下Ctrl+Shift+T键查找类

Java数据类型转换

1. int 转 String

int i=1 ; String s=””; s=i+””; s=String.valueOf(i);

1. String 转int

S=”123”; int I; i=Interget.parseInt(s); i=Interger.valueOf(s).intValue();

1. 数组转String

String[] str={“a”,”b”,”c”};

StringBuffer strings=new StringBuffer();

For(int i=0;i<str.length;i++){

Strings.append(str[i]);

}

String s = strings.toString();

单例设计模式

package com.qqyumidi;

public class SingleTon {

// 利用静态内部类特性实现外部类的单例

private static class SingleTonBuilder {

private static SingleTon singleTon = new SingleTon();

}

// 私有化构造函数

private SingleTon() {

}

public static SingleTon getInstance() {

return SingleTonBuilder.singleTon;

}

public static void main(String[] args) {

SingleTon instance = getInstance();

}

}

Jdk 1.5之后的特性：注解； @Deprecated 意思是“**废弃的，过时的**”

@Override 意思是“**重写、覆盖**”

@SuppressWarnings 意思是“**压缩警告**”

Ajax 和Spring requestmapping 注解的方法 如果需要返回值以json数据或者对象输出，需要加ResponseBody注解

ajax parse用于从一个字符串中解析出json对象,如

var str = '{"name":"huangxiaojian","age":"23"}'

stringify()用于从一个对象解析出字符串，如

a = {a:1,b:2}

结果：

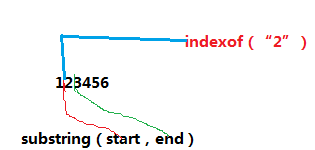
JSON.stringify(a)

"{"a":1,"b":2}"

Gson gson = new Gson();注意乱码问题

@ResponseBody注解在方法上，可使对象按照json格式进行返回，

substring ()



springIoc

传统应用程序是由我们自己在对象中主动通过new来创建并由程序来控制的，控制反转则是由容器帮忙创建外部的资源对象，然后进行依赖注入的，此时程序是被动的接受依赖对象的注入，所以是反转了依赖对象的获取反转了；

<import>标签可以放在<beans>下的任何位置，没有顺序关系。导入多个beans配置文件同事之间协同工作

1. <beans>
2. <**import** resource=”resource1.xml”/>
3. <bean id=”bean1”**class**=””></bean>
4. <bean id=”bean2”**class**=””></bean>
5. <bean name=”bean2”**class**=””></bean>
6. <alias alias="bean3" name="bean2"/>
7. <**import** resource=”resource2.xml”/>
8. </beans>

Bean再spring IOC中的构造方式

·默认是使用无参构造<bean name="bean1" **class**="cn.javass.spring.chapter2.HelloImpl2"/>

1. 其实使用有参构造<bean name="bean2" **class**="cn.javass.spring.chapter2.HelloImpl2">
2. <!-- 指定构造器参数 -->
3. <constructor-arg index="0" value="Hello Spring!"/>
4. </bean>

二．使用静态工厂实例化bean

1. **public** **class** HelloApiStaticFactory {
2. //工厂方法
3. **public** **static** HelloApi newInstance(String message) {
4. //返回需要的Bean实例
5. **return** **new** HelloImpl2(message);
6. }
7. }
8. <bean id="bean3" **class**="cn.javass.spring.chapter2.HelloApiStaticFactory" factory-method="newInstance">
9. <constructor-arg index="0" value="Hello Spring!"/>
10. </bean>
11. --------------------------------------------------------------
12. BeanFactory beanFactory =
13. **new** ClassPathXmlApplicationContext("chaper2/instantiatingBean.xml");
14. HelloApi bean3 = beanFactory.getBean("bean3", HelloApi.**class**);
15. bean3.sayHello();

