# JAVA基础学习日记-2

# --19级学生沈章 记

# Day01:HTML基础学习

## 课1：HTML(概述&演示)

Html内容：

1. Html就是超文本标记语言的简写，是最基础的网页语音。
2. Html是通过标签来定义的语音，代码都是由标签所组成。
3. Html代码不用区分大小写。
4. Html代码由<html>开始</html>结束。里面由头部分<head></head>和体部分<body></body>两部分组成。
5. 头部分是给Html页面增加一些辅助或者属性信息，它里面的内容会最先加载。
6. 体部分是真正存放页面数据的地方。

多数标签都是有开始标签和结束标签，其中有个别标签因为只有单一功能，或者没有要修饰的内容可以在标签内结束。

想要对被标签修饰的内容进行更丰富的操作，就用到了标签中的属性，通过对属性值的改变，增加了更多的效果选择。

属性与属性值之间用“=”连接，属性值可以用双引号或单引号或者不用引导，一般都会用双引号。或公司规定书写规范。

格式： <标签名 属性名=’属性值’>数据内容</标签名>

<标签名 属性名=’属性值’/>

## 课2：HTML(标签的操作思想)

操作思想：

为了操作数据，都需要对数据进行不同标签的封装，通过标签中的属性对封装的数据进行操作，标签就相当于一个容器。对容器中的数据进行操作，就是在不断的改变容器的属性值。

## 课3：HTML(常用工具)

<hr/>：horizontal rule水平分隔线。

<hx>分级标题</hx>

&xxx; 特殊符号

小技巧：为html某一行加注释ctrl+shift+c。

## 课4：HTML(列表标签)

<!-- --演示HTML注释：列表标签This is my HTML page.

列表标签：dl

上层项目：dt

下层项目：dd：封装的内容是会被缩进的。有自动缩进效果。

-->

<dl>

<dt>上层项目内容</dt>

<dd>下次项目内容</dd>

<dd>下次项目内容</dd>

</dl>

<hr/>

<!-- 有序和无序的项目列表

有序：<ol>

无序：<ul>

无论有序和无序，条目的封装用的都是<li>

而且它们都有缩进效果

-->

<ol type=*"a"*>

<li>有序项目列表</li>

<li>有序项目列表</li>

<li>有序项目列表</li>

<li>有序项目列表</li>

</ol>

<ul type=*"disc"*>

<li>无序项目列表</li>

<li>无序项目列表</li>

<li>无序项目列表</li>

<li>无序项目列表</li>

</ul>

## 课5：HTML(图像标签)

5.图像标签：<img>

<!-- 演示图片标签

img. -->

<img src=*"imgs\1.jpg"* height=*"350"* width=*"500"* border=*"10"* alt=*"图片说明文字"*/>

Align:属性定义图片的排列方式，border用来设置图像的边框。Src连接一个文件。

6.图像地图：<map>

应用：当腰在图像中选取某一部分作为连接的时候。如：中国地图每个省所对应的区域。

Map标签要和img标签联合使用。Href是超链接。

<img src=”Sunset.jpg” alt=”图片说明文字” usemap=”#Map” />

<map>

<area shape=”rect” cords=”50,59,116,104” href=”1.html” />

<area shape=”circle” cords=”118,203,40” href=”2.html” />

</map>

## 课6：HTML(表格标签)

7.表格标签<table>

组成：

标题标签：<caption>给表格提供标题。

表头标签<th>：一般对表格的第一行或者第一列进行格式化，就是粗体显示。并不常用。

行标签：<tr>

单元格标签：<td>：加载行标签的里面。可以简单理解为，先有行，在行中再加入单元格。

<table border=*1* bordercolor=*"#0000dd"* cellpadding=*10* cellspacing=*1* width=*500*>

<caption>表格标题</caption>

<tr>

<th>姓名：</th>

<th>年龄：</th>

</tr>

<tr>

<td>王五</td>

<td>44</td>

</tr>

</table>

## 课7：HTML(表格标签\_2)

<table border=*10* bordercolor=*"#0eeeea"* cellpadding=*20* cellspacing=*0* width=*500*>

<tbody><!-- 表格的下一级标签是tbody，不定义也存在 -->

<tr>

<th rowspan=*2*>个人信息</th>

<td>张三</td>

</tr>

<tr>

<td>30</td>

</tr>

</tbody>

</table>

<hr/>

<table border=*1* bordercolor=*"#0000dd"* cellpadding=*20* cellspacing=*10* width=*500*>

<tr>

<th colspan=*2*>个人信息</th>

</tr>

<tr>

<td>张三</td>

<td>30</td>

</tr>

</table>

THEAD、TFOOT包含关于表格列的信息。

TBODY作用是：可以控制表格分行下载，从而提高下载速度。（网页的打开是先，表格的内容全部下载完毕后，才显示出来，分行下载可以先显示部分内容，这样会减少用户等待时间。）

使用TBODY的目的是可以使得这些包含在内的代码不用在整个表格都解析后一起显示，就是说如果有多个行，那么如果得到一个TBODY行，就可以先显示一行。

TBODY这个标签可以控制表格分行下载，当表格内容很大时比较实用，在需要分行下载处加上<tbody></tbody>。

## 课8：HTML(超链接\_1)

超链接标签：<a>

两种用法：

1. 超链接<a href=””>
2. 定位标记<a name=””>
3. 一般在本页中实用，当网页内容过长，定位标记会比拖动滚动条方便快捷。

注：定位标记要和超链接结合使用才有效。

<a href=”#标记>返回标记位置</a>

注：使用定位标记时一定在href值开始加入#标记名。

### 特殊的目标

有 4 个保留的目标名称用作特殊的文档重定向操作：

#### \_blank

浏览器总在一个新打开、未命名的窗口中载入目标文档。

#### \_self

这个目标的值对所有没有指定目标的 <a> 标签是默认目标，它使得目标文档载入并显示在相同的框架或者窗口中作为源文档。这个目标是多余且不必要的，除非和文档标题 <base> 标签中的 target 属性一起使用。

#### \_parent

这个目标使得文档载入父窗口或者包含来超链接引用的框架的框架集。如果这个引用是在窗口或者在顶级框架中，那么它与目标 \_self 等效。

#### \_top

这个目标使得文档载入包含这个超链接的窗口，用 \_top 目标将会清除所有被包含的框架并将文档载入整个浏览器窗口。

**提示：**这些 target 的所有 4 个值都以下划线开始。任何其他用一个下划线作为开头的窗口或者目标都会被浏览器忽略，因此，不要将下划线作为文档中定义的任何框架 name 或 id 的第一个字符。

## 课9：HTML(超链接\_2)

<!-- 定位标记：

专业术语：锚 -->

<a name=*top*>顶部位置</a>

<hr/>

<img src=*"111.jpg"* height=*"500"* width=*"400"* border=*"10"*/>

<hr/>

<a name=*center*>中间位置</a>

<hr/>

<img src=*"111.jpg"* height=*"500"* width=*"400"* border=*"10"*/>

<a href=*"#top"*>顶部位置</a>

<a href=*"#center"*>中间位置</a>

## 课10：HTML(框架)

1. 框架标签<frameset>

注：框架标签不可以放到<body>，一般为了代码的可读性，会到<head>和<body>之间。

例：

<frameset rows=”10%,\*”>

<frame src=”1.html name=”top” />

<frameset rows=”30%,\*”>

<frame src=”2.html name=”left” />

<frame src=”3.html name=”right” />

</frameset>

</frameset>

这段代码会需要已经存在的3个html页面，分别是：1.html，2.html，3.html。

<!--

定义框架，使用标签frameset。

-->

<frameset rows=*"30%,\*"*>

<frame src=*"top.html"* name=*"top"*/>

<frameset cols=*"30%,\*"*>

<frame src=*"left.html"*name=*"left"*/>

<frame src=*"right.html"*name=*"right"*/>

</frameset>

</frameset>

## 课11：HTML(画中画标签)

<iframe src=*"table.html"* height=*"600"* width=*"400"*>这是画中画标签，您如果看到该文字，很遗憾，您的浏览器不支持该标签</iframe>

## 课12：HTML(表单组件-input)

表单标签：<form>

表单标签是最常用的标签，用于与服务器端的交互。

<input>：输入标签：就收用户输入信息。

其中的type属性指定输入标签的类型。

1文本框：text。输入的文本信息直接显示在框中。

2密码框password。输入的文本以原点或者星号的形式显示。

3单选框radio。如：性别选择。（带有预选定复选框属性是checked）

4复选框checkbox。如：兴趣选择。

5隐藏字段hidden在页面上不显示，但在提交的时候随其他内容一起提交。

6提交按钮submit用于提交表单中的内容，提交给属性action所指向的文件。

7图片提交image

8重置按钮reset

9按钮button

10文件域file

每一个表单的组件都必须有name和value。

<!-- 如果要给服务端提交数据，表单中的组件必须有name和value属性。

用于给服务端获取数据方便。

-->

<form action=*""*>

输入名称：<input type=*"text"*/><br/>

输入密码：<input type=*"password"*/><br/>

<!-- 放置在同一组中，需要定义属性name相同 -->

选择性别：<input type=*"radio"* name=*"sex"*/>男

<input type=*"radio"* name=*"sex"* checked=*"checked"*/>女<br/>

所会技术：<input type=*"checkbox"*/>JAVA

<input type=*"checkbox"*/>HTML

<input type=*"checkbox"*/>CSS<br/>

选择文件：<input type=*"file"* name=*"file"*/><br/>

一个图片：<input type=*"image"* src=*"11.jpg"*/><br/>

<!-- 数据不需要客户端知道，但是可以将其提交到服务器端 -->

隐藏组件：<input type=*"hidden"* name=*"mike"* value=*"myvalue"*/><br/>

<!-- 自定义按钮 -->

一个按钮：<input type=*"button"* value=*"旺财"* onclick="alert('哈哈')"/><br/>

<input type=*"reset"* value=*"清除数据"*/>

<input type=*"submit"* value=*"提交数据"*/>

</form>

## 课13：HTML(表单组件-select&textarea)

<select name=*"country"*>

<option value=*"none"*>--选择国家--</option>

<option value=*"usa"*>美国</option>

<option value=*"english"*>英国</option>

<option value=*"china"* selected=*"selected"*>中国</option>

</select>

<textarea rows=*""* cols=*""* name=*"text"*></textarea><br/>

## 课14：HTML(表单格式化)

<form action=*""*>

<table border=*"20"* bordercolor=*"#00fffe"* cellpadding=*"10"* cellspacing=*"0"* height=*"200"* width=*"800"*>

<tr>

<th colspan=*2*>注册表单</th>

</tr>

<tr>

<td>用户名称：</td>

<td>

<input type=*"text"* name=*"user"*/>

</td>

</tr>

<tr>

<td>输入密码：</td>

<td>

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

</td>

</tr>

<tr>

<td>确认密码：</td>

<td>

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

</td>

</tr>

<tr>

<td>选择性别：</td>

<td>

<input type=*"radio"* name=*"sex"* value=*"male"*>男</input>

<input type=*"radio"* name=*"sex"* value=*"female"*>女</input>

</td>

</tr>

<tr>

<td>选择技术：</td>

<td>

<input type=*"checkbox"* name=*"tech"* value=*"JAVA"*>JAVA</input>

<input type=*"checkbox"* name=*"tech"* value=*"HTML"*>HTML</input>

<input type=*"checkbox"* name=*"tech"* value=*"CSS"*>CSS</input>

</td>

</tr>

<tr>

<td>选择国家：</td>

<td>

<select name=*"country"*>

<option value=*"none"*>--选择国家--</option>

<option value=*"USA"*>美国</option>

<option value=*"England"*>英国</option>

<option value=*"China"*>中国</option>

</select>

</td>

</tr>

<tr>

<th colspan=*2*>

<input type=*"reset"* value=*"清除数据"*/>

<input type=*"submit"* value=*"提交数据"*/>

</th>

</tr>

</table>

## 课15：HTML(GET和POST区别)

GET提交和POST提交的区别？

1，

get提交，提交的信息都显示在地址栏中。

Post提交，提交的信息不显示在地址栏中。

2，

get提交，对于敏感的数据信息不安全。

post提交，对于敏感的数据信息安全。

3，

get提交，对于大数据不行，因为地址栏存储体积有限。

post提交，可以提交大体积数据。

4，

get提交，将信息封装到了请求消息的请求行中。

post提交，将信息封装到了请求体中。

在服务端的一个区别。

如果出现将中文提交到tomcat服务器，服务器默认会用iso8859-1进行解码会出现乱码，

通过iso8859-1进行编码，在用指定的中文码表解码，即可。

这种方法对get提交和post提交都有效。

但是对于post提交方法提交的中文，还有另一种解决办法，就是直接使用服务端一个对象

request对象的setCharacterEncoding方法直接设置指定的中文码表就可以将中文数据解析出来。

这个方法只对请求体中的数据进行解码。

综上所述：表单提交，建议使用post。

和服务端交互的三种方式：

1. 地址栏输入url地址。get
2. 超链接。get
3. 表单。get和post。

## 课16：HTML(服务端GET和POST区别)

## 课17：HTML(服务端和客户端校验的问题)

如果在客户端进行增强型的校验，只要一个组件内容是错误的，是无法提交的。只有全对才可以提交。

问，服务器端数据后，还需要校验码？

需要，为了安全性。

如果服务端做了增强型的校验，客户端还需要校验码？

需要，因为要提高用户的上网体验效果，减轻服务器端的压力。

## 课18：HTML(常见的其他标签)

#### 头标签

头标签都放在<head></head>头部分之间。包括：title base meta link

<title>:指定浏览器的标题栏显示的内容。

<base>:

href属性：指定网页中所有的超链接的目录。可以是本地目录，也可以是网络目录。注意值的结尾处一定要用/表示目录。只作用于相对路径的超链接文件。

Target属性：指定打开超链接的方式，如\_blank表示所有的超链接都用新窗口打开显示。

<meta>:

name属性：网页的描述信息。当取keywords时，content属性的内容就作为搜索引擎的关键字进行搜索。

http-equiv属性：模拟HTTP协议的相应消息头。

例：

<meta http-equiv=”refresh” content=”3;url=http://www.sina.com.cn” />表示打开此页面3秒后自动转到新浪页面。

<link>:

ref属性：描述目标文档与当前文档的关系。

type属性：文档类型。

media：指定目标文档在哪种设备上起作用。

例：

<link ref=”stylesheet” type=”text/css” meda=”screen,print” href=”a.css” />

#### Other

<marquee>让内容动起来。

direction属性：left right down up

behavior属性：scroll alternate slide

<pre>:可以将文本内容按在代码区的样子显示在页面上。

XHTML是The Extensible HyperText Markup Language可扩展标识语言的缩写。目前推荐遵循的是W3C于2000年1月26日推荐XML1.0（参考[http://www.w3.org/TR/xhtml1](http://www.w3.org/TR/xhtml1" \t "https://zhidao.baidu.com/question/_blank)）。XML虽然数据转换能力强大，完全可以替代HTML，但面对成千上万已有的站点，直接采用XML还为时过早。因此，我们在HTML4.0的基础上，用XML的规则对其进行扩展，得到了XHTML。简单的说，建立XHTML的目的就是实现HTML向XML的过渡。

代码结构更严谨，是基于XML的一种应用。

XML是可扩展标记语言（Extensible Markup Language）

（标签可以自定义）

XML是对数据信息的描述。HTML是数据显示的描述。

XML代码规定的更为严格，如：标签不结束被视为错误。

XML规范可以被更多的应用程序所解释，将成为一种通用的数据交换语音。

各个服务器，框架都讲XML作为配置文件。

## 课19：HTML(标签的分类)

<body>

<div>这是一个div区域1</div>

<div>这是一个div区域2</div>

<span>span区域1</span>

<span>span区域2</span>

<p>这是一个段落1</p>

<p>这是一个段落2</p>

<!-- 标签分为两大类

1，块级标签（元素）：标签结束后都有换行。div p dl ol ul table title

2，行内标签（元素）：标签救赎后没有换行。span font img select input a

-->

<dl>

<dt>dddd</dt>

<dd>eeee</dd>

</dl>

hahaha

</body>

## **Day02:**

### 课1：CSS(概述&和html结合的方式一二)

CSS是层叠样式表（Cascading Style Sheets）用来定义网页的现实效果。可以解决html代码对样式定义的重复，提高了后期样式代码的可维护性，并增强了网页的现实效果功能，简单一句话：CSS将网页内容和显示样式进行分离，提高了显示功能。

那么CSS和Html是如何在网页代码中相结合的呢？

通过四种方式：

1. style属性方式：
2. style标签方式（内嵌方式）：

<!--

csshe html相结合的第一种方式。

1，每一个html标签中都有一个style样式属性。该属性的值就是css代码。

2，使用标签的方式。

-->

<div style="background-color:#06F;color:#FFF">这是一个div区域1</div>

<div>这是一个div区域2</div>

3，@import url(div.css);

4，

### 课2：CSS(和html结合的方式三四)

5, 样式优先级：

由上到下，由外到内，优先级由低到高。

6，总结CSS代码格式

选择器名称{属性名：属性值；属性名：属性值......}

属性与属性之间用分号隔开

属性与属性值直接按用冒号连接

如果一个属性有多个值的话，那么多个值用空格隔开。

### 课3：CSS(基本选择器&优先级)

1. 选择器：

就是知道CSS要作用的标签，那个标签的名称就是选择器。

选择器共有三种：

A)html标签名选择器，使用的就是html的标签名。

B)class选择器，其实使用的标签中的class属性。

C)id选择器，其实使用的是标签中的id属性。

每一个标签都定义了class属性和id属性。用于对标签进行标识，方便对标签进行操作，

在定义中，多个标签的class属性值可以相同，而id值要唯一，因为JavaScript中进场用。

/\* 预定样式，实现动态的加载。 \*/

*.haha*{

background-color: *#FF3*;

color: *#0C0*

}

/\* 退出id的取值在 页面中是唯一的，因为该属性除了给css使用，还可以被js使用。id通常是为了去标示页面中一些特定区域使用的。 \*/

*#qq*{

background-color: *#000*;

color: *#FFF*

}

<!-- 优先级：

标签选择器<类选择器<id选择器<style属性

-->

<div id=*"qq"*>这是一个div区域1</div>

<div class=*"haha"* id=*"qq"* style="background-color: *#F00*; color: *#000*">这是一个div区域2</div>

<span>span区域1</span>

<span class=*"haha"*>span区域2</span>

### 课4：CSS(关联选择器&组合选择器)

**span** **b** **a** **img**{关联选择器 选择器中的选择器

background-color: *#09F*;

color: *#FFF*

}

/\* 组合选择器 \*/

*.haha***,div** **b**{/\* 对多种选择器进行相同样式定义 \*/

background-color: *#000*;

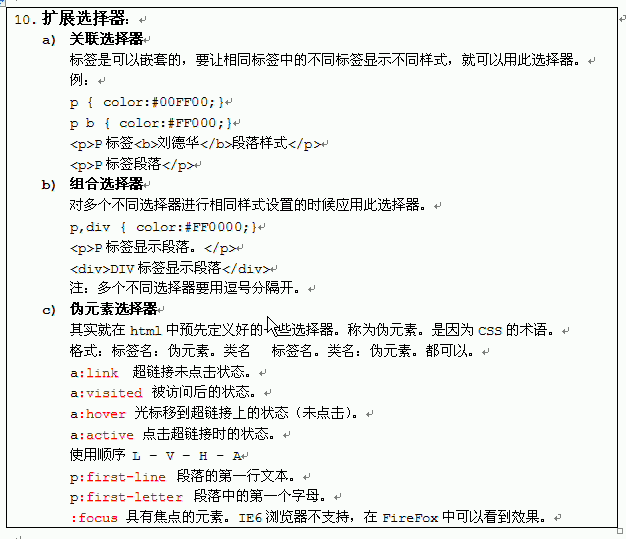
color: *#C00*

}

1. 扩展选择器
2. 关联选择器

标签是可以嵌套的，要让相同标签中的不同标签显示不同

1. 组合选择器
2. 伪元素选择器



### 课5：CSS(伪元素选择器\_1)

其实就在html中预先定义好的一些选择器，称为伪元素选择器。

遇到a:VISITED访问后没有出现划线问题：

ebkit关于a:visited的文字：

　　WebKit is prone to an information-disclosure vulnerability. This issue occurs when Cascading Style Sheets (CSS) use the ':visited' pseudo-class. Attackers 　　may determine which sites a user has visited.

其大意就是a:visited伪类可能会暴露用户浏览信息记录，攻击者可能会据此判断用户曾经访问过的网站，造成不必要的损失，因此这些浏览器决定限制a:visited的功能，所以不是代码的问题，而是浏览器方面的限制。

所以，若是用下划线来判断某链接是否曾被点击过是失效的，那么我们就只能通过设置颜色来区别了，这时候我们就要严格遵从lvha规则了。但在写小demo学习时又会发现有时候页面在经过改动后刷新会自动显示链接的颜色为：visited设置中的颜色，这是因为浏览器缓存的原因，在做小demo时可采用头部meta编码使其不再保留缓存：

<!-- 伪元素

超链接的状态

-->

<style type=*"text/css"*>

**a***:LINK*{

color: *#0FB*;

/\* background-color: #DDD; \*/

text-decoration: *none*;/\* 下划线 \*/

font-size:*18px*;

}

/\* 访问后效果 \*/

**a***:VISITED* {

color: *#000*;

background-color: *#FF9*;

text-decoration:*line-through*;

}

/\* 鼠标悬停 \*/

**a***:HOVER*{

color: *#F00*;

background-color: *#0FE*;

font-size:*24px*;

}

/\* 点击效果 \*/

**a***:ACTIVE* {

color: *#FFF*;

background-color: *#000*;

font-size:*36px*;

}

/\* 顺序：L V H A \*/

**p***:FIRST-LETTER* {

color:*#F00*;

font-size: *40px*;

}

**div***:HOVER* {

color:*#FFF*;

background-color: *#F00*;

}

### 课6：CSS(伪元素选择器\_2)

其实就在html中预先定义好的一些选择器，称为伪元素，是因为CSS的属于。

p:first-line 段落的第一行文本。

p:first-letter 段落中的第一个字母。

:focus 具有焦点的元素。

**input***:FOCUS* {

color: *#FBB*;

background-color: *#CFF*;

}

### 课7：CSS(CSS样式的结合应用)

小技巧：DW向下复制本行内容：crtl+D；删除本行或本表格内容ctrl+shift+d

cellpading：表格单元边界与单元内容之间的间距（单元格里面的内容与单元格边框之间的距离）

cellspacing：表格单元格间距（单元格与单元格之间的距离）

table{

border-bottom:#3EDC47 double 3px;

border-right: #EF6F71 double 3px;

border-left: #EF6F71 double 3px;

border-top: #3EDC47 double 3px;

width: 500px;

}

table td{

border: #F2F972

dotted 1px;

padding: 69px;

}

### 课8：CSS(CSS的盒子模型)

Padding：默认是上和左的内边距。顺序是：上右下右，顺时针方向。

Margin:外边距。同样是顺时针。

<style type="text/css">

body{/\*盒子与标题栏的距离本身默认有固定的距离\*/

margin: 0px;

}

div{

border: #D72CDC solid 1px;

height: 100px;

width: 200px;

padding:

}

#div\_1{

border: #4FC1E4 dashed 10px;

padding: 20px 50px 100px 400px;

}

#div\_2{

border: #00FF15 thick 12px;

margin: 20px 50px 100px 400px;

}

#div\_3{

border: #FF4F00 dotted 7px;

}

</style>

</head>

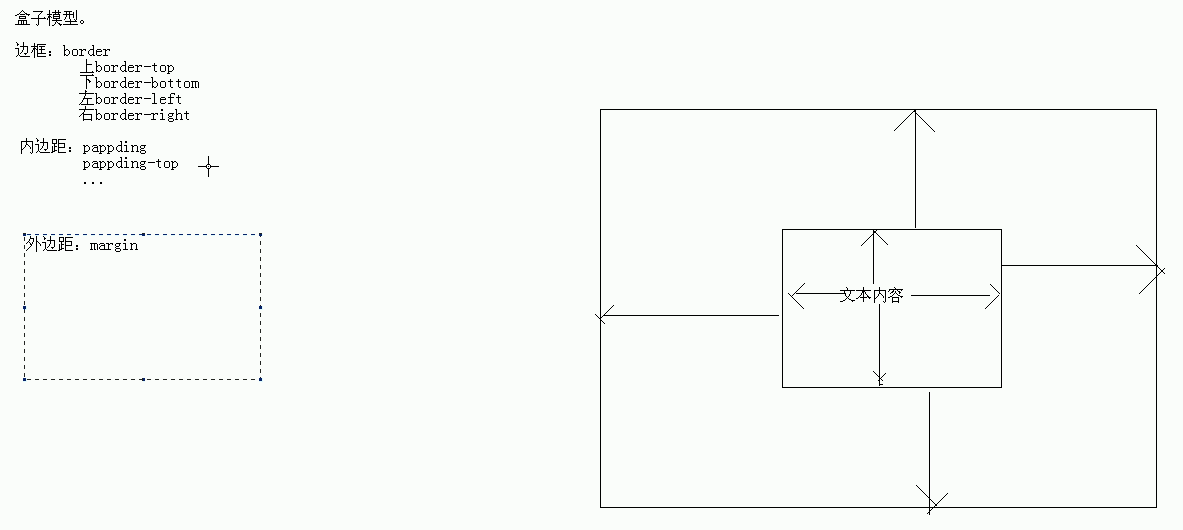
<body>

<div id="div\_1">第一个盒子</div>

<div id="div\_2">第二个盒子</div>

<div id="div\_3">第三个盒子</div>

</body>



### 课9：CSS(CSS的布局-漂浮)

body{

margin: 0px;

}

div{

border: #D72CDC solid 1px;

height: 100px; width: 200px;

}

#div\_1{

background-color: antiquewhite;

float:left;

}

#div\_2{

background-color:aquamarine;

/\*margin: 20px 50px 100px 400px;\*/

float:left;

}

#div\_3{

background-color:brown;

clear: left;

}

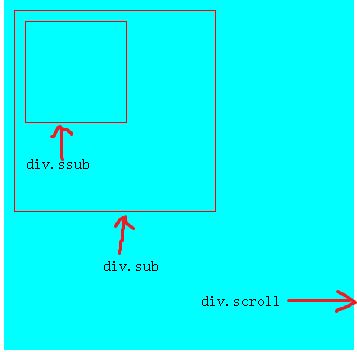
CSS使用float属性导致其他元素错位的问题，该怎么解决？是怎么产生的？

### 课10：CSS(CSS的布局-定位)

* ****position参数：****  
  static: HTML元素的默认定位方式
* absolute: 将对象从文档流中拖出，使用left，right，top，bottom等属性进行绝对定位。而其层叠通过z-index属性定义。绝对定位的元素的位置相对于最近的已定位父元素，如果元素没有已定位的父元素，那么它的位置相对于<html>
* relative: 对象不可层叠，但将依据left，right，top，bottom等属性在正常文档流中偏移位置
* fixed: 元素的位置相对于浏览器窗口是固定位置, 即使窗口是滚动的它也不会移动  
  ****position说明：****  
  设置对象的定位方式，可以让布局层容易位置绝对定位，控制盒子对象更加准确。

注意：绝对定位与[float浮动](http://www.divcss5.com/rumen/r93.shtml)不能同时使用。

****position:absolute****；****position:relative****绝对定位使用通常是父级定义position:relative定位，子级定义position:absolute绝对定位属性，并且子级使用left或right和top或bottom进行绝对定位。



上面截图由3个div组成，外层div名称为scroll，第2个div的名称为sub，第3个div的名称为ssub。现在我就来告诉大家如何使用position中absolute和relative

记住如下的口诀

* 绝对定位（absolute）位置是相对最近已经定位的父元素，如果父元素本身没有使用position定位，则相对于文档（html）定位
* 绝对定位使用通常是父级定义position:relative定位，子级定义position:absolute绝对定位属性，并且子级使用left或right和top或bottom进行绝对定位

### 课11：CSS(CSS的布局-图文混排)

注意：对于图像的长宽，不能用height（width）:xxpx。

### 课12：CSS(CSS的布局-图像签名)

## Day03:

### 课1：01-Javascript(概述和Java的区别)

JavaScript概述

JavaScript是基于对象和事件驱动的脚本语音，主要应用在客户端。

特点：

1. 交互性（它可以做的就是信息的动态交互）
2. 安全性（不允许直接访问本地硬盘）
3. 跨平台性（只要是可以解释JS的浏览器都可以执行，和平台无关）

JavaScript与Java的不同

1. JS是Netscape公司的产品，前身是LiveScript公司的产品；Java原先是Sun公司产品，现在是Oracle公司的产品。
2. JS是基于对象，Java是面向对象。
3. JS只需要解释就可以执行，Java需要先编译成字节码文件，再执行。
4. JS是弱类型，Java是强类型。

JavaScript与Html的结合方式

想要将其他代码融入到Html中，都是以标签的形式。

1. JS代码存放在标签对<script> js code...</script>中。
2. 当有多个html页面使用到相同的js脚本时，可以将js代码封装到一个文件中，只要在script标签的src属性引入一个js文件。

（方便后期维护，扩展）

注：规范中script标签早期有一个属性language，而现在使用type属性。

JS和JSP：

js是javascript的缩写。以下是JSP与JS的区别和联系：

名字：JS：JavaScriptJSP：Java Server Pages

执行过程：JSP先翻译，翻译成Servlet执行如： test.jsp 要变成 test\_jsp.java 然后编译成 test\_jsp.class而 test\_jsp.java 本身就是一个servlet.所以 jsp只是servlet的一个变种，方便书写html内容才出现的。servlet是根本，所有jsp能做的，servlet全能做。

JS：是一种基于对象的客户端[脚本语言](https://www.baidu.com/s?wd=%E8%84%9A%E6%9C%AC%E8%AF%AD%E8%A8%80&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)。目的是解决[服务器端](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8%E7%AB%AF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)语言。如：Perl，遗留的速度问题，为客户提供更流畅的浏览效果。JS可以直接嵌入到html代码中进行解析执行，非常简单易学，可以产生很多动态的效果。

形象比较：JS是在客户端执行的，需要浏览器支持JavaScript。JSP是在[服务器端](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8%E7%AB%AF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)执行的，需要服务器上部署支持Servlet的服务器程序。JS代码能够直接从服务器上download得到，对外是可见的，jsp（和翻译后的Servlet代码）是对外不可见的。

JS和JSP相比较：虽然JS可以再客户端动态生成HTML，但是很难与服务器交互，因此不能提供复杂的服务。如：访问数据库和图像处理等等。JSP在HTML中用<% %>里面实现。JS在HTML中用<Scrippt></Script>实现。

执行过程：JSP：1、JSP文件在用户第一次请求时，会被编译成Servlet，然后由这个Servlet处理用户的请求，所以Servlet可以看成是运行时的Servlet。

2、JSP容器也就是Servlet容器管理JSP页面生命周期分为两个阶段，即转换阶段和执行阶段。

3、转换阶段：当有一个对JSP页面的请求到来时，JSP容器检查JSP页面的语法是否正确，将JSP页面转换为Servlet源文件，然后调用javac命令编译Servlet源文件生成字节码文件。

4、执行阶段：Servlet容器加载转换后的Servlet类。JS：JavaScript 通常是运行在前台，即运行于用户浏览器的，通常不需要服务器的后台支持（AJAX交互例外），混合在HTML中的JavaScript脚本程序直接被浏览器解释执行，以提高页面的美观性和UI操作响应速度为基本目标。

运行位置不同：JSP运行在后台服务器上，混合在HTML中的java程序段用于控制HTML的动态生成，并且通常负责调用后台数据库中的数据，形成能够根据使用情况的变化的，具有丰富数据交互效果的页面。

### 课2：Javascript(Js和html相结合的方式一二)

<!--

js和html相结合的方式：

1，将javascript代码封装到<script>标签中。

2，将javascript代码封装到js文件中，并通过<script>中的src属性导入。

注意：如果<script>标签中使用src属性，那么该标签中封装的javascript不会执行。

所以退出导入js文件都是用单独<script>来完成。

-->

<!-- 导入一个js文件 -->

<!-- <script type="text/javascript" src="demo.js"></script> -->

<!-- 封装javascript代码。 -->

<script type=*"text/javascript"* >

alert("hello javascript");

</script>

### 课3：Javascript(语法的通用体现)

Javascript语法

每一种语言都有自己的语法规则，JS语法与Java很像，所以学习起来比较容易。JS中也一样有变量，语句，函数，数组等常见语言组成元素。

通常高级程序设计语言所包含的语法内容：

1，关键字。该语音中被赋予了特殊含义的单词。

2，标识符。用于标示数据和表达式的符号。通常可以理解为在程序中自定义的名称。比如变量名，函数名。

3，注释。注释说明解释程序。用于调试程序。

4，变量。 用于标示内存中一片空间。用于存储数据，该控件中的数据是可以变化的。

什么时候使用变量，当数据不确定的时候，

5，运算符。可以让数据进行运算的符号。

6，语句。用于对线程的运行流程进行控制的表达式。

7，函数。用于对功能代码进行封装，便于提高复用性。

8，数组。对多数据进行存储，便于操作。就是传说中的容器。

9，对象。只要是基于对象的语音，或者是面向对象的语音，就存在着对象的概念，

对象就是一个封装体。既可以封装数据又可以封装函数。

这些都是高级程序设计语言具备的共性内容，只不过各种语言对这些内容的表现形式有所不同。

但是使用基本思想是一致的。

### 课4：Javascript(JS语法-变量)

1. 变量

通过关键字var来定义，弱类型即是不用指定具体的数据类型，。

注：JS中特殊的常量值：undefined，当变量没有初始化就被使用，该变量的值就是undefined。

注意：JavaScript语句在结尾处是可以不用分号结束的，非严谨语言特点。

但为了符合编程规范，需要像Java一样定义结束符。

而且有些情况是必须加分号的，如：var x = 3; var y = 5; 如果两条语句卸载同一行，就需要分号隔开。

<script type=*"text/javascript"*>

/\*

js中定义变量，使用到关键字。var。

\*/

//定义变量：JS是弱类型的。

**var** x = 3;//var可以不写，但是我们开发时需要通过严谨规范的方式来定义。

x="abc";

x=3.45;

x=**true**;

x='c';

// alert("x="+x);//这是一个函数，将具体的参数通过对话框进行显示。

<!-- 下面的赋值操作结果都相同 -->

<!-- **var** a = 3; **var** b =4; -->

**var** a = 3

**var** b = 4;

</script>

### 课5：Javascript(JS语法-算术运算符)

/\*

运算符：

1，算术运算符。+ - \* / % ++ --

2，赋值运算符。

3，比较运算符。

4，逻辑运算符。

5，位运算符。

6，三元运算符。

\*/

// 算术运算符演示

**var** a = 3710;

// alert("a="+a/1000\*1000);//a=3710;

**var** a1=2.3,b1=4.7;

// alert("a1+b1="+(a1+b1));

// alert(12+1);

// alert("12"+1);//强制转换：1会先被转为string类型

// alert("12"-1);//强转：12被转为数值类型。

// alert(true+1);//因为在js中false默认就是0,或者null。非0或者非空默认为1。

// alert(2%5==0);

**var** n = 3,m;

m=n++;

alert("m="+m+"n="+n);

### 课6：Javascript(JS语法-其他运算符)

// 赋值运算符：= += -= %= /= \*=

**var** i = 3;

i+=2;

// alert("i="+i);

// 比较运算符：> < >= <= != ==

**var** z = 3;

// alert(z==4);

// 逻辑运算符。 && || ^ !

**var** t = 4;

// alert(t>3 && t<6);//true

// alert(t>3 & t<6);//1

// alert(!t);//false

// 位运算符。& | ^ >> << >>> <<<

**var** c = 6;

// alert(c&3);

// alert(c^3);

// alert(c>>>1);//6/2=3

// 三元运算符。 ? :

// alert(3>0?alert("yes"):alert("no"));//undefined

alert(3>0?100:200);

### 课7：Javascript(JS语法-小细节)

/\*

一些细节：

1，undefined来定义。其实它就是一个常量。

\*/

**var** xx=4;

// alert(xx==undefined); //undefined

// 要想获取具体的值的类型，可以通过typeof来完成。

alert(**typeof**("abc"));//string

alert(**typeof**(2.5));//number

alert(**typeof**(**true**));//boolean

alert(**typeof**(78));//number

alert(**typeof**('9'));//string

### 课8：Javascript(JS语句-if语句)

// 顺序结构。

// alert("abc1");

// alert("abc2");

// 判断结构。

**var** x = 3;

**if**(x==4){

alert("yes");

}**else**{

alert("no");

}

### 课9：Javascript(JS语句-switch语句)

注意：JS在代码编写过程中，是区分大小写的。！

/\* 选择结构：

var x = "abc";

switch(x){

case "kk":

alert("a");

break;

case "abc":

alert("b");

break;

default:

alert("c");

break;//break可以省略。

} \*/

### 课10：Javascript(JS语句-循环语句)

// 循环结构：

**var** x = 1;

/\* while(x<10){

// alert("x="+x);

document.write("<font color='red'> x="+x+"</font><br/>");

x="+x+"</font><br/>");

x++;

} \*/

/\* document.write("<font color='blue'>");

while(x<10){

document.write("x="+x+"<br/>");

x++;

}

document.write("</font> "); \*/

document.write("<font color='green'>");

**for**(**var** x=0;x<3;x++){

document.write("x="+x+"<br/>");

}

document.write("</font>");

### 课11：Javascript(JS语句-其他语句)

语句：

1，顺序结构。

2，判断结构。代表语句：if

3，选择结构。switch

4，循环结构。while,for

5，其他语句。break:跳出选择，跳出循环 continue：用于循环语句，结束本次循环，继续下次循环。

**for**(**var** x=0;x<3;x++)

**for**(**var** y=0;y<4;y++){

document.write("x="+x);

// break;

**continue**;

}

### 课12：Javascript(JS语句-练习-九九乘法表)

<!-- 练习九九乘法 -->

<script type=*"text/javascript"*>

document.write("<table>");

**for**(**var** x=1;x<10;x++){

document.write("<tr>");

**for**(**var** y=1;y<=x;y++){//&nbsp：代表的是空格的意思。

// document.write(y+"\*"+x+"="+x\*y+"&nbsp;&nbsp;");

document.write("<td>"+y+"\*"+x+"="+x\*y+"</td>");

}

// document.write("<br/>");

document.write("</tr>");

}

document.write("</table>");

**table,table** **td**{

border:*solid* *5px*;

border-color: *green*;

width: *300px*;

}

### 课13：Javascript(JS数组)

/\*

javascript中的数组演示。

数组用于存储更多的数据，是一个容器。

特点：

1，长度是可变的。

2,元素的类型是任意的。

建议在使用数组时，存储同一类型的元素，操作起来较为方便。

JS中的数组定义的两种方式：

1，var arr= []; var arr = [3,1,5,8];

2，使用了javascript中的Array对象来完成的。

var arr = new Array(); 相当于：var arr = [];

var arr1 = new Array(5);数组定义长度为5。

var arr2 = new Array(5,6,7);定义一个数组，元素是5，6，7;

\*/

**var** arr=[23,78,100];

// alert(typeof(arr));//对象类型Object

// alert("len="+arr.length);

// 遍历数组。

arr[4]=222;

arr[1]="avc";

arr[2]=**true**;

**for**(**var** x=0;x<arr.length;x++){

document.write("arr["+x+"]="+arr[x]+"<br/>");

}

### 课14：Javascript(JS函数)

/\*

js中的函数演示。

函数：就是一个功能的封装体。

定义功能通常需要两个明确。

1，功能的结果。

2，功能实现中的参与运算的未知的内容。

js中的定义函数的格式。

通过指定 的关键字来定义。

function函数名（参数列表）{

函数体：

return返回值;//如果没有具体的返回值，return可以省略不写。

}

\*/

**function** demo(){

alert("demo run");

**return**;

}

// demo();

/\*

需要加法运算

\*/

**function** add(x,y){

**return** x+y;

}

**var** sum = add(3,5);

alert("sum="+sum);

### 课15：Javascript(JS函数\_细节1)

/\*

函数的一些细节。

1，只要使用函数的名称就是对这个函数的调用。

2，函数中有一个数组在对传入的参数进行存储。这个数组就是arguments

\*/

**function** show(x,y){

// alert(arguments.length);

// alert(x+":"+y);

**for**(**var** a=0;a<arguments.length;a++){

document.write(arguments[a]);

}

}

show(4,5,6,7,8);//建议函数中定义了几个参数就传送几个实参。

### 课16：Javascript(JS函数\_细节2)

/\*

函数细节2：

getSum函数运行，并将返回的结果赋值给sum。

getSum本身是一个函数名，而函数本身在js中就是一个对象。getSumjius这个函数对象的引用。

将getSum这个引用的地址赋值给sum。这时sum也指向了这个函数对象。

相当于这个函数对象有两个函数名称。

打印的时候如果sum指向的是函数对象，那么就会将该函数对象以字符串的形式打印出来。

就是该函数的代码定义格式。

例如：

Demo d = new Demo();

syso(d);//Demo@8sdaa;

Thread t = new Thread();

syso(t);//Thread(Thread-0,5,main)

\*/

**function** getSum(){

**return** 100;

}

// var sum = getSum();

**var** sum = getSum;

// alert("sum="+sum);

**function** show2(){

alert("show2 run");

}

alert(show2()); //show2 run undefined

### 课17：Javascript(JS函数\_动态函数)

/\*

JS函数的其他表现形式。

动态函数：使用的是JS中内置的一个对象function。

只不过用的不是很多。

参数列表，函数体都是通过字符串动态指定的。（内部都是字符串类型）

\*/

**var** add = **new** Function("x,y","var sum; sum = x + y;return sum");

**var** sum = add(4,6);

// alert("sum="+sum);

