**day01 xml**

**1.xml是什么**

xml：（Extensible Markup Language）可扩展标记语言。由w3c组织提供，2000年开发出1.0版本,xml是一种数据存储格式，本质上就是一段字符串

**2..什么是B/S 结构,C/S 结构？**

答：B/S为浏览器服务器模型，C/S为客户端服务器模型

B/S优点：无需下载客户端，只需浏览器即可访问，升级程序只需要升级服务端。

B/S缺点：页面展示能力较差，对网速依赖高。

C/S优点：页面展示能力强，对网速依赖较低。

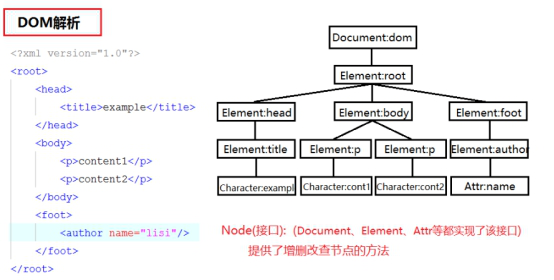
C/S缺点：需要下载客户端，升级程序时，客户端也要升级。

**3.什么是格式良好的 XML**

答：这个问题经常在电话面试中出现。一个格式良好的 XML 意味着该 XML 文档语法上是正确的，比如它有一个根元素，所有的开放标签合适地闭合，属性值必须加引号等等。如果一个 XML 不是格式良好的，那么它可能不能被各种 XML 解析器正确地处理和解析。

**4..DOM和 SAX  解析有什么区别**

xml两种解析方式：DOM解析和SAX解析（参见DOM解析图和SAX解析图）



使用 DOM 解析时，XML 文档以树形结构的形式加载到内存中，可以进行增删改查。

 dom优点：

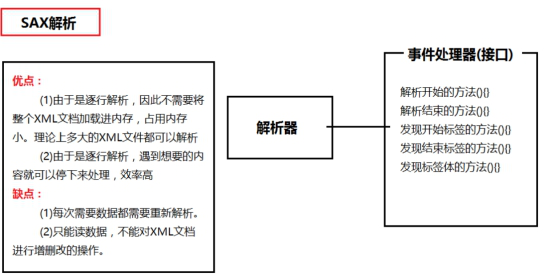
1.可以对文档进行增删改

2.内存中保存了文档树，解析一次，即可重复使用

dom缺点：

1. 如果文件过大，有可能造成内存溢出

2. 需要把所有内容读完才能解析，效率较低



而 SAX 是事件驱动的解析器，只能读，不能增删改。

sax优点：

1. 逐行解析，无需将整个文档加载到内存，占用内存小，理论上多大的xml文档都能读

2. 解析时，遇到想要的内容可以随时停下来，效率高。

sax缺点：

1. 只能读，不能进行增删改

2. 每次读取都需要重新解析。

**day02 html**

**1.什么是html**

超文本标记语言（Hypertext Markup Language）,它规定了自己的语法，用来表示更丰富的意义，比如图片、链接、表格、表单等等。浏览器认识html的标签，所以目前绝大多数网页都是html写的。

**2.html标签**

**1.文本标签<font>**

**2.标题标签<h1>-<h6>**

**3.列表标签<ul>/<li>**

**4.图片标签<img>**

**5.超链接标签<a>**

**6.表格标签<table><tr><th><td>**

**7.表单标签<form>**

浏览器向服务器发送数据有两种方式：

1：通过url发送

2：通过表单提交

**练习:表格,注册表单**

**day03 css**

**1.css概述**

css:层叠样式表（Cascading Style Sheets）层叠样式表，可以用来美化html页面，就比如html是盖房子，而css则是装修。可以实现展示数据的html代码和设置样式的CSS代码的分离，增强了网页的显示能力！

**2.div、span、p标签**

div: 独占一行，标签可以把文档分割为独立的、不同的部分。

span：并非独占一行，多个span会并排排列，直到一行排满才会换行，标签被用来组合文档中的行内元素。

p：独占一行，但是元素会自动在其前后创建一些空白，是段落标签。

**3.引入css**

**3.1.通过style属性引入（案例1）**

**3.2.通过style标签引入（案例2）**

**3.3.引入外部的css文件（案例3）**

**4.选择器**

**4.1.基本选择器（案例4）**

1)元素\标签名选择器

2)类（class）选择器

3)ID选择器

**4.2.扩展选择器（案例5）**

(1) 后代选择器 父选择器 后代选择器{  }

(2) 子元素选择器 父选择器>子元素选择器{  }

(3) 分组选择器 选择器1,选择器2,选择器3,…选择器n{}

(4) 属性选择器 选择器[属性条件..]{}

(5) 相邻兄弟选择器 大哥+小弟{  }

(6) 伪元素选择器:选中的不仅仅是元素本身, 还包括元素的状态,分为如下几种:

:link 表示元素未被点击时的状态

:hover 表示光标(鼠标)悬停时的状态(掌握!)

:active 表示元素被点击时的状态

:visited 表示元素被点击之后的状态

**day04 JavaScript**

**1.JS.** JavaScript 的缩写，一种直译式脚本语言,

1) 不需要编译，解释运行

2）可以插入到html中的编程代码,所有浏览器都支持

3）基于对象，弱类型的语言

4）很容易学习

**3.8.内置对象**

3.8.1.String对象

3.8.2.RegExp对象（正则）

3.8.3.Array对象

3.8.4.Date对象

3.8.5.Math对象

3.8.6.全局对象（global）

**day05 jQuery**

**1.什么是jQuery**

jQuery 是一个“写的更少，但做的更多”的轻量级 JavaScript 函数库。

**2.文档就绪事件**

所谓的文档就绪事件, 就是在整个html文档加载完之后立即触发, 执行一些操作，格式如下：

$(document).ready(function(){

//xxxx

});

 该函数会在整个html文档加载完之后立即执行! 其作用相当于:

window.onload = function(){ //xxx  }

**其简写形式为:**

$(function(){ //xxxx });

**3.jQuery选择器**

**3.1.基本选择器（selector案例一）**

3.1.1.元素名选择器

3.1.2.class选择器

3.1.3.id选择器

3.1.4.\*号匹配符

3.1.5.多元素选择器

**3.2.层级选择器（selector案例二）**

**3.3.基本过滤选择器（selector案例三）**

**3.4.内容选择器（selector案例四）**

**3.5.可见选择器**

**3.6.属性选择器（selector案例五）**

**3.7.子元素选择器**

**3.8.表单选择器（selector案例六）**

**5.事件**

click()

$("div").click(function(){}) – 为所匹配元素绑定点击事件

blur()

$("input").blur(function(){}) – 为所匹配元素绑定失去输入焦点事件

focus()

$("input").focus(function(){}) – 为所匹配元素绑定获得输入焦点事件

change()

$("select").change(function(){}) – 为所匹配元素绑定选项切换事件

ready()

$(document).ready(function(){}) – 文档就绪事件

其作用相当于:  window.onload = function(){}

简写形式为:

$(function(){}) – 在整个文档加载完成后立即执行

**day06-day07 mysql**

**day08-day09 jdbc**

**1.六个步骤实现JDBC程序**

1.注册数据库驱动 Class.forName("com.mysql.jdbc.Driver");

2.获取数据库连接 Connection conn = DriverManager.getConnection("jdbc:mysql:///mydb5", "root", "root");

3.获取传输器

Statement stat = conn.createStatement(); prepareStatement(sql);

4.利用传输器，发送sql到数据库执行，返回执行结果.. ResultSet用于代表sql语句的执行结果

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

executeUpdate(sql);

5.处理结果

while(rs.next()){

int id = rs.getInt(1);

String name = rs.getString("name");

double money = rs.getDouble("money");

System.out.println(id+name+money);

}

} catch (Exception e) {

e.printStackTrace();

throw new RuntimeException();

}finally{

6.释放资源

**2.PreparedStatement优点:**

（1）可以防止sql注入攻击

通过PreparedStatement对象发送sql, 是先把sql语句的骨架发送给数据库编译并确定下来, 后面发送的只能是参数的值, 不能影响sql语句的骨架, 即使参数中包含sql关键字或特殊符号, 也只会当成普通的文本来处理!

（2）通过方法来设置参数, 省去了拼接SQL语句的麻烦!

（3）可以提高程序的效率:

PreparedStatement对象发送的sql语句(骨架)到数据库编译后会被数据缓存下来, 如果下次执行的sql与缓存中的相匹配, 就不再编译而是直接使用缓存中的语句, 可以减少sql语句编译的次数, 提高程序执行的效率!

Statement对象发送的sql语句到数据库之后也会编译, 但是Statement对象是先拼接好再发送sql到数据库, 如果每次参数不同, 整条sql也就不同. 所以每次都需要编译!