三，实例化工厂方法不能直接指定实例的方法属性

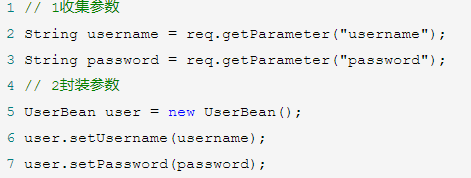
1. <!—1、定义实例工厂Bean -->
2. <bean id="beanInstanceFactory"
3. **class**="cn.javass.spring.chapter2.HelloApiInstanceFactory"/>
4. <!—2、使用实例工厂Bean创建Bean -->
5. <bean id="bean4"
6. factory-bean="beanInstanceFactory"
7. factory-method="newInstance">
8. <constructor-arg index="0" value="Hello Spring!"></constructor-arg>
9. </bean>

\

Mapped statement does not contain for conf没有引用mapping.xml

Session过期时间是不活动的时候开始计算的，如果一直活动则永不过期

Jsp+servlet+javabean数据封装到模型的代码 繁琐



<%@page import=*"www.geshiqi.log.UserBean"* %>

JavaBean的属性是任意类型

<jsp:useBean id=”beanName” class=”package.class” scope=”page|request|session|application”>

<jsp:setProperty name=*"calcBean"* property=*"\*"*/>

<% calcBean.calculate(); %>

<jsp:include>标签与include指令的区别

前者是先处理再合并，后者是先合并再处理，前者会使多个jsp翻译成servlet

1、request：如果客户向服务器发请求，产生的数据，用户看完就没用了，像这样的数据就存在request域,像新闻数据，属于用户看完就没用的。  
　　2、session：如果客户向服务器发请求，产生的数据，用户用完了等一会儿还有用，像这样的数据就存在session域中，像购物数据，用户需要看到自己购物信息，并且等一会儿，还要用这个购物数据结帐。  
　　3、application(servletContext)：如果客户向服务器发请求，产生的数据，用户用完了，还要给其它用户用，像这样的数据就存在application(servletContext)域中，像聊天数据。

JSP 集成index\_jsp 继承httpJspBase类继承HttpServlet类

1 PageContext pageContext;

2 HttpSession session;

3 ServletContext application;

4 ServletConfig config;

5 JspWriter out;

6 Object page = this;

7 HttpServletRequest request,

8 HttpServletResponse response

1 pageContext = \_jspxFactory.getPageContext(this, request, response,null, true, 8192, true);

2 application = pageContext.getServletContext();

3 config = pageContext.getServletConfig();

4 session = pageContext.getSession();

5 out = pageContext.getOut();

Git

1)git remote rm origin

2)git remote add origin https://username:password@github.com/username/test.git

3)git push origin master

表单重复提交！ 1 ，设置线程休眠 2，让表单submit失效3，服务器设置令牌与隐藏域比对

New Cookie(“lastAccessTime”,Stystem.currentTimeMillis()+””)

URLDecoder.decode(cookies[i].getValue(), "UTF-8")

Session 是cookie的一种

创建session时

String sessionId = session.getId();

Cookie cookie = new Cookie(“JSESSIONID”,sessionID);

C:\Users\ADMINI~1\AppData\Local\Temp\1513926724(1).png

Cookie.setPath(request.getContextPath());

Response.addCookie(cookie);

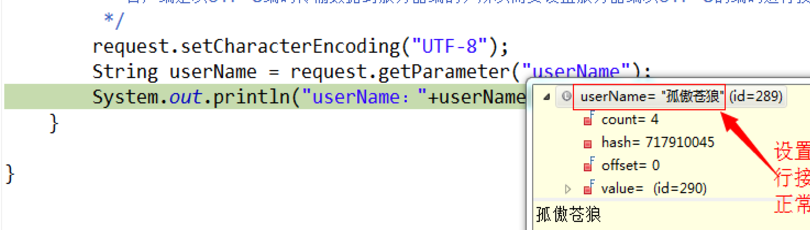
session.invalidate(); 销毁

response.sendRedirect(“web应用/资源URI”); sendRedirect可以去任何URL

request.getRequestDispatcher(“/资源URI”).forward(request,response); forward 只可以去当前web应用的资源文件

MessageFormat.format(arg,arg[])

MessageFormat.format("driver={0},url={1},username={2},password={3}",driver,url, username,password));



Post提交导致的中文乱码可以通过设置request.setCharacterEncoding来解决

**Get则不可以**request默认使用ISO8859-1字符编码来接收数据

name =new String(name.getBytes("ISO8859-1"), "UTF-8") ;//**获取request对象以ISO8859-1字符编码接收到的原始数据的字节数组，然后通过字节数组以指定的编码构建字符串，解决乱码问题**