### 课18：Javascript(JS函数\_匿名函数)

/\*

匿名函数：没有名字的函数。

通常是函数的简写形式。

\*/

**var** add3=**function**(a,b){

**return** a+b;

}

alert(add3(3,11));

**function** haha(){

alert("haha run");

}

**var** xixi = haha;

// 上述代码可以简写成下面的方式。

**var** xixi = **function**(){

alert("haha run");

}

### 课19：Javascript(JS函数\_练习1)

### 课20：Javascript(JS函数\_练习2)

/\*

综合练习：

1，定义功能，完成对数组的最值获取。

2，对数组排序。

3，对数组查找。

4，对数组元素进行反转。

\*/

// 取最值

**var** arr = [66,33,12,64,52];

**function** getMax(arr){

**var** max = 0;

**for**(**var** i=1;i<arr.length;i++){

**if**(arr[max]<arr[i])

max=i;

}

**return** arr[max];

}

**var** maxvalue = getMax(arr);

// alert("最大值："+maxvalue);

// 排序：

**function** sortArray(arr){

**for**(**var** x=0;x<arr.length-1;x++){

**for**(**var** y=x+1;y<arr.length;y++){

**if**(arr[x]<arr[y])

swap(arr,x,y);

}

}

}

**function** swap(arr,a,b){

**var** temp = arr[a];

arr[a]=arr[b];

arr[b]=temp;

}

**function** println(val){

document.write(val+"<br/>");

}

println("排序前："+arr);

sortArray(arr);

println("排序后："+arr);

// 查找:

**function** binarySearch(arr,val){

**var** max=arr.length-1;

**var** min=0;

**var** mid=(max+min)/2;

**while**(max>=min){

mid=(max+min)/2;

**if**(arr[mid]<val){

min=mid+1;

}**else if**(arr[mid]>val){

max=mid-1;

}**else**

**return** mid;

}

**return** -1;

}

// var val = binarySearch(arr,64);

// println(val);

// 反转：

**function** reverseArray(arr){

**for**(**var** x=0,y=arr.length-1;x<y;x++,y--){

swap(arr,x,y);

}

}

reverseArray(arr);

println("反转后："+arr);

## Day04:

### 课1：Javascript(全局和局部变量)

// 在脚本片段中定义的变量，是全局变量。

// for(var x=0;x<3;x++){

// document.write("x="+x);

// }

// function show(){

// var x= 6;//局部变量。

// }

// document.write("x===="+x);

**var** x = 3;//全局变量x。

**function** show(x){//函数局部变量x。

x=8;

}

show(x);

document.write("x="+x);

### 课2：Javascript(常见对象-Object)

/\*

演示一下Object对象。

toString();//将对象变成字符串。toString返回对象的字符串表示。

valueOf();

\*/

**function** show(){

alert("show run");

}

// alert(show.toString());

**var** arr=[3,4,1,6,7];

// alert(arr.toString());

**var** abc = **function**(){

alert("arb run");

}

// alert(abc);

alert(arr.valueOf());

### 课3：Javascript(常见对象-String-基本功能)

出现细节问题：

myeclipse10出现undefined attribute name(src)？

我不知道怎么解决的！重新换个路径就好了。

/\*

演示string对象。

表现形式：

var str = new String("abc");

var str = "abc";

\*/

**var** str = "abcde";

// println(str);

// println("len="+str.length);

// alert(str.bold());//为str的两端加上<B>标签。

// println(str.bold());//加粗

// println(str.fontcolor("red"));

// println(str.link("http://www.sina.com.cn"));//超链接

// println(str.substr(1,3));//从角标1开始，共输出三个。

// println(str.substring(1,3));//从角标1开始，到第三个。

/\*

### 课4：Javascript(常见对象-String-自定义功能)

发现js中的string对象方法有限，想要对字符串操作的其他功能，

比如：去除字符串两端的空格，这时只能自定义。

\*/

// 去除字符串两端的空格。

**function** trim(str){

// 定义两个变量，一个记录开始的位置，一个记录结束的位置。

// 对开始的位置的字符进行判断，如果是空格，就进行递增，直到不是空格为止。

// 对结束的位置的字符进行判断，如果是空格，就进行递减，直到不是空格为止。

// 必须要保证开始<=结束这样才可以进行截取。

**var** start=0,end=str.length-1;

**while**(start<=end&&str.charAt(start)==' '){

start++;

}

**while**(start<=end&&str.charAt(end)==' '){

end--;

}

**return** str.substring(start,end+1);//只包含头，不包含尾。

}

**var** s = " ab c ";

alert("-"+trim(s)+"-");

### 课5：Javascript(常见对象-String-原型属性prototype)

//字符串对象新功能，去除字符串两端的空格。

String.prototype.trim=**function**(){

**var** start=0,end=**this**.length-1;

**while**(start<=end&&**this**.charAt(start)==' '){

start++;

}

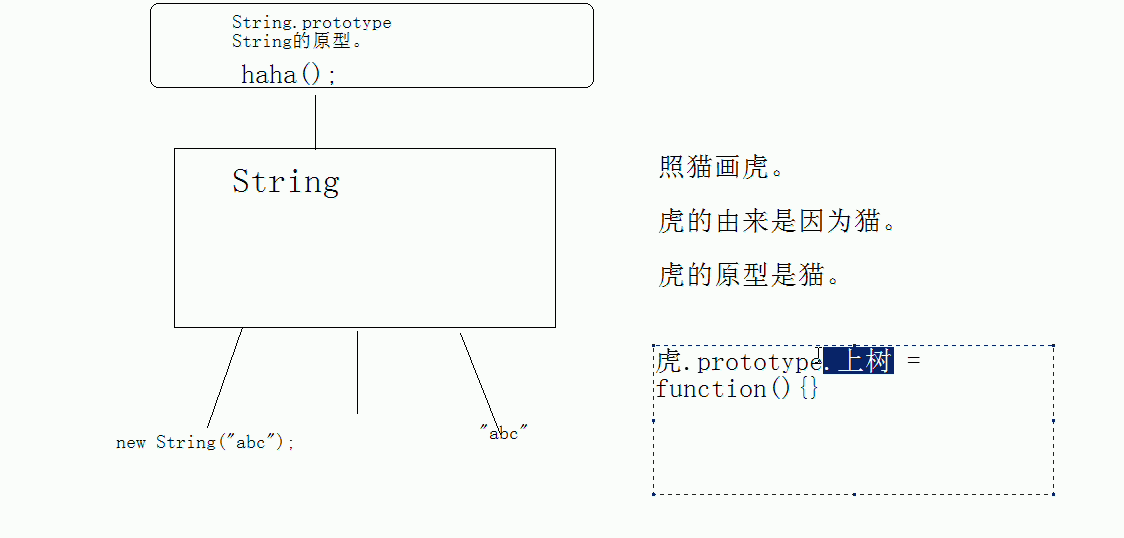
**while**(start<=end&&**this**.charAt(end)==' '){

end--;

}

**return** **this**.substring(start,end+1);//只包含头，不包含尾。

};



### 课6：Javascript(原型练习)

Missing semicolon:缺少操作符或分号。

/\*

字符串新功能，添加一个将字符串转成字符数组。

返回一个数组

\*/

String.prototype.toCharArray = **function**(){

// 定义一个数组：

**var** chs = [];

// 将字符串中的每一位字符串存储到字符数组中。

**for**(**var** x=0;x<**this**.length;x++){

chs[x]=**this**.charAt(x);

}

**return** chs;

};

/\*

\* 添加一个，将字符串进行反转的方法。

\*

\* \*/

String.prototype.reverse=**function**(){

**var** arr=**this**.toCharArray();

**for**(**var** start=0,end=arr.length-1;start<end;start++,end--){

swap(arr,start,end);

}

**return** arr.join("");

};

// 将数组位置置换功能进行封装。

**function** swap(arr,a,b){

**var** temp=arr[a];

arr[a]=arr[b];

arr[b]=temp;

}

### 课7：Javascript(常见对象-Array-基本功能)

/\*

演示数组。

\*/

**var** arr = ["nba","haha","xixi","hehe"];

**var** arr2 = ["qq","xiaoqiang",70];

println(arr);

println(arr2);

**var** newArr = arr.concat("mm",arr2);//将mm作为新数组中的元素，将arr2也作为新数组中的元素。

// println(newArr);

// println(newArr.join("-"));

println(myJoin(newArr,"-"));

println("<hr/>");

println(newArr.pop());//删除并返回最后一个元素。

println(newArr);

**function** myJoin(arr,separator){

**var** str="";

**for**(**var** x=0;x<arr.length;x++){

**if**(x!=arr.length-1)

str += arr[x]+separator;

**else**

str += arr[x];

}

**return** str;

}

**var** arr3 = [1,2,3];

**var** len = arr3.push("arra","arrb","arrb");//为数组添加新的元素，并返回数组的长度。

println(len);

println(arr3);

**var** shift1=arr3.shift();//删除并返回第一个元素。

println(shift1);

println(arr);

println(arr.sort());

// var temp = arr.splice(1,3,333,1004,"yiyi","szzzz");//删除元素，并进行替换。

// println(temp);

println(arr.unshift("uuuu"));//移除第一个元素，并重新添加新元素。

println(arr);

### 课8：Javascript(常见对象-Array-练习-堆栈和队列结构)

/\*

数组练习。

用数组实现js中的堆栈或者队列数据结构。

\*/

**var** arr = [];

// arr.unshift("abc1","abc2","abc3");

arr.unshift("abc1");

arr.unshift("abc2");

arr.unshift("abc3");

println(arr);

// println(arr.pop());

// println(arr.pop());

// println(arr.pop());

println(arr.shift());

println(arr.shift());

println(arr.shift());

### 课9：Javascript(常见对象-Array-练习-自定义功能)

/\*

给数组对象添加新功能，使用到原型属性。

\*/

**var** array = ["abc1","nbc2","nhaa","cdcd"];

**var** maxValue = array.getMax();

println("maxValue:"+maxValue);

println(array.toString());

//数组获取最大值的方法。

Array.prototype.getMax = **function**(){

**var** temp = 0;

**for**(**var** x=1;x<**this**.length;x++){

**if**(**this**[x]>**this**[temp])

temp = x;

}

**return** **this**[temp];

};

//数组的字符串表现形式。

//定义toString方法。相当于java中的复写。

Array.prototype.toString=**function**(){

**return** ("["+**this**.join()+"]");

};

### 课10：Javascript(常见对象-Date-基本方法)

/\*

演示js中的日期。Date

\*/

**var** date = **new** Date();

// println(date);

// println(date.toLocaleString());//日期和时间

// println(date.toLocaleDateString());//日期

**var** year = date.getFullYear();

**var** month = date.getMonth()+1;

**var** day = date.getDate();

**var** week = getWeek(date.getDay());

println(year+"-"+month+"-"+day+"-"+week);

**function** getWeek(num){

**var** weeks = ['星期天','星期1','星期2','星期3','星期4','星期5','星期6'];

**return** weeks[num];

}

//日期和毫秒值转换。

**var** date2 = **new** Date();

// 获取毫秒值。日期对象-->毫秒值

**var** time = date2.getTime();

println("time:"+time);

// 将毫秒值转成日期对象。

// 1,new Date(time);//setTime();

**var** date3 = **new** Date(time);

// 将日期对象和字符串之间转换。

// 日期对象转换成字符串。toLocaleString toLocaleDateString

// 将字符串转成日期对象。具备指定格式的日期字符串-->毫秒值-->日期对象。

**var** str\_date = "9/28/2017";

// parse该方法是 Date 对象的静态方法。一般采用 Date.parse() 的形式来调用，而不是通过 dateobject.parse() 调用该方法。

**var** time2 = Date.parse(str\_date);

**var** date3 = **new** Date(time2);

println(date3);

println(date3.toLocaleString());

### 课11：Javascript(JS中特有语句-with)

/\*

为了简化对象调用内容的书写。

可以使用js中的特有语句with来完成。

格式：

with(对象){

在该区域中可以直接使用指定的内容。不需要写对象。

}

\*/

**var** date = **new** Date();

**with**(date){

**var** year = getFullYear();

**var** month = getMonth()+1;

**var** day = getDate();

**var** week = getWeek(getDay());

}

println(year+"---"+month+"---"+day+"---"+week);

### 课12：Javascript(常见对象-Math-基本功能)

/\*

演示Math对象，该对象中的方法都是静态的。不需要new，直接Math调用即可。

\*/

**var** num1 = Math.ceil(12.35);//返回大于等于指定参数的最小整数。

**var** num2 = Math.floor(12.35);//返回小于等于指定数据的最大整数。

**var** num3 = Math.round(12.35);//四舍五入。

println("num1="+num1);

println("num2="+num2);

println("num3="+num3);

**var** num4 = Math.pow(10, 2);

println("num4="+num4);

println("<hr/>");

**for**(**var** x=0;x<10;x++){

// var num=Math.floor(Math.random()\*10+1);

**var** num=parseInt(Math.random()\*10+1);//全局

println(num);

}

### 课13：Javascript(全局方法&Number对象)

/\*

演示global的全局方法。

\*/

println(parseInt("123")+1);//124

**var** val = parseInt("abc");

println("value="+val);

//通过isNaN方法判断是否非法。

//将指定进制格式的字符串转成十进制。

**var** num = parseInt("110",2);

println("num="+num);

**var** num1 = parseInt('0x3c',16);

println("num1="+num1);

// 将十进制转换成其他进制。使用数字对象完成。

**var** num3 = **new** Number(6);

println("num3="+num3.toString(2));

// var myNum=new Number(value);

// var myNum=Number(value);

// 参数 value 是要创建的 Number 对象的数值，或是要转换成数字的值。

// 返回值

// 当 Number() 和运算符 new 一起作为构造函数使用时，它返回一个新创建的 Number 对象。

// 如果不用 new 运算符，把 Number() 作为一个函数来调用，它将把自己的参数转换成一个原始的数值，并且返回这个值（如果转换失败，则返回 NaN）。

**var** num4 = 60;

println("num4="+num4.toString(16));

### 课14：Javascript(JS中的特有语句-forin)

/\*

JS中的特有语句for in

格式：

for(变量 in 对象)//对对象进行变量的语句。

{}

\*/

**var** arr = [32,80,65];

**for**(i **in** arr){

println("i="+arr[i]);

}

println("<hr/>");

**var** numObj = **new** Number(4);

**for**(x **in** numObj){

println(x);

}

### 课15：Javascript(JS自定义对象)

/\*

JS自定义对象。

如果想要自定义对象，应该先对对象描述。

JS是基于对象，不是面向对象的。不具备描述事物的能力。

我们还想按照面向对象的思想编写JS。

就要先描述，在js中，可以用函数来模拟面向对象中的描述。

\*/

// 用js来描述人。

// 为了与函数进行区分，Person用了开头大写。（规范）

**function** Person(){//相当于构造器功能

alert("person run");

}

**var** p = **new** Person();

// 动态给p对象添加属性，直接使用p.属性名即可。

p.name="zhangsan";

p.age="20";

// 如果定义的p对象的属性赋值为一个函数，即是给p对象添加一个方法。

p.show=**function**(){

alert("show run"+**this**.name+":"+**this**.age);

};

// p.show();

**var** obj = **new** Object();

obj.name = "god father";

obj.age = 2019;

// alert(obj.name+":"+obj.age);

**function** Person(name,age){

// 再给Person对象添加了两个属性。

**this**.name=name;

**this**.age=age;

**this**.setName = **function**(name){

**this**.name = name;

};

**this**.getName = **function**(){

**return this**.name;

};

}

**var** p = **new** Person("旺财",19);

// for(x in p){

// println(x+":"+p.x);

// name:undefined

// age:undefined

// setName:undefined

// getName:undefined

// println(x+":"+p[x]);

// name:undefined

// name:旺财

// age:undefined

// age:19

// }

p.setName("小强");

// alert(p.getName());

// 直接使用{}定义属性和值的键值对方式。键值间通过：链接，键与键之间用逗号隔开。

**var** pp = {

// 定义一些成员。

"name":"xiaoming","age":22,//直接定义键值对

"getName":**function**(){

**return this**.name;

}

};

// 对象调用成员有两种方式，对象.属性名 对象["属性名"]

// alert(pp.name+":"+pp.age+"-"+pp.getName());

// alert(pp["name"]+":"+pp["age"]+"-"+pp["getName()"]);

**var** oMap={

8:"小强",3:"旺财",7:"小明"

};

**var** val1 = oMap[8];

alert("vall:"+val1);

**var** val2 = get(7);

alert("val2:"+val2);

**function** get(key){

**return** oMap[key];

}

### 课16：Javascript(JS自定义对象2)

**var** myobj={

myname:"lisisi",myage:30

};

// alert(myobj.myname+":"+myobj["myage"]);

**var** myobj2={

"myname2":"haha","myage":40

};

// alert(myobj2.myname2+myobj2["myage"]);

**var** myMap={

// names:["aa","bb","cc"],nums:[11,22,33]

names:[{name1:"zhangsan"},{myname:"hahaha"}]

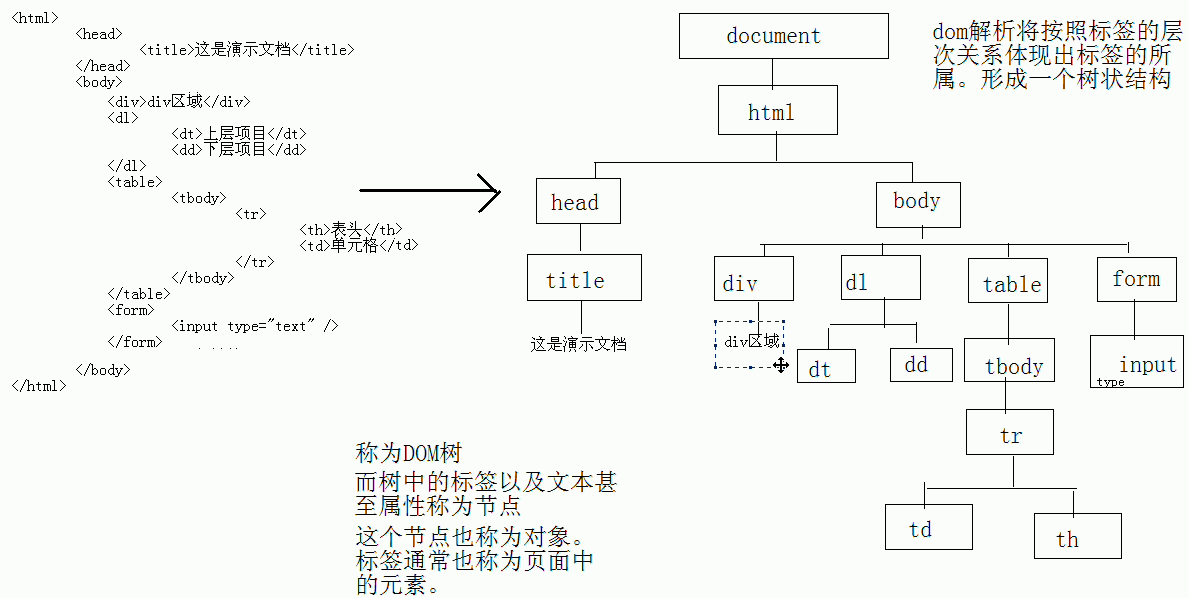
};

// alert(myMap.names[2]);

alert(myMap.names[1].myname);

## Day05:DOM学习

### Charter1:DOM(概述)



DOM:document Object Model(文档对象模型)

用来将标记型文档封装成对象，并将标记型文档中的所有的内容（标签，文本，属性等）都封装成对象。

封装成对象的目的是为了更为方便的操作这些文档以及文档中的所有内容。

因为对象的出现就可以有属性和行为被调用。

文档对象模型：

文档：标记型文档。

对象：封装了属性和行为的实例，可以被直接调用。

模型：所有标记型文档都具备一些共性特征的一个体现。

标记型文档（标签，属性，标签中封装的数据）。

只要是标记型文档，DOM这种技术都可以对其进行操作。

常见的标记型文档：html xml

DOM这种技术是如何对标记型文档操作的呢？

要操作标记型文档必须对其进行解析。

### Charter2:DOM(DOM解析的特点)

DOM技术的解析方式：将标记型文档解析成一棵DOM树，并将树种的内容都封装成节点对象。

注意：这个DOM解析方式的好处：可以对树中的节点进行任意操作，比如：增删改查。

弊端：这种解析需要将整个标记型文档加载进内存。以为着如果标记型文档的体积很大，较为浪费内存空间。

简介另一种解析方式：SAX：是由一些组织定义的一种民间常用的解析方式。并不是W3C标准，而DOM是W3C标准。

SAX解析的方式，基于事件驱动的解析。获取数据的速度很快，但是不能对标记进行增删改动作。

### Charter3：DOM(DOM三级模型)

DOM模型有三种：

DOM level 1:将html文档封装成对象。

DOM level 2：在 level 1基础上加入了新功能，比如解析名称空间。

DOM level 3：将xml文档封装成了对象。

DHTML：动态的HTML。不是一门语音，是多项技术综合体的简称。

其中包含了HTML,CSS,DOM,JavaScript。

这四个技术在动态html页面效果定义时，都处于什么样的角色呢？负责什么样的职责呢？

HTML：负责提供标签，对数据进行封装，目的是便于对该标签中的数据进行操作。

简单说：用标签封装数据。

CSS：负责提供样式属性，对标签中的数据进行样式的定义。

简单说：对数据进行样式定义。

DOM：负责将标签性文档中的所有内容解析，并封装成对象。 在对象中定义了更多的属性和行为，便于对对象的操作。

简单说：将文档和标签以及其他内容变成对象。

JS：负责提供程序设计语音，对页面中的对象进行逻辑操作。

简单说：负责页面的行为定义。就是页面的动态效果。

所以javascript是动态效果的主力编程语音。

DHTML+XMLhttpRequest = AJAX

### Charter4：DOM(DHTML概述)

见上。

### Charter5：DOM(BOM&Navigator对象&History对象)

### Charter6:DOM(Location对象)

// 定义一函数。

**function** windowObjDemo(){

// alert("WINDOW RUN");

// 想要知道这个浏览器的信息。就需要使用window对象中的navigator

**var** name = window.navigator.appName;

**var** version = window.navigator.appVersion;

println(name+":"+version);

}

/\* 演示location \*/

**function** windowObjDemo2(){

**var** pro = location.protocol;

**var** text = location.href;

// alert(text);

// 给location的href属性设置一个值。改变了地址栏的值，并对其值进行了解析如果是http，还进行连接访问。

location.href = "http://www.sina.com.cn";

}

### Charter7:DOM(window常见方法\_1)

**function** windowMethodDemo(){

// var b = confirm("你真的要点击吗？");

// alert("b="+b);

// setTimeout("alert('time out')",4000);

timeId = setInterval("alert('Interval run')",3000);

}

**function** stopTime(){

clearInterval(timeId);

}

**function** windowMove(){

// moveBy(10,10);

// moveTo(50,80);

**for**(**var** x=0;x<20;x++){

moveBy(20,0);

moveBy(0,20);

moveBy(-20,0);

moveBy(0,-20);

}

}

### Charter8:DOM(window常见方法\_2)

**function** windowOpen(){

open("ad.html","\_blank",

"height=200,width=400,status=yes,toolbar=yes,menubar=yes,location=yes");

close();

}

### Charter9:DOM(window常见事件)

遇见的问题：

Window.status状态栏，为什么没有显示？

/\*onunload = function(){

alert("onunload run");

}

onload = function(){

alert("onload run");

}

onbeforeunload = function(){

alert("onbeforeunload run");

}\*/

onload = **function**(){

window.status = "加载完毕";

};

### Charter10:DOM(练习-广告弹窗)

onunload = **function**(){

open("ad.html","\_blank",

"height=200,width=400,status=yes,toolbar=yes,menubar=yes,location=yes");

};

setInterval("focus()",1000);//在指定时间，执行表达式内容

### Charter11:DOM(document获取节点byId)

<!--

document对象的演示。

该对象将标记型文档进行封装。

该对象的作用，是可以对标记型文档进行操作。

最常见的操作就是，想要实现动态效果，需要对节点进行操作。

要先获取这个节点。

要想获取节点，必须要先获取到节点所属的文档对象document。

所以document对象最常见的操作就是获取页面中的节点。

获取节点的方法体现：

1，getElementById();通过标签的id属性值获取该标签节点。返回该标签节点。

2,getElementsByName();通过标签的name属性获取节点，因为name有相同可能，所以返回的是一个数组。

3,getElementsByTagName();通过标签名获取节点。因为标签名会重复，所以返回的是一个数组。

上述element后带s的都是数组。

-->

**function** getNodeDemo(){

/\*

需求：获取页面中的div节点。

思路：

通过document对象完成。因为div节点有id属性。所以可以通过id属性来完成获取。

\*/

**var** divNode = document.getElementById("divid");

//节点都有三个必备的属性。节点名称，节点类型，节点值。参考div元素api手册中。

// alert(divNode.nodeName+":"+divNode.nodeType+":"+divNode.nodeValue);

//DIV:1:null

/\*

常见节点有三种：

1，标签型节点：类型1

2，属性节点：类型2

3，文本节点：类型3

标签型节点是没有值的，属性和文本节点是可以有值的。

\*/

// 获取div节点中的文本。

**var** text = divNode.innerHTML;

// alert(text);

// 改变div中的文本。

divNode.innerHTML = "哈哈，文本被我改掉了".fontcolor("blue");

}

### Charter12:DOM(document获取节点byName)

**function** getNodeDemo2(){

// var nodes = document.getElementsByName("user");

**var** usernodes = document.getElementsByName("user")[0];

// alert(nodes[0].type);

// alert(nodes[0].value);

alert(usernodes.value);

}

<input type=*"button"* value=*"演示document对象获取节点"* onclick="getNodeDemo2()"/>

<div id=*"divid"*>这是一个div区域</div>

<input type=*"text"* name=*"user"*/>

### Charter13:DOM(document获取节点byTagName)

// 获取超链接节点对象。演示getElementByTagName方法。

**function** getNodeDemo3(){

**var** nodes = document.getElementsByTagName("a");

// alert(nodes.length);

// alert(nodes[0].innerHTML);

**for**(**var** x=0;x<2;x++){

alert(nodes[x].innerHTML);

nodes[x].target = "\_blank";//打开一个新的窗口

}

}

### Charter14:DOM(document获取节点byTagName\_2)

/\*

对于页面中的超链接，新闻链接通过新窗口打开，门户链接网站在当前页面打开。

当然是要获取其中被操作的超链接对象，

可是通过document获取超链接，拿到的是页面中的所有超链接节点。

只想获取一部分该如何办呢？

只要获取被操作的超链接所属的节点即可。

再通过这个节点获取到其中的所有超链接节点。

\*/

**function** getNodeDemo4(){

// 先获取超链接所属的div节点。

**var** divNode = document.getElementById("newslink");

// 通过对div对象方法的查找，发现它也具备getElementsByTagName方法。

// 记住：所有的容器型标签都具备该方法，在该标签范围内获取指定名称的标签。

**var** aNodes = divNode.getElementsByTagName("a");

**for**(**var** x=0;x<aNodes.length;x++){

alert(aNodes[x].innerHTML);

aNodes[x].target="\_blank";

}

}

### Charter15:DOM(通过节点层次关系获取节点)

**function** getNodeByLevel(){

// 获取页面中的表格节点。

**var** tabNode = document.getElementById("tabid");

// 获取父节点。

**var** node = tabNode.parentNode;

// alert(node.nodeName); //body

// 获取子节点。

// var nodes = tabNode.childNodes;

// alert(nodes[1].nodeName); //tbody , #text

// 获取兄弟节点。尽量少用兄弟节点，因为在解析的时候，会出现浏览器的不同，解析不一致。

// 会解析出标签间的空白节点。

// 上一个。

**var** nodelast = tabNode.previousSibling;

// alert(nodelast.nodeName);//???#text：浏览器问题？

// 下一个。

**var** nodenext = tabNode.nextSibling.nextSibling;

alert(nodenext.nodeName);//???#text：浏览器问题？

}

### Charter16:DOM(节点操作-创建并添加\_1)

<script type=*"text/javascript"*>

//创建并添加节点。

// 需求：在div\_1节点中添加一个文本节点。

**function** createAndAdd(){

/\*

1，创建一个文本节点。

2，获取div\_1节点。

3，将文本节点添加到div\_1节点当中。

\*/

// 1，创建一个文本节点。

**var** oTextNode = document.createTextNode("这是一个新文本");

// 2，获取div\_1节点。

**var** oDivNode = document.getElementById("div\_1");

// 3，将文本节点添加到div\_1节点当中。(子节点)

oDivNode.appendChild(oTextNode);

}

// 需求：在div\_1中创建并添加一个按钮节点。

**function** createAndAdd2(){

// 1，创建一个按钮节点。

**var** oButNode = document.createElement("input");

oButNode.type = "button";

oButNode.value = "一个按钮";

// 2，获取div\_1节点。

**var** oDivNode = document.getElementById("div\_1");

// 3，将按钮节点添加到div\_1节点当中。(子节点)

oDivNode.appendChild(oButNode);

}

// 通过另一种方式完成添加节点。

**function** createAndAdd3(){

/\*

其实是使用了容器型标签中的一个属性，innerHTML。

这个属性可以设置html文本。

\*/

**var** oDivNode = document.getElementById("div\_1");

oDivNode.innerHTML=("呵呵呵呵");//括号是可以去除的

oDivNode.innerHTML=("<input type='button' value='哎哎哎'/>");//括号是可以去除的

oDivNode.innerHTML=("<a href='http://www.sina.com.cn'>新浪网址</a>");//括号是可以去除的

}

</script>

<input type=*"button"* value=*"创建并添加节点"* onclick="createAndAdd3()"/>

<input type=*"button"* value=*"删除节点"* onclick="deleteNode()"/>

<input type=*"button"* value=*"修改节点"* onclick="updateNode()"/>

<hr/>

<div id=*"div\_1"*>

</div>

<div id=*"div\_2"*>

好好学习，day day up!

</div>

<div id=*"div\_3"*>

div区域演示文字。

</div>

<div id=*"div\_4"*>

演示节点增删改查。

</div>

### Charter17:DOM(节点操作-创建并添加\_2)

见上。

### Charter18:DOM(节点操作-删除节点)

需求：将div\_2节点删除。

**function** deleteNode(){

// 1，获取div\_2的节点。

**var** oDivNode = document.getElementById("div\_2");

// 2，使用div节点的removeNode方法。

// oDivNode.removeNode(false);//较少用。

// oDivNode.removeNode(true);

// 一般使用removeChild方法。删除子节点。

// 获取div\_2的父节点。然后再用父节点的removeChild，将div\_2删除。

oDivNode.parentNode.removeChild(oDivNode);

}

### Charter19:DOM(节点操作-替换&克隆节点)

// 需求：用div\_3节点替换div\_1节点。

**function** updateNode(){

// 1，获取div\_3，div\_1。

**var** oDivNode3 = document.getElementById("div\_3");

**var** oDivNode1 = document.getElementById("div\_1");

// 2,用replaceNode方法替换。

// oDivNode1.replaceNode(oDivNode3);

// 建议使用replaceChild方法。

// oDivNode1.parentNode.replaceChild(oDivNode3,oDivNode1);

}

// 需求：希望用div\_4替换div\_1，并保留div\_3节点。

**function** cloneNode1(){//方法名不能和BOM中的cloneNode方法重名，不然会复写，无法实现克隆功能。

// 1，获取div\_4，div\_1。

**var** oDivNode1 = document.getElementById("div\_1");

**var** oDivNode4 = document.getElementById("div\_4");

// 使用cloneNode方法。

**var** oCopyDiv4 = oDivNode4.cloneNode(**true**);

// 2,使用replaceChild方法。

// oDivNode1.replaceNode(oCopyDiv3);

oDivNode1.parentNode.replaceChild(oCopyDiv4,oDivNode1);

}

## Day05:

### Charter1:DOM(示例-新闻字体)

<!--

需求：新闻字体的大中小样式改变。

思路：

1，先有新闻数据，并用标签封装。

2，定义一些页面样式。

3，确定事件源和事件，以及处理方式中被处理的节点。

事件源：a标签，事件：onclick。

被处理的节点div-newstext

既然要给超链接加入自定义的事件处理，就要先取消超链接的默认点击效果。

可以使用href设置：javascript:void(0)来完成。

-->

// 定义改变字体的方法。

**function** changFont(size,clr){

// 既然要对div newstext中的文本字体进行操作。

// 必须要先获取div节点对象。

**var** oNewsText = document.getElementById("newstext");

// 获取oNewsText节点的style样式属性对象。

oNewsText.style.fontSize = size+"px";

oNewsText.style.color = clr;

}

<a href=*"javascript:void(0)"* onclick="changFont(50,'#ff00dd')">大字体</a>

<a href=*"javascript:void(0)"* onclick="changFont(25,'#aaaaaa')">中字体</a>

<a href=*"javascript:void(0)"* onclick="changFont(10,'#ff0000')">小字体</a>

### Charter2:DOM(示例-新闻字体2)

如果根据用户点击所需要的效果不唯一。

仅通过传递多个参数虽然可以实现效果，

但是1，传参过多，阅读性差。2，js代码耦合性高。3，不利于扩展。

解决：

将多个所需的样式进行封装。

封装到选择器中，只要给指定的标签加载不同的选择器就可以了。

\*/

**function** changeFont(selectorName){

**var** oNewsText = document.getElementById("newstext");

oNewsText.className = selectorName;

}

/\* 超链接访问前，访问后样式一致。 \*/

**a***:link***,a***:visited*{

color:*#ffccff*;

}

**a***:hover*{

color:*#ffff00*;

}

*#newstext*{

width:*500px*;

border:*#00ff00 1px solid*;

}

/\* 预定义一些样式封装到选择器。一般使用类选择器。 \*/

*.norm*{

color:*#000000*;

font-size:*16px*;

background-color: *#cdd8d0*;

}

*.max*{

color:*#808080*;

font-size:*28px*;

background-color: *#9ce9b4*;

}

*.min*{

color:*#ff0000*;

font-size:*12px*;

background-color: *#aaaaaa*;

}

<a href=*"javascript:void(0)"* onclick="changeFont('max')">大字体</a>

<a href=*"javascript:void(0)"* onclick="changeFont('norm')">中字体</a>

<a href=*"javascript:void(0)"* onclick="changeFont('min')">小字体</a>

<div id=*"newstext"* class=*"norm"* >

### Charter3:DOM(示例-展开闭合列表)

<!--

思路：

1，标签封装数据，html

2，定义样式css

3，明确事件源，事件，以及要处理节点,dom。

4，明确具体的操作方式，其实就是事件的处理内容。js。

-->

**var** flag = **true**;

**function** list(){

/\*

思路：

1，就是将dl属性的overflow的值改为visible即可。

2，要先获取dl节点。

3，改变该节点的style.overflow的属性。

\*/

// 1，获取dl节点。

**var** oDlNode = document.getElementsByTagName("dl")[0];

// oDlNode.style.overflow = "visible";//设置对象可视。

**if**(flag){

oDlNode.style.overflow = "visible";//设置对象可视。

flag = **false**;

}**else**{

oDlNode.style.overflow = "hidden";//设置对象可视。

flag = **true**;

}

}

<dl>

<dt onclick="list()">显示条目一</dt>

<dd>展开列表内容</dd>

<dd>展开列表内容</dd>

<dd>展开列表内容</dd>

<dd>展开列表内容</dd>

</dl>

### Charter4:DOM(示例-展开闭合列表2)

**/\*预定义一些样式，类选择器：便于给标签进行样式的动态加载。\*/**

*.open*{

overflow:*visible*;

}

*.close*{

overflow:*hidden*;

}

**dl** **dt**{

color:*#ff8000*;

}

//上述问题，耦合性强；可以对其预订样式

// 故，重新定义list，降低js和css的耦合性。

**function** list3(node){

// alert(node.nodeName);

// 1，获取dl节点。

// var oDlNode = document.getElementsByTagName("dl")[index];

**var** oDlNode = node.parentNode;

// 2,设置dl的className属性值。

// oDlNode.className = "open";

**if**(oDlNode.className=="close"){

oDlNode.className = "open";

}**else**{

oDlNode.className = "close";

}

}

/\*

在多个列表下，一次只需要展开一个列表，其他列表都关闭。

怎么保证开一个，而关其他呢？

思路：

1，获取所有的dl节点。

2，遍历这些节点，只对当前的dl进行展开，其他闭合。

\*/

<dl class=*"close"*>

<dt onclick="list(this)">显示条目一</dt>

<dd>展开列表内容1</dd>

<dd>展开列表内容1</dd>

<dd>展开列表内容1</dd>

<dd>展开列表内容1</dd>

</dl>

忘记为预编译上的注释加注释符/\*\*/，导致点击没有反应！！！

### Charter5:DOM(示例-展开闭合列表3）

/\*

在多个列表下，一次只需要展开一个列表，其他列表都关闭。

怎么保证开一个，而关其他呢？

思路：

1，获取所有的dl节点。

2，遍历这些节点，只对当前的dl进行展开，其他闭合。

\*/

**function** list(node){

// 获取当前的dl节点。

**var** oDlNode = node.parentNode;

// 获取所有的dl节点。

**var** collDlNodes = document.getElementsByTagName("dl");

// 遍历所有dl。

**for**(**var** x=0;x<collDlNodes.length;x++){

**if**(collDlNodes[x]==oDlNode){

**if**(oDlNode.className == "close"){

oDlNode.className = "open";

}**else**{

oDlNode.className = "close";

}

}**else**{

collDlNodes[x].className = "close";

}

}

}

### Charter6:DOM(示例-好友菜单)

不同网页，解析效果不同，导致页面效果不同！

/\* 要求一次仅打开一个，其他关闭 \*/

**function** list(node){

// 获取被操作的节点ul。

// 先通过事件源超链接标签获取其父节点td。然后再通过父节点获取ul节点。

**var** oTdNode = node.parentNode;

**var** oUlNode = oTdNode.getElementsByTagName("ul")[0];//通过标签获取的ul节点数组就只包含一个元素。

// alert(oUlTdNode.nodeName);

// if(oUlTdNode.className=="open"){

// oUlTdNode.className="close";

// }else{

// oUlTdNode.className="open";

// }

// 获取表格中所有的ul。

// 先获取表格节点对象。

**var** oTabNode = document.getElementById("goodlist");

**var** collUlNodes = oTabNode.getElementsByTagName("ul");

**for**(**var** x=0;x<collUlNodes.length;x++){

**if**(collUlNodes[x]==oUlNode){

**if**(oUlNode.className=="open"){

oUlNode.className="close";

}**else**{

oUlNode.className="open";

}

}**else**{

collUlNodes[x].className="close";

}

}

}

<table id=*"goodlist"*>

<tr>

<td>

<a href=*"javascript:void(0)"* onclick="list(this)">好友菜单</a>

<ul>

<li>一个好友</li>

<li>一个好友</li>

<li>一个好友</li>

<li>一个好友</li>

</ul>

</td>

</tr>

### Charter7DOM(示例-创建表格)

**function** crtTable(){

/\*

思路：

可以通过之前学过的createElement创建元素的形式完成。

\*/

// 1,创建表格节点。

**var** oTabNode = document.createElement("table");

// 2，创建tBody节点。

**var** oTbdNode = document.createElement("tbody");

// 3，创建行节点。

**var** oTrNode = document.createElement("tr");

// 4，创建单元格节点。

**var** oTdNode = document.createElement("td");

oTdNode.innerHTML = "这是单元格";

// 让这些节点有关系。进行指定层次的节点添加。

oTrNode.appendChild(oTdNode);

oTbdNode.appendChild(oTrNode);

oTabNode.appendChild(oTbdNode);

document.getElementsByTagName('div')[0].appendChild(oTabNode);

}

### Charter8:DOM(示例-创建表格-使用表格对象)

/\*

上面的方法很麻烦。

既然操作的是表格，

那么最方便的方式是，使用表格节点对象的方法来完成。

表格是由行组成。表格节点对象中insertRow方法就完成了创建行并添加的动作。

行是由单元格组成。通过tr节点对象的insertCell来完成。

\*/

**function** crtTable(){

**var** oTabNode = document.createElement("table");

**for**(**var** x=0;x<5;x++){

**var** oTrNode = oTabNode.insertRow();

**for**(**var** y=0;y<6;y++){

**var** oTdNode = oTrNode.insertCell();

oTdNode.innerHTML =x+"--"+y;

}

}

// 将表格节点添加到div中。

document.getElementsByTagName('div')[0].appendChild(oTabNode);

document.getElementsByName("crtBut")[0].disabled = **true**;

<!--

在页面创建一个5行6列的表格。

1，事件源，比如按钮。

2，必须有一个生成的表格节点存储位置。

-->

<input type=*"button"* value=*"创建表格"* name=*"crtBut"* onclick="crtTable()"/>

<hr/>

<div>

</div>

### Charter9:DOM(示例-创建表格-指定行列)

**function** crtTable(){

**var** oTabNode = document.createElement("table");

// var rowVal = document.getElementsByName("rownum")[0].value;

**var** colVal = document.getElementsByName("colnum")[0].value;

// 这样做，相当于用正则规则进行限定，仅能在一定数范围内输出。

**var** rowVal = parseInt(document.getElementsByName("rownum")[0].value);

**var** colVal = parseInt(document.getElementsByName("colnum")[0].value);

**for**(**var** x=1;x<=rowVal;x++){

**var** oTrNode = oTabNode.insertRow();

**for**(**var** y=1;y<=colVal;y++){

**var** oTdNode = oTrNode.insertCell();

oTdNode.innerHTML =x+"--"+y;

}

}

### Charter10:DOM(示例-删除表格的行和列)

// 为表格赋予属性id。

// oTabNode.id = "tabid";

oTabNode.setAttribute = ("id","tabid");