**SQL注入:**由于后台的SQL是拼接而来的, 其中的参数是用户提交的, 如果用户在提交参数时, 参杂了一些SQL关键字或者特殊符号, 就有可能会导致SQL语句语意的改变, 从而造成一些意外的操作!

**3.数据库连接池**

**开源数据库连接池c3p0** c3p0-config.xml灵活使用

**day10 tomcat**

1.**web资源分类**

**静态web资源（如html 页面）：**指web页面中供人们浏览的数据始终是不变。如Html、CSS、javaScript

**动态web资源：**指web页面中供人们浏览的数据是由程序产生的，不同时间点访问web页面看到的内容各不相同。如JSP/Servlet、ASP、PHP

web容器 > servlet容器

web容器/Servlet容器具体来说就是web服务器

**Servlet容器：**

java中的动态资源开发技术为Servlet/JSP,sun公司定义了Servlet/JSP的技术规范，遵循这样规范就可以写出对应的Servlet、JSP程序,能够运行Servlet/JSP的环境就叫做Servlet容器。

**web容器：**

能够运行web应用的环境叫做web容器。web容器的概念通常要大于Servlet容器，Servlet容器通常都是web容器，但是web容器不一定是Servlet容器，因为运行ASP、PHP的服务器是web容器，但是不能运行Servlet所以不能叫Servlet容器。

**web服务器**

web容器/Servlet容器具体来说就是web服务器，学习web开发，需要先安装一台web服务器，然后再在web服务器中开发相应的web资源，供用户使用浏览器访问。

注意：这里所说的服务器不是指服务器硬件资源，而是指服务器软件。

**2.tomcat 用的是tomcat7**

是一个免费的,开源的web服务器,属于轻量级应用[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8),该服务器支持全部JSP以及Servlet规范,不能完全支持J2EE规范

3.**web应用配置虚拟路径映射的三种方式**

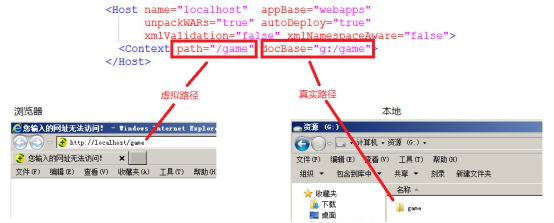
**3.1.1**

在server.xml中的<Host>标签中配置<Context>标签，其中的path属性指定虚拟路径(浏览器访问的路径)，docBase指向真实的web应用目录。

注意，这种方式配置的web应用需要重启服务器后才会生效。

如果将path配置为空，则当前web应用称为虚拟主机的缺省web应用。

如图-11所示：



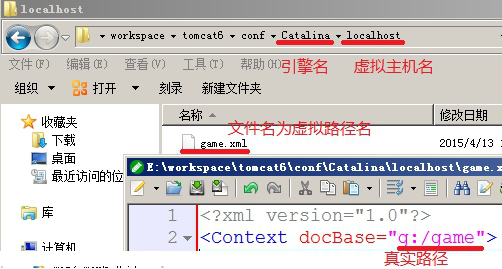
**3.1.2. 方式2**

在[tomcat]/conf/[Catalina]/[host]目录下新建一个xml文件，其中文件名为成当前web应用虚拟路径，在xml文件的内部,用<Context>标签配置docBase指定真实路径，注意这种配置方式不需要指定path虚拟路径，因为文件名就是path.

这种方式不需要重启服务器即可起作用。

如果将xml文件名改为ROOT.xml则当前web应用成为当前虚拟主机的缺省web应用。

如图-12所示：



**3.1.3. 方式3**

直接将web应用放置到虚拟主机管理的目录中。虚拟主机自动管理该目录，文件夹的名字成为web应用的虚拟路径。

这种方式不需要重启服务器即可起作用。

如果将文件夹的名字改为ROOT则当前web应用成为缺省web应用。

localhost虚拟主机管理的是webapps目录，所以将应用放置到webapps目录下通过localhost主机就可以直接访问了。

如图-13所示：



4.**配置虚拟主机**

tomcat/conf/server.xml的<Engin>标签下配置

<Host name=”主机名” appBase=”虚拟主机管理的目录”></Host>

5.DNS服务器

一些专门将域名翻译为对应主机ip的服务器叫做dns服务器（域名解析服务器）

c:/windwos/system32/drivers/etc/hosts文件可以模拟dns的功能

6.打war包

在cmd窗口中进入应用目录，使用jar -cvf news.war \* 即可将当前目录下的内容打成 news.war包

将war包直接丢到虚拟主机管理目录下，发现会自动解压成web应用。。

7.Http协议--超文本传输协议-HyperText Transfer Protocol

HTTP协议就是一套基于tcp/ip协议的应用层协议,它规定了客户端（通常是浏览器）和服务器之间的通信方式。

原理:基于请求响应模型,一次请求对应一次响应,请求只能是客户端发出服务器端只能被动的等待请求，做出响应。

分为HTTP请求和HTTP响应

**8.HTTP请求结构**

一个典型的HTTP请求分为 **一个请求行 若干请求头 一个空行 实体内容**

**一个请求行**:GET /books/java.html HTTP/1.1 请求方式 请求的资源 所遵循的协议

**若干请求头**:Accept: text/html,image/\*    Accept-Charset: ISO-8859-1  Accept-Encoding: gzip,compress Accept-Language: en-us,zh-cn  Host: www.tedu.cn:80  If-Modified-Since: Fri, 17 Feb 2017 18:23:51 GMT Referer: <http://www.tedu.cn/index.jsp>

Cookie Date: Fri, 17 Feb 2017 18:23:51 GMT

**一个空行** 空行的作用为分割请求头和实体内容。

**实体内容** 实体内容中为需要传递给服务器的内容数据。

http1.1支持长连接,在1.0的基础上加入了一些cache的新特性,当缓存对象的Age超过Expire时变为stale对象，cache不需要直接抛弃stale对象，而是与源服务器进行重新激活（revalidation）。

http1.0时的会话方式：

1. 建立连接
2. 发出请求信息
3. 回送响应信息
4. 关掉连接

**9.Http响应:**

一个典型的HTTP响应分为 一个状态行 若干响应头 一个空行 实体内容。

**一个状态行** HTTP/1.1 200 OK(HTTP目前分为1.0和1.1两个版本，其中1.0每次请求都要新建连接，响应结束后断开连接，连接不会复用效率较低。1.1版本在一次请求结束后会保持连接一段时间，下次再次请求时可以复用连接，提高了效率。

状态码:是3位的十进制数,通过不同的状态码表示服务器对请求的不同处理结果。响应状态码分为5类。)

|  |  |
| --- | --- |
| **状态码** |  |
| 100～199 | 表示接收的请求正在处理 |
| 200～299 | 表示成功接收请求并已完成整个处理过程，常用200 |
| 300～399 | 需要进行附加操作以完成请求。例如: 请求的资源已经移动一个新地址，常用302(表示请求重定向)、307和304通知使用缓存中的资源 |
| 400～499 | 客户端的请求有错误，常用404 |
| 500～599 | 服务器端出现错误，常用 500 |

**day11 servlet**

**1.Servlet接口:** --GenericServlet抽象类(没有实现service方法,手写servlet时)--HttpServlet类

1.[init](" \l "init(javax.servlet.ServletConfig))([ServletConfig](mk:@MSITStore:C:\\Users\\park\\Desktop\\java_ee_api_中英文对照版.chm::/javax/servlet/../../javax/servlet/ServletConfig.html) config)

初始化方法，会在Servlet被创建出来后立即执行，做一些初始化的工作

2.[destroy](" \l "destroy())() 销毁方法，会在Servlet被销毁之前执行，做一些善后工作

3.[service](" \l "service(javax.servlet.ServletRequest, javax.servlet.ServletResponse))([ServletRequest](mk:@MSITStore:C:\\Users\\park\\Desktop\\java_ee_api_中英文对照版.chm::/javax/servlet/../../javax/servlet/ServletRequest.html) req, [ServletResponse](mk:@MSITStore:C:\\Users\\park\\Desktop\\java_ee_api_中英文对照版.chm::/javax/servlet/../../javax/servlet/ServletResponse.html) res)

服务方法，每当有请求访问Servlet时，此方法执行，处理请求的代码要写到这个方法里。

**2)Servlet的继承结构**

Servlet接口 – 提供了一个Servlet应该具有的最基本的功能

|

|-- GenericServlet类, 实现了Servlet接口, 并实现了其中大部分的方法, 但是service方法没有实现, 这个方法需要开发人员自己去实现

|

|-- HttpServlet类, 继承了GenericServlet, 并实现了service方法, 在service方法中是根据不同的请求方式, 调用不同的doXxx方法, 因 此 我们在开发中, 只需要写一个类, 继承HttpServlet, 并覆盖 doGet()和 doPost()方法分别来处理Get请求和POST请求即可!!

**2.Servlet的调用过程**



（1）在浏览器输入地址，浏览器先去查找hosts文件，将主机名翻译为ip地址，如果找不到就再去查询dns服务器将主机名翻译成ip地址。

（2）浏览器根据ip地址和端口号访问服务器，组织http请求信息发送给服务器。

（3）服务器收到请求后首先根据Host请求头判断当前访问的是哪台虚拟主机。

（4）服务器根据http请求头中的请求URI判断当前访问的是哪个web应用。

（5）服务器根据http请求头中的请求URI判断当前访问的是web应用中的哪个web资源。

（6）检查web应用的web.xml文件，如果根据路径找到具体的servlet处理类的全路径名交给该servlet处理,如果找不到就交给缺省servlet处理。

（7）这个过程中浏览器只知道自己发出来http请求，不久就收到了http响应，浏览器不知道也不关心服务器内部是如何处理的。浏览器和服务器之间的关系是非常单纯的，只有HTTP协议。

（8）解析请求、封装RequestResponse对象、创建Servlet、调用Service方法都是服务器自动进行的，开发人员只需要写好Servlet配置进容器中即可，无需操心具体的底层实现。---这就是容器啊！多重要！多形象！

**3.servlet的生命周期**

（1）Servlet第一次被访问到时创建对象，创建出来后立即执行init方法执行初始化的操作。

（2）从此以后该对象一直驻留在内存中为后续的对这个Servlet的请求进行服务。

（3）直到服务器关闭或web应用移除出容器时，随着web应用的销毁Servlet对象销毁掉，在销毁之前调用destory方法执行善后工作。

（4） 在存活期间，每次对Servlet 的调用都会导致Service方法的执行。

**4.HttpServlet类**

继承自GenericServlet类，在GenericServlet类的基础上对HTTP协议进行了优化，并且实现了其中的service抽象方法，在其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。通常我们在开发Servlet时，直接继承HttpServlet覆盖对应的doGet()doPost()方法即可，一般不推荐直接覆盖service()方法.