<a href="${pageContext.request.contextPath}/servlet/RequestDemo05?userName=gacl&name=**<%=URLEncoder.encode("徐达沛", "UTF-8")**%>">点击</a>

**转发的问题**

**1可以通过ServletContext的this.getServletContext().getRequestDispatcher().forward(request,response);**

**2 通过request对象提供的request.getRequestDispatcher().forward(request,response)**

Cookie 是客户端技术

Session 是服务端技术Request

I,getRequestURL返回客户端发出请求时的完整URL

II，getRequestURI返回请求行中的资源名部分

III，getQueryString 返回请求行中的参数部分

IV ，getRemoteAddr返回发出请求的客户机的ip地址

V， getPathInfo返回请求URL中的额外路径信息

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

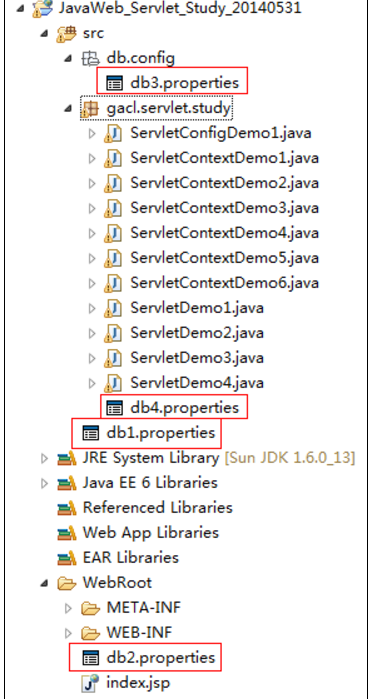
2 response.setHeader("Cache-Control", "no-cache");

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

4 response.setHeader("refresh", "5");

Response 细节问题

1. getOutputStream和getWrite方法分别用于得到输出二进制数据，输出文本数据的ServletOutputStream，Printwrite对象。
2. getOutputStream 和getWriter这两个方法互相排斥，调用了其中的一个方法后就不能再调用另外一个方法
3. Servlet程序向ServletOutputSteam或PrintWriter对象中写入的数据将被Servlet引擎从response里面获取，Servlet引擎将这些数据作为响应消息的正文，然后再与响应状态行和各响应头结合后输出到客户端。 此时就可以把response对象看做成一个容器，是特别针对网页请求返回值构造的容器对象。
4. Servlet的service方法结束后，Servlet疫情将检查getWriter或getOutPutStream方法返回的输出流对象是否已经吊用过close方法，如果没有Servlet引擎将调用close方法关闭输出流对象
5. <%@ page isELIgnored="false" %> jsp el表示无法转义的问题

C:\Users\ADMINI~1\AppData\Local\Temp\1513751218(1).png

C:\Users\ADMINI~1\AppData\Local\Temp\1513751138(1).png

C:\Users\ADMINI~1\AppData\Local\Temp\1513751247(1).png

C:\Users\ADMINI~1\AppData\Local\Temp\1513751189(1).png

C:\Users\ADMINI~1\AppData\Local\Temp\1513751293(1).png

因为ServletContext对象是整个web应用的兑现个 而loader 类装载器是类对象 所以路径会有所不同

**编写文件下载功能时推荐使用OutputStream流，避免使用PrintWriter流，因为OutputStream流是字节流，可以处理任意类型的数据**

**如果"/"是给服务器用的，则代表当前的web工程，如果"/"是给浏览器用的，则代表webapps目录。**

**request.getContextPath()**+"/index.jsp"

**${pageContext.request.contextPath}** && **request.getContextPath()就是项目名称**

<a href="**${pageContext.request.contextPath}**/index.jsp">

1. 使用OutputStream流向客户端浏览器输出汉字，需要通过设置头 （head） 控制浏览器以utf-8编码显示数据；
2. 使用PrintWriter流向客户端浏览器输出汉子，需要设置Encoding setCharacterEncoding("UTF-8") 然后再使用response.setHeader(“content-type”,”text/html;charset=utf-8”) 或者使用meta标签来模拟head response.getWriter().write("<meta http-equiv='content-type' content='text/html;charset=UTF-8'/>")
3. **如果希望服务器输出什么浏览器就能看到什么，那么在服务器端都要以字符串的形式进行输出**。
4. **outputStream.write((1+"").getBytes()); 1+空格会成为字符串，之后正常输出**