//setAttribute存在致命问题！浏览器不兼容问题。

// 利用setAttribute指定e的onclick属性，简单，很好理解。

// 但是IE不支持，IE并不是不支持setAttribute这个函数，而是不支持用setAttribute设置某些属性，

// 例如对象属性、集合属性、事件属性，也就是说用setAttribute设置style和onclick这些属性在IE中是行不通的。

// 为达到兼容各种浏览器的效果，可以用点符号法来设置Element的对象属性、集合属性和事件属性。

**function** delrow(){

// 通过属性Id,获取表格对象。

**var** oTabNode = document.getElementById("tabid");

// alert(oTabNode);//null

**if**(oTabNode==**null**){

alert("表格不存在");

**return**;

}

// 获取要删除的表格。

**var** rowVal = document.getElementsByName("delrow")[0].value;

oTabNode.deleteRow(rowVal);

}

// 删除列，其实就是在删除每一行中同一位置的单元格。

**function** delcol(){

// 通过属性Id,获取表格对象。

**var** oTabNode = document.getElementById("tabid");

**if**(oTabNode==**null**){

alert("表格不存在");

**return**;

}

**var** colVal = document.getElementsByName("delcol")[0].value;

**if**(colVal>=1 && colVal<=oTabNode.rows[0].cells.length){

**for**(**var** x=0;x<oTabNode.rows.length;x++){

oTabNode.rows[x].deleteCell(colVal-1);

}

}**else**{

alert("要删除的列不存在！");

}

}

<input type=*"text"* name=*"delrow"* /> <input type=*"button"* value=*"删除行"* onclick="delrow()"/>

<br/>

<input type=*"text"* name=*"delcol"* /> <input type=*"button"* value=*"删除列"* onclick="delcol()"/>

### Charter11:DOM(示例-行颜色间隔显示并高亮)

**.css设置：**

**table**{

border:*#8080ff* *1px* *solid*;

width:*500px*;

border-collapse: *collapse*;

/\*

collapse:如果可能，边框会合并为一个单一的边框。会忽略 border-spacing 和 empty-cells 属性\*/

}

**table** **td**{

border:*#aaffff* *1px* *solid*;

/\* background-color: #aaaaaa; \*//\* 单元格td背景颜色会覆盖one,two写入的背景颜色。 \*/

padding:*1px*;

}

**table** **th**{

border:*#dddddd* *1px* *solid*;

background-color: *rgb(200,200,200)*;/\* 红、绿、蓝三种颜色的不同比例叠加。 \*/

padding:*1px*;

}

Html:

<style type=*"text/css"*>

*.one*{

background-color: *#faaaaa*;

}

*.two*{

background-color: *#ddddaa*;

}

*.over*{

background-color: *#70d0fa*;

}

</style>

<script type=*"text/javascript"*>

**var** name;

**function** trColor(){

/\*

思路：

1，因为要操作行的样式，所以要先获取表格对象。

2，获取所有被操作的行对象。

3，遍历行并给每一行指定样式。

\*/

**var** oTaNode = document.getElementById("info");

**var** collTrNodes = oTaNode.rows;

**for**(**var** x=1;x<collTrNodes.length;x++){

**if**(x%2==1){

collTrNodes[x].className="one";

}**else**{

collTrNodes[x].className="two";

}

collTrNodes[x].onmouseover = **function**(){

name=**this**.className;

**this**.className="over";

}

collTrNodes[x].onmouseout = **function**(){

**this**.className=name;

}

}

}

onload=**function**(){

trColor();

};

**function** over(node){

name = node.className;

node.className = "over";

}

**function** out(node){

node.className = name;

}

### Charter12:DOM(示例-表格排序)

**var** flag=**true**;

**function** sortTable(){

/\*

思路：

1，排序，就需要数组。获取需要参与排序的行为对象 数组。

2，对行数中的每一行的年龄单元格数组进行比较，并完成行对象在数组中的位置置换。

3，将拍好序的数组重新添加回表格。

\*/

**var** oTabNode = document.getElementById("info");

**var** collTrNodes = oTabNode.rows;

// 定义一个临时容器。存储需要排序的行对象。

**var** trArr = [];

// 遍历原行集合，并将需要排序的行对象存储到临时容器中。

**for**(**var** x=0;x<collTrNodes.length;x++){

trArr[x-1]=collTrNodes[x];

}

// 对临时容器进行排序。

mySort(trArr);

**if**(flag){

// 将排完序的行对象添加进表格。

**for**(**var** x=0;x<trArr.length;x++){

// oTabNode.childNodes[0].appendChild(trArr[x]);

trArr[x].parentNode.appendChild(trArr[x]);//???数组的父节点？

}

flag=**false**;

}**else**{

// 将排完序的行对象添加进表格。

**for**(**var** x=trArr.length-1;x>=0;x--){

// oTabNode.childNodes[0].appendChild(trArr[x]);

trArr[x].parentNode.appendChild(trArr[x]);//???数组的父节点？

}

flag=**true**;

}

}

**function** mySort(trArr){

**for**(**var** x=0;x<trArr.length-1;x++){

**for**(**var** y=x+1;y<trArr.length;y++){

//正则。

**if**(parseInt(trArr[x].cells[1].innerHTML)>parseInt(trArr[y].cells[1].innerHTML)){

**var** temp = trArr[x];

trArr[x] = trArr[y];

trArr[y] = temp;

}

}

}

}

### Charter13:DOM(示例-全选商品列表)

// 实现全选功能。

**function** checkAll(node){

/\*

将全选的box的checked状态赋值给所有的itembox的checked。

\*/

**var** sum = 0;

**var** collItemNodes = document.getElementsByName("item");

**for**(**var** x=0;x<collItemNodes.length;x++){

collItemNodes[x].checked = node.checked;

sum+=parseInt(collItemNodes[x]);

}

**var** sSum = sum+"元";

document.getElementById("sumid").innerHTML=sSum.fontcolor("red");

}

## Day06:

### Charter1:DOM(示例-邮件列表)

<script type=*"text/javascript"*>

**var** name;

// 行颜色间隔显示功能。

**function** trColor(){

// 1，获取表格对象。

**var** oTabNode = document.getElementById("mailtable");

// 2,获取行对象。

**var** collTrNodes = oTabNode.rows;

// 3，对所有需要设置背景的行对象进行遍历。

**for**(**var** x=1;x<collTrNodes.length-1;x++){//最后一行和第一行都不需要遍历

**if**(x%2==1){

collTrNodes[x].className="one";

}**else**{

collTrNodes[x].className="two";

}

collTrNodes[x].onmouseover = **function**(){

name = **this**.className;

**this**.className = "over";

}

collTrNodes[x].onmouseout = **function**(){

**this**.className = name;

}

}

}

onload = **function**(){

trColor();

}

// 复选框的全选动作。

**function** checkAll(node){

// 获取所有的mail复选框。

**var** collMailNodes = document.getElementsByName("mail");

**for**(**var** x=0;x<collMailNodes.length;x++){

collMailNodes[x].checked = node.checked;

}

}

// 定义操作复选框按钮的方法。

**function** checkAllByBut(num){

// 获取所有的mail复选框。

**var** collMailNodes = document.getElementsByName("mail");

**for**(**var** x=0;x<collMailNodes.length;x++){

**if**(num>1){

collMailNodes[x].checked = !collMailNodes[x].checked;

}**else**{

collMailNodes[x].checked = num;

//0和1分别对应false和true。

}

}

}

// 删除所选邮件。

**function** deleteMail(){

**if**(!confirm("你真的要删除该邮件吗？")){

**return**;

}

**var** collMailNodes = document.getElementsByName("mail");

**for**(**var** x=0;x<collMailNodes.length;x++){

**if**(collMailNodes[x].checked){

//集合是名为mail的各个节点。该节点的父节点是td,td的父节点是tr。

**var** oTrNode = collMailNodes[x].parentNode.parentNode;

//通过tr的父节点删除tr节点。

oTrNode.parentNode.removeChild(oTrNode);

// 但凡remove都会改变集合的长度。

x--;

}

}

trColor();

}

</script>

<th>

<input type=*"checkbox"* name=*"all"* onclick="checkAll(this)"/>全选

</th>

<th colspan=*"3"*> <!-- 将4列合并成一列 -->

<input type=*"button"* value=*"全选"* onclick="checkAllByBut(1)" />

<input type=*"button"* value=*"取消全选"*onclick="checkAllByBut(0)" />

<input type=*"button"* value=*"反选"* onclick="checkAllByBut(2)"/>

<input type=*"button"* value=*"删除所选附件"* onclick="deleteMail()"/>

### Charter2:DOM(示例-调查问卷)

<!-- 问卷调查 -->

<!-- script可以放在body中 -->

<script type=*"text/javascript"*>

**function** showContext(node){

**var** oDivNode = document.getElementById("contentid");

**with**(oDivNode.style){

**if**(node.value=='yes'){

display = "block";

}**else**{

display = "none";

}

}

}

</script>

<div>

您要参与问卷调查吗：

<input type=*"radio"* name=*"wenjuan"* value=*"yes"* onclick="showContext(this)"/>是

<input type=*"radio"* name=*"wenjuan"* value=*"no"* checked=*"checked"* onclick="showContext(this)"/> 否

</div>

<br/>

<div id=*"contentid"*>

问卷调查内容：<br/>

您的姓名：<input type=*"text"* /><br/>

您的邮箱：<input type=*"text"* />

</div>

### Charter3:DOM(示例-性格测试)

**function** showResult(){

//1,判断是否有答案被选中。

//获取所有no1的radio。并遍历判断checked状态。

**var** oNo1Nodes = document.getElementsByName("no1");

**var** flag = **false**;

**var** val;

**for**(**var** x=0; x<oNo1Nodes.length; x++){

**if**(oNo1Nodes[x].checked){

flag = **true**;

val = oNo1Nodes[x].value;

**break**;

}

}

**if**(!flag){

// alert("")

document.getElementById("errinfo").innerHTML = "没有任何答案被选中".fontcolor("red");

**return**;

}

**if**(val>=1 && val<=3){

document.getElementById("res\_1").className = "open";

document.getElementById("res\_2").className = "close";

}

**else**{

document.getElementById("res\_1").className = "close";

document.getElementById("res\_2").className = "open";

}

}

<!--2，性格测试。-->

<h2>欢迎您参与性格测试：</h2>

<div>

<h3>第一题：</h3>

<span>您喜欢的水果是什么？</span>

<ul id=*"no1ul"*>

<li><input type=*"radio"* name=*"no1"* value=*"1"* />葡萄</li>

<li><input type=*"radio"* name=*"no1"* value=*"2"* />西瓜</li>

<li><input type=*"radio"* name=*"no1"* value=*"3"* />苹果</li>

<li><input type=*"radio"* name=*"no1"* value=*"4"* />芒果</li>

<li><input type=*"radio"* name=*"no1"* value=*"5"* />樱桃</li>

</ul>

</div>

<div>

<input type=*"button"* value=*"查看测试结果"* onclick="showResult()" />

<span id=*"errinfo"*></span>

<div id=*"res\_1"* class=*"close"*>1-3分：你的性格内向并扭曲，建议...</div>

<div id=*"res\_2"* class=*"close"*>4分以上：你的性格外向并分裂，建议...</div>

</div>

### Charter4:DOM(示例-下拉菜单-选择颜色)

### Charter5:DOM(示例-下拉菜单-选择城市)

**function** selectCity(){

**var** collCities = [['选择城市'],

['海淀区','朝阳区','东城区','西城区'],

['济南','青岛','烟台','威海'],

['沈阳','大连','鞍山','抚顺'],

['石家庄','保定','邯郸','廊坊']];

// var arr ={"beijing":['海淀区','朝阳区','东城区','西城区']};

// 获取两个下拉菜单对象。

**var** oSelNode = document.getElementById("selid");

**var** oSubSelNode = document.getElementById("subselid");

// 获取到底选择的是哪个省。

**var** index = oSelNode.selectedIndex;

// 通过角标到容器中去获取对应的城市数组。

**var** arrCities = collCities[index];

// 将子菜单中的内容清空一下。

// for(var x=0;x<oSubSelNode.length;){

// 从0角标元素开始清除，即从选择城市开始。

// oSubSelNode.removeChild(oSubSelNode.options[x]);

// }

// 清除动作。

oSubSelNode.length = 0;

// 遍历数组。并将该数组的元素封装成option对象，添加到子菜单中。

**for**(**var** x=0;x<arrCities.length;x++){

**var** optNode = document.createElement("option");

optNode.innerHTML = arrCities[x];

oSubSelNode.appendChild(optNode);

}

}

<select id=*"selid"* onchange=*"selectCity()"*>

<option>选择省市</option>

<option>北京</option>

<option>山东</option>

<option>辽宁</option>

<option>河北</option>

</select>

<select id=*"subselid"*>

<option>选择城市</option>

</select>

### Charter6:DOM(示例-添加删除附件)

**function** addFile(){

/\*

因为文件选取框定义到行对象中。

所以只要给表格创建新的行和单元格即可。

\*/

**var** oDivNode = document.getElementById("fileid");

**var** oTrNode = oDivNode.insertRow();

**var** oTdNode\_file = oTrNode.insertCell();

**var** oTdNode\_del = oTrNode.insertCell();

oTdNode\_file.innerHTML="<input type='file'/>";

oTdNode\_del.innerHTML="<a href='javascript:void(0)' onclick='deleteFile(this)'>删除附件</a>";

oTdNode\_del.innerHTML = "<img src='1.jpg' alt='删除图片' onclick='deleteFile(this)'/>";

}

**function** deleteFile(node){

**var** oTrNode = node.parentNode.parentNode;

oTrNode.parentNode.removeChild(oTrNode);

}

<table id=*"fileid"*>

<tr>

<td><a href=*"javascript:void(0)"* onclick="addFile()">添加附件</td>

</tr><br/>

<!-- <tr>

<td><input type="file"/><a href="javascript:void(0)">删除附件</a></td>

</tr> -->

</table>

### Charter7:DOM(示例-表单校验-涉及的事件和信息提示方式)

<!-- 表单校验 -->

<script type=*"text/javascript"*>

//校验用户名

**function** checkUser(){

// alert("haha");

**var** oUserNode = document.getElementsByName("user")[0];

// alert(oUserNode.value);

**var** name = oUserNode.value;

**var** oSpanNode = document.getElementById("userspan");

**if**(name=="abc"){

oSpanNode.innerHTML = /\* "<img src='1.jpg' />"; \*/"用户名正确".fontcolor("green");

}**else**{

oSpanNode.innerHTML = "用户名错误".fontcolor("red");

}

}

</script>

<form action=*""*>

用户名称：<input type=*"text"* name=*"user"* onblur="checkUser()"/>

<!-- 不能用div，因为div结束会换行，可以考虑span -->

<span id=*"userspan"*></span>

<br/>

输入密码：<input type=*"text"* name=*"psw"* />

</form>

### Charter8:DOM(示例-表单校验-涉及的正则校验)

// 语法1：re = /pattern/[flags]

// 语法2：re = new RegExp("pattern",["flags"])//flags:g：全文查找 i：忽略大小写 n：多行查找

//校验用户名

**function** checkUser(){

// alert("haha");

**var** oUserNode = document.getElementsByName("user")[0];

// alert(oUserNode.value);

**var** name = oUserNode.value;

// 定义正则表达式。//表达式：^$：头和尾

// 两种方式的区别：\\关于反斜杠的注释问题，如下：

// "^\\d{4}$"和/^\d{4}$/分别是方式1和2。

// 方式1：

**var** reg = **new** RegExp("^[a-z]{4}$","i");//i忽略大小写

// reg = new RegExp("^[0-9]{4}$"); //必须是四个数字。

// 方式2：

reg = /^[0-9]{4}$/;

**var** oSpanNode = document.getElementById("userspan");

// if(name=="abc"){

**if**(reg.test(name)){

oSpanNode.innerHTML = /\* "<img src='1.jpg' />"; \*/"用户名正确".fontcolor("green");

}**else**{

oSpanNode.innerHTML = "用户名错误".fontcolor("red");

}

}

### Charter9:DOM(示例-表单校验-表单提交的两种方式)

发现快捷键：fn+方向键，实现光标所在代码的移动，左右方向可以在一行内，移动到开头和结尾。

### Charter10:DOM(示例-表单校验-涉及的正则校验)

Charter1:

Charter2:

Charter3:

Charter4:

Charter5:

Charter6:

Charter7:

Charter8:

Charter9:

Charter10:

Charter11:

Charter12:

Charter13:

Charter14:

Charter15:

Charter16:

Charter17:

Charter18:

Charter19:

## Day07:

DW遇到问题1：改变dw的代码字体大小：创建完成一个.html文件后，编辑-->首选项-->字体

2：快捷键：shift+enter：<br/>

3：<img>标签必须有alt。

4：<html lang="en">  
向搜索引擎表示该页面是[html语言](https://www.baidu.com/s?wd=html%E8%AF%AD%E8%A8%80&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)，并且语言为英文网站，其"lang"的意思就是“language”，语言的意思，而“en”即表示english  
你的页面如果是中文页面，可将其改为  
<html lang="zh">  
zh即表示中文

5：Xml5619文档语法错误：dw软件的问题。

6：notepad++对比： Align Matches (对齐匹配)   
　　　 Ignore Spacing (忽略空格)   
　　　 Detect Moves (检测移动)   
　　　 Navigation bar (导航栏)

7：将jscript放在头和放在body有什么区别？

JavaScript应放在哪里  
页面中的JavaScripts会在浏览器加载页面的时候被立即执行，我们并不希望总是这样，有时候我们想让一段脚本在页面加载的时候执行，而有时候我们想在用户触发一个事件的时候执行脚本。

head 部分中的脚本: 需调用才执行的脚本或事件触发执行的脚本放在HTML的head部分中。当你把脚本放在head部分中时，可以保证脚本在任何调用之前被加载。

body 部分中的脚本: 当页面被加载时执行的脚本放在HTML的body部分。放在body部分的脚本通常被用来生成页面的内容。

body 和 head 部分可同时有脚本:你可在文件中放无数的脚本，因此你的文件中可以在body和head部分同时存在脚本。

外部脚本的使用

有时候你可能想在几个页面中运行同样的脚本程序， 而不需在各个页面中重复的写这些代码。这时你就要用到外部脚本。你可以把脚本写在一个外部文件中，保存在扩展名为 .js的文件中。

 1：将JavaScript标识放置... 在头部之间，使之在主页和其余部分代码之前预先装载，从而可使代码的功能更强大； 比如对\*.js文件的提前调用。 也就是说把代码放在区在页面载入的时候，就同时载入了代码，你在区调用时就不需要再载入代码了，速度就提高了，这种区别在小程序上是看不出的，当运行很大很复杂的程序时，就可以看出了。当然也可以将JavaScript标识放置在... 主体之间以实现某些部分动态地创建文档。 这里比如制作鼠标跟随事件，肯定只有当页面加载后再进行对鼠标坐标的计算。或者是filter滤镜与javascript的联合使用产生的图片淡入淡出效果

**2：放入html的head,是页面加载前就运行，放入body中，则加载后才运行javascript的代码~~~**

**所以head里面的先执行。**

3：要是function就没有区别要是会跟放的位置有关一个事head加载完就 alert 一个是要跟在body中加载循序alert

8:遇到代码颜色全都一样，说明该行代码出错！！！！

### 课1：19-jQuery-选择器的使用

1）知识点：

1. Window.onload = = $(document).ready();
2. $(“”)获取元素 标签名、.类名、#id
3. 事件源.方法(function(){

})

4.

2）jQuery特点：

1. 链式编程
2. jq.show(3000).html(内容)
3. 相当于
4. jq.show(3000)
5. jq..html(内容)
6. 隐式迭代

A)隐式适用for循环、迭代

1. 如何实用jQuery
2. 引包：src=””，一般引在开始，解析时可以最先解析。如果没有引的话，会在解析时报错。
3. 入口函数
4. 绑定时间

$(事件源).事件(function(){事件驱动函数})

4）jQuery入口函数：

1.文档加载完毕，图片未加载的时候执行函数

A)$(document).ready(function(){})

B)$(function(){})

2.文档和图片都加载完毕执行函数

A)$(window).ready(function(){})

3.JS入口函数只能有一次，多次会覆盖

4.jq可以有多个入口函数

1. jQuery对象是一个数组，包含原生jsDOM对象
2. 原生JS对象是一个原生对象
3. 先获取DOM对象

jQuery对象名 = $(js对象名)

1. js对象转换为jQuery对象
2. jquery对象名[索引值]
3. Jquery对象名.get[索引值]
4. jQuery选择器

基本：#id、.类、标签名、\*（通配符）

层级：

1. 子代选择器>
2. 后代选择器 空格隔开
3. 基本过滤选择器
4. eq(index):遍历元素数组
5. odd:获取偶数位索引元素
6. even:获取奇数位索引元素

筛选选择器

1. find(selector):找到父元素的所有子元素
2. 父元素.find(‘要查找的子元素’)
3. Parent：

1.父元素.children(‘要查找的子元素’)

1. eq(index)
2. children:亲儿子
3. sibilings：所有的兄弟节点（不包括本身）
4. next:下一个兄弟节点

遇见小问题：

1.input type="button"和button的区别：

<input id="btn1" type="button"/>不是双边标签，标签中间的内容不能自定义，按钮的提示只能通过value属性提示

<button id="btn2"></button>中间可以自定义，比如在中间加入<img />，可以得到一个图片按钮

<button></button>的功能明显比<input type="button">更多

修改input类型的按钮值得用("#btn1").value(str),修改button类型的按钮的值用("#btn2").html(str)

唯一禁止使用的元素是图像映射，因为它对鼠标和键盘敏感的动作会干扰表单按钮的行为。请始终为按钮规定type属性。InternetExplorer的默认类型是"button"，而其他浏览器中（包括W3C规范）的默认值是"submit"。

上面的图像映射指的是？？？

2.opacity???:0;/\*不透明度\*/

3.关于.ready()事件

当 DOM（文档对象模型） 已经加载，并且页面（包括图像）已经完全呈现时，会发生 ready 事件。

由于该事件在文档就绪后发生，因此把所有其他的 jQuery 事件和函数置于该事件中是非常好的做法。正如上面的例子中那样。

ready() 函数规定当 ready 事件发生时执行的代码。

ready() 函数仅能用于当前文档，因此无需选择器。

1. $(document).ready(function(){}),$().ready(function(){})和$(function(){})三个有区别么？
2. 三者都是一样的，  
   最完整的写法是：$(document).ready(function(){})  
   ready() 函数仅能用于当前文档，因此无需选择器。  
   所以document选择器可以不要，那么就可以写成：  
   $().ready(function(){})  
   最后$的默认事件就是ready，所以，ready也可以省略，写成：  
   $(function(){});
3. 动画不执行等问题，最根本的方法，就是看控制台的结果。
4. multiple属性：滚动条

### 课2：20-jQuery-DOM操作

1）知识点：

1. 样式属性操作.css
2. 操作方法：
3. 设置多个样式
4. 设置单个样式
5. 获取样式属性操作

//操作方法1： css(json);

$(function(){

//操作方法1（设置属性）

$("div").css({"width":100,"height":100,"background":"pink","border":"1px solid "});

//操作方法2（设置属性） css(属性，值);

$("div").css("background","pink");

//操作方法3（获取属性） css（属性）;

console.log($("div").css("width"));

});

1. 类操作
2. .addClass添加类

$(“div”).addClass(“class”);

1. .removeClass删除类

$(“div”).removeClass(“class”);

1. .hasClass判断类

$(“div”).hasClass(“class”);

1. .toggleClass：切换类

$(“div”).toggleClass(“class”)

1. 动画
2. 隐藏显示动画
3. .show/hide通过display方法实现显示和隐藏。作用===.css(“display”,“block”);
4. .show/hide(毫秒值):固定时间显示

i.通过宽和高还有透明度还有display控制

ii.宽和高，还有opacity仅在变换时期发生变化，加载完毕后，固定某一定值。

C) .show/hide(字符串)

1. slow慢：600ms；

2. normal正常:400ms；

3. fast快：200ms

D) .show/hide(毫秒值,function(){});

回调函数（callback）。

1. 滑入滑出动画
2. 滑入slideDown
3. 滑出slideUp
4. 滑入滑出切换slideToggle
5. 淡入淡出动画

i.淡入fadeDown

ii.淡出fadeUp

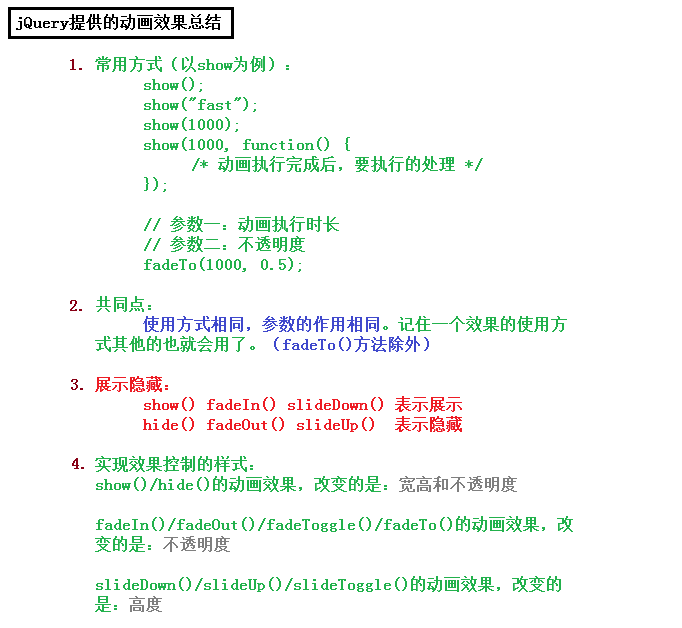
iii.淡入滑出切换fadeToggle

Iv.改变透明度：fateTo(毫秒值，透明度) 0：全透；1：全不透

规律的体现：

jQuery提供的这几个动画效果控制的CSS属性包括：高度、宽度、不透明度。{height:400px; width:300px; opacity:.4;}

这三个CSS属性的共性是：属性值只有一个，并且这个值是数值（去掉单位后）。



1. 自定义动画

animate({param},speed,callback)

动画支持的属性：

[http://www.w3school.com.cn/jquery/effect\_anima](http://www.w3school.com.cn/jquery/effect_animate.asp)[te.asp](http://www.w3school.com.cn/jquery/effect_animate.asp)

1. 停止动画

$("div").stop(); //都不写，默认两个都是false;

// 第一个参数表示后续动画是否要执行

//（true:后续动画不执行 ;false:后续动画会执行）

// 第二个参数表示当前动画是否执行完

//（true:立即执行完成当前动画 ;false:立即停止当前动画）

//$(selector).stop(clearQueue,jumpToEnd);

1. clearQueue:是否取消以队列动画；
2. Toend:是否当前动画立即完成。

DOM创建元素的方法：1.createElement,2..innerhtml,3.document.write

jQuery沿袭了两种方法。

方法1：

创建节点1：$("标签")：类比于js中的document.createElement

console.log($("<li class='aaa'>我是li标签</li>"));

创建节点2：$("标签").html()：类比于js中的innerHTML属性

console.log($("ul").html("<li>我是html创建出来的li标签</li>"));

jQuery的添加元素方法：

1. 在盒子末尾添加。
2. $(selector).append($node);优选
3. $node.appendTo($(selector));
4. 在盒子首部添加。
5. $("ul").prepend(jqNewLi);
6. jqNewLi.prependTo($("ul"));
7. 两种操作兄弟元素的方法。
8. after();

$("li").after(jqNewLi);

1. Before();

$("li").before(jqNewLi);

jQuery清空和删除元素

清空li所在的ul

方法1：.html();

$("ul").html("");

方法2：.empty();

$("ul").empty();

删除指定的li。---自杀

$("li:eq(0)").remove();

jQuery

1. 复制节点

var newNode = $(复制的节点).clone();

返回复制后的新元素$(selector).append(newNode );

5.

小细节：1：cursor:光标的意思。

2：定义字符串，然后显示插入条光标(把字符串转换成数组)

var str = "红鲤鱼与绿鲤鱼与驴";

var arr = str.split("");

3：setInterval指定的是“开始执行”之间的间隔，并不考虑每次任务执行本身所消耗的事件。因此实际上，两次执行之间的间隔会小于指定的时间。比如，setInterval指定每100ms执行一次，每次执行需要5ms，那么第一次执行结束后95毫秒，第二次执行就会开始。如果某次执行耗时特别长，比如需要105毫秒，那么它结束后，下一次执行就会立即开始。

为了确保两次执行之间有固定的间隔，可以不用setInterval，而是每次执行结束后，使用setTimeout指定下一次执行的具体时间。

具体见：<http://caibaojian.com/setinterval.html>

### 课3：21-jQuery-UI

1）操作form表单

获取元素，绑定属性的方法：

var jqinp = $("input").eq(0);

var jqinp2 = $("input:checkbox");

1. $(selector).attr(“属性”，属性值)

特点：一次性

1. 添加checked,selected,display用prop()方法来设置。

特点：多次使用

val()方法：设置或获取表单元素中的标签value的属性值。例如：input,select,textarea的值

将输入的值清零

$("#inValue").val("");

text()方法：（不识别标签）

设置或获取匹配元素的文本内容

html()方法：（识别标签）

1. 尺寸操作

$(selector).height():设置或获取匹配元素的高度值。

内容高度和宽度不包括padding,margin,还有border。

$(selector).width():设置或获取匹配元素的高度值

内容高度和宽度不包括padding,margin,还有border。

1. $(selector).offset().left：

获取或设置元素相对于文档的位置

无参数表示获取，返回值为：{left:num, top:num}，值是相对于document的位置

$(selector).offset();

有参数表示设置，参数推荐使用数值类型

$(selector).offset({left:100, top: 150});

注意点：设置offset后，如果元素没有定位(默认值：static)，则被修改为relative

4.$(selector).position() .left;

获取相对于其最近的具有定位的父元素的位置。

（事实上，就是设置的left,top距离）

注意：只能获取，不能设置。

知识细节：overflow:auto,scroll,hide,inherit,visiable

这个属性定义溢出元素内容区的内容会如何处理。如果值为 scroll，不论是否需要，用户代理都会提供一种滚动机制。因此，有可能即使元素框中可以放下所有内容也会出现滚动条。

1.事件绑定

简单事件绑定 >> bind事件绑定 >> delegate事件绑定 >> on【重点】

$(selector).on(events[,selector][,data],handler);

2.事件解绑

i. unbind() 方式

解绑 bind方式绑定的事件

$(selector).unbind(); //解绑所有的事件

$(selector).unbind(“click”); //解绑指定的事件

II. ndelegate() 方式

解绑delegate方式绑定的事件

$( selector ).undelegate(); //解绑所有的delegate事件

$( selector).undelegate( “click” ); //解绑所有的click事件

iii.**解绑on方式绑定的事件（重点）**

// 解绑匹配元素的所有事件

$(selector).off();

// 解绑匹配元素的所有click事件

$(selector).off(“click”);

// 解绑所有代理的click事件，元素本身的事件不会被解绑

$(selector).off( “click”, “\*\*” );

1. 事件触发

简单事件触发

$(selector).click(); //触发 click事件

trigger方法触发事件，**触发浏览器行为**

$(selector).trigger(“click”);

triggerHandler触发 事件响应方法，**不触发浏览器行为**

**比如:文本框获得焦点的默认行为**

$(selector).triggerHandler(“focus”);

5.jQuery事件对象介绍

event.data 传递给事件处理程序的额外数据

event.currentTarget 等同于this，当前DOM对象

event.pageX  鼠标相对于文档左部边缘的位置

**event.target** 触发事件源，不一定===this

**event.stopPropagation()；** 阻止事件冒泡

**event.preventDefault();**  阻止默认行为

event.type 事件类型：click，dbclick…

event.which 鼠标的按键类型：左1 中2 右3

event.keyCode 键盘按键代码

案例：按键变色

4）链式编程

1.链式编程原理：return this;

通常情况下，只有设置操作才能把链式编程延续下去。因为获取操作的时候，会返回获取到的相应的值，无法返回 this。

end(); // 结束当前链最近的一次过滤操作，并且返回匹配元素之前的状态。

2.隐式迭代

隐式迭代的意思是：在方法的内部会为匹配到的所有元素进行循环遍历，执行相应的方法；而不用我们再进行循环，简化我们的操作，方便我们调用。

如果获取的是多元素的值，大部分情况下返回的是第一个元素的值。

3.each方法

如果要对每个元素做不同的处理，这时候就用到了each方法

作用：遍历jQuery对象集合，为每个匹配的元素执行一个函数

// 参数一表示当前元素在所有匹配元素中的索引号

// 参数二表示当前元素（DOM对象）

$(selector).each(function(index,element){});

Element是一个 js对象，需要转换成jquery对象

5）多库共存

此处多库共存指的是：jQuery占用了$ 和jQuery这两个变量。当在同一个页面中引用了jQuery这个js库，并且引用的其他库（或者其他版本的jQuery库）中也用到了$或者jQuery这两个变量，那么，要保证每个库都能正常使用，这时候就有了多库共存的问题。

解决方式：此后，只能使用jQuery来调用jQuery提供的方法。

i.

让1.11.1放弃$的使用权

$.noConflict();

console.log($.fn.jquery);//1.8.2

console.log(jQuery.fn.jquery);//1.11.1

ii.

放弃两个符号的使用权，同时定义一个新的使用权。

var MrLv = $.noConflict(true);

console.log($.fn.jquery);//1.8.2

console.log(jQuery.fn.jquery);//1.8.2

console.log(MrLv.fn.jquery);//1.11.1

6）插件机制

1.引用插件

animate()自定义动画：不支持背景的动画效果，可以通过插件来实现：jQuery.color.js

2.制作插件

建jQuery插件：

<http://learn.jquery.com/plugins/basic-plugin-creation/>

全局jQuery函数扩展方法

$.pluginName = function() {};

jQuery对象扩展方法

$.fn. pluginName = function() {};

知识细节整理：

1）margin:auto属性

首先auto是做什么的？

定义auto元素，因元素类型和上下文而异。在边距中，auto可以表示两种情况：占用可用空间或0 px。这两个将为元素定义不同的布局。

**“自动”占用可用空间**

通过分配auto元素的左右边距，它们可以平等地占据元素容器中的可用水平空间 - 因此元素将居中。

auto在浮动，内联和绝对元素中不起作用。所有这些元素已经决定了它们的布局，所以没有auto用于边距并期望它像这样集中。

**auto设置垂直会是怎样的呢？**

auto在顶部和底部边距中总是计算为0px（绝对元素除外）。

说实话 auto这个边距问题，我还是没搞懂。

2）块级元素，内联元素，内联块级元素特点：

**块级元素特点：**

1、每个块级元素都从新的一行开始，并且其后的元素也另起一行。（霸道，一个块级元素独占一行）

2、元素的高度、宽度、行高以及顶和底边距都可设置。

3、元素宽度在不设置的情况下，是它本身父容器的100%（和父元素的宽度一致）

常用的块状元素有：

<div>、<p>、<h1>...<h6>、<ol>、<ul>、<dl>、<table>、<address>、<blockquote> 、<form>

设置display:block；可以将元素转换块级元素。

**内联元素特点：**

1、和其他元素都在一行上；

2、元素的高度、宽度及顶部和底部边距不可设置；

3、元素的宽度就是它包含的文字或图片的宽度，不可改变。

常用的内联元素有：

<a>、<span>、<br>、<i>、<em>、<strong>、<label>、<q>

设置display:inline;可以将块状元素转换为内联元素

**内联块级元素特点：**（同时具备内联元素、块级元素的特点）

1、和其他元素都在一行上；

2、元素的高度、宽度、行高以及顶和底边距都可设置！

常用的内联块状元素有：

<img>、<input>

display:inline-block；float:left / right；position：absolute/fixed;可以将元素设置为内联块级元素。

具体见：https://blog.csdn.net/ycq521131/article/details/82590308

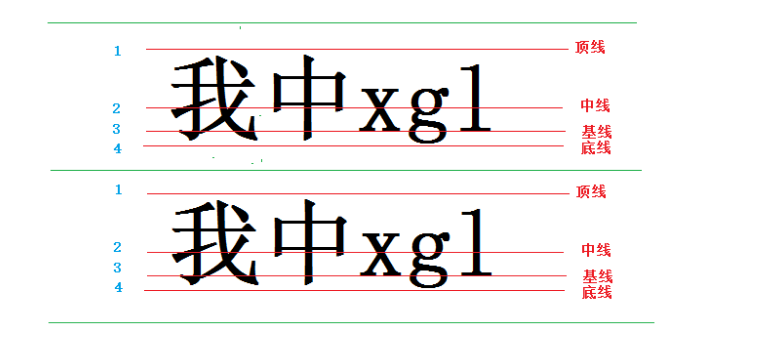
3）

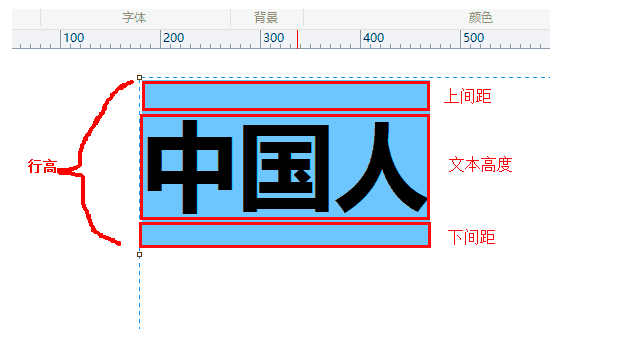
一行文本的行高为：上间距 + 文本的高度+下间距，并且上间距是等于下间距的。  
我们还可以基本上这样认为：行高是两行文字基线之间的距离，也是两行文字顶线之间的距离，两行文字中线之间的距离。

对于块级元素，line-height指定了元素内部line-boxes的**最小**高度

对于非替代行内元素，line-height用于计算line-box的高度

对于替代行内元素，例如input、button，line-height没有影响





4）光标cursor

cursor：pointer意思是是计算机中把鼠标指针的形状弄成一只伸出食指的手，

cursor：default 箭头，

cursor：crosshair 十字，

cursor：progress 沙漏等等。

5）font：12px/24px;

前面12px是指字体大小，后面24px是行高。  
分开写如下：  
font-size: 12px;  
line-height: 24px;

6）&. 和&.fn.异同：

见文档：$.和$.fn.的异同.doc

## Day08-12:bootstrap入门

### 课1：第1章 Bootstrap简介

Bootstrap：是由Twitter发布的一种用于开发响应式布局、移动设备有限的web项目。

将常见的CSS布局小组件和JS插件进行了完整并完善的封装。



认识BootCDN:

CDN的全称是Content Delivery Network，即内容分发网络。

免费 CDN 加速服务

www.ip138.com：可用于查询国内ip地址位置。

（www.bootcss.com：中国 北京 北京 阿里云）

Bootstrap包：

1. Css
2. Fonts
3. js

开发工具：

1. notepad++
2. Dreamweaver cs6
3. HBuilder
4. Webstorm

第一步：引入移动设备优先meta

第二步：

1. Bootstrap 核心 CSS 文件 ： bootstrap.min.css
2. Bootstrap 主题文件 ：bootstrap-theme.min.css
3. Bootstrap 核心 JavaScript ：文件 bootstrap.min.js

### 课2：第2章 Bootstrap栅格系统原理

学习小知识：网页截屏功能：fn+print screen+alt截下当前页面的图片-->打开画图工具-->alt+v

#### 栅格参数



#### 媒体查询：

@media(max-width:767px){

/\*在小于767px的屏幕里，这里的样式菜生效\*/

div{

background: #2AE07D;

}

}

### 课3：第3章 Bootstrap栅格系统用法

1. container & container-fluid

<div class="container">

<h1>Container</h1>

</div>

<div class="container-fluid">

<h1>Container</h1>

</div>

1. 布局容器

列组合：col-md-\*

列偏移：col-md-offset-\*

列嵌套：

<div class="row">

<div class="col-md-8">

<div class="row">

<div class="col-md-6">6</div>

<div class="col-md-6">6</div>

</div>

</div>

<div class="col-md-4">col-md-4</div>

</div>

列排序：col-md-push-\* col-md-pull-\*

3.栅格参数

1. 跨设备组合定义

<div class="row">

<div class="col-md-8 col-xs-6 col-sm-3">8</div>

<div class="col-md-4 col-xs-6 col-sm-3">4</div>

</div>

1. 清除浮动

<div class="row">

<div class="col-xs-6 col-sm-3">col-xs-6 col-sm-3 1

<br/> 1234567890

</div>

<div class="col-xs-6 col-sm-3">col-xs-6 col-sm-3 2</div>

<div class="clearfix visible-xs"></div>

<div class="col-xs-6 col-sm-3">col-xs-6 col-sm-3 3</div>

<div class="col-xs-6 col-sm-3">col-xs-6 col-sm-3 4</div>

</div>

4.禁止响应布局

1.删除viewpt的meta

2.为.container设为固定宽度

3.对导航移除折叠和展开行为

4....