**为什么要继承HttpServlet?**

因为这是一个继承了GenericServlet的类，已经提供doGet()和doPost()方法，可以方便我们开发web项目。而GenericServlet是一个基础的实现，如果要用此类，需要自己写doGet()和doPost()方法，并且需要在service方法中对请求进行判断，会比较麻烦。

**5.request对象获取**

**1).通过request对象获取客户机相关的信息**

getRequestURL方法 -- 返回客户端发出请求完整URL统一资源定位符

getRequestURI方法 -- 返回请求行中的资源名部分 统一资源标识符

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方法 -- 返回客户机的请求方式

**getContextPath**方法 -- 获得当前web应用的虚拟目录名称

注意：在写路径时不要将web应用的虚拟路径的名称写死, 应该在需要写web应用的名称的地方通过getContextPath方法动态获取

**2). 通过request对象获取请求头相关的信息**

getHeader(name)方法 --- String

getHeaders(String name)方法 --- Enumeration<String>

例如:while (values.hasMoreElements()) {

String value = (String) values.nextElement();

System.out.println(value); }

getHeaderNames方法 --- Enumeration<String>

getIntHeader(name)方法  --- int

getDateHeader(name)方法 --- long(日期对应毫秒)

**3). 获取请求参数。**

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

getParameterValues(String name) -- String[] 通过name获得多值 如: 爱好

getParameterNames() -- Enumeration<String> 获得所有请求参数的name

getParameterMap() -- Map<String,String[ ]> key :name value: 多值

**注意:**在转发之前, 如果response缓冲区被写入了数据但是还没有打给浏览器, 在转发时response缓冲区(数据)将会被清空!response.flushBuffer();

**4). 作为域对象来使用**

setAttribute(String name, Object value);用来存储一个对象，也可以称之为存储一个域属性

getAttribute(String name);   用来获取request中的数据

removeAttribute(String name);   用来移除request中的域属性

getAttributeNames();   获取所有域属性的名称

**6.乱码:**

**1)请求参数中的乱码问题** -- 编码时和解码时使用码表不一致造成的

编码: 是在浏览器进行的, 浏览器发送数据使用的是什么编码

解码: 是在服务器端进行的, 服务器端接收数据使用的又是什么编码,默认的码表是iso8859-1.

**POST提交: request.setCharacterEncoding(encode)**

**GET提交: byte[] bytes = username.getBytes("iso8859-1");**

**username = new String(bytes, "utf-8");**

POST提交的请求参数就是在请求实体内容中!, 所以这个方法可以解决POST提交的乱码问题,GET提交的请求参数由于不在请求实体内容中,而是在请求行中的请求资源路径后面拼接着, 所以这行代码对GET提交的参数乱码不起作用.Get提交通过乱码反向编码得回二进制数组,通过二进制数组查询正确的码表, 得出正确的数据匹配规则:

**2)响应向客户端发送数据流乱码问题:**

**\*\*字节流发送数据的中文乱码问题**

服务器端指定了用utf-8来发送数据, 浏览器在接受数据时, 如果不指定将使用默认的平台码GBK, 编解码不一致导致乱码.

response.setHeader("Content-Type", "text/html;charset=utf-8");// 通知浏览器使用utf-8打开服务器发送过去的数据

**\*\*字符流发送数据的中文乱码问题**

利用字符流发送数据, 底层还是要转成字节. 服务器端如果不手动指定, 服务器默认会使用iso8859-1码表, 由于里面没有中文汉字, 所以服务器端发送给客户端就是一堆乱码, 客户端不管使用什么码表都无法转成正常的字符,服务器会根据getCharacterEncoding()方法返回的编码来发送数据, 如果没有指定, 该方法默认返回iso8859-1

\*\* 解决方案:

第一步, 需要指定服务器发送数据使用utf-8

response.setCharacterEncoding("utf-8"); //通知服务器使用utf-8来发送响应实体中数据

第二步: 需要指定浏览器在接收数据时也使用同一个编码来打开数据

response.setHeader("Content-Type", "text/html;charset=utf-8");

等价于<==>

**response.setContentType("text/html;charset="+encode);**

**7.\*.后缀的优先级永远最低!!**

**哪一个更接近哪一个起作用!!**

Servlet1 映射到 /abc/\*

Servlet2 映射到 /\*

Servlet3 映射到 /abc

Servlet4 映射到 \*.do

“/abc/a.html”---1

“/abc”---3

“/abc/a.do”---1

“/a.do”---2

**Web.day12.request,response**

**response**

**1.继承结构**

ServletResponse -- 通用的响应接口, 定义了响应对象应该具有的功能

|

|--HttpServletResponse 在ServletResponse的基础上, 添加了很多和Http协议相关的方法

**2.重定向与转发**

|  |  |
| --- | --- |
| 转发 | 重定向 |
| 服务器端完成 | 客户端完成 |
| 速度快 | 速度慢 |
| 同一次请求 | 两次不同请求 |
| 不会执行转发后的代码 | 会执行重定向之后的代码 |
| 地址栏没有变化 | 地址栏有变化 |
| 必须在同一台服务器中完成 | 可以在不同服务器中完成 |
| request.getRequestDispatcher("new.jsp").forward(request, response); | response.sendRedirect("new.jsp"); |

重定向的原理就是302+location, 通过设置状态码302和location响应头就可以实现重定向的效果

response.setStatus(302);

response.setHeader("Location", "/Day09/index.jsp");

<==>

response.sendRedirect("/Day09/index.jsp");

**3.实现定时刷新** :是通过Refresh响应头, 可以实现在多少秒之后跳转到另外一个资源

response.setHeader("Refresh", "3;url=/Day09/index.jsp");

**请求转发:**

一次请求,一次响应 request对象是同一个

地址栏不会发生变化

只能用于服务器内部的资源跳转, 并且只能是同一应用中的不同资源上进行跳转, 不可用在不同应用和不同服务器中的资源跳转

**请求重定向:**

两次请求,两次响应 request对象不是同一个

地址栏会发生变化

可以用于服务器内部的资源跳转, 也可以用于不同应用和不同服务器之间的资源跳转

**定时刷新:**

两次请求,两次响应 request对象不是同一个

地址栏会发生变化

可以用于服务器内部的资源跳转, 也可以用于不同应用和不同服务器之间的资源跳转

和重定向不同的是

定时刷新可以在刷新到新的地址之间设置一个时间, 在间隔的这段时间内可以输出文本到浏览器并维系一段时间

用法:

\*\*如果是同一服务器中的同一应用内部的资源跳转:

~如果需要利用request域在跳转的资源之间传输数据, 只能用请求转发

~如果不想让地址栏发生变化, 只能用请求转发

~如果需要地址栏发生变化, 只能用重定向或定时刷新

~如果没有什么特殊需求, 三种方式都可以, 但是推荐使用转发, 可以减少请求次数降低服务器的压力.

~如果只是想更新刷新操作, 最好使用重定向或定时刷新, 使用请求转发, 在刷新时会把刚才的操作再做一遍, 可能会导致一些问题, 比如表单重复提交或重复支付订单等

\*\*如果是不同服务器或不同应用内部的资源跳转, 只能用重定向或这定时刷新:

重定向和定时刷新的主要区别在于: 重定向会立即跳转, 而定时刷新可以设置一个时间间隔, 在指定时间后再进行跳转.

如果在跳转之前需要输出提示信息(如: 注册成功, xx秒后跳转到xxx)只能用定时刷新, 否则两种方式都可以.

**4.控制浏览器的缓存行为**

由于不同的浏览器的缓存行为可能是不同的, 我们可以在服务器中通过设置响应头来控制浏览器的缓存行为!!

**控制浏览器不要缓存:**

setDateHeader("Expires", -1);

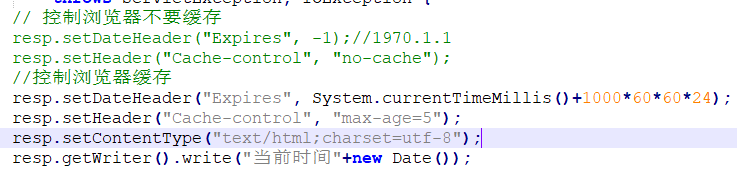
setHeader("Cache-control", "no-cache");

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

**控制浏览器缓存:**

setDateHeader("Expires", System.currentTimeMillis()+1000\*60\*60\*24);

setHeader("Cache-control", "max-age=60");



缓存的优先级:Pragma>Cache-control>Expires

**Web.day13.easymall**

1.搭建环境 创建工程,配置虚拟主机

2.导入静态应用

3.开发注册功能 解决乱码问题,校验数据为空 注册失败消息提示,输入框数据回显 用户名是否存在验证 注册成功跳转

4.校验注册表单

5.开发验证码图片,掌握禁止验证码缓存

6.点击验证码图片,换一张

**Web.day14.ServletConfig+ServletContext**

**1.ServletConfig:**

代表当前Servlet在web.xml文件中配置信息的对象

获取ServletConfig对象:

ServletConfig scf = this.getServletConfig();

getInitParameter(String name);

getInitParameterNames();

功能:

在web.xml中的<Servlet>标签的内部可以为当前Servlet配置零个或多个初始化参数, 这些参数相当于我们为Servlet配置了一些初始化信息, 可以通过ServletConfig对象(在当前Servlet内部可以获取该对象)来获取这些信息。

**2.ServletContext**

代表当前web应用的对象,在web应用被加载后, 服务器会立即创建出代表当前web应用的ServletContext对象, 创建后该对象会一直驻留在内存中唯一的代表当前web应用, 直到服务器关闭或者web应用移出容器时为止, 随着web应用的销毁, ServletContext对象也跟着销毁!

获取ServletContext对象

this.getServletConfig().getServletContext() 或者 this.getServletContext();

功能:

1)在当前Servlet中配置的参数信息, 只能在当前Servlet中通过ServletConfig对象来获取, 在其他的Servlet中无法获取!

如果有一段初始化参数, 希望不是属于某一个Servlet对象, 而是整个web应用都可以使用, 可以将这些参数配置到web.xml的根目录下, 配置在根目录下的这些信息是属于整个web应用的, 可以通过代表整个web应用的ServletContext来获取。

2)作为域对象来使用:ServletContext对象是一个域对象, 利用这个对象上的map就可以在整个web应用内实现资源的共享.