下载

1. 设置conteng-disposition响应头控制浏览器以下载的形式打开文件
2. 获取下载文件的绝对路径
3. 获取下载的文件名
4. 获取要下载的文件输入流
5. 为输入流创建一个缓存区
6. 通过response对象获取OutputStream；流
7. 将FileInputSream流写入到buffer缓冲区
8. 使用OutputStream对象将缓冲区的数据输出到客户端浏览器

类 具体化成对象，对象有属性，用属性可以构造方法来改变对象属性，respons request 代表请求和响应的对象，一次请求一个request对象，request附带属性，获得属性，构建response响应结果。可以通过一个对象来获取另外一个对象，采用这个对象的方法来使用第三或者第四个对象，那么就可以用多个对象的方法了，就是让各个对象专门去做什么，通过这些第三者的方法，完成一件事情。对象之间的关联与转化。“譬如当用一个类加载器读取资源时，首先创建一个类加载器，用类加载器对象的方法getResourceAsStream 关联一个路径，获取到一个输入流对象，因为结果是想通过properties对象对其属性进行获取，所以需要新建一个porperties对象 把输入流加载进来，形成其对象”。

Response对象，与输出流ServletOutputStream对象有互相关联的关系，

servletOutputStream对象是通过getOutputStream方法来获取的，就是有这种关联关系的存在，对象才可以通过其他的对象才实现业务功能，而其他对象姑且看做这个类的属性变量

Java Hashtable 类

Hashtable是原始java.util的一部分，是一个Dictionary抽象类具体的实现，java2中开始实现Map接口，因此他和HashMap类很类似，但是它支持同步。键值经过哈希处理，得到的散列码用作存储该表中值的索引

构造方法有四种：

1. Hashtable（）默认构造方法
2. Hashtable(int size) 构造函数创建指定大小的哈希表
3. Hashtable（int size, float fillRatio） 创建大小指定填充比例
4. Hashtable（Map M）以M中的元素为初始化元素，容量设置为M的两倍

方法接口

1, Enumeration elements() 返回此哈希表值的枚举

2，Object get(Object key)

3, Boolean isEmpty()

4, Enumeration keys()

5, Object put( Object key, Object value)

6, Object remove(Object key)

7,int size()

8 String to String ()

Properties类

Properties集成Hashtable 表示一个持久的属性集，属性列表中每个键及其对应值都是一个字符串

1. String getProperty(String key)
2. String getPropetry(String key, String defaultProperty)
3. Void list(PrintStream streamOut)
4. Void load(inputStream streamln) throws IOExecption
5. Enumeration propertyNames()
6. Object setProperty(String key,String value)
7. Void store(Outputstream streamOut, String description)

Dictionary抽象类已经过时可以用实现java Map接口来通过键来获取值

1. 通过一个键和值来存储在一个map中，当访问的值不存在时，会抛出一个NoSuchElementException异常。
2. 当对象的类型和Map里元素不兼容是，抛出ClassCastException异常
3. 当在不允许使用Null对象的Map中使用Null对象会抛出NullPointerException异常
4. 当尝试修改一个只读map时，会抛出一个UnsupportedOpreationException 异常

Void clear()

Set entrySet()

Boolean equals(Object obj)

Object get(Object k)

Int hashCode()

Set keyset()

Object put(Object k, Object v)

Collection values()

键与值一一对应，若不一一对应，则按ascII码值顺序优先存储

Java Stack Stack()

Object peek() 查看栈顶的对象但不会移除她

Object pop()移除栈顶的对象， 并作为次函数的返回值

Object push(Oject element) 把项压入栈顶

Int search(Object element) 返回对象在堆栈中的位置，

先进后出

赋值,排序,比较,查找

数组

dataType[] ar={,,,} arrayRefVar;

Boolean hasMoreElements()

Object nextElement()

Object.add()

Vector

向量：动态数组

Enumeration

枚举类