### 课4：第4章 Bootstrap排版基础01

1. 排版前的基础

HTML5文档类型：a.<!DOCTYPE html> b.<mata charset=”utf-8”>

移动设备优先

<meta name="viewport" content="width=device-width, initial-scale=1">

initial-scale：初始缩放比例，也即是当页面第一次 load 的时候缩放比例

user-scalable：用户是否可以手动缩放

响应式图片

Bootstrap设置了一些全局样式：

body背景为白，margin:0

字体、大小、行间距都进行了设置：

所有默认样式都在normalize.less和scaffoliding.less

Normalize.css

bootstrap会使用第三方库，Normalize，是一个专门用于将不同浏览器的默认css效果特征统一的css库

排版与链接

2.标题：

h1 36px margin\_top:20px; margin-bottom:10px;

h2 30px margin\_top:20px; margin-bottom:10px;

h3 24px margin\_top:20px; margin-bottom:10px;

h4 18px margin\_top:10px; margin-bottom:10px;

h5 14px margin\_top:10px; margin-bottom:10px;

h6 12px margin\_top:10px; margin-bottom:10px;

<small>

1-3:65% 4-6:75%

### 课5：第5章 Bootstrap排版基础02

1）页面主题：

1. Body全局样式

body {

font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;

font-size: 14px;

line-height: 1.42857143;

color: #333;

background-color: #fff;

}

1. P全局样式

P{

Margin: 0 0 10px;

}

.lead {

margin-bottom: 20px;

font-size: 16px;

font-weight: 300;

line-height: 1.4;

}

1. 对齐方式

lead:text-right：右对齐、text-center：居中对齐

1. 强调文本

small

Strong：粗体强调

em斜体强调

cite

1. 缩略语abbr

<abbr title="跟我一起学习Bootstrap" class=”initialism”>学习Bootstrap</abbr>

.initialism {

font-size: 90%;//原字体的90%

text-transform: uppercase;//改为大写字体

}

1. 地址元素address

address {

margin-bottom: 20px;

font-style: normal;

line-height: 1.42857143;

}

1. 引用blockquote

<blockquote class="pull-right">

### 课6：第6章 bootstrap列表与代码样式

1. 列表
2. 无序列表
3. 有序列表
4. 去点列表：-list-unstyled
5. 内联列表：.list\_inline
6. dl列表
7. 水平列表dl.dl-horizontal
8. 代码
9. <code>显示单行内联
10. <kbd>显示用户输入代码
11. <pre>多行代码块
12. <var>变量
13. <samp>程序结果

### 课7：第7章 Bootstrap表格样式

注意：在做样式的时候一般是先做基本样式，再做其他样式

1. 基础样式table
2. 带背条纹表格.table-striped
3. 带边框.table-border
4. 鼠标悬停：table-hover
5. 紧凑型：table-condensed

### 课8：第8章 Bootstrap表单样式1

关于label的学习，看文档”body学习”。label主要起到作用，关联。

1. 基础表单：
2. 全局样式

.form-control：实现占满格的效果

.form-group：仅仅是margin-bottom的设置

1. 内联表单

form-inline：只能在大于768px上显示;

sr-only隐藏lable

1. 横向表单

在form上应用.form-horizontal

.form-horizontal .form-control：@media (min-width: 768px)

使用栅格系统

1. 表单控件

input：

select：option默认单选；加入multiple，成为多选

textarea：通过rows和cols来控制行和列数

checkbox&radio：.radio, .radio-inline, .checkbox, or .checkbox-inline.

（注意：radio的name需要一致，形成单选形式）

静态控件：<p class="form-control-static">email@example.com</p>

（该内容不可以操作）

1. 控制状态

焦点状态 autofocus

禁用状态 disable

被禁用的fieldset：disable属性（不可输入）

只读状态 readonly

校验状态

1. has-success
2. has-warning
3. has-error

Bootstrap 对表单控件的校验状态，如 error、warning 和 success 状态，都定义了样式。使用时，添加 .has-warning、.has-error 或 .has-success 类到这些控件的父元素即可。任何包含在此元素之内的 .control-label、.form-control 和 .help-block 元素都将接受这些校验状态的样式。



has-feedback：通过组件显示

input-lg input-sm；input和select：调整输入框或者选择框的大小

辅助文本：helpBlock

<span id="helpBlock" class="help-block">A block of help text that breaks onto a new line and may extend beyond one line.</span>



### 课9：第9章 Bootstrap表单样式2

### 课10：第10章 Bootstrap表单样式3

### 课11：第11章 Bootstrap按钮

Bootstrap按钮样式：

1. 按钮样式：
2. btn-danger
3. btn-default
4. btn-primary
5. btn-success
6. btn-info
7. btn-warning
8. btn-link
9. 按钮大小
10. 默认样式（不试用下列按钮样式大小）
11. btn-lg
12. btn-sm
13. btn-xs
14. btn-block块级（以将其拉伸至父元素100%的宽度）
15. 多标签支持 role=”button”
16. 活动状态 .active 有点击效果
17. 禁用状态 disable 没有点击效果

### 课12：第12章 Bootstrap图片与辅助样式

遇到细节问题：

图片颠倒或者翻转90度，给图片加设置：

-moz-transform:rotate(-90deg);  
 -webkit-transform:rotate(-90deg);

1. 响应式图片： .img-responsive
2. 图片形状：

img-rounded 四角呈弧形

img-circle 图像呈椭圆形状

img-thumbnail 有轮廓边框

1. 文本样式

柔和灰text-muted

主要蓝text-primary

成功绿text-success

信息蓝text-info

警告黄text-warning

危险红text-danger

1. 文本背景样式

主要蓝bg-primary

成功绿bg-success

信息蓝bg-info

警告黄bg-warning

危险红bg-danger

1. 辅助图标

关闭图标close（&times;）叉叉按钮两种形式：

<button type="button" class="close">&times;</button>

<a href="#" class="close">&times;</a>

向下箭头caret

<span class="caret"></span>

1. 内容浮动

.pull-right向右浮动

.pull-left

.cleaerfix 清除浮动

<div class="clearfix visible-sm hidden-xs">//在sm范围内可视浮动，在Xs范围内隐藏浮动

<div class="pull-left">居左</div>

<div class="pull-right">居右</div>

<div>ccc</div>

</div>

1. 隐藏与显示

.show显示

.hidden隐藏内容（style display:none）

.visible-md 在md范围内显示

### 课13：第13章 Bootstrap小图标

1. 小图标基本用法

<span class="glyphicon glyphicon-flag" ></span>

<a href="#"><span class="glyphicon glyphicon-duplicate"></span></a>

1. 实现原理
2. 应用场景
3. 可以在各种元素内容用
4. 工具栏

.btn-toolbar ：实现左浮动（float:left）

.btn-group

1. 输入框

<div class="col-sm-3 col-sm-offset-2">

<div class="input-group">

<span class="input-group-addon">

<span class="glyphicon glyphicon-hourglass"></span>

</span>

<input type="text" class="form-control"/>

</div>

</div>

### 课14：第14章 Bootstrap下拉菜单

1. 基本用法

<div class="container">

<div class="dropdown">

<button type="button" class="btn btn-info" data-toggle="dropdown">

<span class="glyphicon glyphicon-star"></span>

星座

<span class="caret"></span>

</button>

<ul class="dropdown-menu">

<li><a href="#">处女座</a></li>

<li><a href="#">天马座</a></li>

<li><a href="#">水瓶座</a></li>

<li><a href="#">天蝎座</a></li>

<li><a href="#">狮子座</a></li>

</ul>

</div>

</div>

1. 对齐

<ul class="dropdown-menu dropdown-menu-right">

1. 标题

<li class="dropdown-header">星座选择</li>

1. 分隔线

<li class="divider"><a href="#">天蝎座</a></li>

1. 禁用菜单

<li class="disabled"><a href="#">水瓶座</a></li>

1. 激活状态（默认选项）

<li class="active"><a href="#">天马座</a></li>

### 课15：第15章 Bootstrap按钮组

1. 基本用法

<div class="btn-group">

1. 按钮工具栏

<div class="btn-toolbar">

1. 按钮尺寸设置

<div class="btn-group btn-group-lg">

<a class="btn btn-info btn-sm">按钮组</a>

<a class="btn btn-info">按钮组</a>

<a class="btn btn-info">按钮组</a>

</div>

1. 嵌套分组

<div class="btn-group dropdown">

<button type="button" class="btn btn-info dropdown-toggle" data-toggle="dropdown"> //需要这两个来连接button和表

下拉

<span class="caret"></span>

</button>

<ul class="dropdown-menu">

<li class="dropdown-header">星座选择</li>

<li><a href="#">处女座</a></li>

<li class="active"><a href="#">天马座</a></li>

<li class="disabled"><a href="#">水瓶座</a></li>

<li class="divider"><a href="#">天蝎座</a></li>

<li><a href="#">狮子座</a></li>

</ul>

</div>

！注意： class="btn btn-info dropdown-toggle" data-toggle="dropdown"

必须先写btn 然后加入各种封装

1. 垂直分组

<div class="btn-group-vertical">

1. 自适应分组

### 课16：第16章 Bootstrap按钮下拉菜单

1. 组合式下拉菜单
2. 分离式下拉菜单
3. 按钮大小

btn-lg btn-sm ...

1. 向上弹起的菜单

dropup

### 课17：第17章 Bootstrap输入框

addon（插件）

1. 基本用法

<p>基本用法1</p>

<div class="input-group">

<span class="input-group-addon">@</span>

<input type="text" class="form-control"/>

<span class="input-group-addon">1234</span>

</div>

1. 尺寸设置
2. 复选框加addon

<span class="input-group-addon">@

<input type="checkbox"/>

</span>

1. 按钮加addon

<span class="input-group-btn">

<button type="button" class="btn btn-info">2222</button>

</span>

1. 下拉菜单按钮作为addon
2. 分段按钮作为addon

### 课18：第18章 Bootstrap导航

1. 选项卡导航 .nav .nav-tabs

注意：导航是没有默认样式的，而button是有btn-default的

1. 胶囊式选项卡导航 .nav .nav-pills
2. 堆叠式导航 .nav .nav-pills .nav-stacked
3. 自适应导航 .nav-justified
4. 分隔符 .nav.divider（用于堆叠垂直式）
5. 禁用
6. 二级导航

<li><a href="#" class="dropdown-toggle" data-toggle="dropdown">首页<span class="caret"></span></a>

<ul class="dropdown-menu">

<li><a href="#">1</a></li>

<li><a href="#">1</a></li>

<li><a href="#">1</a></li>

<li><a href="#">1</a></li>

</ul>

</li>

### 课19：第19章 Boostrap导航条

遇到的细节问题：

1. navbar-wrapper没有
2. 基础导航条

<nav class="navbar navbar-default">

<div class="nav-header"> //头信息

<a href="#" class="navbar-brand">导航</a>

</div>

<ul class="nav navbar-nav">

<li class="active"><a href="#">首页</a></li>

<li><a href="#">首页</a></li>

<li><a href="#">首页</a></li>

<li><a href="#">首页</a></li>

</ul>

</nav>

1. 导航条中的表
2. 导航条中的文本、链接、按钮

<p class="navbar-text">文本</p>

<a href="#" class="navbar-link">链接</a>

<button type="button" class="btn btn-primary navbar-btn">按钮</button>

1. 顶部固定和底部固定

注意：body中默认的padding-top是50px。所以我们在顶部被覆盖的情况下，可以改变顶部padding距离。

<nav class="navbar navbar-default navbar-fixed-top">

<nav class="navbar navbar-default navbar-fixed-bottom">

1. 全屏方角导航 .navbar-static-top
2. 反色导航条 .navbar-inverse
3. 响应式导航条

<nav class="navbar navbar-inverse">

<div class="nav-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-responsive-collapse">

<span class="sr-only">Toogle navagation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a href="#" class="navbar-brand">导航</a>

</div>

<div class="collapse navbar-collapse navbar-responsive-collapse">

<ul class="nav navbar-nav">

<li class="active"><a href="#">首页</a></li>

<li><a href="#">首页</a></li>

<li><a href="#">首页</a></li>

<li><a href="#">首页</a></li>

</ul>

</div>

</nav>

### 课20：第20章 Bootstrap面包屑导航和分页导航

1. 面包屑导航

<ul class="breadcrumb">

<li><a href="#">首页</a></li>

<li><a href="#">新闻</a></li>

<li><a href="#">国内新闻</a></li>

<li><a href="#">高考专题</a></li>

</ul>

1. 分页导航

<ul class="pagination pagination-sm">

<li><a href="#">&laquo;</a></li>

<li><a href="#">1</a></li>

<li><a href="#">2</a></li>

<li><a href="#">3</a></li>

<li><a href="#">4</a></li>

<li><a href="#">5</a></li>

<li><a href="#">&raquo;</a></li>

</ul>

1. 下一页导航

<ul class="pager">

<li><a href="#">上一页</a></li>

<li><a href="#">下一页</a></li>

</ul>

### 课21：第21章 徽章巨幕和页头

1. 徽章

给链接、导航等元素嵌套 <span class="badge"> 元素，可以很醒目的展示新的或未读的信息条目。

<a href="#">Inbox <span class="badge">42</span></a>

<button class="btn btn-primary" type="button"> Messages

<span class="badge">4</span>

</button>

1. 巨幕

<div class="jumbotron">

<h1>Hello, world!</h1>

<p>...</p>

<p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a></p></div>

如果需要让巨幕组件的宽度与浏览器宽度一致并且没有圆角，请把此组件放在所有 .container 元素的外面，并在组件内部添加一个 .container 元素。

<div class="jumbotron">

<div class="container">

...

</div></div>

1. 页头

页头组件能够为 h1 标签增加适当的空间，并且与页面的其他部分形成一定的分隔。它支持 h1 标签内内嵌 small 元素的默认效果，还支持大部分其他组件（需要增加一些额外的样式）。

<div class="page-header"> <h1>Example page header <small>Subtext for header</small></h1> </div>

### 课22：第22章 缩略图和警告框

疑问：alt="pic"？？？

1. 缩略图thumbnail

<div class="col-sm-6 col-md-4">

<div class="thumbnail">

<img src="../jq-day01/images/04.jpg" alt="alt">

<div class="caption">

<h3>Thumbnail label</h3>

<p>...</p>

<p>

<a href="#" class="btn btn-primary" role="button">Button</a>

<a href="#" class="btn btn-default" role="button">Button</a>

</p>

</div>

</div>

</div>

</div>

1. 默认警告框
2. 可关闭的警告框
3. 多彩样式警告框
4. 警告框中的链接

&times;：关闭功能

<div class="container">

<div class="alert alert-info">

<button type="button" class="close" data-dismiss="alert">&times;</button>

<p>这里是提示信息</p>

</div>

<br>

<div class="alert alert-info alert-dismissable">

<button type="button" class="close" data-dismiss="alert">&times;</button>

<p>这里是提示信息<a href="#" class="alert-link">百度</a></p>

</div>

</div>

### 课23：第23章 进度条和媒体对象

1. 进度条基本样式
2. 进度条多彩样式
3. 条纹进度条
4. 进度条动画样式
5. 堆叠样式进度条
6. 媒体对象默认样式
7. 媒体列表

1）

<div class="container">

<div class="progress">

<div class="progress-bar" style="width:25%"></div>

</div>

<h1>堆叠</h1>

<div class="progress">

<div class="progress-bar progress-bar-info" style="width:25%"></div>

<div class="progress-bar progress-bar-success" style="width:25%"></div>

<div class="progress-bar progress-bar-striped" style="width:25%"></div>

<div class="progress-bar progress-bar-warning" style="width:25%"></div>

</div>

</div>

<div class="progress">

<div class="progress-bar progress-bar-warning active progress-bar-striped" style="width:75%"></div>

</div>

2）

<ul class="media-list">

<li class="media">

<div class="media-body">

<h2 class="media-heading">这里是标题</h2>

<p>

这里写内容，这里写内容，这里写内容<br>

这里写内容，这里写内容，这里写内容<br>

这里写内容，这里写内容，这里写内容<br>

</p>

</div>

<div class="media-right media-middle">

<a href="#">

<img class="media-object " src="../jq-day01/images/01.jpg " alt="pic"/>

</a>

</div>

</li>

<li class="media">

<div class="media-body">

<h2 class="media-heading">这里是标题</h2>

<p>

这里写内容，这里写内容，这里写内容<br>

这里写内容，这里写内容，这里写内容<br>

这里写内容，这里写内容，这里写内容<br>

</p>

</div>

<div class="media-right media-middle">

<a href="#">

<img class="media-object " src="../jq-day01/images/01.jpg " alt="pic"/>

</a>

</div>

</li>

</ul>

### 课24：第24章 Js选项卡

1. 选项卡组成与布局

选项卡组件

底部可切换的选项卡面板

1. 声明式用法
2. Javascript
3. 事件

### 课25：第25章 工具提示框

1. 基本用法

data-toggle

data-placement

1. js用法

$(“select”).tooltip();

$(“select”).tooltip(‘show’);

$(“select”).tooltip(‘hide’);

$(“select”).tooltip(‘toggle’);

$(“select”).tooltip({option});

1. 参数

data-animation 应用一个fade动画

data-html true|false

是否支持html作为提示语

data-placement 方向（left、right、top、bottom、auto）

data-selector

data-original-title提示语

data-trigger： click hover focus manual

data-delay延迟时间

(show:500,hid:100)

data-container string|false

将tooltip附加到元素上

data-template 字符串 提示语模板

1. 事件

show.bs.tooltip

shown.bs.tooltip

hid.bs.tooltip

hidden.bs.tooltip

### 课26：第26章 弹出框与警告框

1. 弹出框

基本用法popover

$(‘select.popover();

data-toggle-’popover’

data-content=””

data-original-title=””

1. 警告框

布局与样式

data-dismiss=”alert”

$().alert()

$().alert(‘close’)

### 课27：第27章 折叠效果

1. 声明式用法

data-toggle=”collapse”

data-target=”#demo”

<div id=”demo” class=”collapse”></div>

1. javascript用法

$(‘select’).collaspse();

parent选项，将指定容器中内容都隐藏

toggle默认为true，是否开启反转功能

1. 事件

show.bs.collapse

shown.bs.collaspse

hide.bs.collapse

hidden.bs.collapse

1. $(‘#myCollapsible’).on(‘hidden.bs.collapse’,function()

{});

### 课28：第28章 折叠练习

### 课29：第29章 焦点轮播图

知识细节：

1. 让轮播图carousel居中：在class中加入center-block
2. 加入class=”item”，使其display=none
3. 轮播图 自动轮动时间设置：data-ride=”carousel” 改为手动设置 data-interval=”2000”
4. 基本用法

Data-ride=”carousel”

图片容器carousel-inner

圆圈carousel-indicators

1. 自定义属性

Data-ride=”carousel”

Data-slide=’prev|next’

Data-slide-to=’3’

1Data-interval=’5000’自动循环时间

Data-pause=”hover

Data-wrap=’true’是否持续循环

1. javascript用法

$(‘.carousel’).carousel()

.carousel(‘cycle’)默认从左到右

.carousel(‘pause’)暂停

.carousel(number)循环到number

.carousel(‘prev’)返回到前一张

.carousel(‘next’)转到下一张

1. 事件

Slide.bs.carousel

slide方法后，还未开始下张图

$(‘#myCarousel’).on(‘slide.bs.carousel’)

Function(){}

### 课30：第30章 网站后管理 导航

### 课31：第31章 网站后管理 侧边功能栏

遇见问题：

1. -webkit-\*（safari、chorome）,-moz-\*（firefox）,-ms-\*（ie）
2. <i>：呈现斜体的文本
3. rem：根元素字体大小

之前，用的是em：参考“父元素的字符大小”；存在问题：随着嵌套元素的结构越来越深，字符大小会逐渐改变。

而rem是根据“根元素的字符大小”来确定的。（根元素：html元素）

### 课32：第32章 网站后管理 完成主要区域

## D**ay1:**

知识点认识：

1. .xsd：xml schema definition ：xml的模板样式定义

什么是 XML Schema？

XML Schema 的作用是定义 XML 文档的合法构建模块，类似 DTD。

#### **XML Schema:**

* 定义可出现在文档中的元素
* 定义可出现在文档中的属性
* 定义哪个元素是子元素
* 定义子元素的次序
* 定义子元素的数目
* 定义元素是否为空，或者是否可包含文本
* 定义元素和属性的数据类型
* 定义元素和属性的默认值以及固定值

课1-Spring框架的概述

（一）Spring概念：Spring是分层的JavaSE\EE full-stack（一站式）轻量级开源框架。

以IoC（Inverse of Control 反转控制）和AOP（Aspect Oriented Programming面向切割编程为内核）

Spring的出现，解决了EJB的臃肿，低效，脱离现实等问题。

分层：

SUN提供的EE的三层结构：web层、业务层、数据访问层（持久层，集成层）

Struts2是web层基于MVC（Model View Controller）设计模式框架。

Hibernate是持久的一个ORM的框架。

一站式：

Spring框架有堆三层的每层解决方案：

Web层：Spring MVC。

持久层：JDBC Templete。

业务层：Spring 的Bean管理。

（二）Spring的核心：

IOC反转控制

控制反转：将对象的创建权，交由Spring完成。

AOP 面向切割编程。是面向对象的功能延伸。不是替换面向对象，是用来解决OO（Object Oriented）中一些问题。

1. Spring的版本：

Spring3.x 和Spring4.x

Spring需要整合Hibernate4。

（四）Spring优点：

1 方便解耦，简化开发

Spring就是一个大工厂，可以将所有对象创建和依赖关系维护，交给Spring管理

2 AOP编程的支持

Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能

3 声明式事务的支持

只需要通过配置就可以完成对事务的管理，而无需手动编程。

4方便程序的测试

Spring对Junit4支持，可以通过注解方便的测试Spring程序；

5方便集成各种优秀框架。

Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz等）的直接支持。

6降低JavaEE API的使用难度

Spring 对JavaEE开发中非常难用的一些API（JDBC、JavaMail、远程调用等），都提供了封装，使这些API应用难度大大降低。

课2：Spring的入门程序

1. 下载开发包。

Spring3.2开发包+Spring开发依赖包。

docs :Spring框架api和规范。

libs :Spring开发的jar包。

schema: XML的约束文档。

（二）创建web工程引入Jar包。

首先，导入核心jar包。（beans,context,core,expression）

开发的日志记录的包:（阿帕奇）

com.springsource.org.apache.commons.logging-1.1.1.jar --- 用于整合其他的日志的包(类似Hibernate中slf4j)

com.springsource.org.apache.log4j-1.2.15.jar

出现细节问题：

1 关于代码和文字注释出现乱码，可以在preference中找到spelling，workspaces中的

text file encoding：GBK or utf-8。或者右击某工程-->properties-->text file encoding。

2 @Test 为什么无法加载导入该包？

（三）创建配置文件

1创建 xxxxxxx.xml文件（src）。

2引入XML约束：（xsd-config.html --> 引入beans约束）

1. 配置中配置类

<!-- 通过一个<bean>标签设置类中的信息，通过id属性为类起个标识. -->

<bean id ="userService" class="cn.itcast.spring3.demo1.HelloServiceImpl"></bean>

</beans>

（五）传统方式和创建工厂类

@Test

// 传统方式

public void demo1(){

// 造成程序紧密耦合。

HelloService helloService = new HelloServiceImpl();

helloService.sayHello();

}

public void demo2(){

// 创建一个工厂类。

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

HelloService helloService = (HelloService)applicationContext.getBean("userService");

helloService.sayHello();

}

课3：IOC和DI（DI—Dependency Injection依赖注入）

IOC：将对象的创建权给与Spring管理。

DI：在Spring创建对象的过程中，把对象依赖的属性注入到类中。

（创建Bean对象时，动态的将依赖对象注入到Bean组件）

面向对象中对象之间的关系：

依赖：

继承：is a

聚合：--聚集

--组合

课4：Spring加载配置文件

ApplicationContext：应用上下文，加载Spring配置文件。

加载配置文件：

加载classpath：

new ClassPathXmlApplicationContext("applicationContext.xml");//加载classpath下面配置文件.

加载磁盘路径：（将applicationContext.xml文件放在工程上，然后可以进行该路径加载）

new FileSystemXmlApplicationContext("applicationContext.xml");//加载磁盘下配置文件.

小技巧：ctrl+T 可以看到该类的各个子类。

为源文件加载路径

ApplicationContext+attach源文件

HierarchicalBeanFactory+attach源文件

其中：ApplicationContext extends HierarchicalBeanFactory

HierarchicalBeanFactory extends BeanFactory

也就是说ApplicationContext 继承了BeanFactory

BeanFactory在使用到这个类的时候,getBean()方法的时候才会加载这个bean类.（延迟加载）

ApplicationContext对BeanFactory提供了扩展:

国际化处理

事件传递

Bean自动装配

各种不同应用层的Context实现

早期开发使用BeanFactory。

--BeanFactory beanfactory = new XmlBeanFactory(new ClassPathResource("applicationContext.xml"));

--BeanFactory beanfactory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));

课5：Spring的BeanFactory与ApplicationContext区别

见上。

课6：Myeclipes配置Schema约束

配置XML提示：

Window--->xml catalog--->add 找到schema的位置 ,将复制的路径 copy指定位置,选择schema location.

课7：Spring框架Bean的时候方式

提供了三种方式实例化Bean.

**\* 构造方法实例化:(默认无参数)**

\* 静态工厂实例化:

\* 实例工厂实例化:

遇到细节问题：log4j:WARN Please initialize the log4j system properly.

解决：为该工程创建：log4j.properties

内容：

log4j.rootLogger=WARN, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

三种实例化Bean方式：

<!-- 默认情况下使用的就是无参数的构造方法 -->

<bean id = *"bean1"* class = *"cn.itcast.spring3.demo2.Bean1"*></bean>

<!-- 第二种使用静态工厂实例化 -->

<bean id = *"bean2"* class = *"cn.itcast.spring3.demo2.Bean2Factory"* factory-method = *"getBean2"*></bean>

<!-- 第二种使用实例工厂实例化 -->

<bean id = *"bean3"* factory-bean = *"bean3Factory"* factory-method=*"getBean3"* ></bean>

<bean id = *"bean3Factory"* class=*"cn.itcast.spring3.demo2.Bean3Factory"*></bean>

课8：Spring的id,name和scope顺序

Id和name的区别：

id遵守XML约束的id的约束.id约束保证这个属性的值是唯一的,而 且必须以字母开始，可以使用字母、数字、连字符、下划线、句话、冒号

name没有这些要求

\*\*\*\*\* 如果bean标签上没有配置id,那么name可以作为id.

\*\*\*\*\* 开发中Spring和Struts1整合的时候, /login.

<bean name=”/login” class=””>

现在的开发中都使用id属性即可.

scope属性。

scope属性 :

**singleton:单例的(默认的值)，仅实例化一次。**

**prototype:多例的**

request:web开发中.创建了一个对象,将这个对象存入request范围,request.setAttribute();

session:web开发中.创建了一个对象,将这个对象存入session范围,session.setAttribute();

globalSession:一般用于Porlet应用环境.指的是分布式开发.不是porlet环境,globalSession等同于session;

实际开发中主要使用singleton,prototype

遇见小问题：

1在xml中的快捷键alt+/快速输入绝对路径，必须需要先建立该文件。

2 为.xml注释，快捷键ctrl+shift+/; 为.xml删除注释，快捷键ctrl+shift+\;

3 在.xml中结束一个bean语句，后面需要加/。如：<bean ......> </bean> --><<bean ...... />

课9：Spring类的完整生命周期及后处理Bean

Bean生命周期

配置Bean的初始化和销毁的方法:

配置初始化和销毁的方法:

\* init-method=”setup”

\* destroy-method=”teardown”

执行销毁的时候,必须手动关闭工厂,而且只对scope=”**singleton**”有效.

（ApplicationContext中没有close命令，。而在该类的子类中存在ClassPathXmlApplicationContext关闭命令）

Bean的生命周期的11个步骤:

1.instantiate bean对象实例化

2.populate properties 封装属性

3.如果Bean实现BeanNameAware 执行 setBeanName

4.如果Bean实现BeanFactoryAware 或者 ApplicationContextAware 设置工厂 setBeanFactory 或者上下文对象 setApplicationContext

**5.如果存在类实现 BeanPostProcessor（后处理Bean） ，执行postProcessBeforeInitialization**

6.如果Bean实现InitializingBean 执行 afterPropertiesSet

7.调用<bean init-method="init"> 指定初始化方法 init

**8.如果存在类实现 BeanPostProcessor（处理Bean） ，执行postProcessAfterInitialization**

9.执行业务处理（add，find）

10.如果Bean实现 DisposableBean 执行 destroy

11.调用<bean destroy-method="customerDestroy"> 指定销毁方法 customerDestroy

在CustomerService类的add方法之前进行权限校验?

增强动作：

// 动态代理

if(beanName.equals("customerService")){

Object proxy = Proxy.newProxyInstance(bean.getClass().getClassLoader(), bean.getClass().getInterfaces(), new InvocationHandler() {

// 调用目标方法的时候，调用invoke方法。

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

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

System.out.println(System.currentTimeMillis());

Object result = method.invoke(bean, args);

System.out.println(System.currentTimeMillis());

return result;

}

return method.invoke(bean, args);

}

});

return proxy;

课10：Spring的Bean的属性注入（value普通属性）

注入的三种方式：接口方式注入，构造器注入，setter注入。

构造器注入：

<!-- 构造方法的注入 -->

<!-- <constructor-arg name = "name" value = "宝马"/>

<constructor-arg name = "price" value = "100000"/> -->

<constructor-arg index = "0" value = "奔驰"/>

<constructor-arg index = "1" value = "2000000"/>

setter注入：

注意：

在property属性注入时，使用的是setName这些方法。

课11：Spring的属性注入-对象属性

setter方法属性对象注入：

<property name="car2" ref="car2"/>

<constructor-arg index = *"0"* ref="car2"/>

课12：Spring的Bean的属性注入-名称空间

名称空间P注入属性

Spring2.5版本引入了名称空间p.

p:<属性名>="xxx" 引入常量值

p:<属性名>-ref="xxx" 引用其它Bean对象

引入名称空间:

**xmlns:p="http://www.springframework.org/schema/p"**

<bean id="car2" class="cn.itcast.spring3.demo5.Car2" p:name="宝马" p:price="400000"/>

<bean id="person" class="cn.itcast.spring3.demo5.Person" p:name="童童" p:car2-ref="car2"/>

课13：Spring的Bean的属性注入-spEL注入（Spring Expression Language表达式语言注入）

overview of new features：新特性的概述。

Spring3.0提供注入属性方式:

语法：#{表达式}

<bean id="" value="#{表达式}">

<bean id="car2" class="cn.itcast.spring3.demo5.Car2">

<property name="name" value="#{'大众'}"></property>

<property name="price" value="#{'120000'}"></property>

</bean>

<bean id="person" class="cn.itcast.spring3.demo5.Person">

<!--<property name="name" value="#{personInfo.name}"/>-->

<property name="name" value="#{personInfo.showName()}"/>

<property name="car2" value="#{car2}"/>

</bean>

<bean id="personInfo" class="cn.itcast.spring3.demo5.PersonInfo">

<property name="name" value="张三"/>

</bean>

课14：Spring的Bean的属性注入-集合属性注入

<bean id="collectionBean" class="cn.itcast.spring3.demo6.CollectionBean">

<!-- 注入List集合 -->

<property name="list">

<list>

<value>童童</value>

<value>小凤</value>

</list>

</property>

<!-- 注入set集合 -->

<property name="set">

<set>

<value>杜宏</value>

<value>如花</value>

</set>

</property>

<!-- 注入map集合 -->

<property name="map">

<map>

<entry key="刚刚" value="111"/>

<entry key="娇娇" value="333"/>

</map>

</property>

<property name="properties">

<props>

<prop key="username">root</prop>

<prop key="password">123</prop>

</props>

</property>

</bean>

课15：Spring的配置文件引入的问题

1 加载配置文件：

方法1

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml",”bean2.xml”);

方法2

<import resource="applicationContext2.xml"/>

课16：Spring的Bean的装配-注解的方式

IOC装配Bean注解

小技巧：单独一个value = “xxx”， 其中value单词可以省略

@Component("userService") //单独一个value = “xxx”， 其中value单词可以省略

//@Service(value = "userService")

注意：

注解后不需要加;（//@Service(value = "userService");是错误的）

Spring2.5 引入使用注解去定义Bean

@Component 描述Spring框架中Bean

Spring的框架中提供了与@Component注解等效的三个注解:

@Repository 用于对DAO实现类进行标注

@Service 用于对Service实现类进行标注

@Controller 用于对Controller实现类进行标注

\*\*\*\*\* 三个注解为了后续版本进行增强的.

下图总的注解两步骤：

第一步，可以不写，在同时需要xml和注解时，需要添加。



课17：Spring的Bean的属性注入-注解的方式

普通属性;

@Value(value="itcast")

private String info;

对象属性:

@Autowired:自动装配默认使用类型注入.

通过@Autowired的required属性，设置一定要找到匹配的Bean

@Autowired(required = true)

@Autowired

@Qualifier("userDao") --- 按名称进行注入.

@Autowired

@Qualifier("userDao")

private UserDao userDao;

等价于

@Resource(name="userDao")

private UserDao userDao;

课18：Spring的XML和注解的结合使用

Bean其他的属性的配置。

配置Bean初始化方法和销毁方法：

\* init-method 和 destroy-method.

@PostConstruct 初始化

@PreDestroy 销毁

配置Bean的作用范围:

@Scope

注意：

在.java文件中，Double类型的数据，后面需要加xxxd。

在.xml中不需要。

选择xml还是注解？

XML:

\* bean管理

注解;

\* 注入属性的时候比较方便.

两种方式结合;一般使用XML注册Bean,使用注解进行属性的注入.

<context:annotation-config/>

s

@Autowired

@Qualifier("orderDao")

private OrderDao orderDao;

在context:component-scan中，可以将路径改为 base-package="cn.itcast.spring3"。这就可以扫描多个包。

课19：Spring整合web开发

注意改名：需要refactor和properties中都要改名。

Deployment：发布。

正常整合Servlet和Spring没有问题的

但是每次执行Servlet的时候加载Spring配置,加载Spring环境.

\* 解决办法:在Servlet的init方法中加载Spring配置文件?

\* 当前这个Servlet可以使用,但是其他的Servlet的用不了了!!!

\* 将加载的信息内容放到ServletContext中.ServletContext对象时全局的对象.服务器启动的时候创建的.在创建ServletContext的时候就加载Spring的环境.

\* ServletContextListener:用于监听ServletContext对象的创建和销毁的.

导入;spring-web-3.2.0.RELEASE.jar

在web.xml中配置:

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

修改程序的代码:

WebApplicationContext applicationContext = WebApplicationContextUtils.getWebApplicationContext(getServletContext());

WebApplicationContext applicationContext = (WebApplicationContext) getServletContext().getAttribute(WebApplicationContext.ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE);

课20：Spring与Junit整合

1. 程序中有Junit环境.
2. 导入一个jar包.spring与junit整合jar包.

\* spring-test-3.2.0.RELEASE.jar

1. 测试代码:

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations="classpath:applicationContext.xml")

public class SpringTest {

@Autowired

private UserService userService;

@Test

public void demo1(){

userService.sayHello();

}

}

课21：今天的内容总结

Day2

课1：上次课内容回顾

课2：AOP的概述

AOP Aspect Oriented Programing 面向切面编程

AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码（性能监视、事务管理、安全检查、缓存）

Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码

AspecJ是一个基于Java语言的AOP框架，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入

AOP底层原理：

就是代理机制:

\* 动态代理:(JDK中使用)

\* JDK的动态代理,对实现了接口的类生成代理.

Spring的AOP代理：

JDK动态代理:对实现了接口的类生成代理

CGLib代理机制:对类生成代理

AOP术语：

Joinpoint(连接点):所谓连接点是指那些被拦截到的点。在spring中,这些点指的是方法,因为spring只支持方法类型的连接点.

Pointcut(切入点):所谓切入点是指我们要对哪些Joinpoint进行拦截的定义.

Advice(通知/增强):所谓通知是指拦截到Joinpoint之后所要做的事情就是通知.通知分为前置通知,后置通知,异常通知,最终通知,环绕通知(切面要完成的功能)

Introduction(引介):引介是一种特殊的通知在不修改类代码的前提下, Introduction可以在运行期为类动态地添加一些方法或Field.

Target(目标对象):代理的目标对象

Weaving(织入):是指把增强应用到目标对象来创建新的代理对象的过程.

spring采用动态代理织入，而AspectJ采用编译期织入和类装在期织入

Proxy（代理）:一个类被AOP织入增强后，就产生一个结果代理类

Aspect(切面): 是切入点和通知（引介）的结合

课3：JDK的动态处理

怎样获得代理？

Loader：加载器

public class JDKProxy implements InvocationHandler{

private UserDao userDao;

public JDKProxy(UserDao userDao) {

super();

this.userDao = userDao;

}

public UserDao createProxy() {

UserDao proxy = (UserDao) Proxy.newProxyInstance(userDao.getClass()

.getClassLoader(), userDao.getClass().getInterfaces(), this);

return proxy;

}

// 调用目标对象的任何一个方法 都相当于调用invoke();

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

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

// 记录日志:

System.out.println("日志记录=================");

Object result = method.invoke(userDao, args);

return result;

}

return method.invoke(userDao, args);

}

}

课4：CGLIB的动态代理

小技巧：选中开头后，摁下[shift键](https://www.baidu.com/s?wd=shift%E9%94%AE&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)，然后选中结尾，那么从开头到结尾加上中间部分都会被选中。

6个包加载入web-inf --> lib，而CGLIB本身已经融入Spring3.2，所以可以不用加入6个包。

CGLIB(Code Generation Library)是一个开源项目！是一个强大的，高性能，高质量的Code生成类库，它可以在运行期扩展Java类与实现Java接口。 Hibernate支持它来实现PO(Persistent Object 持久化对象)字节码的动态生成

Hibernate生成持久化类的javassist.

CGLIB生成代理机制:其实生成了一个真实对象的子类.

下载cglib的jar包.

\* 现在做cglib的开发,可以不用直接引入cglib的包.已经在spring的核心中集成cglib.

public class CGLibProxy implements MethodInterceptor{

private ProductDao productDao;

public CGLibProxy(ProductDao productDao) {

super();

this.productDao = productDao;

}

public ProductDao createProxy(){

// 使用CGLIB生成代理:

// 1.创建核心类:

Enhancer enhancer = new Enhancer();

// 2.为其设置父类:

enhancer.setSuperclass(productDao.getClass());

// 3.设置回调:

enhancer.setCallback(this);

// 4.创建代理:

return (ProductDao) enhancer.create();

}

public Object intercept(Object proxy, Method method, Object[] args,

MethodProxy methodProxy) throws Throwable {

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

System.out.println("日志记录==============");

Object obj = methodProxy.invokeSuper(proxy, args);

return obj;

}

return methodProxy.invokeSuper(proxy, args);

}

}

**结论:Spring框架,如果类实现了接口,就使用JDK的动态代理生成代理对象,如果这个类没有实现任何接口,使用CGLIB生成代理对象.**

课5：Spring的AOP-不带有切点的切面

Spring传统AOP。

AOP:不是由Spring定义.AOP联盟的组织定义.

Spring中的通知:(增强代码)

前置通知 org.springframework.aop.MethodBeforeAdvice

\* 在目标方法执行前实施增强

后置通知 org.springframework.aop.AfterReturningAdvice

\* 在目标方法执行后实施增强

环绕通知 org.aopalliance.intercept.MethodInterceptor

\* 在目标方法执行前后实施增强

异常抛出通知 org.springframework.aop.ThrowsAdvice

\* 在方法抛出异常后实施增强

引介通知 org.springframework.aop.IntroductionInterceptor(课程不讲.)

\* 在目标类中添加一些新的方法和属性

Spring 中的切面类型。

Advisor : Spring中传统切面.

\* Advisor:都是有一个切点和一个通知组合.

\* Aspect:多个切点和多个通知组合.

Advisor : 代表一般切面，Advice本身就是一个切面，对目标类所有方法进行拦截(\* 不带有切点的切面.针对所有方法进行拦截)

PointcutAdvisor : 代表具有切点的切面，可以指定拦截目标类哪些方法(带有切点的切面,针对某个方法进行拦截)

IntroductionAdvisor : 代表引介切面，针对引介通知而使用切面（不要求掌握）

Spring的AOP的开发。

针对所有方法的增强：

CustomerDaoImpl实现类。

第一步:导入相应jar包.

\* spring-aop-3.2.0.RELEASE.jar

\* com.springsource.org.aopalliance-1.0.0.jar

第二步:编写被代理对象:

\* CustomerDao接口

\* CustoemrDaoImpl实现类

第三步:编写增强的代码:

public class MyBeforeAdvice implements MethodBeforeAdvice{

/\*\*

\* method:执行的方法

\* args:参数

\* target:目标对象

\*/

public void before(Method method, Object[] args, Object target)

throws Throwable {

System.out.println("前置增强...");

}

}

第四步:生成代理:(配置生成代理:)

\* 生成代理Spring基于ProxyFactoryBean类.底层自动选择使用JDK的动态代理还是CGLIB的代理.

\* 属性:

target : 代理的目标对象

proxyInterfaces : 代理要实现的接口

如果多个接口可以使用以下格式赋值

<list>

<value></value>

....

</list>

proxyTargetClass : 是否对类代理而不是接口，设置为true时，使用CGLib代理

interceptorNames : 需要织入目标的Advice

singleton : 返回代理是否为单实例，默认为单例

optimize : 当设置为true时，强制使用CGLib

<!-- 定义目标对象 -->

<bean id="customerDao" class="cn.itcast.spring3.demo3.CustomerDaoImpl"></bean>

<!-- 定义增强 -->

<bean id="beforeAdvice" class="cn.itcast.spring3.demo3.MyBeforeAdvice"></bean>

<!-- Spring支持配置生成代理: -->

<bean id="customerDaoProxy" class="org.springframework.aop.framework.ProxyFactoryBean">

<!-- 设置目标对象 -->

<property name="target" ref="customerDao"/>

<!-- 设置实现的接口 ,value中写接口的全路径 -->

<property name="proxyInterfaces" value="cn.itcast.spring3.demo3.CustomerDao"/>

<!-- 需要使用value:要的名称 -->

<property name="interceptorNames" value="beforeAdvice"/>

</bean>

**\*\*\*\*\* 注入的时候要注入代理对象:**

@Autowired

// @Qualifier("customerDao")// 注入是真实的对象,必须注入代理对象.