获取方法:

setAttribute();

getAttribute();

removeAttribute();

getAttributeNames();

生命周期:和WEB应用的命一般长

作用范围:在整个WEB应用内都可以被看见

主要功能:在整个WEB应用内实现资源的共享

ServletContext在作为作用域时,也被称为Application作用域.

3)获取web资源:如何获取web应用的资源文件路径

sc.getRealPath("xxx"); 传入一个相对于web应用根目录的资源文件的路径, 这个方法会在传入的路径的前面动态的拼接上当前web应用根目录的硬盘路径, 从而拼接出当前资源文件的硬盘路径, 由于web应用的根目录的硬盘路径是动态获取的. 没有写死, 即使换一个发布环境, 也能获取到正确的路径

WEB应用在开发和部署时使用的是不同的路径,开发是在workspace里面,部署是在WEB服务器上,在开发时很可能不知道未来应用部署的绝对路径.在读取一些资源时,需要指定该资源的绝对路径,就需要通过方法来动态获取当前应用所在的绝对路径.解决方案就是通过调用ServletContext对象的getRealPath(path)方法,其中path指定的是相对于Web应用的路径,该方法会返回该资源的绝对路径.

当没有ServletContext对象可以使用时, 可以通过类加载器来加载资源, 类加载器提供了方法来加载资源, 但是要求传入的文件路径必须是相对于类加载器加载类的路径

getResource(xxx)

getResourceAsStream(xxx)

**Web.day15.cookeie,session**

**1.会话技术:**

为了实现某一个功能, 客户端和服务器之间可能会产生多次的请求和响应, 从客户端访问服务器开始, 到最后访问服务器结束, 客户端关闭为止, 这期间产生的多次请求和响应加在一起就称之为是客户端和服务器之间的一次会话

HTTP协议规定一次请求一次响应后断开连接,每一次请求都是一个崭新的请求.但是一次会话往往需要跨越多个请求,如何保存在每次请求中产生的临时数据,是必须要解决的一个问题.解决方案有2种:

一种是将临时数据保存在客户端浏览器上,用户每次访问时,将会话的临时数据在请求中带给服务器;(Cookie)

另一种方案是将会话数据保存在服务器上,给每个用户一个身份标识,根据用户的身份标识,找到保存在服务器上的数据.(Session)

1.2.如何来保存会话过程中产生的数据?

~requset域太小

~ServletContext域太大

**2.Cookie**

Cookie的原理是通过Set-Cookie响应头和Cookie请求头将会话中产生的数据保存在客户端.

客户端请求服务器, 服务器将需要保存的数据通过Set-Cookie响应头发给客户端, 客户端收到后会将数据保存在浏览器的内部

当客户端再次请求服务器时, 通过Cookie请求头将上次保存的数据再带给服务器, 服务器通过Cookie头来获取数据, 通过这种方式可以保存会话中产生的数据.

Cookie是将需要保存的数据保存在了客户端, 是客户端技术. 每个客户端各自保存各自的数据, 再次访问服务器时会带着自己的数据.



2.2.1.创建Cookie

Cookie cookie = new Cookie(String name, String value);

getName()

getvalue()

setValue()

2.2.2.添加Cookie

response.addCookie(Cookie cookie);//向响应中添加一个Cookie, 可以在一次响应中添加多个Cookie

2.2.3.获取Cookie

Cookie [] cs = requset.getCookies();//返回请求中所有Cookie组成的数组, 如果请求中没有任何Cookie信息, 则返回null.

2.2.4.设置Cookie存活时间

setMaxAge();//指定Cookie保存的时间, 以秒为单位

//如果不明确的指定, Cookie默认是会话级别的Cookie, Cookie会保存在浏览器的内存中, 一旦浏览器关闭, Cookie也会随着浏览器内存的释放而销毁.通过setMaxAge()方法可以设置Cookie的存活时间, 设置了存活时间后, Cookie将会以文件的形式保存在浏览器的临时文件夹中, 在指定的时间到来之前, 即使多次开关浏览器, Cookie信息也会一直存在.

2.2.5.设置Cookie路径

setPath(String path); 向下兼容，不向上兼容//设置当前Cookie在浏览器访问哪一个路径及其子孙路径的时候带回来

//如果不指定, 默认的path值就是发送Cookie的Servlet的所在的路径

~2.2.6.设置domain(不推荐设置)

setDomain(xxx);

设置浏览器访问哪一个域名时带着当前Cookie,现在的浏览器一般都拒绝接受第三方的Cookie, 甚至有的浏览器只要发现Cookie被设置了domain, 不管是不是第三方Cookie, 也会拒绝接受! 所以最好不要设置这个方法!

2.2.7.删除Cookie

没有直接删除Cookie的方法!!!如果想要删除一个Cookie, 可以向浏览器发送一个 同名 同path 同domain的Cookie, 只需要将Cookie的maxAge设置为0, 由于浏览器是根据 名+path+domain 来区分Cookie的, 所以当两个cookie的上述条件相同时, 浏览器就会认为是同一个Cookie, 那么后发的Cookie会覆盖之前的, 而后发的Cookie的存活时间为0, 所以浏览器收到后也会立即删除!!

~2.2.8.Cookie的细节:

一个Cookie只能标识一种信息，它至少含有一个标识该信息的名称（NAME）和设置值（VALUE）。一个WEB站点可以给一个WEB浏览器发送多个Cookie，一个WEB浏览器也可以存储多个WEB站点提供的Cookie。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。





**3.easymall在页面中显示上次访问时间**

**4..easymall实现记住用户名**

1)编写LoginServlet

if("true".equals(remname)){

Cookie remnameCookie = new Cookie("remname", URLEncoder.encode(username, "utf-8"));

//设置最大生存时间(保存30天)

remnameCookie.setMaxAge(3600\*24\*30);

//设置path, 让浏览器访问当前WEB应用下任何一个资源都能带Cookie!!

remnameCookie.setPath(request.getContextPath()+"/");

//将Cookie添加到response中发送给浏览器

response.addCookie(remnameCookie);

}else{//取消记住用户名 -- 删除Cookie

Cookie remnameCookie = new Cookie("remname", "");

remnameCookie.setMaxAge(0);

remnameCookie.setPath(request.getContextPath()+"/");

response.addCookie(remnameCookie);

2)编写login.jsp, 从Cookie中获取用户名存入用户名输入框

3)勾选记住用户名复选框

**5.Session**

Session是一个域对象

setAttribute(String name, Object value);

getAttribute(String name);

removeAttribute(String name)

getAttributeNames()

生命周期:

当第一次调用request.getSession()方法时创建Session

超时: 如果一个Session超时30分钟(可以在web.xml中来修改, 在根目录下通过<session-config>来配置)未被使用, 则认为Session超时, 销毁session

自杀: 当调用session.invalidate()方法的时session立即销毁!!

意外身亡: 当服务器非正常关闭时, 随着应用的销毁, session销毁. 当服务器正常关闭, 则未超时的session将会以文件的形式保存在tomcat服务器work目录下, 这个过程叫做session的钝化. 当服务器再次启动时, 钝化着的session还可以恢复过来, 这个过程叫做session的活化。

作用范围: 当前会话范围

主要功能: 保存当前会话相关的数据

3.2.Session的原理 Session是基于一个JSESSIOINID的Cookie工作的

3.2.1.怎么解决浏览器关闭之后, 无法使用浏览器关闭之前的Session??(注意: 在访问一个jsp时, 默认一上来就会创建session!!)

3.2.2.禁用Cookie的情况下使用Session

URL重写: 就是在传入的URL地址后拼接JSESSIOINID返回一个新的地址, 用来在禁用Cookie的情况下用url地址来携带JSESSIOINID

**6.实现购物车**

**7.登录功能的实现,session中保存登录标记,判断当前用户的登录状态**

**8.实现后台注册代码**

**9.Cookie和Session的比较:**

(1) Cookie是将会话中产生的数据保存在客户端, 是客户端的技术

(2) Session是将会话中产生的数据保存在服务器端, 是服务器端的技术

(3) Cookie保存的信息的时间比较长, 但是安全性不高. 可能随着用户的操作, Cookie会被清空, 所以Cookie存储数据的稳定性比较差. 因此Cookie适合存放要保存时间较长, 但安全性要求不高的信息

(4) Session通常保存信息的时间比较有限, 但安全性比较高, 因为是保存在服务器端, 不会随着用户的操作而导致Session意外丢失, 因此session适合存放安全性要求比较高, 但是不需要长时间保存的数据.

**Web.day16.ajax+jsp**

**1.ajax:**asynchronous js and xml: 异步的js和xml

可以利用js访问服务器, 而且是异步访问! 通常服务器给浏览器响应的是一个完整的页面, 而在AJAX中, 由于是利用js访问服务器, 再由js接受响应, 局部刷新页面, 所以服务器不用给浏览器响应整个页面了, 而只是数据。

\* 服务器响应的数据:

> text 纯文本 "用户名已存在!!"

> xml

> json: js提供的一种数据交互格式, 在js中很受欢迎

**2.同步交互和异步交互**

同步:

\* 向服务器发一个请求, 必须等待响应结束, 才能发送第二个请求, 在服务器处理期间, 浏览器不能干别的事儿

\* 刷新整个页面

异步:

\* 向服务器发一个请求, 不用等待响应结束, 就可以发送第二个请求, 在服务器处理期间, 浏览器可以干别的事儿

\* 可以使用js接受服务器的响应, 再利用js局部刷新页面

**3.AJAX的优点和缺点**

优点:

\* 异步交互, 提高了用户体验!

\* 服务器只响应部分数据, 而不是整个页面, 所以降低了服务器的压力!

缺点:

\* ajax不能应用所有的场景

\* ajax会无端的增加访问服务器的次数, 给服务器带来了压力!!

