设置一次就可以监听所有session，HttpSessionBindingListener通常都是一对一的。正是这种区别成就了HttpSessionBindingListener的优势，我们可以让每个listener对应一个loginmsg，这样就不需要每次再去session中读取loginmsg，进一步可以将所有操作在线列表的代码都移入listener，更容易维护。

session.setAttribute("test4", new TestHttpSessionBindingListener(loginmsg));

HttpSessionBindingListener valueUnbound()

valueUnbound的触发条件是以下三种情况：

a.执行session.invalidate()时。

b.session超时，自动销毁时。

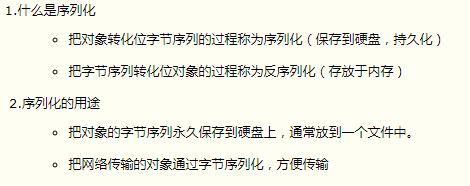
c.执行session.setAttribute("test4", "其他对象");

或session.removeAttribute("test4");将listener从session中删除时。

因此，只要不将listener从session中删除，就可以监听到session的销毁。

参考网址：<https://www.cnblogs.com/siv8/p/5904105.html>

3、HttpSessionActivationListener 监听session对象的序列化与反序列化



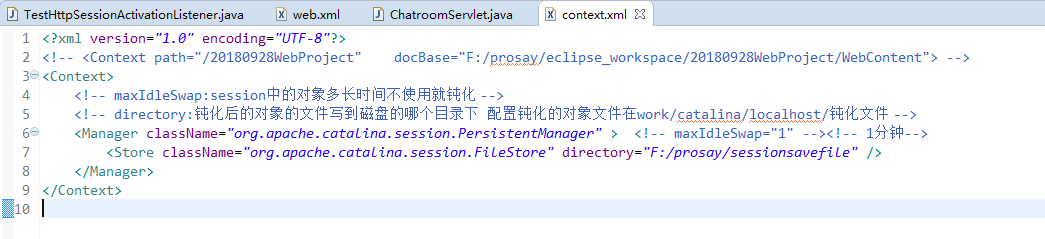
使用场景一:

最常见的是Web服务器中的Session对象，当有10万用户并发访问，就有可能出现10万个Session对象，内存可能吃不消，于是Web容器就会把一些seesion先序列化到硬盘中，等要用了，再把保存在硬盘中的对象还原到内存中。

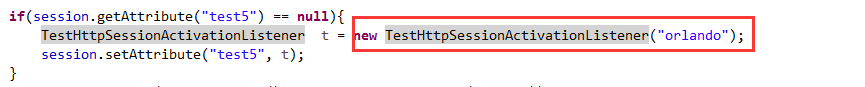
使用场景二:

在用户访问的时候，假如服务器突然关闭了，这个时候，用户的session就不存在了，假如是购物网站，也就相当于，用户好不容易选好的物品，刚刚添加到购物车，结果，因为服务器的突然关闭一下，什么都没了，这样很不好，于是我们就需要实现会话的持久化。

配置：



调用：



sessionWillPassivate()

序列化操作会在session失效时自动触发,当服务宕机时，保存到硬盘了

sessionDidActivate()