@Qualifier("**customerDaoProxy**")

private CustomerDao customerDao;

对JAVA中的.xml文件的理解：

java项目中,[xml文件](https://www.baidu.com/s?wd=xml%E6%96%87%E4%BB%B6&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)一般都是用来存储一些配置信息。

java项目完成之后,每个模块应该都是独立的,模块之间的关系都可以使用xml来进行维护,spring就是这样的一个框架  
  
 一个好的项目,需要有良好的可拓展性,如果把所有的逻辑关系还有配置信息都写入代码中,会使程序的可拓展性变差,为了解决这个问题,xml就可以对整个项目进行调度(spring)  
  
 还有使用xml作为数据储存,不过用起来很少,多数还是用来存放配置信息。

注意：

1）applicationContext.xml配置信息，一定主要不能单独放在某包内，一定得放在工程中。

2）在测试与Junit4配合使用时，Log4j日志不能打开，需要Off挡。

课6：Spring的AOP-带有切点的切面

带有切点的切面（针对目标对象的某些方法进行增强）

第一步:创建被代理对象.

\* OrderDao

第二步:编写增强的类:

public class MyAroundAdvice implements MethodInterceptor{

public Object invoke(MethodInvocation methodInvocation) throws Throwable {

System.out.println("环绕前增强...");

Object result = methodInvocation.proceed();// 执行目标对象的方法

System.out.println("环绕后增强...");

return result;

}

}

第三步:生成代理:

<!-- 带有切点的切面 -->

<!-- 定义目标对象 -->

<bean id="orderDao" class="cn.itcast.spring3.demo4.OrderDao"></bean>

<!-- 定义增强 -->

<bean id="aroundAdvice" class="cn.itcast.spring3.demo4.MyAroundAdvice"></bean>

<!-- 定义切点切面: -->

<bean id="myPointcutAdvisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">

<!-- 定义表达式,规定哪些方法执行拦截 -->

<!-- . 任意字符 \* 任意个 -->

<!-- <property name="pattern" value=".\*"/> -->

<!-- <property name="pattern" value="cn\.itcast\.spring3\.demo4\.OrderDao\.add.\*"/> -->

<!-- <property name="pattern" value=".\*add.\*"></property> -->

<property name="patterns" value=".\*add.\*,.\*find.\*"></property>

<!-- 应用增强 -->

<property name="advice" ref="aroundAdvice"/>

</bean>

<!-- 定义生成代理对象 -->

<bean id="orderDaoProxy" class="org.springframework.aop.framework.ProxyFactoryBean">

<!-- 配置目标 -->

<property name="target" ref="orderDao"></property>

<!-- 针对类的代理 -->

<property name="proxyTargetClass" value="true"></property>

<!-- 在目标上应用增强 -->

<property name="interceptorNames" value="myPointcutAdvisor"></property>

</bean>

课7：Spring的AOP-自动代理

为SpringTest3 加上断点，会发现是自动加上代理的。

可以通过配置属性<property name=*"optimize"* value=*"true"*></property>

前面的案例中，每个代理都是通过ProxyFactoryBean织入切面代理，在实际开发中，非常多的Bean每个都配置ProxyFactoryBean开发维护量巨大

⭐配置数量太大该怎么办呢？

自动创建代理(\*\*\*\*\*基于后处理Bean.在Bean创建的过程中完成的增强.生成Bean就是代理.)

BeanNameAutoProxyCreator 根据Bean名称创建代理

DefaultAdvisorAutoProxyCreator 根据Advisor本身包含信息创建代理

\* AnnotationAwareAspectJAutoProxyCreator 基于Bean中的AspectJ 注解进行自动代理

#### BeanNameAutoProxyCreator :按名称生成代理

<!-- 定义目标对象 -->

<bean id="customerDao" class="cn.itcast.spring3.demo3.CustomerDaoImpl"></bean>

<bean id="orderDao" class="cn.itcast.spring3.demo4.OrderDao"></bean>

<!-- 定义增强 -->

<bean id="beforeAdvice" class="cn.itcast.spring3.demo3.MyBeforeAdvice"></bean>

<bean id="aroundAdvice" class="cn.itcast.spring3.demo4.MyAroundAdvice"></bean>

<!-- 自动代理:按名称的代理 基于后处理Bean,后处理Bean不需要配置ID-->

**<bean class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">**

**<property name="beanNames" value="\*Dao"/>**

**<property name="interceptorNames" value="beforeAdvice"/>**

**</bean>**

#### DefaultAdvisorAutoProxyCreator :根据切面中定义的信息生成代理

<!-- 定义目标对象 -->

<bean id="customerDao" class="cn.itcast.spring3.demo3.CustomerDaoImpl"></bean>

<bean id="orderDao" class="cn.itcast.spring3.demo4.OrderDao"></bean>

<!-- 定义增强 -->

<bean id="beforeAdvice" class="cn.itcast.spring3.demo3.MyBeforeAdvice"></bean>

<bean id="aroundAdvice" class="cn.itcast.spring3.demo4.MyAroundAdvice"></bean>

**<!-- 定义一个带有切点的切面 -->**

**<bean id="myPointcutAdvisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">**

**<property name="pattern" value=".\*add.\*"/>**

**<property name="advice" ref="aroundAdvice"/>**

**</bean>**

**<!-- 自动生成代理 -->**

**<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"></bean>**

**区分基于ProxyFattoryBean的代理与自动代理区别?**

**\*\*\*\*\* ProxyFactoryBean:先有被代理对象,将被代理对象传入到代理类中生成代理.**

**自动代理基于后处理Bean.在Bean的生成过程中,就产生了代理对象,把代理对象返回.生成Bean已经是代理对象.**

课8：Spring的AOP-基于Aspect的注解

AspectJ是一个面向切面的框架，它扩展了Java语言。AspectJ定义了AOP语法所以它有一个专门的[编译器](http://baike.baidu.com/view/487018.htm" \t "http://baike.baidu.com/_blank)用来生成遵守Java字节编码规范的Class文件。

AspectJ是一个基于Java语言的AOP框架

Spring2.0以后新增了对AspectJ切点表达式支持

@AspectJ 是AspectJ1.5新增功能，通过JDK5注解技术，允许直接在Bean类中定义切面

新版本Spring框架，建议使用AspectJ方式来开发AOP

AspectJ表达式:

\* 语法:execution(表达式)

execution(<访问修饰符>?<返回类型><方法名>(<参数>)<异常>)

\* execution(“\* cn.itcast.spring3.demo1.dao.\*(..)”) ---只检索当前包

\* execution(“\* cn.itcast.spring3.demo1.dao..\*(..)”) ---检索包及当前包的子包.

\* execution(\* cn.itcast.dao.GenericDAO+.\*(..)) ---检索GenericDAO及子类

AspectJ增强:

@Before 前置通知，相当于BeforeAdvice

@AfterReturning 后置通知，相当于AfterReturningAdvice

@Around 环绕通知，相当于MethodInterceptor

@AfterThrowing抛出通知，相当于ThrowAdvice

@After 最终final通知，不管是否异常，该通知都会执行

@DeclareParents 引介通知，相当于IntroductionInterceptor (不要求掌握)

基于注解：

第一步:引入相应jar包.

\* aspectj依赖aop环境.

\* spring-aspects-3.2.0.RELEASE.jar

\* com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

第二步:编写被增强的类:

\* UserDao

第三步:使用AspectJ注解形式:

@Aspect

public class MyAspect {

@Before("execution(\* cn.itcast.spring3.demo1.UserDao.add(..))")

public void before(){

System.out.println("前置增强....");

}

}

第四步:创建applicationContext.xml

\* 引入aop的约束:

\* <aop:aspectj-autoproxy /> --- 自动生成代理:

\* 底层就是AnnotationAwareAspectJAutoProxyCreator

<aop:aspectj-autoproxy />

<bean id="userDao" class="cn.itcast.spring3.demo1.UserDao"></bean>

<bean id="myAspect" class="cn.itcast.spring3.demo1.MyAspect"></bean>

课9：Spring的AOP-基于Aspect的通知类型

@Before 前置通知，相当于BeforeAdvice

\* 就在方法之前执行.没有办法阻止目标方法执行的.

@AfterReturning 后置通知，相当于AfterReturningAdvice

\* 后置通知,获得方法返回值.

@Around 环绕通知，相当于MethodInterceptor

\* 在可以方法之前和之后来执行的,而且可以阻止目标方法的执行.

@AfterThrowing抛出通知，相当于ThrowAdvice

@After 最终final通知，不管是否异常，该通知都会执行

@DeclareParents 引介通知，相当于IntroductionInterceptor (不要求掌握)

课10：Spring的AOP-基于Aspect的切点定义

@Pointcut("execution(\* cn.itcast.spring3.demo1.UserDao.find(..))")

private void myPointcut(){}

面试:

\* Advisor和Aspect的区别?

\* Advisor:Spring传统意义上的切面:支持一个切点和一个通知的组合.

\* Aspect:可以支持多个切点和多个通知的组合.

课11：Spring的AOP-基于Aspect的XML配置

第一步:编写被增强的类:

\* ProductDao

第二步:定义切面

第三步:配置applicationContext.xmll

前置通知:

\* 代码:

public void before(){

System.out.println("前置通知...");

}

\* 配置:

<aop:config>

<!-- 定义切点: -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo2.ProductDao.add(..))" id="mypointcut"/>

<aop:aspect ref="myAspectXML">

<!-- 前置通知 -->

<aop:before method="before" pointcut-ref="mypointcut"/>

</aop:aspect>

</aop:config>

后置通知:

\* 代码:

public void afterReturing(Object returnVal){

System.out.println("后置通知...返回值:"+returnVal);

}

\* 配置:

<aop:config>

<!-- 定义切点: -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo2.ProductDao.add(..))" id="mypointcut"/>

<aop:aspect ref="myAspectXML">

<!-- 后置通知 -->

<aop:after-returning method="afterReturing" pointcut-ref="mypointcut" returning="returnVal"/>

</aop:aspect>

</aop:config>

环绕通知:

\* 代码:

public Object around(ProceedingJoinPoint proceedingJoinPoint) throws Throwable{

System.out.println("环绕前增强....");

Object result = proceedingJoinPoint.proceed();

System.out.println("环绕后增强....");

return result;

}

\* 配置:

<aop:config>

<!-- 定义切点: -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo2.ProductDao.add(..))" id="mypointcut"/>

<aop:aspect ref="myAspectXML">

<!-- 前置通知 -->

<!-- <aop:before method="before" pointcut-ref="mypointcut"/> -->

<!-- 后置通知 -->

<!-- <aop:after-returning method="afterReturing" pointcut-ref="mypointcut" returning="returnVal"/> -->

<!-- 环绕通知 -->

**<aop:around method="around" pointcut-ref="mypointcut"/>**

</aop:aspect>

</aop:config>

异常通知:

\* 代码;

public void afterThrowing(Throwable e){

System.out.println("异常通知..."+e.getMessage());

}

\* 配置;

<aop:config>

<!-- 定义切点: -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo2.ProductDao.add(..))" id="mypointcut"/>

<aop:aspect ref="myAspectXML">

<!-- 异常通知 -->

<aop:after-throwing method="afterThrowing" pointcut-ref="mypointcut" throwing="e"/>

</aop:aspect>

</aop:config>

最终通知:

\* 代码:

public void after(){

System.out.println("最终通知....");

}

\* 配置:

<aop:config>

<!-- 定义切点: -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo2.ProductDao.add(..))" id="mypointcut"/>

<aop:aspect ref="myAspectXML">

<!-- 最终通知 -->

<aop:after method="after" pointcut-ref="mypointcut"/>

</aop:aspect>

</aop:config>

课12：Spring的JDBC模板：快速入门

JdbcTemplate模板与DbUtils工具类比较类似.

Spring对持久层技术支持。

Mysql遇到错误：

（一）ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'user spring3\_day02' at line 1

原因：输入细节错误。

1：首先，打开mysql的运行状态，（电脑的管理-->服务中找到mysql，使状态开启。）

2：运行mysql -u -root -p

3：create database spring3\_day02;

4：use spring3\_day02;

可是，我前后输入的命令是完全一样的，为什么第一次失败。后面的成功了！

Mysql的密码修改：

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

（）修改密码）；

刷新权限 flush privileges;

修改root用户的密码 update user set password=PASSWORD("123456") where user='root';

org.springframework.jdbc.CannotGetJdbcConnectionException: Could not get JDBC Connection; nested exception is java.sql.SQLException: Access denied for user 'root'@'localhost' (using password: YES)

Caused by: java.sql.SQLException: Access denied for user 'root'@'localhost' (using password: YES)

？？？？？？？？？？？？

（二）有database，但是没有table？

删除数据库：drop database if exists spring3\_day02;

第一步:引入相应jar包:

\* spring-tx-3.2.0.RELEASE.jar

\* spring-jdbc-3.2.0.RELEASE.jar

\* mysql驱动.

第二步:创建applicationContext.xml

第三步:编写一个测试类:

课13：Spring的JDBC模板：默认连接池的配置

#### Spring默认的连接池:

<!-- 配置Spring默认的连接池 -->

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

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

<property name="url" value="jdbc:mysql:///spring3\_day02"/>

<property name="username" value="root"/>

<property name="password" value="123"/>

</bean>

课14：Spring的JDBC模板：DBCP连接池的配置

#### DBCP连接池:

导入jar包:

\* com.springsource.org.apache.commons.dbcp-1.2.2.osgi.jar

\* com.springsource.org.apache .commons.pool-1.5.3.jar

<!-- 配置DBCP连接池 -->

<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

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

<property name="url" value="jdbc:mysql:///spring3\_day02"/>

<property name="username" value="root"/>

<property name="password" value="123"/>

</bean>

课15：Spring的JDBC模板：C3P0连接池配置

#### C3P0连接池:

导入jar包:

\* com.springsource.com.mchange.v2.c3p0-0.9.1.2.jar

<!-- 配置c3p0连接池 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

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

<property name="jdbcUrl" value="jdbc:mysql:///spring3\_day02"/>

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

<property name="password" value="123"/>

</bean>

课16：Spring的JDBC模板：设置参数到属性文件

在src下创建jdbc.properties

jdbc.driver = com.mysql.jdbc.Driver

jdbc.url = jdbc:mysql:///spring3\_day02

jdbc.user = root

jdbc.password = 123

需要在applicationContext.xml 中使用属性文件配置的内容.

\* 第一种写法:

<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="location" value="classpath:jdbc.properties"></property>

</bean>

\* 第二种写法:

<context:property-placeholder location="classpath:jdbc.properties"/>

课17：Spring的JDBC模板：完成增删改的操作

Spring框架中提供了对持久层技术支持的类:

JDBC : org.springframework.jdbc.core.support.JdbcDaoSupport

Hibernate 3.0 : org.springframework.orm.hibernate3.support.HibernateDaoSupport

iBatis : org.springframework.orm.ibatis.support.SqlMapClientDaoSupport

编写DAO的时候:

Public class UserDao extends **JdbcDaoSupport**{

}

进行CRUD的操作;

\* 保存:update(String sql,Object... args)

\* 修改:update(String sql,Object... args)

\* 删除:update(String sql,Object... args)

课18：Spring的JDBC模板：查询的操作

查询:

\* 简单查询:

\* select count(\*) from user; --- queryForInt(String sql);

\* select name from user where id = ?; --- queryForObject(String sql,Class clazz,Object... args);

\* 复杂查询:(返回对象,和对象集合)

\* select \* from user where id = ? --- queryForObjectString sql,RowMapper<T> rowMapper,Object... args);

\* select \* from user; --- query(String sql,RowMapper<T> rowMapper,Object... args);

课19：今天的内容总结

Spring AOP

\* AOP:面向切面编程.不是替代面向对象,对面向对象扩展,延伸.

\* AOP的思想:

\* 代理机制.(横向抽取).

\* Spring使用AOP的时候,根据类的情况,自动选择代理机制:

\* JDK :针对实现了接口的类生成代理.

\* CGLIB :针对没有实现接口的类生成代理.生成某个类的子类.

\* Spring的AOP的开发中:

\* 术语 :

\* JoinPoint:连接点.可以被增强的方法.

\* PointCut:切入点.真正被增强的方法.

\* Advice:通知.增强的代码.

\* Target:目标对象.被增强的类.

\* Weaving:

\* Aspect:切面.在切入点应用增强.

\* 传统的Spring AOP.

\* 不带切点的切面:

\* 带有切点的切面:

\* 生成代理:基于ProxyFactoryBean.

**\* 缺点;每个类都需要配置ProxyFactroyBean**.

\* 自动代理:(基于后处理Bean)在类的生命周期，类的创建过程中，已经创建了代理。

\* Bean名称自动代理:

\* 切面信息自动代理:

**\* Spring中AspectJ的支持.**

\* 为了简便开发引入AspectJ的支持.

\* 注解:

\* @Aspect:

\* @Before

\* @Around

\* @AfterReturing

\* @AfterThrowing

\* @After

\* PointCut

\* XML:

\* 引入aop约束:

<aop-config>

<aop:pointcut id=”” expression=””/>

<aop:aspect ref=””>

<aop:before />

</aop:aspect>

</aop-config>

Spring JDBCTemplate:

\* 配置连接池:

\* 默认的:

\* DBCP:

\* C3P0:(\*\*\*\*\*)

\* 提取了properties

\* JdbcTemplate的CRUD的操作.

Day03:

课1：内容回顾

课2：事务（transaction）的回顾

事务:是逻辑上一组操作，要么全都成功，要么全都失败.

事务特性:

ACID:

原子性:事务不可分割

一致性:事务执行的前后，数据完整性保持一致.

隔离性:一个事务执行的时候，不应该受到其他事务的打扰

持久性:一旦结束，数据就永久的保存到数据库.

如果不考虑隔离性:

脏读:一个事务读到另一个事务未提交数据

不可重复读:一个事务读到另一个事务已经提交数据（update）导致一个事务多次查询结果不一致

虚读:一个事务读到另一个事务已经提交数据（insert）导致一个事务多次查询结果不一致

事务的隔离级别:

未提交读:以上情况都有可能发生。

已提交读:避免脏读，但不可重复读，虚读是有可能发生。

可重复读:避免脏读，不可重复读，但是虚读有可能发生。

串行的:避免以上所有情况.

课3：Spring的事务开发常用API的详解

目前开发都是分层开发：事务处在Service层。

PlatformTransactionManager:平台事务管理器.

|  |  |
| --- | --- |
| [commit](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/PlatformTransactionManager.html" \l "commit(org.springframework.transaction.TransactionStatus))([TransactionStatus](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionStatus.html" \o "interface in org.springframework.transaction) status)   |  | | --- | | [getTransaction](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/PlatformTransactionManager.html" \l "getTransaction(org.springframework.transaction.TransactionDefinition))([TransactionDefinition](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \o "interface in org.springframework.transaction) definition) | |
| [rollback](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/PlatformTransactionManager.html" \l "rollback(org.springframework.transaction.TransactionStatus))([TransactionStatus](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionStatus.html" \o "interface in org.springframework.transaction) status) |

TransactionDefinition:事务定义

ISOLation\_XXX:事务隔离级别.

PROPAGATION\_XXX:事务的传播行为.(不是JDBC中有的，为了解决实际开发问题.)

过期时间:

TransactionStatus:事务状态

是否有保存点

是否一个新的事务

事务是否已经提交

关系:PlatformTransactionManager通过TransactionDefinition设置事务相关信息管理事务，管理事务过程中，产生一些事务状态:状态由TransactionStatus记录.

API详解:

PlatformTransactionManager:接口.

Spring为不同的持久化框架提供了不同PlatformTransactionManager接口实现

**org.springframework.jdbc.datasource.DataSourceTransactionManager : 使用Spring JDBC或iBatis 进行持久化数据时使用**

**org.springframework.orm.hibernate3.HibernateTransactionManager : 使用Hibernate3.0版本进行持久化数据时使用**

org.springframework.orm.jpa.JpaTransactionManager 使用JPA进行持久化时使用

org.springframework.jdo.JdoTransactionManager 当持久化机制是Jdo时使用

org.springframework.transaction.jta.JtaTransactionManager 使用一个JTA实现来管理事务，在一个事务跨越多个资源时必须使用。

TransactionDefinition:

\* [ISOLATION\_DEFAULT](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \l "ISOLATION_DEFAULT):默认级别. Mysql repeatable\_read oracle read\_commited

|  |
| --- |
| [ISOLATION\_READ\_UNCOMMITTED](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \l "ISOLATION_READ_UNCOMMITTED) |
| [ISOLATION\_READ\_COMMITTED](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \l "ISOLATION_READ_COMMITTED) |
| [ISOLATION\_REPEATABLE\_READ](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \l "ISOLATION_REPEATABLE_READ) |
| [ISOLATION\_SERIALIZABLE](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \l "ISOLATION_SERIALIZABLE) |

\* 事务的传播行为:(不是JDBC事务管理，用来解决实际开发的问题.)传播行为：解决业务层之间的调用的事务的关系.

PROPAGATION\_REQUIRED :支持当前事务，如果不存在 就新建一个

\* A,B 如果A有事务，B使用A的事务，如果A没有事务，B就开启一个新的事务.(A,B是在一个事务中。)

PROPAGATION\_SUPPORTS :支持当前事务，如果不存在，就不使用事务

\* A,B 如果A有事务，B使用A的事务，如果A没有事务，B就不使用事务.

PROPAGATION\_MANDATORY :支持当前事务，如果不存在，抛出异常

\* A,B 如果A有事务，B使用A的事务，如果A没有事务，抛出异常.

PROPAGATION\_REQUIRES\_NEW 如果有事务存在，挂起当前事务，创建一个新的事务

\* A,B 如果A有事务，B将A的事务挂起，重新创建一个新的事务.(A,B不在一个事务中.事务互不影响.)

PROPAGATION\_NOT\_SUPPORTED 以非事务方式运行，如果有事务存在，挂起当前事务

\* A,B 非事务的方式运行，A有事务，就会挂起当前的事务.

PROPAGATION\_NEVER 以非事务方式运行，如果有事务存在，抛出异常

PROPAGATION\_NESTED 如果当前事务存在，则嵌套事务执行

\* 基于SavePoint技术.

\* A,B A有事务，A执行之后，将A事务执行之后的内容保存到SavePoint.B事务有异常的话，用户需要自己设置事务提交还是回滚.

\* 常用:(重点)

PROPAGATION\_REQUIRED

PROPAGATION\_REQUIRES\_NEW

PROPAGATION\_NESTED

课4：Spring的事务-环境搭建

Spring的事务管理分成两类:

\* 编程式事务管理:

\* 手动编写代码完成事务管理.

\* 声明式事务管理:

\* 不需要手动编写代码,配置.

事物环境操作搭建：

CREATE TABLE `account` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(20) NOT NULL,

`money` double DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

INSERT INTO `account` VALUES ('1', 'aaa', '1000');

INSERT INTO `account` VALUES ('2', 'bbb', '1000');

INSERT INTO `account` VALUES ('3', 'ccc', '1000');

创建一个web项目:

\* 导入相应jar包

\* 引入配置文件:

\* applicationContext.xml、log4j.properties、jdbc.properties

创建类:

\* AccountService

\* AccountDao

在Spring中注册:

<!-- 业务层类 -->

<bean id="accountService" class="cn.itcast.spring3.demo1.AccountServiceImpl">

<!-- 在业务层注入Dao -->

<property name="accountDao" ref="accountDao"/>

</bean>

<!-- 持久层类 -->

<bean id="accountDao" class="cn.itcast.spring3.demo1.AccountDaoImpl">

<!-- 注入连接池的对象,通过连接池对象创建模板. -->

**<property name="dataSource" ref="dataSource"/>**

</bean>

其中，需要添加源文件：为JdbcDaoSupport 添加。

就可以不需要设置JdbcTemplate。

编写一个测试类:

课5：Spring的事务-手动编码方式

#### 手动编码的方式完成事务管理:

需要事务管理器:**真正管理事务对象.**

**\* Spring提供了事务管理的模板(工具类.)**

**第一步:注册事务管理器:**

<!-- 配置事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!-- 需要注入连接池,通过连接池获得连接 -->

<property name="dataSource" ref="dataSource"/>

</bean>

第二步:注册事务模板类:

<!-- 事务管理的模板 -->

<bean id="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate">

<property name="transactionManager" ref="transactionManager"/>

</bean>

第三步:在业务层注入模板类:(模板类管理事务)

<!-- 业务层类 -->

<bean id="accountService" class="cn.itcast.spring3.demo1.AccountServiceImpl">

<!-- 在业务层注入Dao -->

<property name="accountDao" ref="accountDao"/>

<!-- 在业务层注入事务的管理模板 -->

**<property name="transactionTemplate" ref="transactionTemplate"/>**

</bean>

第四步:在业务层代码上使用模板:

public void transfer(final String from, final String to, final Double money) {

**transactionTemplate.execute(new TransactionCallbackWithoutResult() {**

**@Override**

**protected void doInTransactionWithoutResult(TransactionStatus status) {**

**accountDao.out(from, money);**

**int d = 1 / 0;**

**accountDao.in(to, money);**

**}**

**});**

}

问题：为什么protected的方法，内部传入的数据，需要用final修饰。

**public** **void** transfer(**final** String from, **final** String to, **final** Double money) {

transactionTemplate.execute(**new** TransactionCallbackWithoutResult() {

@Override

**protected** **void** doInTransactionWithoutResult(TransactionStatus status) {

accountDao.out(from, money);

// int i = 1/0;

accountDao.in(to, money);

}

})

课6：Spring的事务-声明式事务（原始）

手动编码方式缺点:

\* 代码量增加,代码有侵入性.

基于TransactionProxyFactoryBean.

导入:aop相应jar包.

第一步:注册平台事务管理器:

<!-- 事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!-- 注入连接池 -->

<property name="dataSource" ref="dataSource"/>

</bean>

第二步:创建业务层代理对象:

<!-- 配置生成代理对象 -->

<bean id="accountServiceProxy" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<!-- 目标对象 -->

<property name="target" ref="accountService"/>

**<!-- 注入事务管理器 -->**

**<property name="transactionManager" ref="transactionManager"/>**

**<!-- 事务的属性设置 -->**

**<property name="transactionAttributes">**

**<props>**

**<prop key="transfer">PROPAGATION\_REQUIRED</prop>**

**</props>**

**</property>**

</bean>

第三步:编写测试类:

**\*\*\*\*\* 千万注意:注入代理对象**

@Autowired

@Qualifier("**accountServiceProxy**")

private AccountService accountService;

**prop格式：PROPAGATION,ISOLATION,readOnly,-Exception,+Exception**

**\* 顺序:传播行为、隔离级别、事务是否只读、发生哪些异常可以回滚事务（所有的异常都回滚）、发生了哪些异常不回滚.**

**\*\*\*\*\* 缺点:就是需要为每一个管理事务的类生成代理.需要为每个类都需要进行配置.**

课7：Spring的事务-声明式事务（原始）属性设置

课8：Spring的事务-声明式事务（基于切面自动代理）

**第一步:导入相应jar包.**

**\* aspectj**

**第二步:引入相应约束:**

**\* aop、tx约束.**

**<beans xmlns="http://www.springframework.org/schema/beans"**

**xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xmlns:context="http://www.springframework.org/schema/context"**

**xmlns:aop="http://www.springframework.org/schema/aop"**

**xmlns:tx="http://www.springframework.org/schema/tx"**

**xsi:schemaLocation="http://www.springframework.org/schema/beans**

**http://www.springframework.org/schema/beans/spring-beans.xsd**

**http://www.springframework.org/schema/context**

**http://www.springframework.org/schema/context/spring-context.xsd**

**http://www.springframework.org/schema/aop**

**http://www.springframework.org/schema/aop/spring-aop.xsd**

**http://www.springframework.org/schema/tx**

**[http://www.springframework.org/schema/tx/spring-tx.xsd">](http://www.springframework.org/schema/tx/spring-tx.xsd\">)**

**第三步:注册事务管理器;**

<!-- 事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

第四步:定义增强(事务管理)

<!-- 定义一个增强 -->

<tx:advice id="**txAdvice**" transaction-manager="transactionManager">

<!-- 增强(事务)的属性的配置 -->

<tx:attributes>

<!--

isolation:DEFAULT :事务的隔离级别.

propagation :事务的传播行为.

read-only :false.不是只读

timeout :-1

no-rollback-for :发生哪些异常不回滚

rollback-for :发生哪些异常回滚事务

-->

<tx:method name="transfer"/>

</tx:attributes>

</tx:advice>

**第五步:定义aop的配置(切点和通知的组合)**

<!-- aop配置定义切面和切点的信息 -->

<aop:config>

<!-- 定义切点:哪些类的哪些方法应用增强 -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo3.AccountService+.\*(..))" id="mypointcut"/>

<!-- 定义切面: -->

<aop:advisor advice-ref="**txAdvice**" pointcut-ref="mypointcut"/>

</aop:config>

第六步:编写测试类:

\* 注入Service对象,不需要注入代理对象(生成这个类的时候,已经是代理对象.)

课9：Spring的事务-声明式事务（基于注解）

第一步:事务管理器:

<!-- 事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

第二步:注解事务:

<!-- 开启注解的事务管理 -->

<tx:annotation-driven transaction-manager="transactionManager"/>

第三步:在Service上使用注解

@Transactional

\* 注解中有属性值:

\* isolation

\* propagation

\* readOnly

...

课10：三大框架整合：jar包的整理（Struts+Spring+Hibernate）

Struts2导入jar包:

\* struts2/apps/struts2-blank.war/WEB-INF/lib/\*.jar

\* 导入与spring整合的jar

\* struts2/lib/struts2-spring-plugin-2.3.15.3.jar --- 整合Spring框架

\* struts2/lib/struts2-json-plugin-2.3.15.3.jar --- 整合AJAX

\* struts2/lib/struts2-convention-plugin-2.3.15.3.jar --- 使用Struts2注解开发.

\* 配置

web.xml

<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>

struts.xml

<struts>

<constant name="struts.devMode" value="true" />

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

</package>

</struts>

Spring导入jar包:

Spring3.2 开发最基本jar包

spring-beans-3.2.0.RELEASE.jar

spring-context-3.2.0.RELEASE.jar

spring-core-3.2.0.RELEASE.jar

spring-expression-3.2.0.RELEASE.jar

com.springsource.org.apache.commons.logging-1.1.1.jar

com.springsource.org.apache.log4j-1.2.15.jar

AOP开发

spring-aop-3.2.0.RELEASE.jar

spring-aspects-3.2.0.RELEASE.jar

com.springsource.org.aopalliance-1.0.0.jar

com.springsource.org.aspectj.weaver-1.6.8.RELEASE.jar

Spring Jdbc开发

spring-jdbc-3.2.0.RELEASE.jar

spring-tx-3.2.0.RELEASE.jar

Spring事务管理

spring-tx-3.2.0.RELEASE.jar

Spring整合其他ORM框架

spring-orm-3.2.0.RELEASE.jar

Spring在web中使用

spring-web-3.2.0.RELEASE.jar

Spring整合Junit测试

spring-test-3.2.0.RELEASE.jar

(Spring没有引入c3p0和数据库驱动)

课11：三大框架整合：配置文件的导入.

小知识：

1，Open type快捷键：ctrl+shift+T。

2，复制类的全路径：在右下角的outLine窗口中找到对应的java文件右键copyqulifiedName。

\* 配置:

applicationContext.xml

Log4j.properties

在web.xml中配置监听器;

<!-- 配置Spring的监听器 -->

<listener>

<!-- 监听器默认加载的是WEB-INF/applicationContext.xml -->

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 指定Spring框架的配置文件所在的位置 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

Hibernate的jar包导入:

\* 核心包:hibernate3.jar

\* lib/required/\*.jar

\* lib/jpa/\*.jar

\* 引入hibernate整合日志系统的jar包:

\* 数据连接池:

\* 数据库驱动:

\* 二级缓存:(可选的.)

\* backport-util-concurrent.jar

\* commons-logging.jar

\* ehcache-1.5.0.jar

\* Hibernate的配置:

\* hibernate.cfg.xml

\* 映射:

\* 格式:类名.hbm.xml

课12：三大框架整合：提交请求到Action

1.新建包结构:

\* cn.itcast.action

\* cn.itcast.service

\* cn.itcast.dao

\* cn.itcast.vo

2.创建实体类:

\* Book

3.新建一个jsp页面:（Java Server Pages）

\* addBook.jsp

<s:form action="book\_add" namespace="/" method="post" theme="simple">

图书名称:<s:textfield name="name"/><br/>

图书价格:<s:textfield name="price"/><br/>

<s:submit value="添加图书"/>

</s:form>

1. 编写Action:

public class BookAction extends ActionSupport implements ModelDriven<Book>{

// 模型驱动类

private Book book = new Book();

public Book getModel() {

return book;

}

// 处理请求的方法:

public String add(){

System.out.println("web层的添加执行了...");

return NONE;

}

}

5.配置struts.xml

<action name="book\_\*" class="cn.itcast.action.BookAction" method="{1}">

</action>

课13：三大框架整合：Action的创建两种方式

#### Struts2自己管理Action:(方式一)

<action name="book\_\*" class="cn.itcast.action.BookAction" method="{1}">

\* Struts2框架自动创建Action的类.

#### Action交给Spring管理:(方式二)

可以在<action>标签上通过一个伪类名方式进行配置:

<action name="book\_\*" class="**bookAction**" method="{1}"></action>

在spring的配置文件中:

<!-- 配置Action -->

<bean id="**bookAction**" class="cn.itcast.action.BookAction"></bean>

(\*\*\*\*\*)注意:Action交给Spring管理一定要配置scope=”**prototype**”

推荐使用二:

\* 在Spring中管理的类,可以对其进行AOP开发.统一的管理.

课14：三大框架整合：自动注入Service

出现问题：

没有找到struts-plugin.xml。？？

课15：三大框架整合：Spring和Hibernate的整合方式一

课16：三大框架整合：Spring和Hibernate的整合方式二

课17：HibernateTemplate模板API

课18：OpenSessionInView的使用

课19：三大框架整合：注解方式

课20：今天内容的总结

Day04:

遇见小问题：

1 打开myeclipes，发现出现以下问题：Configure java build path 。重新配置JRE System library-->sun jdk1.6。

2 工程上带红色叹号：为其改变JDK的配置，类似上述情况。导入一个web project到MyEclipese中，在Package Explorer中工程前面有一个红色感叹号，查看子目录下面没有任何符号。原因出在引用包上。在工程名上面右键，选择Properties->[Java](https://www.baidu.com/s?wd=Java&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank) Build Path，把前面带红色感叹号的包改为sun jdk1.6。

小技巧：

1 为在工程栏中只显示工程，可以选择向下三角箭头，然后选择working set。

2 创建working set，可以右键-->new -->java-->java working set。

课1：1原生态jdbc编程中的问题总结

关于Java上的映射理解：

我们在计算机科学和数学里经常用到映射概念,这里打个比方就容易理解了

不废话 直接作出最直白的解释 :

生活里存在大量的对应 :

比如一个人的身份证号对应一个人 一张电影票对应一个人 一个丈夫配一个妻子

或者一个丈夫配多个妻子？？！！（对，就是你脑补的那些画面）

说白了 映射就是一种特殊的对应关系

生活里 一箭射一只鸟 多箭射一只鸟 一箭射多只鸟

映射里 每一箭必须射到鸟 还不能一箭射多只鸟

对的 就跟你这个单身狗一样 家里催着你必须结婚 还只能结一个

在计算机科学里就相应的这么理解:

映射就是将两个对象对应起来

对应的对象叫象

被对应的对象叫原象

以java中的map举例

Map names = new HashMap();

names.put("9527", "唐伯虎");

这里就将9527和唐伯虎对应起来，建立了两者的映射(对应)

9527就是唐伯虎。

Java中有非常好的例子。实现了Map接口的HashMap和TreeMap。前面已经提到过Map类是实现键值对的双向链表，这里就是完成了key和value的映射，当然key要是唯一的才可以。

1.1创建数据库：

sql\_data.sql；sql\_table.sql

先导入sql\_table.sql，再导入sql\_data.sql(记录系统的初始化数据)

通常需要提供初始化数据的数据库脚本。

1.2 Jdbc编程问题：

企业开发中，根据项目大小、特点进行技术选型 ，jdbc操作数据库时效率是很高的，jdbc也是技术选型的参考。

1.3

参考教案

需要数据库的驱动包：

mysql-connector-java-5.1.7-bin.jar

ojdbc14-10.2.0.1.0.jar

上边是mysql的驱动，下边是oracle的驱动。

发现了 com.mysql.jdbc.Driver在哪了。com.mysql.jdbc.Driver.class 在Referenced Library中。

PreparedStatement:

先来说说，什么是java中的Statement：Statement是java执行数据库操作的一个重要方法，用于在已经建立数据库连接的基础上，向数据库发送要执行的SQL语句。具体步骤：

　　1.首先导入java.sql.\*；这个包。

　　2.然后加载驱动，创建连接，得到Connection接口的的实现对象，比如对象名叫做conn。

　　3.然后再用conn对象去创建Statement的实例，方法是：Statement stmt = conn.creatStatement("SQL语句字符串");

　　Statement 对象用于将 SQL 语句发送到数据库中。实际上有三种 Statement 对象，它们都作为在给定连接上执行 SQL语句的包容器：Statement、PreparedStatement（它从 Statement 继承而来）和CallableStatement（它从 PreparedStatement 继承而来）。它们都专用于发送特定类型的 SQL 语句：Statement 对象用于执行不带参数的简单 SQL 语句；PreparedStatement 对象用于执行带或不带参数的预编译 SQL 语句；CallableStatement 对象用于执行对数据库已存储过程的调用。

　　综上所述，总结如下：Statement每次执行sql语句，数据库都要执行sql语句的编译，最好用于仅执行一次查询并返回结果的情形，效率高于PreparedStatement.但存在sql注入风险。PreparedStatement是预编译执行的。在执行可变参数的一条SQL时，PreparedStatement要比Statement的效率高，因为DBMS预编译一条SQL当然会比多次编译一条SQL的效率高。安全性更好，有效防止SQL注入的问题。对于多次重复执行的语句，使用Prepared

Statement效率会更高一点。执行SQL语句是可以带参数的，并支持批量执行SQL。由于采用了Cache机制，则预编译的语句，就会放在Cache中，下次执行相同的SQL语句时，则可以直接从Cache中取出来。

PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET name= ? WHERE ID = ?");

pstmt.setString(1, "李四");

pstmt.setInt(2, 1);

pstmt. executeUpdate();

那么CallableStatement扩展了PreparedStatement的接口，用来调用存储过程，它提供了对于输入和输出参数的支持，CallableStatement 接口还有对 PreparedStatement 接口提供的输入参数的sql查询的支持。

PreparedStatement: 数据库会对sql语句进行预编译，下次执行相同的sql语句时，数据库端不会再进行预编译了，而直接用数据库的缓冲区，提高数据访问的效率（但尽量采用使用？号的方式传递参数），如果sql语句只执行一次，以后不再复用。 从安全性上来看，PreparedStatement是通过?来传递参数的，避免了拼sql而出现sql注入的问题，所以安全性较好。

在开发中，推荐使用 PreparedStatement

课2：mybatis架构

遇见小问题：

怎样打开myeclipse左边的视图栏：window-->show view-->package explorer-->show in-->navigator

SqlMapConfig.xml（mybatis全局配置文件，名称不固定）

运行环境（数据源、事务）

mapper.xml(配置sql语句), mapper.xml.....