**4.js实现ajax的步骤**

第一步, 获取XMLHttpRequest对象xmlHttp

第二步, 打开与服务器的连接xmlHttp.open(method, url, async);async: 是否为异步传输, true 表示为异步传输 一般都是true

第三步, 发送请求xmlHttp.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

第四步, 注册监听:(xmlHttp.readyState)

0状态: 表示刚创建XMLHttpRequest对象, 还未调用open()方法

1状态: 表示刚调用open()方法, 但是还没有调用send()方法发送请求

2状态: 调用完了send()方法了, 请求已经开始

3状态: 服务器已经开始响应, 但是不代表响应结束

4状态: 服务器响应结束!(通常我们只关心这个状态) if(xmlHttp.readyState == 4)

**5.jQuery实现AJAX**

**1). load方法**

$(selector).load(url,data,callback);

selector -- 选择器, 将从服务器获取到的数据加载到指定的元素中

url -- 发送请求的URL地址

data -- 可选, 向服务器发送的数据 key/value数据 如:{"username" : "张飞", "psw" : "123"}

callback -- 可选, load方法完成后所执行的函数

$("#username\_msg").load("<%= request.getContextPath() %>/AjaxCheckUsernameServlet", {"username": username});

**2). $.get方法**

$.get(url, [data], [callback]);

url -- 发送请求的URL地址

data -- 可选, 向服务器发送的数据

callback -- 可选, 请求成功后所执行的函数

$.get("<%= request.getContextPath() %>/AjaxCheckUsernameServlet", {"username": username}, function(result){

$("#username\_msg").html("<font style='color:red'>"+result+"</font>");});

**3). $.ajax方法**

$.ajax(url, [data], [async], [callback]);

url -- 发送请求的URL地址

data -- 可选, 发送至服务器的key/value数据

async -- 可选, 默认为true, 表示异步交互

type -- 可选, 请求方式 , 默认为"GET"。

success -- 可选, 请求成功后执行的函数, 函数参数:

result -- 服务器返回的数据

$.ajax({

"url" : "<%= request.getContextPath() %>/AjaxCheckUsernameServlet",

"data" : {"username": username},

"async" : true,

"type" : "POST",

"success" : function(result){

$("#username\_msg").html("<font style='color:red'>"+result+"</font>") } } );

**6.JSP**

由SUN公司提供的动态web资源的开发技术, 看起来非常像html, 但是可以在JSP页面中写java代码, 所以JSP是一种动态web资源开发技术.

JSP的出现是为了解决Servlet在响应时不适合向外输出页面的问题,Servlet本质上是一段java代码, 非常适合处理逻辑, 但是不合适向外输出页面。HTML适合用来开发页面, 展示数据, 但是HTML开发出来的页面本质上就是一个文档, 无法展示动态的数据。

JSP非常适合编写HTML代码, 适合作为响应页面向外输出, 同时JSP里可以写java代码, 也可以展示动态的数据。所以JSP可以解决既要处理逻辑又要向外输出页面的难题。

JSP在第一次访问时, 会被翻译成一个Servlet, 对JSP访问后看到的页面 其实就是翻译后的Servlet在向外输出！！

JSP语法

**6.1.模版元素**: html代码

◇ 直接写在jsp页面中的html内容称之为jsp页面中的模版元素

◇ 模版元素在翻译过来的Servlet中被out.write()原样输出到浏览器中

out.write("\tday13....index.jsp........\r\n");

**6.2.JSP表达式:**

<%= 脚本表达式 %> 常量 变量 表达式 ◇ 在翻译过来的Servlet中, 计算表达式的值原样输出 out.print( 100+123 );

**6.3.JSP脚本片段**: <% 若干java语句 %> Scriptlet

◇ 在翻译过来的servlet中, 脚本片段被复制粘贴到对应位置执行

◇ 多个脚本片段之间的变量可以互相访问

◇ 在某一个脚本片段中的java代码可以是不完整的, 但是要求在翻译过来的servlet中整体的代码必须是完整符合java语法的

◇ 任何文本，HTML标记，JSP元素必须在脚本片段之外

**~~6.4.JSP声明:(不常用)** 格式: <%! java代码 %>

◇ 写在jsp声明中的内容, 在翻译过来的servlet中会和servlet方法平级成为类的成员 成员变量刷新值变化

( 注意: 写在JSP脚本片段中的内容, 在翻译过来的servlet中是方法的局部变量! ) 局部变量刷新值不变

**6.5.JSP注释:** 格式: <%-- JSP注释 --%>

<%-- out.write("aaa"); --%> 被JSP注释注释的内容, 在翻译的过程中被抛弃, 不会被翻译

<% //out.write("bbb"); %> 被java注释注释的内容, 在翻译的过程中被当作脚本片段翻译到servlet中, 但是由于被注释了, 所以也不会执行

<!-- <% out.write("ccc"); %> --> 被HTML注释注释的内容, 在翻译过程中直接当作模版元素原样输出到浏览器, 但是浏览器认为是注释, 所以不予显示

!!!在jsp中最好使用jsp注释, html注释慎用!!

**6.6.JSP指令:**<%@ 指令名称 若干属性声明... %>

不会直接产生输出, 用来指挥解析引擎如何来翻译当前JSP页面中其他部分的内容

**6.6.1.page指令** 格式: <%@ page ... %> 在JSP 2.0规范中共定义了三个指令： page指令 Include指令 taglib指令

-- 用来声明当前JSP页面的基本属性的, page指令可以写在JSP页面的任意位置, 但是为了可读性考虑, 一般情况下最好放在JSP页面的最前面

~~[ 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

可以在一个page指令的import属性中导入多个包,**用","分开** 也可以通过多个page指令来分别导包

!!![ pageEncoding="characterSet | ISO-8859-1" ] 服务器翻译JSP时使用的编码集.

如果想防止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.

**2.6.2.include指令** <%@ include file="" %> -- 可以实现页面包含的效果

◇ include指令实现的包含叫做静态包含:

多个JSP文件翻译成一个servlet, 最终由这一个servlet向外输出数据, 这是源文件级别的包含

◇ 其他方式的包含叫做动态包含:

被包含的页面各自翻译成servlet, 包含的过程其实就是各个servlet分别执行后在输出流上的合并

◇ 总结: 静态包含在效率上更高一些, 尽量使用静态包含.

**2.6.3.taglib指令(TODO)**

导入tld标签库文件

JSTL标签库

**7.JSP的九大隐式对象( 记住!!! )**

JSP翻译引擎在将JSP翻译成servlet的过程中, 在servlet里预先定义了九个对象, 因此我们可以在JSP页面中直接使用这九个对象

**page对象**

**request对象**

**response对象**

**config对象**

**ServletContext application对象**

**session对象**

**~exception对象**

**~out对象**

**!!pageContext**

~**~3.1.ou**t相当于response.getWriter();

区别在于out对象自带缓冲区, 如果out和response.getWriter()混用可能会造成输出顺序上混乱,page指令中buffer和autoflush就是用来设置out缓冲区的总结起来一句话, 如果想要在JSP页面中输出内容, 直接使用out, 尽量别使用response.getWriter();

**!!!3.2.pageContext**

代表当前JSP页面的运行环境的对象, 通过该对象可以访问页面中的共享数据

PageContext对象功能:

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

!!!(1) 作为入口对象获取其他八大隐式对象

getPage()方法 返回page隐式对象

getRequest()方法 返回request隐式对象

getResponse()方法 返回response隐式对象

getServletConfig()方法 返回config隐式对象

getServletContext()方法 返回application隐式对象

getSession()方法 返回session隐式对象

getException()方法 返回exception隐式对象

getOut()方法 返回out隐式对象

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

!!!(2) 本身也是一个域对象, 也可以作为入口对象来操作其他三大作用域中的数据

a)本身是个域对象

setAttribute(String name, Object obj);

getAttribute(String name);

removeAttribute(String name);

getAttributeNames();

生命周期: 访问JSP页面开始时创建, 访问JSP页面结束时销毁

作用范围: 当前JSP页面

主要功能: 在当前JSP页面中共享数据

ServletContext > Session > request > pageContext

b)作为入口对象, 可以操作其他三大作用域

setAttribute(String name, Object value,int scope)

getAttribute(String name,int scope)

removeAttribute(String name,int scope)

其中pageContext中代表域的常量：

PageContext.APPLICATION\_SCOPE

PageContext.SESSION\_SCOPE

PageContext.REQUEST\_SCOPE

PageContext.PAGE\_SCOPE

额外提供了:

!!findAttribute(String name); //按照由小到大的顺序在四大作用域中搜寻指定名称的属性, 如果找到就返回, 如果都找不到就返回一个null

**ServletContext > session > request > pageContext**

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

~~(3) 提供了便捷方法实现请求转发和包含

request.getRequestDispatcher("/7.jsp").forward(request, response);

request.getRequestDispatcher("/7.jsp").include(request, response);

<%@include file="xxx.jsp" %>

转发: pageContext.forward("/index.jsp");

包含: pageContext.include("/index.jsp");

**Web.day17.git 略**

**Web.day18jsp+jstl**

**1.el表达式:**

Expression Language，用来替代<%= %>脚本表达式。

基本结构: ${ 表达式 }

EL只能获取域中的数据！！

EL只能获取不能设置!!!

EL只能获取不能遍历!!!