当服务在次启动时，自动反序列化拿session中的指定对象，从硬盘读取并活化了

手动注销驱动



Ajax

异步通信的js and xml

同步：电话

异步：短信

单工：同一时间可以同时收发

双工：同一时间只能发送或者接收

JSON

{} 表示对象

[] 表示数组或者集合

： 表示键值对的分隔符号，前面表示键，后面表示值

， 表示多个元素之间的分隔符号

“” 表示字符串

() 表示分组

数据类型

字符串

布尔值

数字

JSON数据如何生成？（服务端解决）JavaWeb --底层原理：反射



JSON数据如何解析？（客户端解决）JavaScript/jQuery

分页技术

1、数据库分页（以MySQL为例，SQL语言完成的数据帅选限定）

存储过程和自定义函数来实现

limit （页面-1）\* 记录数， 记录数

优点：可以最大限度的提高网络的使用效率，程序的响应速度很快。

缺点：在高并发访问的情况下，会造成数据库的负担；

在翻页的过程中，将会一直需要保持与数据库的联系。

2、应用级分页

优点：降低数据库的负担

缺点：第一次访问速度比较慢

数据不实时

应用服务器资源占用较大

3、混合分页

通过数据库分页，筛选一部分数据，这一部分数据，在应用中分页

4、预加载分页

自定义标签

自定义标签的工作原理和组成

组成：

1. 标签处理类组成的标签库
2. 标签库描述文件TLD

如何开发自定义标签

继承 SimpleTagSupport 类，重写 doTag 方法

编写部署描述文件TLD ,存放于 WEB-INF 文件夹中

SimpleTagSupport 是一个简单标签的父类，

不能完成循环的功能，也没有标签的主体,

也可以采用实现接口的方式来完成标签处理类 SimpleTag

TagSupport 是一个基本的标签父类

可以完成循环功能，能够处理简单的标签主体

也可以采用实现接口的方式 Tag(不能循环) IterationTag(可以循环)

doStartTag()

Tag.***EVAL\_BODY\_INCLUDE 处理标签主体***

Tag.***SKIP\_BODY 不处理标签主体***

doAfterBody()

IterationTag.***EVAL\_BODY\_AGAIN 重复处理标签主体***

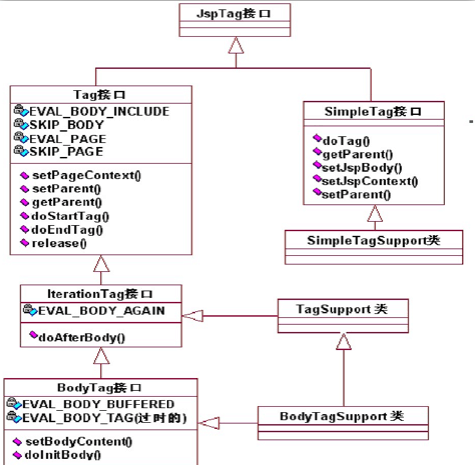
Tag.***SKIP\_BODY 执行一次以后，不再重复处理标签主体***

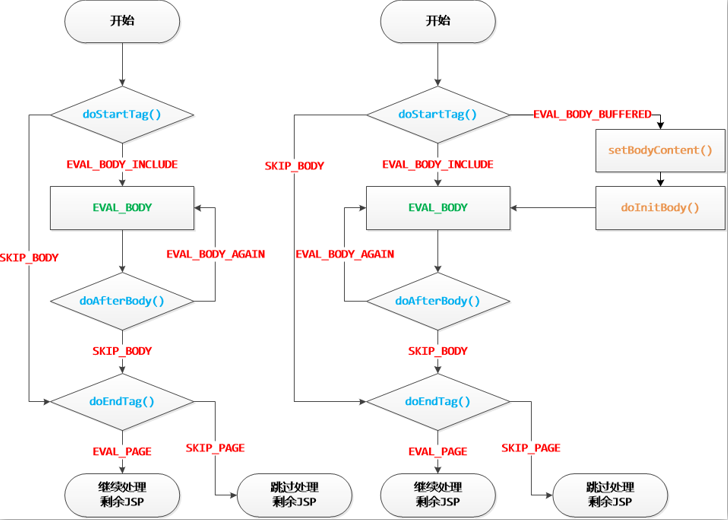
doEndTag()

Tag.***EVAL\_PAGE 继续处理剩余的jsp***

Tag.***SKIP\_PAGE 执行以后不再处理剩余的jsp***

***BodyTagSupport***





"(tlib-version,jsp-version,short-

name,uri?,display-name?,small-icon?,large-icon?,description?,validator?,listener\*,tag+)".

() 表示子标记

， 表示多个标记的分隔符号

结尾没有任何符号的表示必须的子标记（1次）

？ 表示可有可无，有的话只能出现1次

\* 表示可有可无，有的话可以出现N次

+ 表示必须出现 1 次以上