SqlSessionFactory（会话工厂）

作用：创建SqlSession

SqlSession（向面用户的接口，数据库操作方法）

作用：操作数据库

Executor（数据库操作的执行器）

Executor是一个接口，有两个实现 （默认执行器和缓存执行器）

mysql

MappedStatement（mybatis的封装对象）

封装：sql语句(占位符)

输入 映射

java简单类型

pojo类型

hashmap

输出 映射

java简单类型

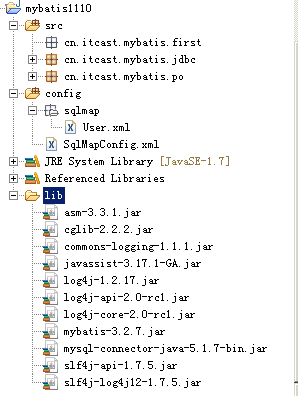
pojo类型

hashmap

课3：3mybatis工程结构

关于POJO的详细解说：具体见链接：https://blog.csdn.net/kevin\_loving/article/details/80491663

POJO是不包含业务逻辑的透明组件，就像其命名那样，它的确简单，只用作单纯的DTO(Data Transport Object　数据传输对象)或是VO(Value Object　值对象)．它通常的构成形式为一个包含私有变量作为属性以及访问其属性的getter和setter方法的类，根据需要可以为其添加constructor．对于初识POJO的人来说，它很像JavaBean，但按照其应与其他组件易组合的设计目的，它不应包含业务逻辑的处理，这也是与JavaBean不同的地方。



log4j.properties

SqlMapConfig.xml

课4：入门程序-查询用户信息

根据id查询用户

1，建立pojo

2，User.xml配置

建议命名规则：表名+mapper.xml

早期ibatis命名规则：表名.xml

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<!-- namespace命名空间，为了对sql语句进行隔离，方便管理 ，mapper开发dao方式，使用namespace有特殊作用

mapper代理开发时将namespace指定为mapper接口的全限定名

-->

<mapper namespace=*"test"*>

<!-- 在mapper.xml文件中配置很多的sql语句，执行每个sql语句时，封装为MappedStatement对象

mapper.xml以statement为单位管理sql语句

-->

<!-- 根据id查询用户信息 -->

<!--

id：唯一标识 一个statement

#{}：表示 一个占位符，如果#{}中传入简单类型的参数，#{}中的名称随意

parameterType：输入 参数的类型，通过#{}接收parameterType输入 的参数

resultType：输出结果 类型，不管返回是多条还是单条，指定单条记录映射的pojo类型

-->

<select id=*"findUserById"* parameterType=*"int"* resultType=*"cn.itcast.mybatis.po.User"*>

SELECT \* FROM USER WHERE id= #{id}

</select>

<!-- 根据用户名称查询用户信息，可能返回多条

${}：表示sql的拼接，通过${}接收参数，将参数的内容不加任何修饰拼接在sql中。

-->

<select id=*"findUserByName"* parameterType=*"java.lang.String"* resultType=*"cn.itcast.mybatis.po.User"*>

select \* from user where username like '%${value}%'

</select>

<!-- 添加用户

parameterType：输入 参数的类型，User对象 包括 username,birthday,sex,address

#{}接收pojo数据，可以使用OGNL解析出pojo的属性值

#{username}表示从parameterType中获取pojo的属性值

selectKey：用于进行主键返回，定义了获取主键值的sql

order：设置selectKey中sql执行的顺序，相对于insert语句来说

keyProperty：将主键值设置到哪个属性

resultType：select LAST\_INSERT\_ID()的结果 类型

-->

3，创建SqlSessionFactory:

// 会话工厂

private SqlSessionFactory sqlSessionFactory;

// 创建工厂

@Before

public void init() throws IOException {

// 配置文件（SqlMapConfig.xml）

String resource = "SqlMapConfig.xml";

// 加载配置文件到输入 流

InputStream inputStream = Resources.getResourceAsStream(resource);

// 创建会话工厂

sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

}

4.1，// 测试根据id查询用户(得到单条记录)

@Test

public void testFindUserById() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

// 第一个参数：statement的位置，等于namespace+statement的id

// 第二个参数：传入的参数

User user = null;

try {

user = sqlSession.selectOne("test.findUserById", 2);

} catch (Exception e) {

e.printStackTrace();

} finally {

// 关闭sqlSession

sqlSession.close();

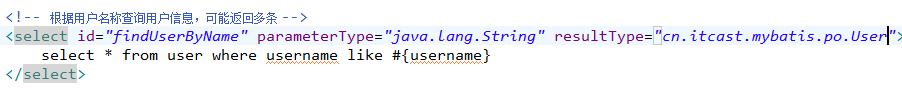
}

System.out.println(user);

}

4.2，根据用户名称模糊查询用户信息

根据用户名称模糊查询用户信息可能返回多条记录。



@Test

public void testFindUserByName() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

// 第一个参数：statement的位置，等于namespace+statement的id

// 第二个参数：传入的参数

List<User> list = null;

try {

list = sqlSession.selectList("test.findUserByName", "小明");

} catch (Exception e) {

e.printStackTrace();

} finally {

// 关闭sqlSession

sqlSession.close();

}

System.out.println(list.get(0).getUsername());

}

4.3，使用${}接收参数

<!-- 根据用户名称查询用户信息，可能返回多条

${}：表示sql的拼接，通过${}接收参数，将参数的内容不加任何修饰拼接在sql中。

-->

<select id="findUserByName" parameterType="java.lang.String" resultType="cn.itcast.mybatis.po.User">

select \* from user where username like '%${value}%'

</select>

## mybatis开发过程小结

1、编写SqlMapConfig.xml

2、编写mapper.xml

定义了statement

3、编程通过配置文件创建SqlSessionFactory

4、通过SqlSessionFactory获取SqlSession

5、通过SqlSession操作数据库

如果执行添加、更新、删除需要调用SqlSession.commit()

6、SqlSesion使用完成要关闭

知识点：

1，URL与URI的关系：

URL正是使用 Web 浏览器等访问 Web 页面时需要输入的网页地址  
 URI 用字符串标识某一互联网资源，而 URL表示资源的地点（互联  
 网上所处的位置）。可见 URL是 URI 的子集。

2，DAO层？

介于业务逻辑层和数据库之间，进行数据的访问和操作。

DAO层包括三个部分。

DAO接口：定义了一系列操作数据库的方法。

DAO实现类：实现了操作数据库的方法。（对于不同数据库，有不同的实现）

DAO工厂类：返回一个DAO实现类对象。

3，Statement

Statement 是 Java 执行数据库操作的一个重要接口，用于在已经建立数据库连接的基础上，向数据库发送要执行的SQL语句。

Statement， 用于执行静态的sql语句或者说拼装好的sql语句

PreparedStatement 对象用于执行带或不带 IN 参数的预编译 SQL 语句。

CallableStatement 对象用于执行对数据库已存在的存储过程的调用。

PreparedStatement接口相比于Statement，有以下几个重要的特点：

提高安全，防SQL注入：Statement这种直接拼接sql的方式，容易发生sql注入，因此你就应该始终以PreparedStatement代替Statement（mybatis框架默认的就是PreparedStatement）。

预编译技术提高性能：PreparedStatement 创建后，向Server端发送执行带有占位符的SQL语句，把SQL语句在Server端的执行计划和执行逻辑缓存起来，通过参数来调用，减少Sql编译等过程，这里值得强调是在实际使用中mysql不一定开启了预编译技术，具体见后面分析，这个和具体的mysql版本也有一定的关系。

参数转义、编码简洁：Statement 所有的参数需要我们自己拼接好，这样就比较恶心而且有的参数比较难拼接。

课5：5用户添加

// 测试根据id删除用户(得到单条记录)

@Test

public void testDeleteUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

try {

sqlSession.delete("test.deleteUser", 35);

// 需要提交事务

sqlSession.commit();

} catch (Exception e) {

e.printStackTrace();

} finally {

// 关闭sqlSession

sqlSession.close();

}

}

如有以下需求：user对象插入到数据库后，新记录的主键要通过user对象返回，通过user获取主键值。

解决思路：

通过LAST\_INSERT\_ID()获取刚插入记录的自增主键值，在insert语句执行后，执行select LAST\_INSERT\_ID()就可以获取自增主键。

User.xml修改：

<!-- 添加用户

parameterType：输入 参数的类型，User对象 包括 username,birthday,sex,address

#{}接收pojo数据，可以使用OGNL解析出pojo的属性值

#{username}表示从parameterType中获取pojo的属性值

selectKey：用于进行主键返回，定义了获取主键值的sql

order：设置selectKey中sql执行的顺序，相对于insert语句来说

keyProperty：将主键值设置到哪个属性

resultType：select LAST\_INSERT\_ID()的结果 类型

-->

<insert id="insertUser" parameterType="cn.itcast.mybatis.po.User">

<selectKey keyProperty="id" order="AFTER" resultType="int">

select LAST\_INSERT\_ID()

</selectKey>

INSERT INTO USER(username,birthday,sex,address) VALUES(#{username},#{birthday},#{sex},#{address})

</insert>

<!-- oracle

在执行insert之前执行select 序列.nextval() from dual取出序列最大值，将值设置到user对象 的id属性

-->

<!-- <insert id="insertUser" parameterType="cn.itcast.mybatis.po.User">

<selectKey keyProperty="id" order="BEFORE" resultType="int">

select 序列.nextval() from dual

</selectKey>

使用mysql的uuid机制生成主键：

使用uuid生成主键的好处是不考虑数据库移植后主键冲突问题。

实现思路：

先查询uuid得到主键，将主键设置到user对象中，将user对象插入数据库。

<!-- mysql的uuid生成主键 -->

<!-- <insert id="insertUser" parameterType="cn.itcast.mybatis.po.User">

<selectKey keyProperty="id" order="BEFORE" resultType="string">

select uuid()

</selectKey>

INSERT INTO USER(id,username,birthday,sex,address) VALUES(#{id},#{username},#{birthday},#{sex},#{address})

</insert> -->

课6：用户修改和删除

<!-- 用户删除 -->

<delete id="deleteUser" parameterType="int">

delete from user where id=#{id}

</delete>

<!-- 用户更新

要求：传入的user对象中包括 id属性值

-->

<update id="updateUser" parameterType="cn.itcast.mybatis.po.User">

update user set username=#{username},birthday=#{birthday},sex=#{sex},address=#{address} where id=#{id}

</update>

// 测试根据id删除用户(得到单条记录)

@Test

public void testDeleteUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

try {

sqlSession.delete("test.deleteUser", 35);

// 需要提交事务

sqlSession.commit();

} catch (Exception e) {

e.printStackTrace();

} finally {

// 关闭sqlSession

sqlSession.close();

}

}

// 测试根据id更新用户(得到单条记录)

@Test

public void testUpdateUser() {

// 通过sqlSessionFactory创建sqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 通过sqlSession操作数据库

// 创建更新数据对象，要求必须包括 id

User user = new User();

user.setId(35);

user.setUsername("燕青");

user.setAddress("河南郑州");

// user.setBirthday(new Date());

user.setSex("1");

try {

sqlSession.update("test.updateUser", user);

// 需要提交事务

sqlSession.commit();

} catch (Exception e) {

e.printStackTrace();

} finally {

// 关闭sqlSession

sqlSession.close();

}

System.out.println("用户的id=" + user.getId());

}

课7：mybatis和hibernate的适用场景

## Mybatis解决jdbc编程的问题

1. 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

1. Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

1. 向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决：Mybatis自动将java对象映射至sql语句，通过statement中的parameterType定义输入参数的类型。

1. 对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

解决：Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

mybatis与hibernate重要区别：

企业开发进行技术选型 ，考虑mybatis与hibernate适用场景。

mybatis：入门简单，程序容易上手开发，节省开发成本 。mybatis需要程序员自己编写sql语句，是一个不完全 的ORM框架，对sql修改和优化非常容易实现 。

mybatis适合开发需求变更频繁的系统，比如：互联网项目。

hibernate：入门门槛高，如果用hibernate写出高性能的程序不容易实现。hibernate不用写sql语句，是一个 ORM框架。

hibernate适合需求固定，对象数据模型稳定，中小型项目，比如：企业OA系统。

总之，企业在技术选型时根据项目实际情况，以降低成本和提高系统 可维护性为出发点进行技术选型。

小知识：ORM的解释：

对象关系映射（英语：(Object Relational Mapping，简称ORM，或O/RM，或O/R mapping）。

课8：sqlSession使用范围

### SqlMapConfig.xml

是mybatis全局配置文件，只有一个，名称不固定的，主要mapper.xml，mapper.xml中配置 sql语句

### mapper.xml

mapper.xml是以statement为单位进行配置。（把一个sql称为一个statement），satatement中配置 sql语句、parameterType输入参数类型（完成输入映射）、resultType输出结果类型（完成输出映射）。

还提供了parameterMap配置输入参数类型（过期了，不推荐使用了）

还提供resultMap配置输出结果类型（完成输出映射），明天重点讲通过resultMap完成复杂数据类型的映射（一对多，多对多映射）

### #{}

表示一个占位符，向占位符输入参数，mybatis自动进行java类型和jdbc类型的转换。

程序员不需要考虑参数的类型，比如：传入字符串，mybatis最终拼接好的sql就是参数两边加单引号。

#{}接收pojo数据，可以使用OGNL解析出pojo的属性值

### ${}

表示sql的拼接，通过${}接收参数，将参数的内容不加任何修饰拼接在sql中。

${}也可以接收pojo数据，可以使用OGNL解析出pojo的属性值

缺点：不能防止sql注入。

### selectOne

用于查询单条记录，不能用于查询多条记录，否则异常：

org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 4

## selectList:

用于查询多条记录，可以用于查询单条记录的。

课9：原始dao开发方式

## SqlSession作用范围

是使用局部变量、成员变量。。。。？？？

### SqlSessionFactoryBuilder

SqlSessionFactoryBuilder是以工具类方式来使用，需要创建sqlSessionFactory就new一个SqlSessionFactoryBuilder。

### sqlSessionFactory

正常开发时，以单例方式管理sqlSessionFactory，整个系统运行过程中sqlSessionFactory只有一个实例，将来和spring整合后由spring以单例方式管理sqlSessionFactory。

### SqlSession

sqlSession是一个面向用户（程序员）的接口，程序员调用sqlSession的接口方法进行操作数据库。

sqlSession能否以单例 方式使用？？

由于sqlSession是线程不安全，所以sqlSession最佳应用范围在方法体内，在方法体内定义局部变量使用sqlSession。

## 原始dao开发方式

程序员需要写dao接口和dao 的实现 类

### dao接口

**public** **interface** UserDao {

//根据id查询用户信息

**public** User findUserById(**int** id) **throws** Exception;

### dao接口实现

**public** **class** UserDaoImpl **implements** UserDao {

**private** SqlSessionFactory sqlSessionFactory;

// 将SqlSessionFactory注入

**public** UserDaoImpl(SqlSessionFactory sqlSessionFactory) {

**this**.sqlSessionFactory = sqlSessionFactory;

}

@Override

**public** User findUserById(**int** id) **throws** Exception {

// 创建SqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

// 根据id查询用户信息

User user = sqlSession.selectOne("test.findUserById", id);

sqlSession.close();

**return** user;

**}**

**//根据用户ID,姓名，列表查询用户。**

**public User findUserById(int id) throws Exception {**

**// 创建SqlSession**

**SqlSession sqlSession = sqlSessionFactory.openSession();**

**// 根据id查询用户信息**

**User user = sqlSession.selectOne("test.findUserById", id);**

**sqlSession.close();**

**return user;**

**}**

public List<User> findUserByUsername(String username) throws Exception {

// 创建SqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

List<User> list = sqlSession.selectList("test.findUserByName", username);

sqlSession.close();

return list;

}

@Override

public void insertUser(User user) throws Exception {

// 创建SqlSession

SqlSession sqlSession = sqlSessionFactory.openSession();

sqlSession.insert("test.insertUser", user);

sqlSession.commit();

sqlSession.close();

}

课10：mapper代理开发方式

程序员只需要写dao接口，dao接口实现对象由mybatis自动生成代理对象。

本身dao在三层架构中就是一个通用的接口。

### 上边原始dao开发方式的问题

1 dao的实现类中存在重复代码，整个mybatis操作的过程代码模板重复（先创建sqlsession、调用sqlsession的方法、关闭sqlsession）

2、dao的实现 类中存在硬编码，调用sqlsession方法时将statement的id硬编码。

课11：11mapper接口返回单个对象和集合对象

<!-- namespace命名空间，为了对sql语句进行隔离，方便管理 ，mapper开发dao方式，使用namespace有特殊作用

mapper代理开发时将namespace指定为mapper接口的全限定名

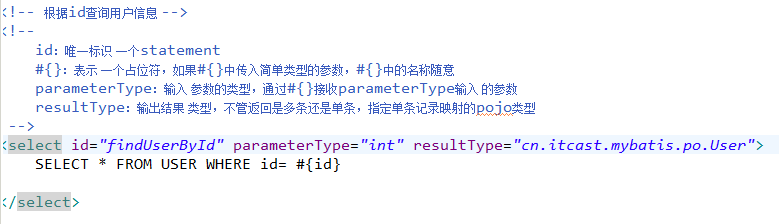
-->

<mapper namespace=*"cn.itcast.mybatis.mapper.UserMapper"*>

### mapper.xml（映射文件）

mapper映射文件的命名方式建议：表名Mapper.xml

namespace指定为mapper接口的全限定名



此步骤目的：通过mapper.xml和mapper.java进行关联。

2、mapper.xml中statement的id就是mapper.java中方法名

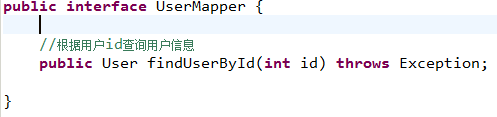
3、mapper.xml中statement的parameterType和mapper.java中方法输入参数类型一致

4、mapper.xml中statement的resultType和mapper.java中方法返回值类型一致.

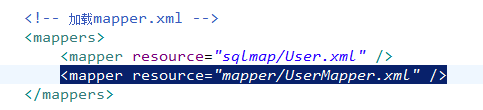
### mapper接口

mybatis提出了mapper接口，相当 于dao 接口。

mapper接口的命名方式建议：表名Mapper



### 将mapper.xml在SqlMapConfig.xml中加载



### mapper接口返回单个对象和集合对象

不管查询记录是单条还是多条，在 statement中resultType定义一致，都是单条记录映射的pojo类型。

mapper接口方法返回值，如果是返回的单个对象，返回值类型是pojo类型，生成的代理对象内部通过selectOne获取记录，如果返回值类型是集合对象，生成的代理对象内部通过selectList获取记录。

//根据用户id查询用户信息

**public** User findUserById(**int** id) **throws** Exception;

//根据用户名称 查询用户信息

**public** List<User> findUserByName(String username) **throws** Exception;

课12：mapper代理添加修改删除用户

课13：13mapper代理开发的问题总结

#### 返回值的问题

如果方法调用的statement，返回是多条记录，而mapper.java方法的返回值为pojo，此时代理对象通过selectOne调用，由于返回多条记录，所以报错：

org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 4

#### 输入参数的问题

使用mapper代理的方式开发，mapper接口方法输入 参数只有一个，可扩展性是否很差？？

可扩展性没有问题，因为dao层就是通用的，可以通过扩展pojo（定义pojo包装类型）将不同的参数（可以是pojo也可以简单类型）传入进去。

课14：14属性配置和全局参数

# sqlMapConfig.xml

SqlMapConfig.xml中配置的内容和顺序如下：

properties（属性）

settings（全局配置参数）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

## properties属性定义

可以把一些通用的属性值配置在属性文件中，加载到mybatis运行环境内。

比如：创建db.properties配置数据库连接参数。

<!-- 属性定义

加载一个properties文件

在 properties标签 中配置属性值

-->

<properties resource=*"db.properties"*>

<!-- <property name="" value=""/> -->

</properties>

<!-- 和spring整合后 environments配置将废除-->

<environments default=*"development"*>

<environment id=*"development"*>

<!-- 使用jdbc事务管理-->

<transactionManager type=*"JDBC"* />

<!-- 数据库连接池-->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${jdbc.driver}"*/>

<property name=*"url"* value=*"${jdbc.url}"*/>

<property name=*"username"* value=*"${jdbc.username}"*/>

<property name=*"password"* value=*"${jdbc.password}"*/>

</dataSource>

</environment>

</environments>

注意： MyBatis 将按照下面的顺序来加载属性：

* 在 properties 元素体内定义的属性首先被读取。
* 然后会读取properties 元素中resource或 url 加载的属性，它会覆盖已读取的同名属性。
* 最后读取parameterType传递的属性，它会覆盖已读取的同名属性。

建议使用properties，不要在properties中定义属性，只引用定义的properties文件中属性，并且properties文件中定义的key要有一些特殊的规则。

## settings全局参数配置

mybatis运行时可以调整一些全局参数（相当于软件的运行参数），参考：mybatis-settings.xlsx

根据使用需求进行参数配置。

注意：小心配置，配置参数会影响mybatis的执行。

ibatis的全局配置参数中包括很多的性能参数（最大线程数，最大待时间。。。），通过调整这些性能参数使ibatis达到高性能的运行，mybatis没有这些性能参数，由mybatis自动调节。

课15：15别名定义

## typeAliases(常用)

可以将parameterType、resultType中指定的类型，通过别名引用。

### mybaits提供了很多别名

|  |  |
| --- | --- |
| 别名 | 映射的类型 |
| \_byte | byte |
| \_long | long |
| \_short | short |
| \_int | int |
| \_integer | int |
| \_double | double |
| \_float | float |
| \_boolean | boolean |
| string | String |
| byte | Byte |
| long | Long |
| short | Short |
| int | Integer |
| integer | Integer |
| double | Double |
| float | Float |
| boolean | Boolean |
| date | Date |
| decimal | BigDecimal |
| bigdecimal | BigDecimal |

<!-- 定义 别名 -->

<typeAliases>

<!--

单个别名的定义

alias：别名，type：别名映射的类型 -->

<!-- <typeAlias type="cn.itcast.mybatis.po.User" alias="user"/> -->

<!-- 批量别名定义

指定包路径，自动扫描包下边的pojo，定义别名，别名默认为类名（首字母小写或大写）

-->

<package name=*"cn.itcast.mybatis.po"*/>

</typeAliases>

课16：mapper配置

### typeHandlers

类型处理器将java类型和jdbc类型进行映射。

mybatis默认提供很多类型处理器，一般情况下够用了。

<select id=*"findUserById"* parameterType=*"int"* resultType=*"cn.itcast.mybatis.po.User"*>

SELECT \* FROM USER WHERE id= #{id}

</select>

<!--加载mapper映射

如果将和spring整合后，可以使用整合包中提供的mapper扫描器，此处的mappers不用配置了。

-->

<mappers>

<!-- 通过resource引用mapper的映射文件 -->

<mapper resource=*"sqlmap/User.xml"* />

<!-- <mapper resource="mapper/UserMapper.xml" /> -->

<!-- 通过class引用mapper接口

class：配置mapper接口全限定名

要求：需要mapper.xml和mapper.java同名并且在一个目录 中

-->

<!-- <mapper class="cn.itcast.mybatis.mapper.UserMapper"/> -->

<!-- 批量mapper配置

通过package进行自动扫描包下边的mapper接口，

要求：需要mapper.xml和mapper.java同名并且在一个目录 中

-->

<package name=*"cn.itcast.mybatis.mapper"*/>

</mappers>

课17：parameterType传递包装类型

# 输入和输出映射

通过parameterType完成输入映射，通过resultType和resultMap完成输出映射。

## parameterType传递pojo包装对象

可以定义pojo包装类型扩展mapper接口输入参数的内容。

需求：

自定义查询条件查询用户信息，需要向statement输入查询条件，查询条件可以有user信息、商品信息。。。。

包装类型

//用户信息

**private** User user;

//自定义user的扩展对象

**private** UserCustom userCustom;

### mapper.xml

<!-- 自定义查询条件查询用户的信息

parameterType：指定包装类型

%${userCustom.username}%：userCustom是userQueryVo中的属性，通过OGNL获取属性的值

-->

<select id=*"findUserListResultMap"* parameterType=*"userQueryVo"* resultMap=*"userListResultMap"*>

select id id\_,username username\_,birthday birthday\_ from user where username like '%${userCustom.username}%'

</select>

### mapper.java

//自定义查询条件查询用户信息

**public** List<User> findUserList(UserQueryVo userQueryVo) **throws** Exception;

测试

// 通过包装类型查询用户信息

@Test

**public** **void** testFindUserList() **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

// 创建代理对象

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

// 构造查询条件

UserQueryVo userQueryVo = **new** UserQueryVo();

UserCustom userCustom = **new** UserCustom();

userCustom.setUsername("小明");

userCustom.setSex("1");

userQueryVo.setUserCustom(userCustom);

//id集合

List<Integer> ids = **new** ArrayList<Integer>();

ids.add(16);

ids.add(22);

userQueryVo.setIds(ids);

List<User> list = userMapper.findUserList(userQueryVo);

sqlSession.close();

System.*out*.println(list);

}

### 异常

如果parameterType中指定属性错误，异常，找不到getter方法：

org.apache.ibatis.exceptions.PersistenceException:

### Error querying database. Cause: org.apache.ibatis.reflection.ReflectionException: There is no getter for property named 'userCusto' in 'class cn.itcast.mybatis.po.UserQueryVo'

### Cause: org.apache.ibatis.reflection.ReflectionException: There is no getter for property named 'userCusto' in 'class cn.itcast.mybatis.po.UserQueryVo'

注意：如果将来和spring整合后，不是通过调用getter方法来获取属性值，通过反射强读取pojo的属性值。

课18：resultType的使用

## resultType

指定输出结果的类型（pojo、简单类型、hashmap..），将sql查询结果映射为java对象 。

### 返回简单类型

mapper.xml

<!-- 输出简单类型

功能：自定义查询条件，返回查询记录个数，通常用于实现 查询分页

-->

<select id=*"findUserCount"* parameterType=*"userQueryVo"* resultType=*"int"*>

select count(\*) from user

//查询用户，返回记录个数

**public** **int** findUserCount(UserQueryVo userQueryVo) **throws** Exception;

// 返回查询记录总数

@Test

**public** **void** testFindUserCount() **throws** Exception {

SqlSession sqlSession = sqlSessionFactory.openSession();

// 创建代理对象

UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);

// 构造查询条件

UserQueryVo userQueryVo = **new** UserQueryVo();

UserCustom userCustom = **new** UserCustom();

userCustom.setUsername("小明");

userQueryVo.setUserCustom(userCustom);

**int** count = userMapper.findUserCount(userQueryVo);

sqlSession.close();

System.*out*.println(count);

}

注意：

如果查询记录结果集为一条记录且一列再使用返回简单类型。

课19：resultType和resultMap的区别

## resultMap(入门)

resultType ：指定输出结果的类型（pojo、简单类型、hashmap..），将sql查询结果映射为java对象 。

使用resultType注意：sql查询的列名要和resultType指定pojo的属性名相同，指定相同 属性方可映射成功，如果sql查询的列名要和resultType指定pojo的属性名全部不相同，list中无法创建pojo对象的。

resultMap：将sql查询结果映射为java对象。

如果sql查询列名和最终要映射的pojo的属性名不一致，使用resultMap将列名和pojo的属性名做一个对应关系 （列名和属性名映射配置）

课20：resultMap入门

<!-- 定义resultMap，列名和属性名映射配置

id：mapper.xml中的唯一标识

type：最终要映射的pojo类型

-->

### 使用resultMap

<!-- 使用resultMap作结果映射

resultMap：如果引用resultMap的位置和resultMap的定义在同一个mapper.xml，

直接使用resultMap的id，如果不在同一个mapper.xml要在resultMap的id前边加namespace

-->

<select id=*"findUserListResultMap"* parameterType=*"userQueryVo"* resultMap=*"userListResultMap"*>

select id id\_,username username\_,birthday birthday\_ from user where username like '%${userCustom.username}%'

</select>

### resultMap配置

<resultMap id=*"userListResultMap"* type=*"user"* >

<!-- 列名

id\_,username\_,birthday\_

id：要映射结果集的唯 一标识 ，称为主键

column：结果集的列名

property：type指定的哪个属性中

-->

<id column=*"id\_"* property=*"id"*/>

<!-- result就是普通列的映射配置 -->

<result column=*"username\_"* property=*"username"*/>

<result column=*"birthday\_"* property=*"birthday"*/>

</resultMap>

### mapper.java

//查询用户，使用resultMap进行映射

**public** List<User> findUserListResultMap(UserQueryVo userQueryVo)**throws** Exception;

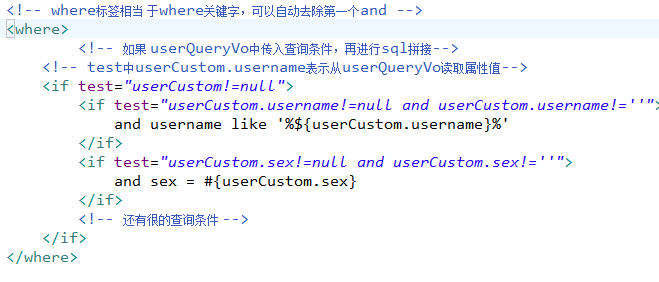
课21：动态sql-ifwhere和sql片段

动态sql(重点)

mybatis重点是对sql的灵活解析和处理。

## 需求

将自定义查询条件查询用户列表和查询用户列表总记录数改为动态sql



## sql片段

通过sql片段可以将通用的sql语句抽取出来，单独定义，在其它的statement中可以引用sql片段。

通用的sql语句，常用：where条件、查询列

<!-- 将用户查询条件定义为sql片段

建议对单表的查询条件单独抽取sql片段，提高公用性

注意：不要将where标签放在sql片段

-->

<sql id=*"query\_user\_where"*>

<!-- 如果 userQueryVo中传入查询条件，再进行sql拼接-->

<!-- test中userCustom.username表示从userQueryVo读取属性值-->

<if test=*"userCustom!=null"*>

<if test=*"userCustom.username!=null and userCustom.username!=''"*>

and username like '%${userCustom.username}%'

</if>

<if test=*"userCustom.sex!=null and userCustom.sex!=''"*>

and sex = #{userCustom.sex}

</if>

select id,username,birthday from user

<!-- where标签相当 于where关键字，可以自动去除第一个and -->

<where>

<!-- 引用sql片段，如果sql片段和引用处不在同一个mapper必须前边加namespace -->

<include refid=*"query\_user\_where"*></include>

<!-- 下边还有很其它的条件 -->

<!-- <include refid="其它的sql片段"></include> -->

</where>

</select>

<!-- where标签相当 于where关键字，可以自动去除第一个and -->

<where>

<!-- 引用sql片段，如果sql片段和引用处不在同一个mapper必须前边加namespace -->

<include refid=*"query\_user\_where"*></include>

<!-- 下边还有很其它的条件 -->

<!-- <include refid="其它的sql片段"></include> -->

</where>

课22：foreach的使用

## foreach

在statement通过foreach遍历parameterType中的集合类型。

需求：

根据多个用户id查询用户信息。

### 在userQueryVo中定义list<Integer> ids

在userQueryvo中定义list<Integer> ids存储多个id

### 修改where语句

使用foreach遍历list：

<!-- 根据id集合查询用户信息 -->

<!-- 最终拼接的效果：

SELECT id ,username ,birthday FROM USER WHERE username LIKE '%小明%' AND id IN (16,22,25)

collection：集合的属性

open：开始循环拼接的串

close：结束循环拼接的串

item：每次循环取到的对象

separator：每两次循环中间拼接的串

-->

<foreach collection=*"ids"* open=*" AND id IN ( "* close=*")"* item=*"id"* separator=*","*>

#{id}

</foreach>

<!--

SELECT id ,username ,birthday FROM USER WHERE username LIKE '%小明%' AND (id = 16 OR id = 22 OR id = 25)

<foreach collection="ids" open=" AND ( " close=")" item="id" separator="OR">

id = #{id}

</foreach>

-->

### 测试代码

//id集合

List<Integer> ids = **new** ArrayList<Integer>();

ids.add(16);

ids.add(22);

userQueryVo.setIds(ids);

List<User> list = userMapper.findUserList(userQueryVo);

Day02:

课1：订单商品数据模型分析

# 复习

mybatis是什么？

mybatis是一个持久层框架，是apache下的开源项目，前身是itbatis，是一个不完全的ORM框架，mybatis提供输入和输出的映射，需要程序员自己写sql语句，mybatis重点对 sql语句的灵活操作。

适合用于：需求变化频繁， 数据模型不固定的项目，例如：互联网项目。

mybatis架构？

SqlMapConfig.xml（名称不固定），配置内容：数据源、事务、properties、typeAliases、settings、mapper配置。

SqlSessionFactory--会话工厂，作用是创建SqlSession，实际开发中以单例模式管理 SqlSessionFactory。

SqlSession--会话，是一个面向用户（程序员）的接口，使用mapper代理方法开发是不需要程序员直接调用sqlSession的方法。是线程不安全，最佳适用场合方法体内。

mybatis开发dao的方法：

1、原始dao开发方法，需要程序员编写dao接口和实现类，此方法在当前企业中还有使用，因为ibatis使用的就是原始dao开发方法。

2、mapper代理方法，程序员只需要写mapper接口（相当于dao接口），mybatis自动根据mapper接口和mapper接口对应的statement自动生成代理对象（接口实现类对象）。

开发需要遵循规则：

1）mapper.xml中namespace是mapper接口的全限定名

2）mapper.xml中statement的id为mapper接口方法名

3）mapper.xml中statement的输入映射类型（parameterType）和mapper接口方法输入参数类型一致

4) mapper.xml中statement的输出映射类型（resultType）和mapper接口方法返回结果类型一致

resultType和resultMap都可以完成输出映射：

resultType映射要求sql查询的列名和输出映射pojo类型的属性名一致

resultMap映射时对sql查询的列名和输出映射pojo类型的属性名作一个对应关系。

动态sql：

#{}和${}完成输入参数的属性值获取，通过OGNL获取parameterType指定pojo的属性名。

#{}：占位符号，好处防止sql注入

${}：sql拼接符号

if

where

foreach

# 课程计划

1、使用resultMap完成高级映射(重点)

学习商品订单数据模型（一对一、一对多、多对多）

resultMap实现一对一、一对多、多对多

延迟加载

2、查询缓存(重点)

一级缓存

二级缓存

3、mbatis和spring整合(重点)

4、mybatis逆向工程(常用)

# 商品订单数据模型

学会在企业中如何去分析陌生表的数据模型：

1、学习单表记录了什么东西（去学习理解需求）

2、学习单表重要字段的意义（优先学习不能为空的字段）

3、学习表与表之间的关系（一对一、一对多、多对多）

通过表的外键分析表之间的关系

注意：分析表与表之间的关系时是要建立在具体 的业务意义基础之上

用户表user：

记录了购买商品的用户

订单表orders：

记录了用户所创建的订单信息

订单明细表orderdetail：

记录了用户创建订单的详细信息

商品信息表items：

记录了商家提供的商品信息

分析表与表之间的关系：

用户user和订单orders：

user---->orders：一个用户可以创建多个订单 一对多

orders-->user：一个订单只能由一个用户创建 一对一

订单orders和订单明细orderdetail：

orders-->orderdetail：一个订单可以包括多个订单明细 一对多

orderdetail-->orders：一个订单明细只属于一个订单 一对一

订单明细orderdetail和商品信息items：

orderdetail-->items：一个订单明细对应一个商品信息 一对一

items--> orderdetail：一个商品对应多个订单明细 一对多

课2：2resultType实现一对一查询

OrdersMapperCustom.xml:自定义的订单映射配置。

## 需求

查询订单信息关联查询用户信息

## sql语句

查询语句：

先确定主查询表：订单信息表

再确定关联查询表：用户信息

通过orders关联查询用户使用user\_id一个外键，只能关联查询出一条用户记录就可以使用内连接

SELECT

orders.\*,

user.username,

user.sex

FROM

orders,

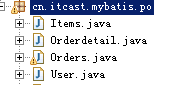
USER

WHERE orders.user\_id = user.id

## 使用resultType实现

### 创建po类

基础的单表的 po类：



### 一对一查询映射的pojo

创建pojo包括 订单信息和用户信息，resultType才可以完成映射。

创建OrderCustom作为自定义pojo，继承sql查询列多的po类。

**public** **class** OrderCustom **extends** Orders {

//补充用户信息

**private** String username;

**private** String sex;

**private** String address;

### mapper.xml

定义mapper.xml文件，

<!-- 一对一查询使用reusltType完成

查询订单关联查询用户信息

-->

<select id=*"findOrderUserList"* resultType=*"orderCustom"*>

SELECT

orders.\*,

user.username,

user.sex

FROM

orders,

USER

WHERE orders.user\_id = user.id

</select>

### mapper.java

**public** **interface** OrdersMapperCustom {

// 一对一查询，查询订单关联查询用户，使用resultType

**public** List<OrderCustom> findOrderUserList() **throws** Exception;

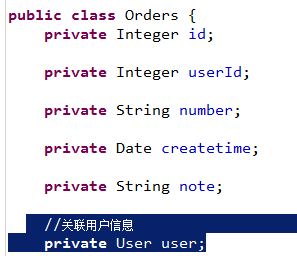
课3：3resultMap实现一对一查询

## 使用resultMap实现一对一

### resultMap映射思路

resultMap提供一对一关联查询的映射和一对多关联查询映射，一对一映射思路：将关联查询的信息映射到pojo中，如下：

在Orders类中创建一个User属性，将关联查询的信息映射到User属性中。



### resultMap定义

<!-- 一对一查询resultMap -->

<resultMap type=*"orders"* id=*"ordersUserResultMap"*>

<!-- 完成了订单信息的映射配置 -->

<!-- id：订单关联用户查询的唯 一 标识 -->

<id column=*"id"* property=*"id"*/>

<result column=*"user\_id"* property=*"userId"*/>

<result column=*"number"* property=*"number"*/>

<result column=*"createtime"* property=*"createtime"*/>

<result column=*"note"* property=*"note"*/>

<!-- 下边完成关联信息的映射

association：用于对关联信息映射到单个pojo

property：要将关联信息映射到orders的哪个属性中

javaType：关联信息映射到orders的属性的类型，是user的类型

-->

<association property=*"user"* javaType=*"user"*>

<!-- id：关联信息的唯 一标识 -->

<!-- property： 要映射到user的哪个属性中-->

<id column=*"user\_id"* property=*"id"*/>

<!-- result就是普通列的映射 -->

<result column=*"username"* property=*"username"*/>

<result column=*"sex"* property=*"sex"*/>

</association>

### mapper.java

// 一对一查询，使用resultMap

**public** List<Orders> findOrderUserListResultMap() **throws** Exception;

## 小结

resultType：要自定义pojo 保证sql查询列和pojo的属性对应，这种方法相对较简单，所以应用广泛。

resultMap：使用association完成一对一映射需要配置一个resultMap，过程有点复杂，如果要实现延迟加载就只能用resultMap实现 ，如果为了方便对关联信息进行解析，也可以用association将关联信息映射到pojo中方便解析。

课4：4resultMap实现一对多查询

# 一对多查询

## 需求

查询所有订单信息及订单下的订单明细信息。

## sql语句

主查询表：订单表

关联查询表：订单明细

SELECT

orders.\*,

user.username,

user.sex ,

orderdetail.id orderdetail\_id,

orderdetail.items\_num,

orderdetail.items\_id

FROM

orders,

USER,

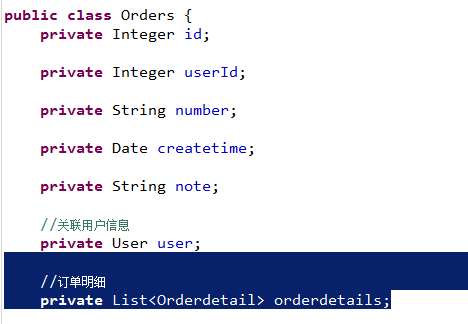
orderdetail

WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id

## resultMap进行一对多映射思路

resultMap 提供collection完成关联信息映射到集合对象中。

在orders类中创建集合属性：



## mapper.xml

<!-- 一对多查询使用reusltMap完成

查询订单关联查询订单明细

-->

<select id=*"findOrderAndOrderDetails"* resultMap=*"orderAndOrderDetails"* >

SELECT

orders.\*,

user.username,

user.sex ,

orderdetail.id orderdetail\_id,

orderdetail.items\_num,

orderdetail.items\_id

FROM

orders,

USER,

orderdetail

WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id

</select>

## resultMap定义

<!-- 一对多，查询订单及订单明细 -->

<resultMap type=*"orders"* id=*"orderAndOrderDetails"* extends=*"ordersUserResultMap"*>

<!-- 映射订单信息，和用户信息，这里使用继承ordersUserResultMap -->

<!-- 映射订单明细信息

property：要将关联信息映射到orders的哪个属性中

ofType：集合中pojo的类型

-->

<collection property=*"orderdetails"* ofType=*"cn.itcast.mybatis.po.Orderdetail"*>

<!-- id：关联信息订单明细的唯 一标识

property：Orderdetail的属性名

-->

<id column=*"orderdetail\_id"* property=*"id"*/>

<result column=*"items\_num"* property=*"itemsNum"*/>

<result column=*"items\_id"* property=*"itemsId"*/>

</collection>

</resultMap>

mapper.java



课5：resultMap实现一对多复杂查询

## 需求

查询所有用户信息，关联查询订单及订单明细信息及商品信息，订单明细信息中关联查询商品信息

## sql

主查询表：用户信息

关联查询：订单、订单明细，商品信息

SELECT

orders.\*,

user.username,

user.sex ,

orderdetail.id orderdetail\_id,

orderdetail.items\_num,

orderdetail.items\_id,

items.name items\_name,

items.detail items\_detail

FROM

orders,

USER,

orderdetail,

items

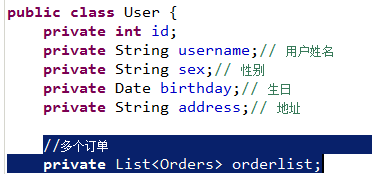
WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id AND items.id = orderdetail.items\_id

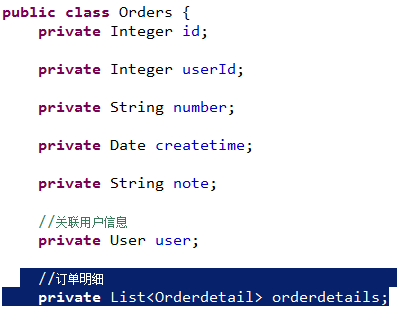
## pojo定义

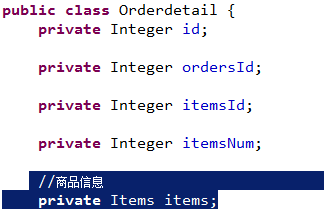
在user.java中创建映射的属性：集合 List<Orders>orderlist

在Orders中创建映射的属性：集合List<Orderdetail> orderdetails

在Orderdetail中创建商品属性：pojo Items items







## mapper.xml

<!-- 一对多查询使用reusltMap完成

查询用户及订单和订单明细，关联商品，的信息

-->

<select id=*"findUserOrderDetail"* resultMap=*"userOrderDetailResultMap"* >

SELECT

orders.\*,

user.username,

user.sex ,

orderdetail.id orderdetail\_id,

orderdetail.items\_num,

orderdetail.items\_id,

items.name items\_name,

items.detail items\_detail

FROM

orders,

USER,

orderdetail,

items

WHERE orders.user\_id = user.id AND orders.id = orderdetail.orders\_id AND items.id = orderdetail.items\_id

</select>

## resultMap

<!-- 一对多查询，查询用户及订单明细和商品信息 -->

<resultMap type=*"user"* id=*"userOrderDetailResultMap"*>

<!-- 用户信息映射 -->

<id column=*"user\_id"* property=*"id"*/>

<result column=*"username"* property=*"username"*/>

<result column=*"sex"* property=*"sex"*/>

<!-- 订单信息 -->

<collection property=*"orderlist"* ofType=*"cn.itcast.mybatis.po.Orders"*>

<id column=*"id"* property=*"id"*/>

<result column=*"user\_id"* property=*"userId"*/>

<result column=*"number"* property=*"number"*/>

<result column=*"createtime"* property=*"createtime"*/>

<result column=*"note"* property=*"note"*/>

<!-- 订单明细映射 -->

<collection property=*"orderdetails"* ofType=*"cn.itcast.mybatis.po.Orderdetail"*>