可以获取:

常量,变量(只能获取域中的值，没有不会输出null),数组(越界了什么也不输出，直接输出数组对象，会输出地址),集合(直接输出集合对象，会输出集合内容),map(直接输出map对象，会输出map内容,乱序)

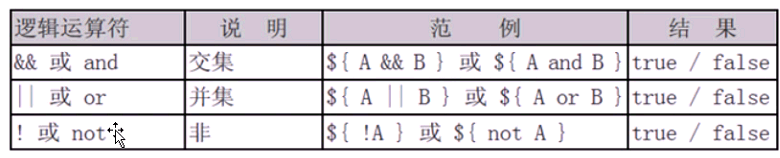
运算：

算术运算:+("+"只是一个运算符, 不能作为连接符使用.数字加法先转换再加，不识别就异常报错) - \* /(精确除法)

关系运算:



逻辑运算:



三元表达式: 表达式 ? "xx" : "xx"

empty运算：判断是否为null

**2.jsp 11个隐藏内置对象**

1个：!!!pageContext:代表当前JSP页面环境的pageContext对象,有了它就意味着el中就有了其他八大隐式对象

4个作用域：代表四大作用域的隐式对象, 用来明确指定从哪个域中查找数据

!!!pageScope

!!!requestScope

!!!sessionScope

!!!applicationScope

2个请求参数：请求参数组成的map

!!!param Map<String, String> 不属于request作用域

paramValues Map<String, String[]>

2个请求头：请求头组成的map,key为请求头的名字

header Map<String, String> 不属于request作用域

headerValues Map<String, String[]>

1个所有cookie信息组成的map

!!!cookie Map<String, Cookie> key为Cookie的名字, 值为Cookie对象

1个web应用的初始化参数 web.xml配置 所有web应用初始化信息组成的map,key初始化参数的名字, value为参数对应的值

initParam -- Map<String, String>

**3.jstl标签库**

通用的JSP标签库 提供了实现java代码中最常见功能的操作标签(taglib指令),为javaweb开发人员提供的一套标准通用的标签库, 可以和el配合起来替JSP页面中的大部分的java代码,javaee4.0及其之前的版本中, 在使用前需要导入jstl开发包, 在javaee5.0及其以后版本中内置了jstl开发包, 所以不需要我们手动导入就可以直接使用了!!

IMG_268

jstl的子库：

core - 核心库

~<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。

输出常量 value

输出变量 标签里面加el表达式

转义输出 转义成字符串+escapeXml="false"==标签的内容

输出默认值 default

!!!<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。

向四大作用域中增加域属性

向map中增加、修改键值对 target property value

修改javabean的属性

<c:remove> 标签用于删除各种Web域中的属性,如果不指定scope属性，会删除所有作用域中的该属性

<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>捕获到的异常 可以通过声明var属性 用指定的名称 写入 pageContext域 后续处理

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

test 写判断表达式

scope 声明将test的结果存入哪个域

var 写入域时的属性名

!!!<c:choose> - <c:when> <c:otherwise>

标签用于指定多个条件选择的组合边界，实现多重判断，可以构造类似 “if-else if-else” 的复杂条件判断结构。如果两个符合条件，只有一个生效。

!!!<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: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

fn函数库：jstl提供的可以供el调用的函数,其中提供了大量操作字符串的函数

**4.四大作用域总结**

pageContext域 --- pageScope -- page域

生命周期：访问jsp开始创建，结束时销毁

作用范围：当前jsp页面

主要功能：在当前jsp页面中共享数据

request域 --- requestScope

生命周期：请求开始时创建，请求结束后销毁

作用范围：整个请求链

主要功能：在整个请求链中共享数据

session域 -- sessionScope

生命周期：第一次调用request.getSession()时创建。超时、自杀、意外身亡

作用范围：整个会话

主要功能：在整个会话范围内共享数据

ServletContext域 -- applicationScope -- application域

生命周期：服务器启动应用加载后立即创建，服务器关闭或web应用移除出容器时销毁

作用范围：整个web应用

主要功能：在整个应用范围内共享数据

什么时候用什么域？

如果数据只在jsp页面中使用，用page域

如果数据需要在转发包含时传递，用request域

如果数据现在我需要用，过一会我还需要用，用session域

如果数据现在我需要用，过一会别人也需要用，存ServletContext域中

**5.JSP标签技术**

在JSP页面中写入大量的java代码会导致JSP页面中html代码和java代码混杂在一起, 会造成页面非常的混乱, 难于维护

于是在JSP的2.0版本中, sun提出了JSP标签技术, 推荐使用标签来代替JSP页面中java代码, 并且推荐, JSP2.0以后不要在JSP页面中出现任何一行java代码。

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

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

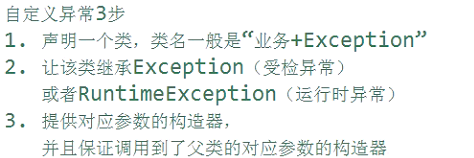
<jsp:param> 配合前两个标签使用，可以在包含或转发时，带一些参数过去

6.改造EasyMall项目, 将其中的java代码用EL+JSTL进行替换。

**Web.day19web开发模式**

**1.复习:**

**接口:不同类事物的共同的行为特征的抽象**



**2.单例模式:**

解决的问题:保证一个类在一个程序中只能有唯一的一个实例

2种情况:

1. 需要唯一的一个对象进行统一的管理,如果对象多了,就会出问题

2. 相对于频繁创建对象,使用唯一的对象可以提高效率

如何实现单例模式?

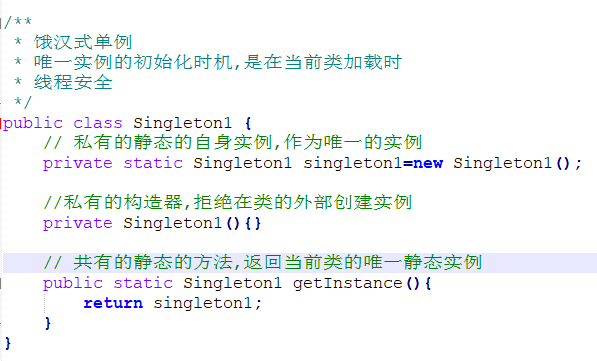
1. 私有的构造器

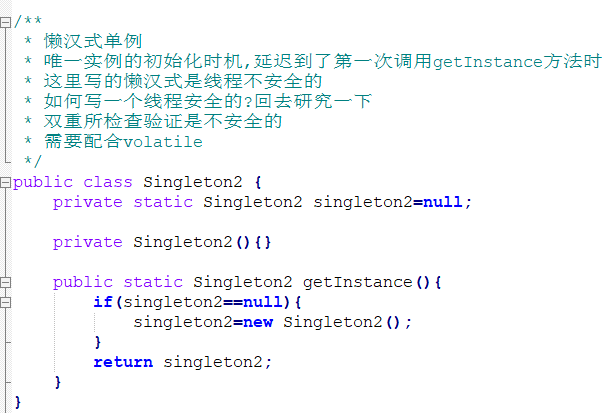
2. 私有的静态的自身实例,作为唯一的实例

3. 公有的静态的返回当前类的唯一实例的方法

饿汉式：线程安全

懒汉式：线程不安全volatile





**3.MVC设计模式**

软件可以认为有 Model View Controller 来组成 MVC设计模式 要求这三部分 应该尽量独立 互不干扰 使程序结构清晰 便于开发和维护

现在的javaweb应用基本都使用的是javaee的经典三层架构。

**4.JAVAEE经典三层架构**

将整个JAVAEE的开发过程分为 Web Service Dao 三层

分层的优势：

~1.分层目的在于 代码具有更加优良的结构 便于开发和调试

~2.便于在层与层之间实现代码的复用 减少代码冗余

~3.在更改模块时可以不影响其他模块的使用 实现模块的复用

为了实现第三个优点，三层架构要求，层与层之间尽量的独立，不要胡乱传递层特有的对象，例如不要将web层特有的对象传递给其他层，dao层 特有的对象返回给其他层，因为，一旦这样做，这些某一层特有的对象就入侵到了其他层，一旦需要替换这一层时，不可避免的要跟着修改其他层中使用了这一层特有对象的地方的代码。这种由于乱传递对象而造成 层与层之间关系变得过于紧密的情况，称之为层与层之间发生了耦合。

这种层与层之间过于紧密的关系是有害的，应该尽量的避免或消除，解决这些耦合的过程就称之为 解耦。最终希望实现 高内聚 低耦合 的状态。

**解耦的实现：**

不要胡乱传递层特有的对象人为的造成耦合。但是再小心，层与层之间早晚要发生关系的，这些耦合是无法避免的，此时 需要将耦合管理起来 - 管理耦合有很多种方案 - 我们介绍 - 接口+配置文件+工厂 实现解耦。

**接口：**

使层可以面向接口编程 而不需要关注下面层具体的实现,在切换底层时 不需要修改上层代码 因为接口没有变动,为层的实现提供了统一的标准 方便 实现不同实现类

**工厂+配置文件：**

基于配置来实现产生接口的实现类,由工厂来根据配置来生产接口的具体实现类对象，在底层实现类变化时，可以不需要修改代码 只需要改配置即可,单例 泛型 反射

**Web.day20 filter+全站乱码处理**

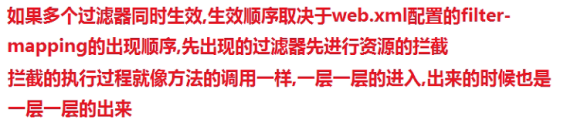
**1.Filter 过滤器概述**

是**javaEE三大组件**之一:Servlet Filter Listener

可以拦截对资源的访问 拦截下来后可以是否允许通过 或者 在允许通过时 在之前或之后做一些额外的操作

所谓的**拦截** 其实就时将代表请求的request对象和代表响应的response对象拦截下来 进行控制

一个过滤器可能拦截**多个**资源 一个资源也可能被**多个**过滤器所拦截,这种多个过滤器拦截一个资源的模式 称之为 **责任链模式**



**2.过滤器开发**

**步骤：**

1)写一个类实现Filter接口

2)在web.xml配置过滤器

**Filter接口**

1)init(FilterConfig):初始化的方法 当Filter被初始化时 调用此方法 执行初始化操作

2)doFilter(ServletReqeust,ServletResponse,FilterChain):核心方法 在存活期间 过滤器拦截到对资源的访问 会造成此方法的执行 需要在这个方法中设计过滤器的核心逻辑代码.FilterChain:代表拦截当前资源的过滤器链对象，一个资源可能被多个过滤器所拦截，拦截的顺序取决于Filter的<filter-mapping>的配置顺序。那么按照过滤器的拦截顺序，就组成一个拦截的链，按照顺序取连接资源。FilterChain提供了一个叫doFilter的方法，一旦调用就表示放行当前过滤器，执行过滤器链的下一个节点，如果下一个节点仍然是一个过滤器，则接着进行过滤器，如果后续没有其他过滤器放行到最重要访问的资源。

3)destory():销毁方法 在Filter被销毁之前调用 实现善后操作

**配置过滤器**

<filter> -- 配置一个过滤器

<filter-name>FirstFilter</filter-name> -- 过滤器的名字

<filter-class>com.tarena.filter.FirstFilter</filter-class> -- 过滤器的类

</filter>

<filter-mapping> -- 过滤器的拦截路径配置,可以配置多个

<filter-name>FirstFilter</filter-name> -- 为哪个名字的过滤器配置

<url-pattern>/\*</url-pattern> -- 拦截哪个路径资源可以配置多个

<servlet-name>XxxServlet</servlet-name> -- 拦截哪个名字的Servlet

<dispatcher></dispatcher> -- 指定过滤器拦截哪种方式对资源的访问，可以取值为REQUEST FORWARD INCLUDE ERROR，如果不配置，默认只拦截REQUEST方式的访问。可以配置多个。

</filter-mapping>

**4.filter的生命周期**

当服务器启动,web应用加载后,立即创建出这个web应用中的所有过滤器对象,创建出来后立即调用过滤器的init方法执行初始化操作.从此这些过滤器对象驻留在内存中为后续的拦截进行服务.每当拦截到资源时,都会导致dofilter方法执行.最终直到服务器关闭或web应用移除出容器时,随着web应用的销毁,过滤器对象销毁,销毁之前调用destory方法执行善后工作。

**5.细节**

如果一个资源被多个过滤器所拦截 多个过滤器的拦截顺序 取决与web.xml中filter-mapping 配置的顺序

多个过滤器的执行 类似于方法一层一层调用的过程 一层一层往里钻 再一层一层往外出

**6.案例:**全站乱码解决过滤器

30天内自动登录

**7.继承:**只能先继承再创建对象 如果用原来的类创建对象 对这些对象没有影响

**装饰:**可以改造已有对象身上的方法 但是当原来对象身上的方法 比较多的时候 开发起来比较麻烦

**动态代理:**可以改造已有对象身上的方法 并且 只需要改造要改造的方法 其他方法 不需要额外编写 非常方便

**8.MD5加密算法**

数据摘要算法 - 数据指纹算法,任意长度二进制计算出128位二进制的摘要信息 通常转换为32位的16进制来表示

明文相同算出的MD5一定相同

明文不同算出的MD5一定不相同(有可能相同 只是几率非常非常非常的低 一般认为是唯一的)

只能由明文算成密文 永远无法从密文算成明文

MD5在计算机领域有大量的应用：

加密存储数据

文件的完整性验证

数字签名等等。。。

**Web.day21.Listener vs文件上传**

**1.Listener 监听器**

Servlet技术规范中sun预先定义了八大监听器用来监听web开发中对应的事件,监听器可以用来监听web应用执行过程中相关的事件 来进行对应的处理

**开发监听器步骤**

写一个类实现指定监听器接口

在web.xml文件中配置监听器

<listener>

<listener-class>cn.tedu.listener.MyXxxListener</listener-class>

</listener>

**监听三大作用域创建和销毁的监听器**

**1)ServletContextListener**

该监听器负责监听ServletContext的创建和销毁

生命周期：在web应用启动时创建出来 之后一直驻留在内存中唯一的代表当前web应用 直到web应用移除出容易或服务器关闭时 随着web应用的销毁ServletContext对象跟着被销毁

作用范围：整个web应用范围

主要功能：在整个web应用范围内 整个web应用存活期间 共享数据

**2)HttpSessionListener**

负责监听HttpSession对象的创建和销毁事件

生命周期：在第一次调用request.getSession()时创建 自杀 - session.invalidate 超时 - 30分钟 意外身亡 - 服务器非正常关闭

作用范围：整个会话

主要功能：在整个会话范围内 整个会话的生命周期期间 共享数据

钝化：在服务器正常关闭时 仍然在存活期间的session 会被序列化后保存在tomcat的work目录下 这个过程称之为session的钝化

活化：在服务器正常启动时 会将钝化的session再恢复到内存中 继续使用 这个过程称之为session的活化

**3)ServletRequestListener**

生命周期：请求开始时创建 请求结束时销毁

作用范围：整个请求

主要功能：在请求过程中 在请求范围内 共享数据

**监听三大作用域中属性增加、修改或减少的监听器**

**4)ServletContextAttributeListener**

**5)HttpSessionAttributeListener**

**6)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。

void attributeAdded(ServletContextAttributeEvent scae) 添加属性时调用

void attributeReplaced(ServletContextAttributeEvent scae) 替换属性时调用

void attributeRemoved(ServletContextAttributeEvent scae) 删除属性时调用

使javabean自己感知 自己在session域中状态变化的监听器

**7)HttpSessionBindingListener** - 使javabean自己感知自己在session域中 被加入 或 移除的 状态变化的监听器

**8)HttpSessionActivationListener** - 使javabean自己感知自己在session域中 随着session被钝化 活化 状态变化的监听器

\*\*这两个监听器比较特殊 不需要单独写类来实现 也不需要在web.xml中进行配置 只需要让javabean自己来实现即可

EasyMall中监听器的应用

开发EasyMallServletContextListener 在ServletContext创建时 向 ServletContext域中增加 app 属性 值为应用的路径 在ServletContext销毁时删除该属性 这样 以后需要在jsp页面中写应用路径时 直接写$(app)即可

**2.文件上传**

在web开发中经常需要从客户端向服务器上传文件，如：上传照片、上传新闻图片、上传附件等等。这些都需要通过WEB开发中的文件上传技术实现。

**文件上传步骤**

提供一个带有文件上传项的表单

文件上传的输入框必须有name属性才能被上传

文件上传的表单必须是post提交

文件上传的表单必须设置enctype=multipart/form-data

在servlet中读取处理上传的文件，保存到服务器中。

方式1：

可以通过javaee原生的api request.getInputStream 来获取请求中实体内容的流,获取到流中的数据后,解析处理 -- 非常麻烦

方案2：

使用开源工具实现

Apache提供的文件上传包 - Commons-fileupload

**上传组件(Apache commons-fileupload)使用过程(1),(2),(3)**

下载并导入对应的包

Commons-fileupload

commons-io

**(1)文件上传工厂 DiskFileItemFactory**

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

sizeThreshold - 指定内存缓冲区的大小

repository - 指定临时文件存放的位置

文件上传 时需要将请求的实体内容 全部读取后 才能做处理

此时需要将实体内容缓冲起来 内存缓冲快 但是耗费内存 文件缓冲慢 但是可以存放大量数据

所以此处提供了两个选项 如果数据大小小于sizeThreshold则使用内存做缓冲 速度快 如果文件大小超过了该限制则在repository指定的位置下创建临时文件来缓冲数据

**(2)上传文件核心类 ServletFileUpload**

ServletFileUpload fileUpload = new ServletFileUpload(factory);

//判断当前提交数据表单是否是一个enctype为multipart/form-data类型的表单

boolean isMultipartContent(HttpServletRequest request)

//指定处理文件名时使用的编码集

setHeaderEncoding(java.lang.String encoding)

//控制单个文件大小最大值

setFileSizeMax(long fileSizeMax)

//控制总的文件大小最大值

setSizeMax(long sizeMax)

//解析request对象 获取 FileItem的集合

List<FileItem> parseRequest(HttpServletRequest request)

//设置上传文件监听器

setProgressListener(ProgressListener pListener)

**(3)对list进行迭代，得到文件上传项 FileItem，并判断是否是上传文件。**

List<FileItem> parseRequest(HttpServletRequest request)

//判断当前FileItem是不是一个普通字段项 如果返回true表示这是一个普通字段项 返回false表示是一个文件上传项

boolean isFormField()

//如果是普通字段项

String getFieldName() //获取字段项的名称

String getString() //获取字段项的值

String getString(String encode) //获取字段项的值

//如果是文件上传项

String getName() //获取文件名

InputStream getInputStream() //获取文件内容的流

delete() //删除临时文件

**上传文件注意问题**

**1)ie上传文件 文件名 Bug 处理:当ie上传文件时，文件名称会携带路径。**

if(fname.contains("\\")){

fname = fname.substring(fname.lastIndexOf("\\"));

}

**2)文件上传保存位置问题**

文件上传保存的位置一定不能被外界直接访问 防止 用户用户浏览器访问 下载资源 或执行jsp恶意代码

要么保存在WEB-INF下保护起来 要么放在本地磁盘 其他位置 保证通过浏览器无法直接访问

**3)上传文件重名问题**

多个上传名称相同时 文件会发生覆盖

应该想办法让文件名 尽量不要重复 - 在文件名的前面拼接UUID来保证文件名绝对不会重复

**4)上传文件目录下文件过多 - hash分目录存储**

一个文件夹下文件过多，会造成访问缓慢，甚至有可能无法访问。所以，应该想办法将这些文件分目录存储。

分目录存储的算法可以有很多：

(1).按照上传时间进行目录分离 （周、月 ）

(2).按照上传用户进行目录分离 ----- 为每个用户建立单独目录

(3).按照固定数量进行目录分离 ------ 假设每个目录只能存放3000个文件 ，每当一个目录存满3000个文件后，创建一个新的目录

**(4).按照唯一文件名的hashcode 进行目录分离**

计算文件名的hash - 散列

将hash值转换为16进制的字符串 - 共8位 值0-9 a-f

将这个hash字符串截取出每一位作为一级目录

最终将上文件放置到8级hash目录下，共有16^8个可能的文件夹分目录存放文件，从而保证一个文件夹文件不会过多

**文件上传进度监听**

upload.jsp页面中 为表单提交事件增加函数触发 增加定时器 每隔一段时间 通过ajax发送是那个请求 访问服务器 获取最新的文件上传进度 展示到页面

$(document).ready(function(){

$("#f").submit(function(){

window.setInterval(function(){

$.get("${pageContext.request.contextPath }/servlet/UploadProgressServlet", function(data){