<!-- id：关联信息订单明细的唯 一标识

property：Orderdetail的属性名

-->

<id column=*"orderdetail\_id"* property=*"id"*/>

<result column=*"items\_num"* property=*"itemsNum"*/>

<result column=*"items\_id"* property=*"itemsId"*/>

<!-- 商品信息 -->

<association property=*"items"* javaType=*"cn.itcast.mybatis.po.Items"*>

<id column=*"item\_id"* property=*"id"*/>

<result column=*"items\_name"* property=*"name"*/>

<result column=*"items\_detail"* property=*"detail"*/>

</association>

</collection>

</collection>

</resultMap>

## mapper.java



课6：多对多查询需求

# 多对多查询（自己实现 ）

一对多是多对多的特例。

需求1：

查询显示字段：用户账号、用户名称、用户性别、商品名称、商品价格(最常见)

企业开发中常见明细列表，用户购买商品明细列表，

使用resultType将上边查询列映射到pojo输出。

需求2：

查询显示字段：用户账号、用户名称、购买商品数量、商品明细（鼠标移上显示明细）

使用resultMap将用户购买的商品明细列表映射到user对象中。

课7：延迟加载

## 使用延迟加载意义

在进行数据查询时，**为了提高数据库查询性能**，尽量使用单表查询，因为单表查询比多表关联查询速度要快。

如果查询单表就可以满足需求，一开始先查询单表，当需要关联信息时，再关联查询，当需要关联信息再查询这个叫延迟加载。

mybatis中resultMap提供延迟加载功能，通过resultMap配置延迟加载。

## 配置mybatis支持延迟加载

|  |  |  |  |
| --- | --- | --- | --- |
| 设置项 | 描述 | 允许值 | 默认值 |
| lazyLoadingEnabled | 全局性设置懒加载。如果设为‘false’，则所有相关联的都会被初始化加载。 | true | false | false |
| aggressiveLazyLoading | 当设置为‘true’的时候，懒加载的对象可能被任何懒属性全部加载。否则，每个属性都**按需加载**。 | true | false | true |

<!-- 全局配置参数 -->

<settings>

<!-- 延迟加载总开关 -->

<settingname=*"lazyLoadingEnabled"*value=*"true"*/>

<!-- 设置按需加载 -->

<settingname=*"aggressiveLazyLoading"*value=*"false"*/>

</settings>

## 延迟加载实现

### 实现思路

需求：

查询订单及用户的信息，一对一查询。

刚开始只查询订单信息

当需要用户时调用 Orders类中的getUser()方法执行延迟加载 ，向数据库发出sql。

### mapper.xml

<!-- 一对一查询延迟加载

开始只查询订单，对用户信息进行延迟加载

-->

<select id=*"findOrderUserListLazyLoading"* resultMap=*"orderCustomLazyLoading"*>

SELECT

orders.\*

FROM

orders

</select>

### resultMap

<!-- 一对一查询延迟加载 的配置 -->

<resultMap type=*"orders"* id=*"orderCustomLazyLoading"*>

<!-- 完成了订单信息的映射配置 -->

<!-- id：订单关联用户查询的唯 一 标识 -->

<id column=*"id"* property=*"id"* />

<result column=*"user\_id"* property=*"userId"* />

<result column=*"number"* property=*"number"* />

<result column=*"createtime"* property=*"createtime"* />

<result column=*"note"* property=*"note"* />

<!-- 配置用户信息的延迟加载 select：延迟加载执行的sql所在的statement的id，如果不在同一个namespace需要加namespace

sql：根据用户id查询用户信息 column：关联查询的列 property:将关联查询的用户信息设置到Orders的哪个属性 -->

<association property=*"user"*

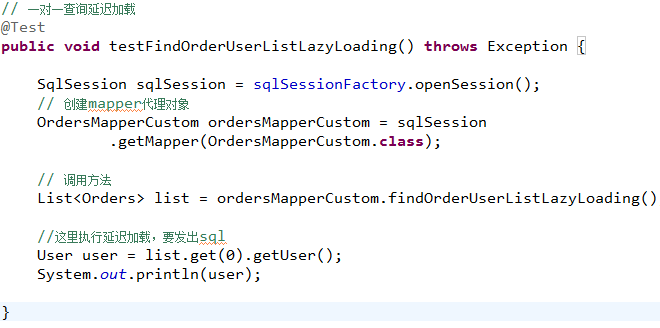
select=*"cn.itcast.mybatis.mapper.UserMapper.findUserById"* column=*"user\_id"*></association>

</resultMap>

### mapper.java



### 测试代码



## 一对多延迟加载

一对多延迟加载的方法同一对一延迟加载，在collection标签中配置select内容。

本部分内容自学。

课8：resultType、resultMap、延迟加载的应用场合

# esultType、resultMap、延迟加载使用场景总结

延迟加载：

延迟加载实现的方法多种多样，在只查询单表就可以满足需求，为了提高数据库查询性能使用延迟加载，再查询关联信息。

mybatis提供延迟加载的功能用于service层。

resultType：

作用：

将查询结果按照sql列名pojo属性名一致性映射到pojo中。

场合：

常见一些明细记录的展示，将关联查询信息全部展示在页面时，此时可直接使用resultType将每一条记录映射到pojo中，在前端页面遍历list（list中是pojo）即可。

resultMap：

使用association和collection完成一对一和一对多高级映射。

association：

作用：

将关联查询信息映射到一个pojo类中。

场合：

为了**方便获取关联**信息可以使用association将关联订单映射为pojo，比如：查询订单及关联用户信息。

collection：

作用：

将关联查询信息映射到一个list集合中。

场合：

为了**方便获取关联信息**可以使用collection将关联信息映射到list集合中，比如：查询用户权限范围模块和功能，可使用collection将模块和功能列表映射到list中。

# 查询缓存

## 缓存的意义

将用户经常查询的数据放在缓存（内存）中，用户去查询数据就不用从磁盘上(关系型数据库数据文件)查询，从缓存中查询，从而提高查询效率，解决了高并发系统的性能问题。

用户、程序

**系统持久层(mybatis)**

**缓存区域(数据库取出数据缓存**)

数据库

缓存数据

系统业务层(spring)

缓存区域（业务数据进行缓存）

系统控制层(springmvc)

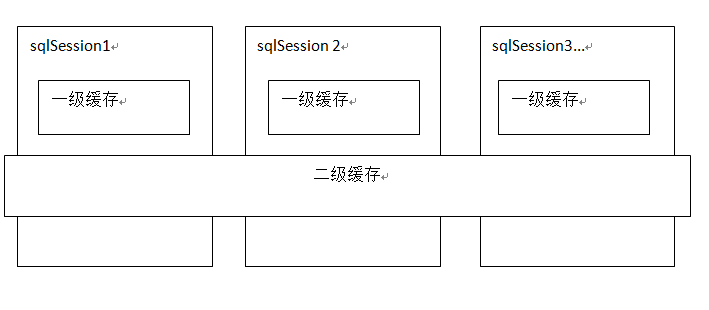
缓存区域(响应的jsp页面进行缓存)

缓存数据

缓存数据

## mybatis持久层缓存

mybatis提供一级缓存和二级缓存

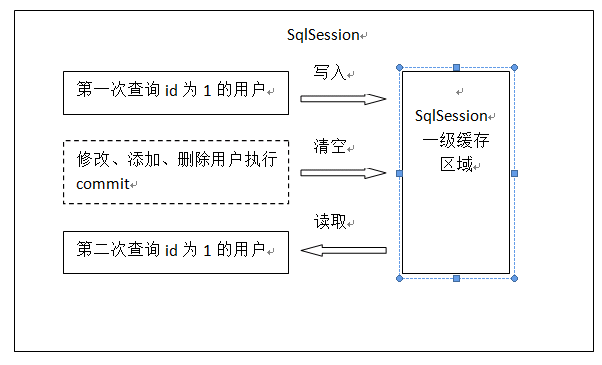


mybatis一级缓存是一个SqlSession级别，sqlsession只能访问自己的一级缓存的数据，二级缓存是跨sqlSession，是mapper级别的缓存，对于mapper级别的缓存不同的sqlsession是可以共享的。

课9：一级缓存

## 一级缓存

### 原理



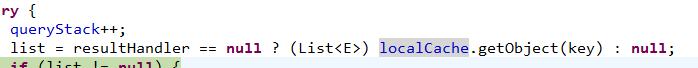
第一次发出一个查询sql，sql查询结果写入sqlsession的一级缓存中，缓存使用的数据结构是一个map<key,value>

key：hashcode+sql+sql输入参数+输出参数（sql的唯一标识）

value：用户信息

，同一个sqlsession再次发出相同的sql，就从缓存中取不走数据库。如果两次中间出现commit操作（修改、添加、删除），本sqlsession中的一级缓存区域全部清空，下次再去缓存中查询不到所以要从数据库查询，从数据库查询到再写入缓存。

每次查询都先从缓存中查询：



如果缓存中查询到则将缓存数据直接返回。

如果缓存中查询不到就从数据库查询：



### 一级缓存配置

mybatis默认支持一级缓存不需要配置。

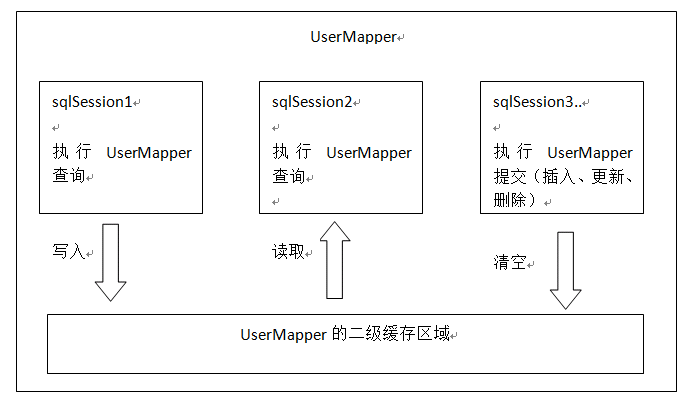
**注意：mybatis和spring整合后进行mapper代理开发，不支持一级缓存，mybatis和spring整合，spring按照mapper的模板去生成mapper代理对象，模板中在最后统一关闭sqlsession。**

### 一级缓存测试



## 二级缓存

### 原理



二级缓存的范围是mapper级别（mapper同一个命名空间），mapper以命名空间为单位创建缓存数据结构，结构是map<key、value>。

每次查询先看是否开启二级缓存，如果开启从二级缓存的数据结构中取缓存数据，

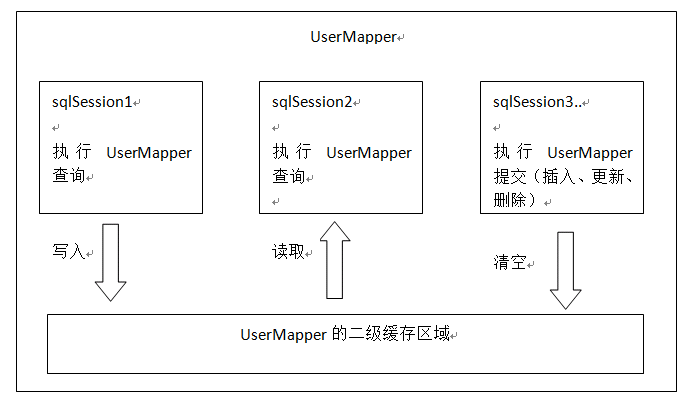


如果从二级缓存没有取到，再从一级缓存中找，如果一级缓存也没有，从数据库查询。

课10：二级缓存

## 二级缓存

### 原理



二级缓存的范围是mapper级别（mapper同一个命名空间），mapper以命名空间为单位创建缓存数据结构，结构是map<key、value>。

每次查询先看是否开启二级缓存，如果开启从二级缓存的数据结构中取缓存数据，



如果从二级缓存没有取到，再从一级缓存中找，如果一级缓存也没有，从数据库查询。

### mybatis二级缓存配置

在核心配置文件SqlMapConfig.xml中加入

<settingname=*"cacheEnabled"*value=*"true"*/>

|  |  |  |  |
| --- | --- | --- | --- |
|  | 描述 | 允许值 | 默认值 |
| cacheEnabled | 对在此配置文件下的所有cache 进行全局性开/关设置。 | true false | true |

要在你的Mapper映射文件中添加一行：  <cache /> ，表示此mapper开启二级缓存。

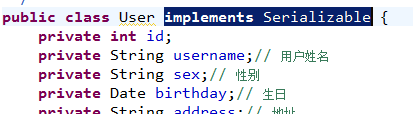
### 查询结果映射的pojo序列化

mybatis二级缓存需要将查询结果映射的pojo实现 java.io.serializable接口，如果不实现则抛出异常：

org.apache.ibatis.cache.CacheException: Error serializing object. Cause: java.io.NotSerializableException: cn.itcast.mybatis.po.User

二级缓存可以将内存的数据写到磁盘，存在对象的序列化和反序列化，所以要实现java.io.serializable接口。

如果结果映射的pojo中还包括了pojo，都要实现java.io.serializable接口。



### 二级缓存禁用

对于变化频率较高的sql，需要禁用二级缓存：

在statement中设置useCache=false可以禁用当前select语句的二级缓存，即每次查询都会发出sql去查询，默认情况是true，即该sql使用二级缓存。

<select id="findOrderListResultMap" resultMap="ordersUserMap"useCache="false">

### 刷新缓存

如果sqlsession操作commit操作，对二级缓存进行刷新（全局清空）。

设置statement的flushCache是否刷新缓存，默认值是true。

### 测试代码



课11：mybatis和ehcache整合

### mybatis的cache参数（了解）

mybatis的cache参数只适用于mybatis维护缓存。

flushInterval（刷新间隔）可以被设置为任意的正整数，而且它们代表一个合理的毫秒形式的时间段。默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新。

size（引用数目）可以被设置为任意正整数，要记住你缓存的对象数目和你运行环境的可用内存资源数目。默认值是1024。

readOnly（只读）属性可以被设置为true或false。只读的缓存会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。可读写的缓存会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是false。

<!-- namespace命名空间，为了对sql语句进行隔离，方便管理 ，mapper开发dao方式，使用namespace有特殊作用 mapper代理开发时将namespace指定为mapper接口的全限定名 -->

[<mapper namespace="](H:/java/java/后端框架/mybatis/mybatis02/0319/源代码/mybatis1110/src/cn/itcast/mybatis/mapper/UserMapper.xml)**[cn.itcast.mybatis.mapper.UserMapper](H:/java/java/后端框架/mybatis/mybatis02/0319/源代码/mybatis1110/src/cn/itcast/mybatis/mapper/UserMapper.xml)**[">](H:/java/java/后端框架/mybatis/mybatis02/0319/源代码/mybatis1110/src/cn/itcast/mybatis/mapper/UserMapper.xml)

<!-- 在mapper.xml文件中配置很多的sql语句，执行每个sql语句时，封装为MappedStatement对象 mapper.xml以statement为单位管理sql语句 -->

<!-- 开启二级缓存 -->

<!-- 单位：毫秒 -->

[<cache type="](H:/java/java/后端框架/mybatis/mybatis02/0319/源代码/mybatis1110/src/cn/itcast/mybatis/mapper/UserMapper.xml)**[org.mybatis.caches.ehcache.EhcacheCache](H:/java/java/后端框架/mybatis/mybatis02/0319/源代码/mybatis1110/src/cn/itcast/mybatis/mapper/UserMapper.xml)**[">](H:/java/java/后端框架/mybatis/mybatis02/0319/源代码/mybatis1110/src/cn/itcast/mybatis/mapper/UserMapper.xml)<property value="**12000**" name="**timeToIdleSeconds**"/><property value="**3600**" name="**timeToLiveSeconds**"/>

<!-- 同ehcache参数maxElementsInMemory -->

<property value="**1000**" name="**maxEntriesLocalHeap**"/>

<!-- 同ehcache参数maxElementsOnDisk -->

<property value="**10000000**" name="**maxEntriesLocalDisk**"/><property value="**LRU**" name="**memoryStoreEvictionPolicy**"/></cache>

如下例子：

<cache  eviction="FIFO"  flushInterval="60000"  size="512"  readOnly="true"/>

这个更高级的配置创建了一个 FIFO 缓存,并每隔 60 秒刷新,存数结果对象或列表的 512 个引用,而且返回的对象被认为是只读的,因此在不同线程中的调用者之间修改它们会导致冲突。可用的收回策略有, 默认的是 LRU:

1. LRU – 最近最少使用的:移除最长时间不被使用的对象。
2. FIFO – 先进先出:按对象进入缓存的顺序来移除它们。
3. SOFT – 软引用:移除基于垃圾回收器状态和软引用规则的对象。
4. WEAK – 弱引用:更积极地移除基于垃圾收集器状态和弱引用规则的对象。

## mybatis和ehcache缓存框架整合

mybatis二级缓存通过ehcache维护缓存数据。

### 分布缓存

将缓存数据数据进行分布式管理。

系统1

springmvc

mybatis

系统2

springmvc

mybatis

用户请求

缓存框架（远程服务器集群）

redis、memcached、ehcache

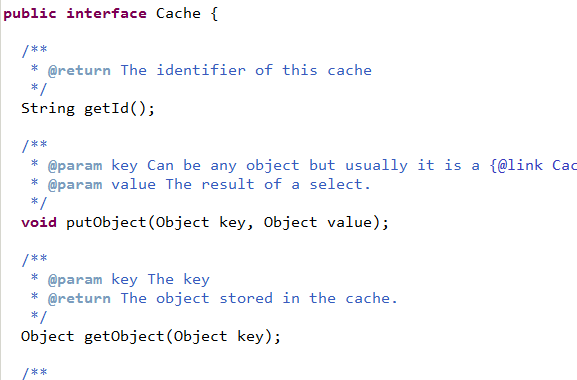
存取缓存数据

存取缓存数据

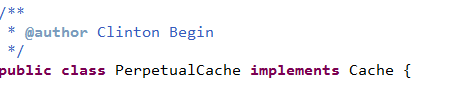
### mybatis和ehcache思路

通过mybatis和ehcache框架进行整合，就可以把缓存数据的管理托管给ehcache。

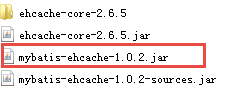
在mybatis中提供一个cache接口，只要实现cache接口就可以把缓存数据灵活的管理起来。



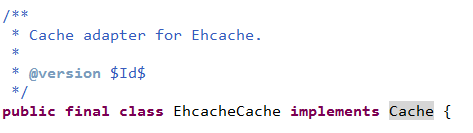
mybatis中默认实现：



### 下载和ehcache整合的jar包



ehcache对cache接口的实现类：

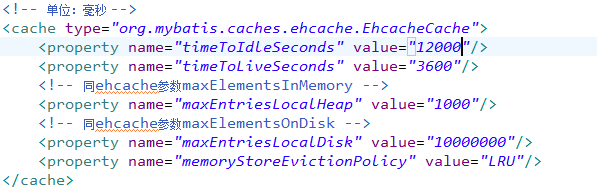


### 配置ehcache.xml



### 整合测试

在mapper.xml添加ehcache配置：



课12：

二级缓存场景及局限性

对查询频率高，变化频率低的数据建议使用二级缓存。

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用mybatis二级缓存技术降低数据库访问量，提高访问速度，业务场景比如：耗时较高的统计分析sql、电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由mybatis每隔一段时间自动清空缓存，根据数据变化频率设置缓存刷新间隔flushInterval，比如设置为30分钟、60分钟、24小时等，根据需求而定。

## mybatis局限性

mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用mybatis的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为mybaits的二级缓存区域以mapper为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此类问题需要在业务层根据需求对数据有针对性缓存。

课13：mybatis和spring整合工程结构

## mybaits和spring整合的思路

1、让spring管理SqlSessionFactory

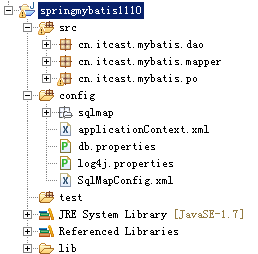
2、让spring管理mapper对象和dao。

使用spring和mybatis整合开发mapper代理及原始dao接口。

自动开启事务，自动关闭 sqlsession.

3、让spring管理数据源( 数据库连接池)

## 创建整合工程



## 加入jar包

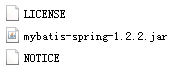
1、mybatis3.2.7本身的jar包

2、数据库驱动包

3、spring3.2.0

4、spring和mybatis整合包

从mybatis的官方下载spring和mybatis整合包



课14：一些问题总结

## log4j.properties

## SqlMapconfig.xml

mybatis配置文件：别名、settings，数据源不在这里配置

## applicationContext.xml

1、数据源（dbcp连接池）

2、SqlSessionFactory

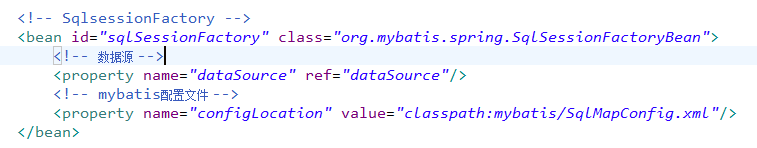
3、mapper或dao

课15：mybatis整合spring开发原始dao

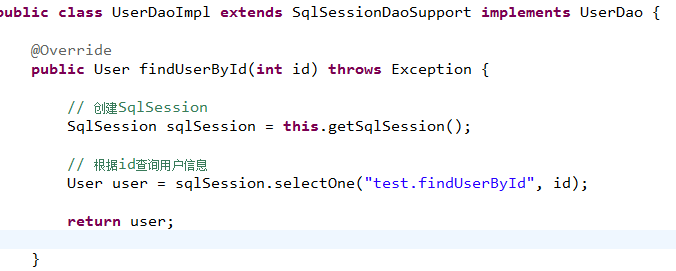
## 整合开发原始dao接口

### 配置SqlSessionFactory

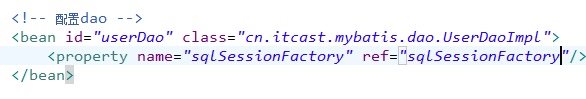
在 applicationContext.xml配置SqlSessionFactory



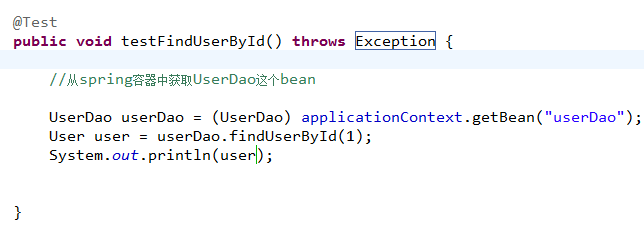
### 开发dao



### 配置 dao



### 测试dao接口



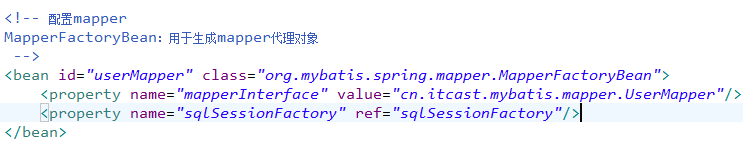
课16：mybatis整合spring开发mapper接口

## 整合开发mapper代理方法

### 开发mapper.xml和mapper.java



### 使用MapperFactoryBean



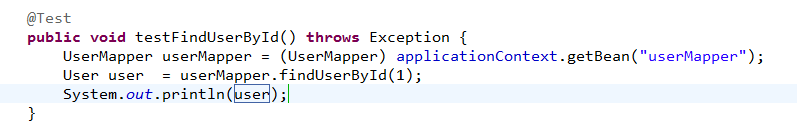
使用此方法对于每个mapper都需要配置，比较繁琐。

### 使用MapperScannerConfigurer（扫描mapper）



使用扫描器自动扫描mapper，生成代理对象，比较方便。

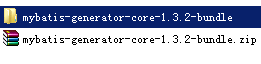
### 测试mapper接口



课17：逆向工程配置和生成代码

## 什么是mybatis的逆向工程

mybatis官方为了提高开发效率，提高自动对单表生成sql，包括 ：mapper.xml、mapper.java、表名.java(po类)



在企业开发中通常是在设计阶段对表进行设计 、创建。

在开发阶段根据表结构创建对应的po类。

mybatis逆向工程的方向：由数据库表----》java代码

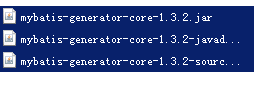
## 逆向工程 使用配置

运行逆向工程 方法：



本教程使用通过java程序运行逆向工程 。

逆向工程 运行所需要的jar包：

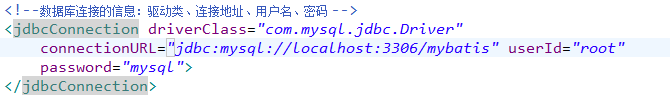


数据库驱动包

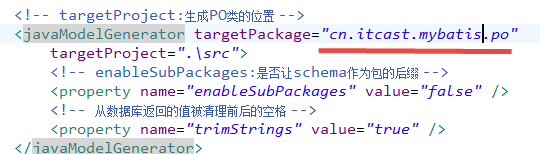
### xml配置

需要使用用配置的地方：

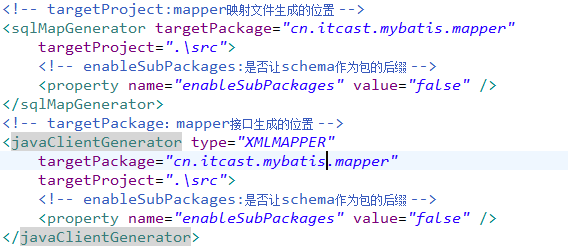
1、连接数据库的地址和驱动



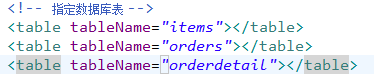
2、需要配置po类的包路径



3、需要配置mapper包的路径

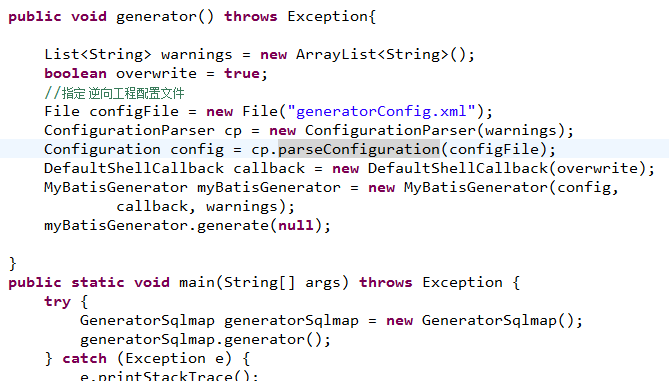


4、指定数据表



### ava程序

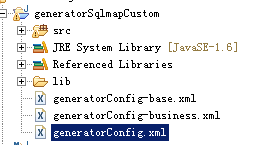
通过java程序生成mapper类、po类。。



## 使用逆向工程生成的代码

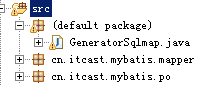
### 第一步配置generatorConfig.xml

参考：

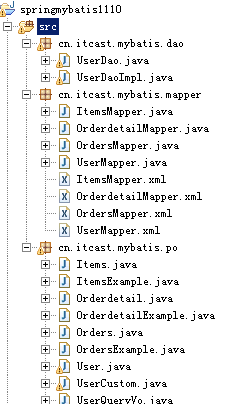


### 第二步配置执行java程序

执行java程序后，所生成的代码已经生成到工程中。



### 第三步将生成的代码拷贝到工程 中



课18：逆向工程代码测试

## 测试生成的代码

参考：



Day01:

课1：1mvc在bs系统中的应用课

## Spring web mvc介绍

Spring web mvc和Struts2都属于表现层的框架,它是Spring框架的一部分,我们可以从Spring的整体结构中看得出来：



## Web MVC

解释BS：

BS即Browser/Server(浏览器/服务器)结构，就是只安装维护一个服务器，而客户端采用浏览器运行软件。

mvc设计模式在b/s系统下应用：

1发起请求

C 控制器controller

作用：接收请求，响应处理结果，没有业务逻辑

5 响应结果

M 模型

pojo

service

dao

2请求业务处理

V 视图 View

将模型数据展示给用户

4 向view中填充模型数据

3返回模型

注意：b/s下模型无法将数据设置到视图中

1. 用户发起request请求至控制器(Controller)

控制接收用户请求的数据，委托给模型进行处理

1. 控制器通过模型(Model)处理数据并得到处理结果

模型通常是指业务逻辑

1. 模型处理结果返回给控制器
2. 控制器将模型数据在视图(View)中展示

web中模型无法将数据直接在视图上显示，需要通过控制器完成。如果在C/S应用中模型是可以将数据在视图中展示的。

1. 控制器将视图response响应给用户

通过视图展示给用户要的数据或处理结果。

课2：2springmvc架构

### 架构图



### 架构流程

1. 用户发送请求至前端控制器DispatcherServlet
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。
3. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。
4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器
5. 执行处理器(Controller，也叫后端控制器)。
6. Controller执行完成返回ModelAndView
7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器
9. ViewReslover解析后返回具体View
10. DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。
11. DispatcherServlet响应用户

### 组件说明

以下组件通常使用框架提供实现：

* DispatcherServlet：前端控制器

用户请求到达前端控制器，它就相当于mvc模式中的c，dispatcherServlet是整个流程控制的中心，由它调用其它组件处理用户的请求，dispatcherServlet的存在降低了组件之间的耦合性。

* HandlerMapping：处理器映射器

HandlerMapping负责根据用户请求找到Handler即处理器，springmvc提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

* Handler：处理器

Handler 是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。

由于Handler涉及到具体的用户业务请求，所以一般情况需要程序员根据业务需求开发Handler。

* HandlAdapter：处理器适配器

通过HandlerAdapter对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

* ViewResolver：视图解析器

ViewResolver负责将处理结果生成View视图，ViewResolver首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成View视图对象，最后对View进行渲染将处理结果通过页面展示给用户。 springmvc框架提供了很多的View视图类型，包括：jstlView、freemarkerView、pdfView等。

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由程序员根据业务需求开发具体的页面。

课3：3springmvc入门程序--工程结构

遇到问题：

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

网上查询是缺少jar包。

拷贝Tomcat8安装包，在eclipse中重新配置Tomcat，结果总是配置不上，起不了服务，后来大神跟我说你这是Tomcat8.5与jdk1.8不兼容，于是又灰头土脸拷了一份Tomcat8.0，配置好之后又出现HttpServletRequest类报错，当时没有明白，后来想想好简单，这个类其实是在服务jar包中的，说明服务的jar没有配置好，项目→右键→properties→java build bath→Libraries ，看看里面有没有Tomcat这个包，如果没有就add Library→ServerRuntime→对应的Tomcat版本，就是这么简单，真实大写的尴尬。

# 入门程序

## 需求

实现商品列表查询

## 需要的jar包

使用spring3.2.0（带springwebmvc模块）



## 前端控制器

在web.xml中配置：

<display-name>springfirst1110</display-name>

<!-- 前端控制器 -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 加载springmvc配置 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<!-- 配置文件的地址

如果不配置contextConfigLocation，

默认查找的配置文件名称classpath下的：servlet名称+"-serlvet.xml"即：springmvc-serlvet.xml

-->

<param-value>classpath:springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!--

可以配置/ ，此工程 所有请求全部由springmvc解析，此种方式可以实现 RESTful方式，需要特殊处理对静态文件的解析不能由springmvc解析

可以配置\*.do或\*.action，所有请求的url扩展名为.do或.action由springmvc解析，此种方法常用

不可以/\*，如果配置/\*，返回jsp也由springmvc解析，这是不对的。

-->

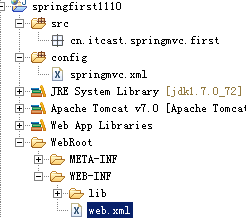
<url-pattern>\*.action</url-pattern>

</servlet-mapping>

课4：4springmvc入门程序-编写调试

## springmvc.xml

在springmvc.xml中配置springmvc架构三大组件（处理器映射器、适配器、视图解析器）



## 处理器映射器

在springmvc.xml中配置：

BeanNameUrlHandlerMapping： 根据请求url（XXXX.action）匹配spring容器bean的 name

找到对应的bean（程序编写的Handler）

<!-- 根据bean的name进行查找Handler 将action的url配置在bean的name中 -->

<bean

class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"* />

所有处理器映射器都实现HandlerMapping接口。

## 处理器适配器

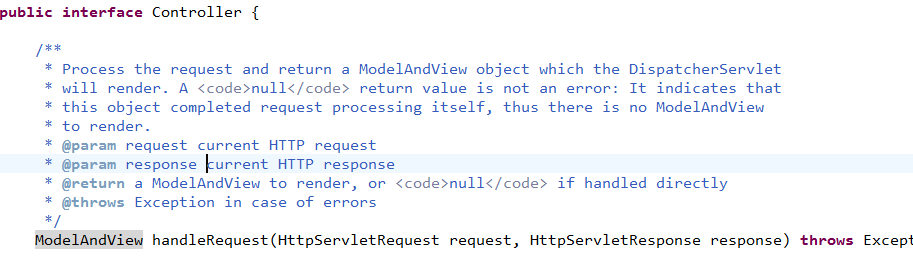
在springmvc.xml配置：

<bean class=*"org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"/*>

所有的适配器都是实现了HandlerAdapter接口。

## Handler编写

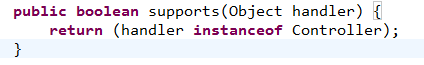
需要实现Controller接口：



程序编写Handler根据适配器的要求编写。

SimpleControllerHandlerAdapter适配器要求：

通过supports方法知道Handler必须要实现哪个接口：



**Handler代码如下：**

**public** **class** ItemController1 **implements** Controller {

@Override

**public** ModelAndView handleRequest(HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

// 使用静态数据将商品信息列表显示在jsp页面

// 商品列表

List<Items> itemsList = **new** ArrayList<Items>();

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setCreatetime(**new** Date());

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

ModelAndView modelAndView = **new** ModelAndView();

//将数据填充到request

// request.setAttribute("itemsList", itemsList);

modelAndView.addObject("itemsList", itemsList);

//指定转发的jsp页面

modelAndView.setViewName("/WEB-INF/jsp/itemsList.jsp");

**return** modelAndView;

}

}

## 配置Handler

在springmvc.xml配置Handler由spring管理Handler

<!-- 配置Handler 由于使用了BeanNameUrlHandlerMapping处理映射器，name配置为url -->

<bean id=*"itemController1"* name=*"/itemList.action"*

class=*"cn.itcast.springmvc.first.ItemController1"* />

## 配置视图解析器

配置视图解析，能够解析jsp视图：

<!-- 配置视图解析器 要求将jstl的包加到classpath -->

<!-- ViewResolver -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/jsp/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

## 工程部署

访问：http://localhost:8080/springfrist1110/itemList.action



## 异常

### HandlerMapping没有找到Handler

HandlerMapping没有找到Handler，404后边不显示jsp路径



### jsp页面地址错误



课5：非注解处理器映射器和适配器学习

## BeanNameUrlHandlerMapping(映射器)

根据请求url（XXXX.action）匹配spring容器bean的 name

找到对应的bean（程序编写的Handler）

## SimpleUrlHandlerMapping(映射器)

<!--简单url映射 集中配置bean的id对应 的url -->

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props>

<prop key=*"/itemsTest1.action"*>itemController1</prop>

<prop key=*"/itemsTest2.action"*>itemController2</prop>

</props>

</property>

</bean>

注意：在springmvc.xml配置了多个处理器映射器，多个处理器映射器可以共存。

## SimpleControllerHandlerAdapter(适配器)

要求程序编写的Handler(Controller)需要实现 Controller接口。

## HttpRequestHandlerAdapter（适配器）

在springmvc.xml配置：HttpRequestHandlerAdapter

要求Handler 实现 HttpRequestHandler接口

开发Handler

@Override

**public** **void** handleRequest(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

// 使用静态数据将商品信息列表显示在jsp页面

// 商品列表

List<Items> itemsList = **new** ArrayList<Items>();

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setCreatetime(**new** Date());

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

request.setAttribute("itemsList", itemsList);

//转发到jsp页面

request.getRequestDispatcher("/WEB-INF/jsp/itemsList.jsp").forward(request, response);

}

### 配置Handler

<!-- handler2 -->

<bean id=*"itemController2"* class=*"cn.itcast.springmvc.first.ItemController2"*></bean>

使用简单url映射器：

<!--简单url映射 集中配置bean的id对应 的url -->

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props>

<prop key=*"/itemsTest1.action"*>itemController1</prop>

<prop key=*"/itemsTest2.action"*>itemController2</prop>

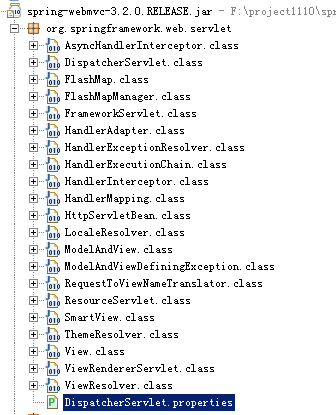
</props>

</property>

</bean>

课6：注解处理器映射器和适配器

# DispatcherServlet.properoties



DispatcherServlet前端控制器加载 DispatcherServlet.properoties 配置文件，从而默认加载各各组件，

如果在springmvc.xml中配置了处理器映射器和适配器，以sprintmvc.xml中配置的为准

遇见问题：Decompiler反编译插件安装不上？（fatak alert : protocol\_version）

## 注解映射器

spring3.1之前默认加载映射器是

org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping，3.1之后要使用：

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping

在springmvc.xml中配置RequestMappingHandlerMapping：

使用RequestMappingHandlerMapping需要在Handler 中使用@controller标识此类是一个控制器，使用@requestMapping指定Handler方法所对应的url。

## 注解适配器

spring3.1之前默认加载映射器是

org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter，3.1之后要使用：

org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter

RequestMappingHandlerAdapter，不要求Handler实现任何接口，它需要和RequestMappingHandlerMapping注解映射器配对使用，主要解析Handler方法中的形参。

## 注解开发Hanlder

@Controller

**public** **class** ItemController3 {

//商品列表，@RequestMapping中url建议和方法名一致，方便开发维护

@RequestMapping("/queryItems")

**public** ModelAndView queryItems(){

// 使用静态数据将商品信息列表显示在jsp页面

// 商品列表

List<Items> itemsList = **new** ArrayList<Items>();

Items items\_1 = **new** Items();

items\_1.setName("联想笔记本");

items\_1.setPrice(6000f);

items\_1.setCreatetime(**new** Date());

items\_1.setDetail("ThinkPad T430 联想笔记本电脑！");

Items items\_2 = **new** Items();

items\_2.setName("苹果手机");

items\_2.setPrice(5000f);

items\_2.setDetail("iphone6苹果手机！");

itemsList.add(items\_1);

itemsList.add(items\_2);

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.addObject("itemsList", itemsList);

//指定逻辑视图名

modelAndView.setViewName("itemsList");

**return** modelAndView;

}

## 配置Handler

<!-- 注解的handler，单个配置 -->

<!-- <bean class="cn.itcast.springmvc.first.ItemController3"/> -->

<!-- 使用spring组件扫描 -->

<context:component-scan base-package=*"cn.itcast.springmvc.first"* />

建议使用组件扫描，组件扫描可以扫描@Controller、@Service、@component、@Repsitory

课7：入门程序小结源码分析

## 小结

DispatcherServlet：前端控制器，相当于中央调度器，可以降低组件之间的耦合度。

HandlerMapping：处理器映射器,负责根据url查找Handler

HandlerAdapter：处理器适配器，负责根据适配器要求的规则去执行处理器。可以通过扩展适配器支持不同类型的Handler。

viewResolver：视图解析器，根据逻辑视图名解析成真正的视图，

<!-- 配置视图解析器 要求将jstl的包加到classpath -->

<!-- ViewResolver -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

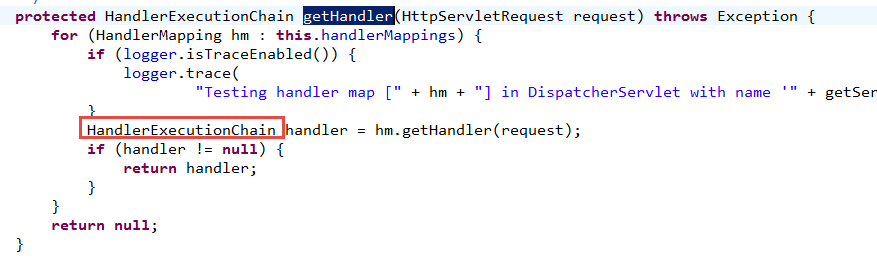
<property name=*"prefix"* value=*"/WEB-INF/jsp/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

真正视图地址==前缀+逻辑视图名+后缀

1、DispatcherServlet通过HandlerMapping查找Handler

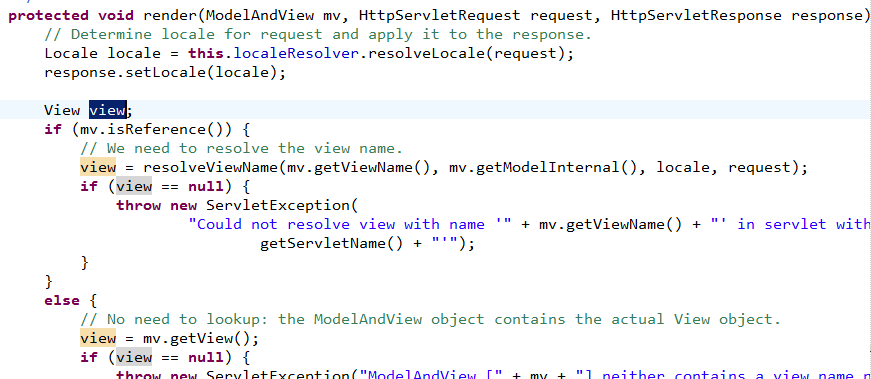


2、DispatcherServlet通过适配器去执行Handler，得到ModelAndview



3、视图解析

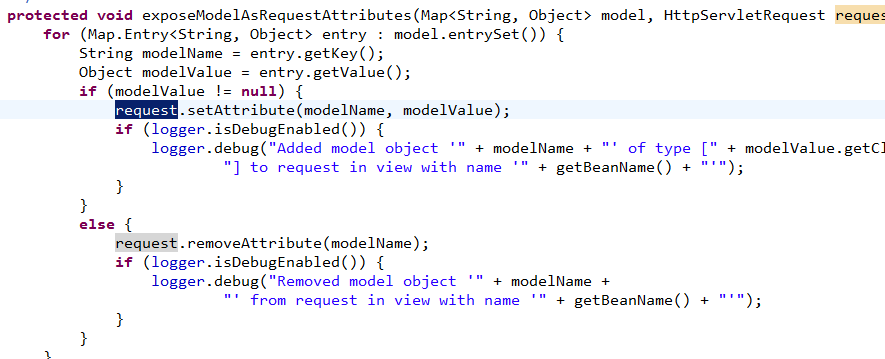
视图解析完成得到一个view：



4、进行视图渲染

将Model中的数据 填充到request域。





课8：springmvc和mybatis整合工程结构

## 整合思路

在mybatis和spring整合的基础上 添加springmvc。

spring要管理springmvc编写的Handler（controller）、mybatis的SqlSessionFactory、mapper

第一步：整合dao，spring和mybatis整合

第二步：整合service，spring管理service接口，service中可以调用spring容器中dao(mapper)

第三步：整合controller，spring管理controller接口，在controller调用service

## jar包

mybatis：3.2.7

spring：3.2.0

mybatis的jar

mybatis和spring整合包

spring的所有jar包(包括 springmvc的包)

数据库驱动包

log4j日志..