$("#in\_div").width(data);

});

}, 10);

})

});

UploadServlet中 为问件上传注册监听器 计算各种指标 并且将计算的指标 存入session

UploadProgressServlet中 从session中获取上传进度信息 作为响应发回给ajax访问

**Web.day22.事务+商品添加**

**1.事务的概念**

事务是逻辑上的一组操作 要求这一组操作要么同时完成 要么同时不完成。

**2.事务操作**

数据库默认就有事务，mysql中默认情况下一条语句独占一个事务。

也可以通过手动控制事务：

**sql控制事务：**

start transaction; #开启一个事务 这条语句之后的所有的sql都将处在同一个事务中。这些sql都不会立即起作用 - 没有改变底层真正的数据。

commit; #提交事务 这个事务中所有的sql对数据库的影响立即发生 - 真正改变底层数据。

rollback; #回滚事务 撤销这个事务中所有的操作可能的对数据库的影响。

**JDBC控制事务：**

conn.setAutoCommit(false);#开启了事务，接下来在这个连接执行的所有sql都将处在同一个事务当中。

....

conn.commit();#提交事务，将这个连接上执行的事务提交，对数据库产生影响。

conn.rollback();#回滚事务，取消这个连接上执行的事务。

要注意，回滚到回滚点时，回滚点之前的代码仍然是未提交也未回滚的状态，如果希望对数据库产生影响仍然需要进行提交的操作。

**3.事务的四大特性 - ACID**

**原子性（Atomicity）** - 事务中的一组操作是不可分割的一个整体，要么一起成功，要么一起失败。

**一致性（Consistency）** - 事务前后 无论事务是否成功 数据库应该都保持一个完整性的状态。

数据库中数据完整性:数据库中数据 是业务完整 且约束完整的。

业务完整: 事务的所有操作之前,A账户+B账户是2000元,那么事务操作之后,A账户+B账户还应该是2000元

约束完整: 事务的所有操作结束后,不会破坏之前的约束

**隔离性（Isolation）** - 多个并发事务之间应该互相隔离 互不影响

**持久性（Durability）** - 一个事务成功 对数据库产生的影响是永久性的 无论发生什么情况 这种影响都不会被取消

**4.隔离性的问题**

数据库的其他三大特性数据库可以帮我们保证，而隔离性我们需要再讨论。我们知道数据库的隔离性问题本质上就是多线程并发安全性问题。可以用锁来解决多线成并发安全问题，但是如果用了锁，必然会造成程序的性能大大的下降.对于数据库这种高并发要求的程序来说这是不可接受的.

**加锁** - 同一时间内只能有一个人操作数据 - 完美的保证隔离性 - 但是这样一来数据库就相当于工作在单线程的状态下 同一时间只能有一个事务操作 并发的效率非常低下

分析下隔离性产生的细节：

如果两个线程并发修改，必然产生多线程并发安全问题，必须隔离开

如果两个线程并发查询，必然没有问题，不需要隔离

如果一个线程修改，一个线程查询，在不同的应用场景下有可能有问题，有可能没问题。

**5.四大隔离级别**

演示不同隔离级别下的并发问题

select @@tx\_isolation 查询当前事务隔离级别

set [session/global] transaction isolation level xxxxxx;

global设置的是数据库服务器默认响应连接时使用的数据库隔离级别

session设置的是当前会话使用的数据库隔离级别

1).当把事务的隔离级别设置为read uncommitted时，会引发脏读、不可重复读和虚读

**脏读**:一个事务读取到了另一个事务未提交的数据

A窗口

set transaction isolation level read uncommitted;

start transaction;

select \* from account;

-----发现a帐户是1000元，转到b窗口

B窗口

start transaction;

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

-----不要提交，转到a窗口查询

select \* from account

-----发现a多了100元，这时候a读到了b未提交的数据（脏读）

2).当把事务的隔离级别设置为read committed时，会引发不可重复读和虚读，但避免了脏读

**不可重复读**：一个事务读取到另一个事务已经提交的数据

A窗口

set transaction isolation level read committed;

start transaction;

select \* from account;

-----发现a帐户是1000元，转到b窗口

B窗口

start transaction;

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

commit;

-----转到a窗口

select \* from account;

-----发现a帐户多了100,这时候，a读到了别的事务提交的数据，两次读取a帐户读到的是不同的结果（不可重复读）

3).当把事务的隔离级别设置为repeatable read(**mysql默认级别**)时，会引发虚读，但避免了脏读、不可重复读

**虚读(幻读)**：一个事务读取全表数据时 读取到另一个事务向表中新增、删除操作提交的结果

\*\*虚读(幻读)问题 有可能出现 有可能不出现 概率非常低

A窗口

set transaction isolation level repeatable read;

start transaction;

select \* from account;

----发现表有4个记录，转到b窗口

B窗口

start transaction;

insert into account(name,money) values('ggg',1000);

commit;

-----转到 a窗口

select \* from account;

----可能发现表有5条记如，这时候发生了a读取到另外一个事务插入的数据（虚读）

4).当把事务的隔离级别设置为Serializable时，会避免所有问题

本质上是靠**锁**来实现的

**从安全性说：**Serializable > Repeatable read > Read committed > Read uncommitted

**从效率说：**Read uncommitted >Read committed > Repeatable read > Serializable

真正在开发的过程中 应该根据自身的业务需求 选择一个 在能防止想要防止的隔离性问题的级别中 性能 最好的一个隔离级别

mysql的默认隔离级别时 Repeatable read

A窗口

set transaction isolation level Serializable;

start transaction;

select \* from account;

-----转到b窗口

B窗口

start transaction;

insert into account(name,money) values('ggg',1000);

-----发现不能插入，只能等待a结束事务才能插入

**6.数据库中的锁机制**

数据库中是有锁的 但是锁 如果控制不好 对效率影响非常大 所以数据库设计者 对锁做了特别的设计：

两个查询 --> 没有必要互斥

两个修改 --> 必须互斥

一个查询 另一个 修改 --> 具体看情况 Serializable隔离级别下需要排斥 其他隔离级别不需要

**共享锁**

共享锁和共享锁可以共存 共享锁和排他锁不能共存

1在非Serializable级别中查询不加任何锁

2在Seralizable级别中查询加共享锁

**排他锁**

排他锁和任何锁都不能共存

在任意隔离级别下做增删改都加排他锁

**死锁：**

当两边都时Serializable隔离级别时

两边都先进行查询 再尝试进行修改 则互相等待对方释放共享锁 都无法接着执行 造成了死锁

死锁的解决有两种办法：避免死锁 解决死锁

mysql可以自动检测到死锁,发现死锁后,错误退出一方,执行另一方来解决了死锁。

**7.案例easymall中商品模块的添加,事务控制**

事务加在哪里?

Service不能用Connection，又要控制事务，矛盾了，怎么办？

有些耦合性无法避免 --> 想办法管理起来 --> 将所有控制事务的代码整合到一个工具类中来进行管理

开发TransactionManager，在其中管理Connection，并对外提供getConn、startTran、commitTran、rollbackTran、release方法。之后所有和事务 相关的操作都不要直接使用Conn 而是通过TransactionManager来实现管理

解决耦合性的问题 - 本质上是将耦合转移到了TransactionManager中同一管理。虽然没有彻底的解决耦合，但是统一管理起来，方便未来开发和维护。

删除商品--事务

ThreadLocal - 线程本地变量

TransactionManager中的Connection如果管理？

如果所有的线程都使用同一个Connection 则事务混乱 - 多线程安全问题

方案1：在类的内部通过 静态Connection 加锁 的方式来管理 - 可以解决问题 但是 所有事务排队 效率非常低

方案2：通过ThreadLocal编码 实现 所有的线程都各自携带各自的Connection对象 从而让管理各自的事务 - 不会有阻塞 效率高

**8.ThreadLocal:本地线程变量(重要)**

Thread对象内置了一个Map来存取消息，但是这个map外界无法直接操作，需要通过ThreadLocal来实现对Thread中的Map进行数据的存取。

本质上是一种利用线程的执行过程由程序的上游 向程序下游传递信息的机制,又由于 而每个线程都可以保存自己对应的对象 后续获取来使用 每个线程给自有各自的对象 自然就不会有线程安全问题了

**ThreadLocal方法**

ThreadLocal tl = new ThreadLocal();

tl.set(obj); --向当前线程中的map保存对象，key为当前线程变量，值为存入的数据。

tl.get(); --从当前线程中获取对象，如果获取不到对象，调用initialValue()创建一个新的对象。

tl.remove(); --从当前线程中删除对象。

initialValue() --创建对象。

**Web.day23-24事务 vs 更新丢失**

1.更新丢失

两个并发的事务 基于同一个查询结果进行更新操作 后提交的事务 忽略了先提交的事务对数据库的影响 造成的问题 称之为更新丢失问题

例子：重复充值,秒杀抢购

**解决方案：**

将数据库隔离级别设置为Serializable就可以天然防止更新丢失问题的产生 - 但是很少这样解决 因为 效率太低

**悲观锁**

悲观锁悲观的认为 每次查询都会造成更新丢失 所以在查询时 手动添加排它锁 排斥 查询 从而解决更新丢失问题

select \* from xxx for update;#for update在查询时手动增加了排他锁

**乐观锁**

乐观锁乐观的认为 每次查询都不会造成更新丢失 在修改时 检测更新丢失的发生来进行纠正

**悲观锁 和 乐观锁 都不是数据库中真正存在的锁 而是两种解决方案的名字**

**如果 查询多 而 更新少 用 乐观锁**

**如果 更新多 而 查询少 用 悲观锁**

演示不同隔离级别下的并发问题

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

实现装饰设计模式:

（1）写一个装饰类, 要求装饰类和被装饰者所属的类实现同一个接口或者继承同一个父类

（2）装饰类必须提供构造方法接收被装饰者, 并将被装饰者保存在类的内部

（3）对于想要改造的方法直接进行改造, 对于不想改造的方法, 直接调用原有对象(被装设者)上的方法