## 工程结构-配置文件

applicationContext-dao.xml---配置数据源、SqlSessionFactory、mapper扫描器

applicationContext-service.xml---配置service接口

applicationContext-transaction.xml--事务管理

sprintmvc.xml---springmvc的配置，配置处理器映射器、适配器、视图解析器

SqlMapConfig.xml---mybatis的配置文件，配置别名、settings、mapper

### applicationContext-dao.xml

配置mybatis的数据源、sqlSessionFactory、mapper扫描器

### applicationContext-transation.xml

在此配置文件配置事务，声明式事务控制。

<!-- 使用声明式事务配置，可以有效规范代码 -->

<!-- 事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

<property name=*"dataSource"* ref=*"dataSource"*/>

</bean>

<!-- 通知 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"save\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"insert\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"update\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"delete\*"* propagation=*"REQUIRED"*/>

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

<tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"*/>

</tx:attributes>

</tx:advice>

<!-- aop -->

<aop:config>

<aop:advisor advice-ref=*"txAdvice"*

pointcut=*"execution(\* cn.itcast.ssm.service.impl.\*.\*(..))"*/>

</aop:config>

## 前端控制器配置



## 配置springmvc.xml

<!-- 使用spring组件扫描 -->

<context:component-scan base-package=*"cn.itcast.ssm.controller"* />

<!-- 通过annotation-driven可以替代下边的处理器映射器和适配器 -->

<!-- <mvc:annotation-driven conversion-service="conversionService">

</mvc:annotation-driven>

-->

<!-- 注解处理器映射器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*>

</bean>

<!-- 注解适配器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<!-- 在webBindingInitializer中注入自定义属性编辑器、自定义转换器 -->

<property name=*"webBindingInitializer"* ref=*"customBinder"*></property>

</bean>

<!-- 配置视图解析器 要求将jstl的包加到classpath -->

<!-- ViewResolver -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/jsp/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

课9：springmvc和mybatis整合开发商品列表

## 商品列表开发

### 需求

查询商品列表

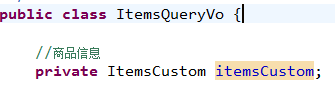
### mapper

功能描述：根据条件查询商品信息，返回商品列表

一般情况下针对查询mapper需要自定义mapper。

首先针对单表进行逆向工程，生成代码。

#### 包装类：



#### mapper.xml

<mapper namespace=*"cn.itcast.ssm.mapper.ItemsMapperCustom"*>

<!-- 商品查询的sql片段

建议是以单表为单位定义查询条件

建议将常用的查询条件都写出来

-->

<sql id=*"query\_items\_where"*>

<if test=*"itemsCustom!=null"*>

<if test=*"itemsCustom.name!=null and itemsCustom.name!=''"*>

and name like '%${itemsCustom.name}%'

</if>

<if test=*"itemsCustom.id!=null"*>

and id = #{itemsCustom.id}

</if>

</if>

</sql>

<!-- 商品查询

parameterType：输入 查询条件

-->

<select id=*"findItemsList"* parameterType=*"cn.itcast.ssm.po.ItemsQueryVo"*

resultType=*"cn.itcast.ssm.po.ItemsCustom"*>

SELECT \* FROM items

<where>

<include refid=*"query\_items\_where"*/>

</where>

</select>

#### **mapper.java**

### service

**public** **class** ItemsServiceImpl **implements** ItemsService {

//注入mapper

@Autowired

**private** ItemsMapperCustom itemsMapperCustom;

@Autowired

**private** ItemsMapper itemsMapper;

//商品查询列表

@Override

**public** List<ItemsCustom> findItemsList(ItemsQueryVo itemsQueryVo)

**throws** Exception {

**return** itemsMapperCustom.findItemsList(itemsQueryVo);

}

### 在applicationContext-service.xml中配置service

<!-- 商品管理 的service -->

<bean id=*"itemsService"* class=*"cn.itcast.ssm.service.impl.ItemsServiceImpl"*/>

</beans>

### controller

**public** **class** ItemsController {

//注入service

@Autowired

**private** ItemsService itemsService;

@RequestMapping("/queryItems")

**public** ModelAndView queryItems(HttpServletRequest request) **throws** Exception {

System.*out*.println(request.getParameter("id"));

//调用service查询商品列表

List<ItemsCustom> itemsList = itemsService.findItemsList(**null**);

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.addObject("itemsList", itemsList);

// 指定逻辑视图名

modelAndView.setViewName("itemsList");

**return** modelAndView;

}

**jsp**



### 在web.xml配置spring监听器

<!-- 配置spring容器监听器 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/classes/spring/applicationContext-\*.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

课10：springmvc和mybatis整合开发商品列表调试

课11：注解基础-商品修改功能service开发

## 商品修改

### 需求

功能描述：商品信息修改

操作流程：

1、在商品列表页面点击修改连接

2、打开商品修改页面，显示了当前商品的信息

根据商品id查询商品信息

3、修改商品信息，点击提交。

更新商品信息

### mapper

使用逆向工程生成代码：

根据商品id查询商品信息

更新商品信息



### service

//根据商品id查询商品信息

**public** ItemsCustom findItemsById(**int** id) **throws** Exception;

//更新商品信息

/\*\*

\* 定义service接口，遵循单一职责，将业务参数细化 （不要使用包装类型，比如map）

\* <p>Title: updateItems</p>

\* <p>Description: </p>

\* **@param** id 修改商品的id

\* **@param** itemsCustom 修改商品的信息

\* **@throws** Exception

\*/

**public** **void** updateItems(Integer id,ItemsCustom itemsCustom)**throws** Exception;

课12：RequestMapping特性学习

## @RequestMapping

### 设置方法对应的url（完成url映射）

一个方法对应一个url

@RequestMapping("/queryItems")

**public** ModelAndView queryItems(HttpServletRequest request) **throws** Exception {

### 窄化请求映射

在class上定义根路径

@Controller

//定义url的根路径，访问时根路径+方法的url

@RequestMapping("/items")

**public** **class** ItemsController {

好处：更好的规范系统 的url，避免 url冲突。

### 限制http请求的方法

通过requestMapping限制url请求的http方法，

如果限制请求必须是post，如果get请求就抛出异常：



商品修改方法，限制为http的get：



//方法返回 字符串，字符串就是逻辑视图名，Model作用是将数据填充到request域，在页面展示

@RequestMapping(value="/editItems",method={RequestMethod.*GET*})

**public** String editItems(Model model,Integer id)**throws** Exception{

//调用 service查询商品信息

ItemsCustom itemsCustom = itemsService.findItemsById(id);

model.addAttribute("item", itemsCustom);

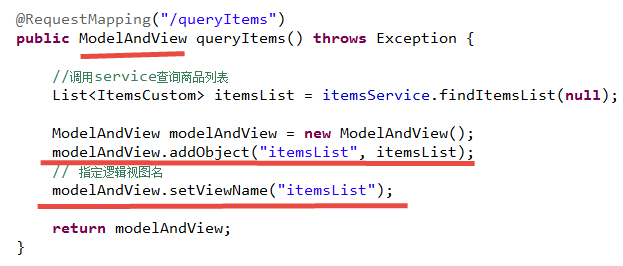
//return "editItem\_2";

**return** "editItem";

}

课13：controller方法返回值

### 返回ModelAndView



### 返回字符串

如果controller方法返回jsp页面，可以简单将方法返回值类型定义 为字符串，最终返回逻辑视图名。



### 返回void



使用此方法，容易输出json、xml格式的数据：

通过response指定响应结果，例如响应json数据如下：

response.setCharacterEncoding("utf-8");

response.setContentType("application/json;charset=utf-8");

response.getWriter().write("json串");

### redirect重定向

如果方法重定向到另一个urk，方法返回值为“redirect:url路径”

使用redirect进行重定向，request数据无法共享，url地址栏会发生变化的。

### forward转发

使用forward进行请求转发，request数据可以共享，url地址栏不会。

方法返回值为“forward:url路径”

//商品修改提交

//itemsQueryVo是包装类型的pojo

@RequestMapping("/editItemSubmit")

// public String editItemSubmit(Integer id,ItemsCustom itemsCustom,

// ItemsQueryVo itemsQueryVo)throws Exception{

**public** String editItemSubmit(Integer id,ItemsCustom itemsCustom)**throws** Exception{

//调用service接口更新商品信息

itemsService.updateItems(id, itemsCustom);

//请求重定向

**return** "redirect:queryItems.action";

//转发

// return "forward:queryItems.action";

}

课14：参数绑定过程

### 参数绑定过程

http请求

HanlderAdapter(适配器)

Handler

以方法为单位进行编写

方法形参

key、value数据

早期springmvc是使用

PropertyEditor属性编辑器进行参数绑定(仅支持由字符串传为其它类型)

后期springmvc是使用

converter转换器进行参数绑定（支持任意类型转换）

将key/value数据绑定到方法形参

### 默认支持的参数类型

处理器形参中添加如下类型的参数处理适配器会默认识别并进行赋值。

### HttpServletRequest

通过request对象获取请求信息

### HttpServletResponse

通过response处理响应信息

### HttpSession

通过session对象得到session中存放的对象

### Model

通过model向页面传递数据，如下：

//调用service查询商品信息

Items item = itemService.findItemById(id);

model.addAttribute("item", item);

页面通过${item.XXXX}获取item对象的属性值。

课15：RequestParam注解

### @RequestParam

如果request请求的参数名和controller方法的形参数名称一致，适配器自动进行参数绑定。如果不一致可以通过

@RequestParam 指定request请求的参数名绑定到哪个方法形参上。

对于必须要传的参数，通过@RequestParam中属性required设置为true，如果不传此参数则报错。

对于有些参数如果不传入，还需要设置默认值，使用@RequestParam中属性defaultvalue设置默认值。

课16：简单类型和pojo类型绑定

### 可以绑定简单类型

可以绑定整型、 字符串、单精/双精度、日期、布尔型。

### 可以绑定简单pojo类型

简单pojo类型只包括简单类型的属性。

绑定过程：

request请求的参数名称和pojo的属性名一致，就可以绑定成功。

问题：

如果controller方法形参中有多个pojo且pojo中有重复的属性，使用简单pojo绑定无法有针对性的绑定，

比如：方法形参有items和User，pojo同时存在name属性，从http请求过程的name无法有针对性的绑定到items或user。

### 可以绑定包装的pojo

包装的pojo里边包括了pojo。

页面参数定义：

<tr>

<td>商品名称</td>

<td><input type=*"text"* name=*"itemsCustom.name"* value=*"*${item.name }*"*/></td>

</tr>

<tr>

<td>商品价格</td>

<td><input type=*"text"* name=*"itemsCustom.price"* value=*"*${item.price }*"*/></td>

</tr>

<tr>

<td>商品简介</td>

<td>

<textarea rows=*"3"* cols=*"30"* name=*"itemsCustom.detail"*>${item.detail }</textarea>

</td>

</tr>

包装类型的属性也是itemsCustom：

**public** **class** ItemsQueryVo {

//商品信息

**private** ItemsCustom itemsCustom;

按照上边的规则进行包装类型的绑定。

### POST乱码处理：

<!-- post乱码处理 -->

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>utf-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

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

</filter-mapping>

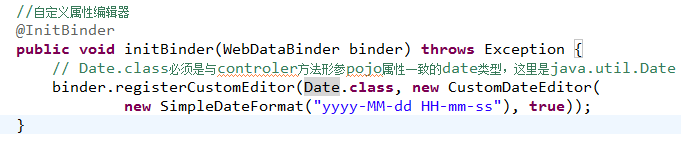
课17：自定义参数绑定-属性编辑器

### 自定义绑定使用属性编辑器

springmvc没有提供默认的对日期类型的绑定，需要自定义日期类型的绑定。

#### 使用WebDataBinder（了解）

在controller类中定义：



使用这种方法问题是无法在多个controller共用。

#### 使用WebBindingInitializer（了解）

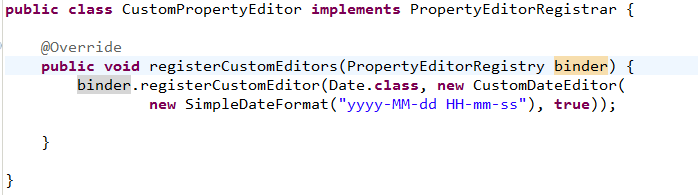
使用WebBindingInitializer让多个controller共用 属性编辑器。

自定义WebBindingInitializer，注入到处理器适配器中。

如果想多个controller需要共同注册相同的属性编辑器，可以实现PropertyEditorRegistrar接口，并注入webBindingInitializer中。

如下：

编写CustomPropertyEditor：

配置如下：

<!-- 注册属性编辑器 -->

<bean id=*"customPropertyEditor"* class=*"cn.itcast.ssm.propertyeditor.CustomPropertyEditor"*></bean>

<!-- 自定义webBinder -->

<bean id=*"customBinder"*

class=*"org.springframework.web.bind.support.ConfigurableWebBindingInitializer"*>

<property name=*"propertyEditorRegistrars"*>

<list>

<ref bean=*"customPropertyEditor"*/>

</list>

</property>

</bean>

<!--注解适配器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<property name=*"**webBindingInitializer"* ref=*"customBinder"*></property>

</bean>

课18：自定义参数绑定-自定义转换器：

#### 实现Converter接口：

定义日期类型转换器和字符串去除前后空格转换器。



**public** **class** StringTrimConverter **implements** Converter<String, String> {

@Override

**public** String convert(String source) {

**try** {

//去掉字符串两边空格，如果去除后为空设置为null

**if**(source!=**null**){

source = source.trim();

**if**(source.equals("")){

**return** **null**;

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

**return** source;

}

}

#### 配置转换器

参考教案：

配置方式1针对不使用<mvc:annotation-driven>

配置方式2针对使用<mvc:annotation-driven>的配置

Day02:

课1：1数据回显

# 复习

springmvc框架：

用户请求url到DispatcherServlet前端控制器，相当于中央调度器，降低系统各组件之间耦合度

DispatcherServlet前端控制器通过HandlerMapping根据url找到Handler。

DispatcherServlet前端控制器通过HandlerAdapter处理器适配器执行Handler。

DispatcherServlet前端控制器拿着Handler返回的ModelAndView通过视图解析器ViewResolver去进行视图解析。

视图解析：将程序中写的逻辑视图名，转成真正的视图（springmvc通过view表示各各不同类型的视图）。

DispatcherServlet前端控制器调用View的渲染方法进行视图渲染（将ModelAndView中的Model放到request域）。

要掌握springmvc的注解开发，企业中常用springmvc注解开发。

使用专门注解处理器映射器（RequestMappingHandlerMapping）和处理器适配器（RequestMappingHandlerAdapter）。

<mvc:annotation-driven/>可以代替上边的处理器映射器和适配器的配置。

在Handler(Controller)中定义很多的方法，一个方法通过RequestMapping和url进行映射。

方法返回值：ModelAndView、string(jsp的逻辑视图名)、void（通过response将数据输出成json）

方法输入参数（形参）：springmvc需要将请求的key/value(串，id=001&type=t002)、解析、绑定到Handler(Controller)中方法的形参上。

springmvc默认支持多类型的参数绑定。

默认支持哪些类型：

HttpServletRequest、response、session、Model(用于将数据填充到request域)

@requestParam注解：用于绑定单个请求参数，常用于简单类型参数（Integer、String 、Float。。。）绑定。

不用 @requestParam要求请求参数的名称和方法形参名一致方可绑定。

对于简单类型参数中的日期型，建议使用自定义参数绑定，对日期型数据个化定义日期的格式。

自定义参数绑定：建议使用Convertor进行参数绑定。

还可以绑定pojo、包装的pojo。

# 课程安排

注解开发：

数据回显：表单提交出现错误，重新回到表单，用户重新填写数据，刚才提交的参数在页面上回显。

集合类型（String[]、List<>、map(自学)）的参数绑定。

springmvc上传图片（重点）

json数据交互（提交json数据、响应json数据）（重点）

**Validation(springmvc使用校验方式，使用Hibernate Validator（和Hibernate的ORM没有任何关系）)**

**异常处理器（可以用于系统的统一异常处理，架构的内容）**

springmvc提供RESTful支持

拦截器（用于权限控制）

学生练习

# 数据回显

## 需求

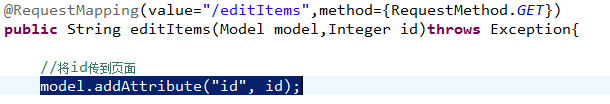
表单提交出现错误，重新回到表单，用户重新填写数据，刚才提交的参数在页面上回显。

## 对简单类型的数据回显

对商品修改数据回显：

注意在进入修改页面的controller方法中和提交修改商品信息方法model.addAttribute方法设置的key一致。

修改商品显示方法：



修改商品页面：

<body>

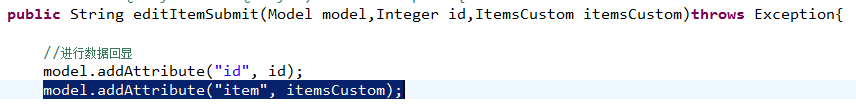
<form id=*"itemForm"* action=*"*${pageContext.request.contextPath }*/items/editItemSubmit.action"* method=*"post"* >

<input type=*"hidden"* name=*"id"* value=*"*${item.id }*"*/>

## pojo类型数据回显

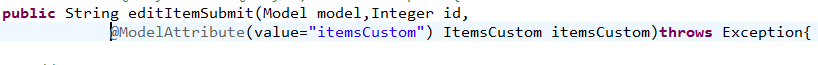
### 方法1：

使用Model.addtribute方法进行数据回显：



### 方法2：

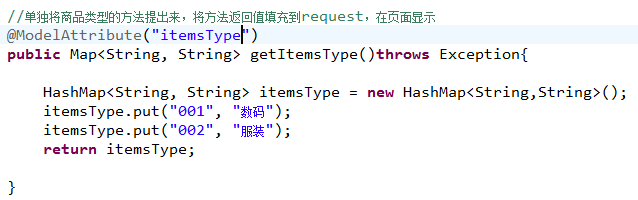
使用@ModelAttribute，作用于将请求pojo数据放到Model中回显到页面



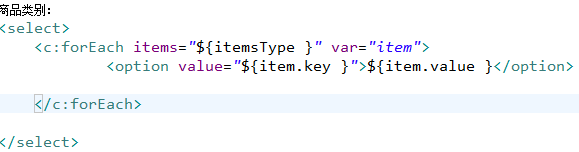
在ModelAttribute方法指定的名称就是要填充Model中的key，在页面中就要通过key取数据。

### @ModelAttribute将方法返回值传到页面

需求：商品类别信息在商品信息页面显示。



页面：



使用@ModelAttribute将公用的取数据的方法返回值传到页面，不用在每一个controller方法通过Model将数据传到页面。

课2：参数绑定-绑定list数据

## 绑定数组

需求：在商品查询列表页面，用户选择要删除的商品，批量删除商品。

在controller方法中如何将批量提交的数据绑定成数组类型。

### 页面定义

<c:forEach items=*"*${itemsList }*"* var=*"item"*>

<tr>

<td><input type=*"checkbox"* name=*"delete\_id"* value=*"*${item.id}*"* /></td>

### controller方法定义

//删除 商品

@RequestMapping("/deleteItems")

**public** String deleteItems(Integer[] delete\_id)**throws** Exception{

//调用service方法删除 商品

//....

**return** "success";

}

<td><input type=*"submit"* value=*"查询"*/>

<input type=*"button"* value=*"批量删除"* onclick="deleteItems()"/>

</td>

<script type=*"text/javascript"*>

**function** deleteItems(){

//将form的action指向删除商品的地址

document.itemsForm.action="${pageContext.request.contextPath }/items/deleteItems.action";

//进行form提交

document.itemsForm.submit();

}

</script>

## 绑定List<Object>

需求：批量修改商品信息提交。

先进入批量修改商品页面，填写信息，点击提交。

### 页面定义

<tr>

<td><input type=*"text"* name=*"itemsList[*${s.index }*].name"* value=*"*${item.name }*"*/></td>

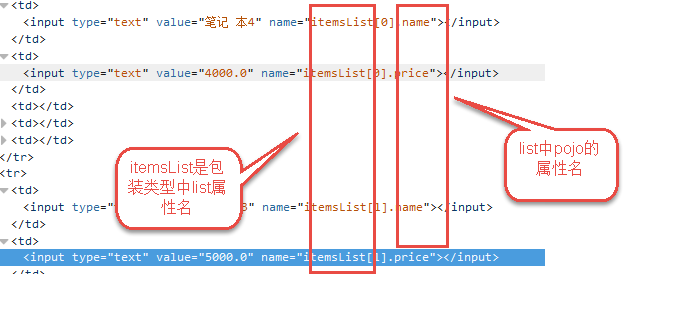
<td><input type=*"text"* name=*"itemsList[*${s.index }*].price"* value=*"*${item.price }*"*/></td>

注释：

itemsList：controller方法形参包装类型中list的属性名。

itemsList[0]或itemsList[1]。。，[]中是序号，从0开始。

itemsList[].name：name就是controller方法形参包装类型中list中pojo的属性名



### controller方法定义

使用包装类型接收页面批量提交的数据，绑定成List。

**public** **class** ItemsQueryVo {

//商品信息

**private** ItemsCustom itemsCustom;

//定义一个list

**private** List<ItemsCustom> itemsList;

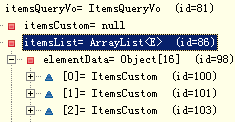
//批量修改商品提交

@RequestMapping("/editItemsListSubmit")

**public** String editItemsListSubmit(ItemsQueryVo itemsQueryVo)**throws** Exception{

**return** "success";

}



#### 绑定Map

在包装类中定义Map对象，并添加get/set方法，action使用包装对象接收。

包装类中定义Map对象如下：

**Public class** QueryVo {

private Map<String, Object> itemInfo = new HashMap<String, Object>();

//get/set方法..

}

页面定义如下：

<tr>

<td>学生信息：</td>

<td>

姓名：<inputtype=*"text"*name=*"itemInfo['name']"*/>

年龄：<inputtype=*"text"*name=*"itemInfo['price']"*/>

.. .. ..

</td>

</tr>

Contrller方法定义如下：

public String useraddsubmit(Model model,QueryVo queryVo)throws Exception{

System.out.println(queryVo.getStudentinfo());

}

课3：springmvc和struts的区别

springmvc是通过方法的形参接收参数，在使用时可以以单例方式使用，建议使用单例。

struts是通过成员变量接收参数，在使用时必须以多例方式使用。

springmvc是基于方法开发，struts基于类开发。

springmvc将一个请求的Method和Handler进行关联绑定，一个method对应一个Handler。

springmvc开发以方法为单位进行开发，方法更帖进service(业务方法)。

经过实际测试，发现struts标签解析速度比较慢，建议在实际开发时使用jstl。

课4：springmvc图片上传

### 需求

在商品修改页面，增加图片上传的功能。

操作流程：

用户进入商品修改页面

上传图片

点击提交（提交的是图片和商品信息）

再次进入修改页面，图片在商品修改页面展示

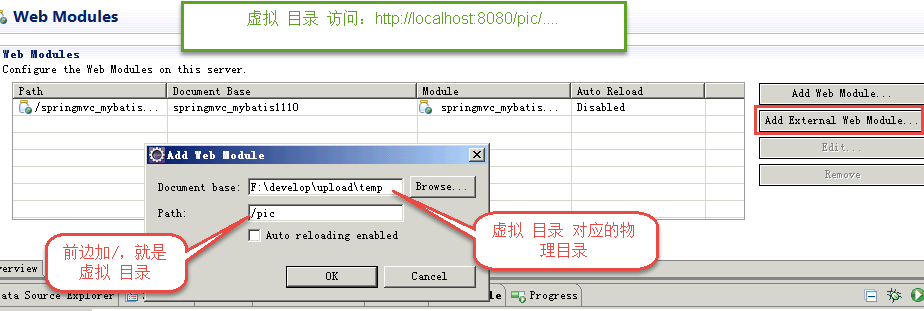
### 图片存储问题

切记：不要把图片上传到工程 目录 ，不方便进行工程 维护。

实际电商项目中使用专门图片服务器(http，比如apache、tomcat)。

本教程使用图片虚拟目录，通过虚拟目录 访问硬盘上存储的图片目录 。

虚拟 目录 设置：



注意：

图片目录中尽量进行目录分级存储，提高访问速度（提交i/o）。

## 配置图片上传解析器

springmvc使用commons-fileupload进行图片上传。

commons-fileupload对应的springmvc的图片上传解析器：

*org.springframework.web.multipart.commons.CommonsMultipartResolver*

<!-- 文件上传 -->

<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 设置上传文件的最大尺寸为5MB -->

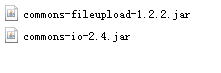
<property name=*"maxUploadSize"*>

<value>5242880</value>

</property>

</bean>

加入commons-fileupload的jar包



## 编写上传图片的页面

<td>商品图片</td>

<td>

<c:if test=*"*${itemsCustom.pic !=null}*"*>

<img src=*"/pic/*${itemsCustom.pic}*"* width=*100* height=*100*/>

<br/>

</c:if>

<input type=*"file"* name=*"pictureFile"*/>

</td>

## 编写controller方法

**public** String editItemSubmit(Model model,Integer id,

@Validated(value={ValidGroup1.**class**}) @ModelAttribute(value="itemsCustom") ItemsCustom itemsCustom,

BindingResult bindingResult,

//上传图片

MultipartFile pictureFile

)**throws** Exception{

//输出校验错误信息

//如果参数绑定时有错

**if**(bindingResult.hasErrors()){

//获取错误

List<ObjectError> errors = bindingResult.getAllErrors();

//准备在页面输出errors，页面使用jstl遍历

model.addAttribute("errors", errors);

**for**(ObjectError error:errors){

//输出错误信息

System.out.println(error.getDefaultMessage());

}

//如果校验错误，回到商品修改页面

**return** "editItem";

}

//进行数据回显

model.addAttribute("id", id);

//model.addAttribute("item", itemsCustom);

//进行图片上传

**if**(pictureFile!=**null** && pictureFile.getOriginalFilename()!=**null** && pictureFile.getOriginalFilename().length()>0){

//图片上传成功后，将图片的地址写到数据库

String filePath = "F:\\develop\\upload\\temp\\";

//上传文件原始名称

String originalFilename = pictureFile.getOriginalFilename();

//新的图片名称

String newFileName = UUID.randomUUID() +originalFilename.substring(originalFilename.lastIndexOf("."));

//新文件

File file = **new** java.io.File(filePath+newFileName);

//将内存中的文件写入磁盘

pictureFile.transferTo(file);

//图片上传成功，将新图片地址写入数据库

itemsCustom.setPic(newFileName);

}

课5：requestBody和responseBody

# json数据的交互

## 需求

json数据格式是比较简单容易理解，json数据格式常用于远程接口传输，http传输json数据，非常方便页面进行提交/请求结果解析，对json数据的解析。

## springmvc解析json加入json解析包

Springmvc默认用MappingJacksonHttpMessageConverter对json数据进行转换，需要加入jackson的包，如下：



## 在处理器适配器中注入MappingJacksonHttpMessageConverter

让处理器适配器支持json数据解析，需要注入MappingJacksonHttpMessageConverter。

<!-- 注解适配器 -->

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<!-- 在webBindingInitializer中注入自定义属性编辑器、自定义转换器 -->

<property name=*"webBindingInitializer"* ref=*"customBinder"*></property>

<!-- 加入 json数据的消息转换器 MappingJacksonHttpMessageConverter依赖Jackson的包 -->

<property name=*"messageConverters"*>

<list>

<bean

class=*"org.springframework.http.converter.json.MappingJacksonHttpMessageConverter"*></bean>

</list>

</property>

</bean>

## @RequestBody和@ResponseBody

@RequestBody：将请求的json数据转成java对象

@ResponseBody：将java对象转成json数据输出。

http请求数据是json格式（json串）

contentType=appliation/json;charset=utf-8

@ResponseBody方法返回值 controller方法形参（@RequestBody pojo对象）

使用@RequestBody将json转成java对象(pojo)

@ResponseBody可以将java对象转成json输出

如果请求的是key/value就不需要使用@RequestBody

## 请求json响应json

controller方法：

@RequestMapping("/requestJson")

**public** @ResponseBody ItemsCustom requestJson(@RequestBody ItemsCustom itemsCustom)**throws** Exception{

**return** itemsCustom;

}

页面：

<head>

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

<title>json测试</title>

<script type=*"text/javascript"* src=*"*${pageContext.request.contextPath }*/js/jquery-1.4.4.min.js"*></script>

<script type=*"text/javascript"*>

//请求json响应json

**function** requestJson(){

$.ajax({

url:"${pageContext.request.contextPath }/requestJson.action",

type:"post",

contentType:"application/json;charset=utf-8",

//请求json数据,使用json表示商品信息

data:'{"name":"手机","price":1999}',

success:**function**(data){

alert(data.name);

}

});

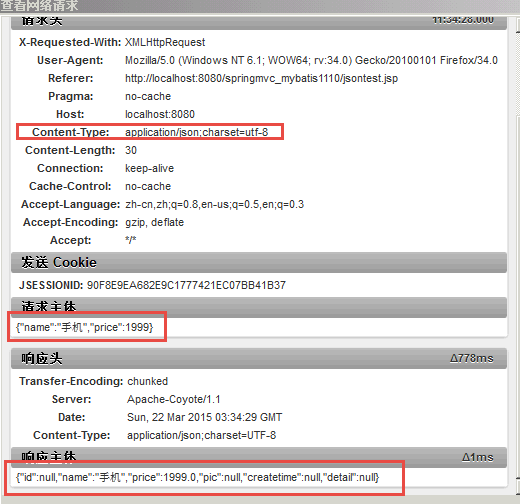
}

<body>

<input type=*"button"* value=*"请求json响应json"* onclick="requestJson()"/>

<input type=*"button"* value=*"请求key/value响应json"* onclick="responseJson()"/>

</body>



## 请求key/value响应json

controller方法：

//请求key/value响应json

@RequestMapping("/responseJson")

**public** @ResponseBody ItemsCustom responseJson(ItemsCustom itemsCustom)**throws** Exception{

**return** itemsCustom;

}

页面：见上面的页面内容。



## 小结

如果前端处理没有特殊要求建议使用第二种，请求key/value，响应json，方便客户端解析请求结果 。

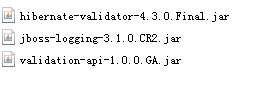
课6：validation基本使用

### validation校验（了解）

对前端的校验大多数通过js在页面校验，这种方法比较简单，如果对安全性考虑，还要在后台校验。

springmvc使用JSR-303（javaEE6规范的一部分）校验规范，springmvc使用的是Hibernate Validator（和Hibernate的ORM）

## 加入Hibernate Validator的jar



## 在处理器适配器中配置校验器

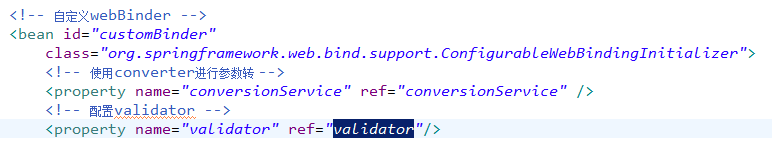
<!-- 注解适配器 -->

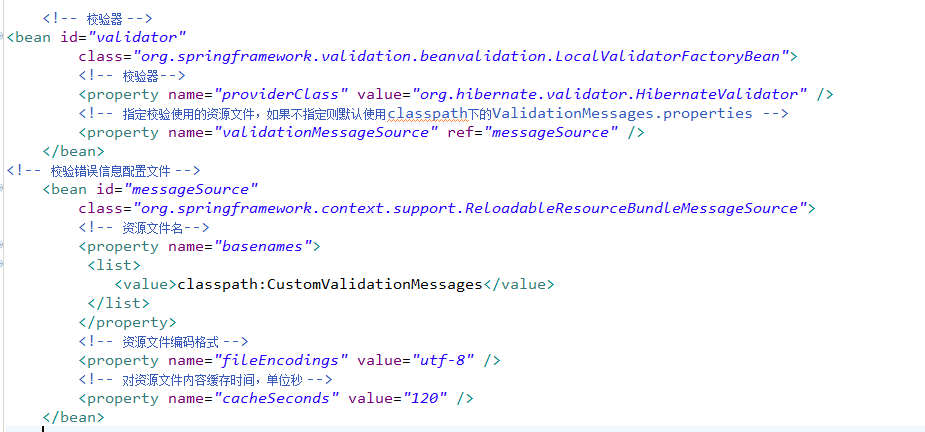
<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<!-- 在webBindingInitializer中注入自定义属性编辑器、自定义转换器 -->

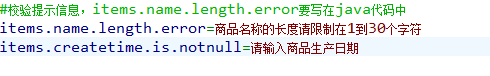
<property name=*"webBindingInitializer"* ref=*"customBinder"*></property>





## 创建CustomValidationMessages

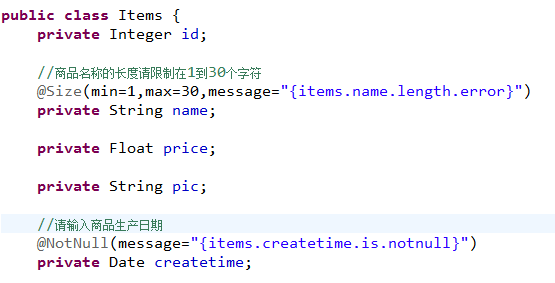
在classpath下创建CustomValidationMessages.properties



## 校验规则

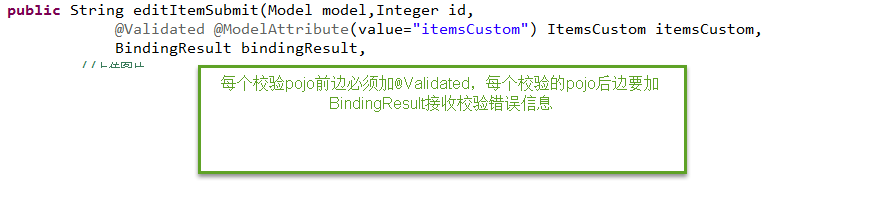
需求：

商品信息提交时校验 ，商品生产日期不能为空，商品名称长度在1到30字符之间

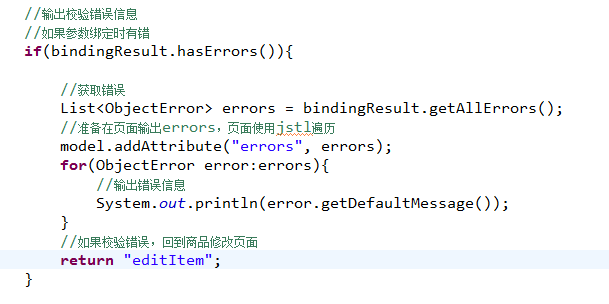


## 捕获错误

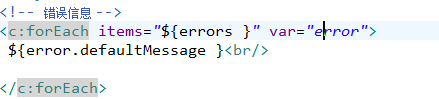
需要修改controller方法，在要校验的pojo前边加上@Validated，



错误信息输出：



## 在页面上展示错误



课7：validation分组校验

## 分组校验

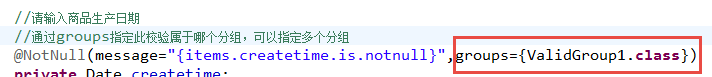
需求：

针对不同的controller方法通过分组校验达到个性化校验的目的，修改商品修改功能，只校验生产日期不能为空。

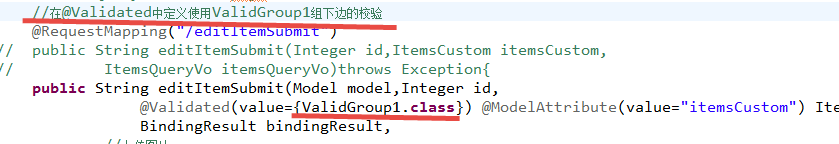
第一步：创建分组接口



第二步：定义校验规则属于哪个分组



第三步：在controller方法定义使用校验的分组



课8：springmvc统一异常处理&异常处理图解

## 需求

一般项目中都需要作异常处理，基于系统架构的设计考虑，使用统一的异常处理方法。

系统中异常类型有哪些？

包括预期可能发生的异常、运行时异常（RuntimeException），运行时异常不是预期会发生的。

针对预期可能发生的异常，在代码手动处理异常可以try/catch捕获，可以向上抛出。

针对运行时异常，只能通过规范代码质量、在系统测试时详细测试等排除运行时异常。

## 统一异常处理解决方案

### 定义异常

针对预期可能发生的异常，定义很多异常类型，这些异常类型通常继承于Exception。

这里定义一个系统自定义异常类：

CustomException，用于测试。

**public** **class** CustomException **extends** Exception {

//异常信息

**private** String message;

**public** CustomException(String message){

**super**(message);

**this**.message = message;

}

**public** String getMessage() {

**return** message;

}

**public** **void** setMessage(String message) {

**this**.message = message;

}

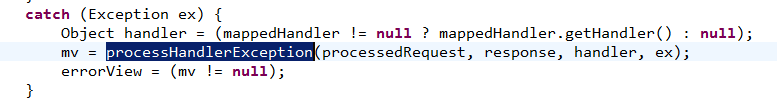
}

### 异常处理

要在一个统一异常处理的类中要处理系统抛出的所有异常，根据异常类型来处理。

统一异常处理的类是什么？

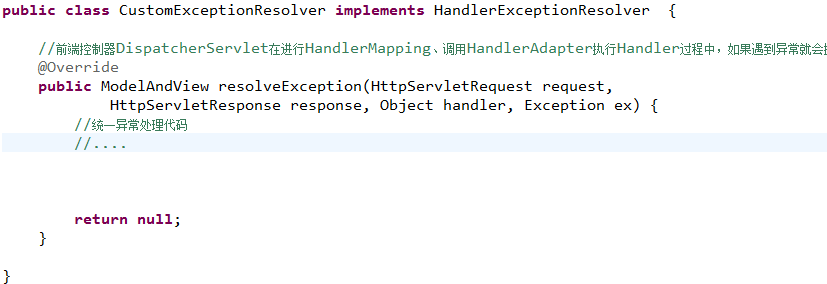
前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常，进行异常处理。



在系统中自定义统一的异常处理器，写系统自己的异常处理代码。

#### 定义统一异常处理器类

统一异常处理器实现HandlerExceptionResolver接口。



#### 配置统一异常处理器

<!-- 定义统一异常处理器 -->

<bean class=*"cn.itcast.ssm.exception.CustomExceptionResolver"*></bean>

#### 异常处理逻辑

根据不同的异常类型进行异常处理。

系统自定义的异常类是CustomException ，在controller方法中、service方法中手动抛出此类异常。

针对系统自定义的CustomException异常，就可以直接从异常类中获取异常信息，将异常处理在错误页面展示。

针对非CustomException异常，对这类重新构造成一个CustomException，异常信息为“未知错误”，此类错误需要在系统测试阶段去排除。

在统一异常处理器CustomExceptionResolver中实现上边的逻辑。

**public** **class** CustomExceptionResolver **implements** HandlerExceptionResolver {

//前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常就会执行此方法

//handler最终要执行的Handler，它的真实身份是HandlerMethod

//Exception ex就是接收到异常信息

@Override

**public** ModelAndView resolveException(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex) {

//输出异常

ex.printStackTrace();

//统一异常处理代码

//针对系统自定义的CustomException异常，就可以直接从异常类中获取异常信息，将异常处理在错误页面展示

//异常信息

String message = **null**;

CustomException customException = **null**;

//如果ex是系统 自定义的异常，直接取出异常信息

**if**(ex **instanceof** CustomException){

customException = (CustomException)ex;

}**else**{

//针对非CustomException异常，对这类重新构造成一个CustomException，异常信息为“未知错误”

customException = **new** CustomException("未知错误");

}

//错误 信息

message = customException.getMessage();

request.setAttribute("message", message);

**try** {

//转向到错误 页面

request.getRequestDispatcher("/WEB-INF/jsp/error.jsp").forward(request, response);

} **catch** (ServletException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **new** ModelAndView();

}

}

#### 测试抛出异常由统一异常处理器捕获

可以在controller方法、service方法、dao实现类中抛出异常，要求dao、service、controller遇到异常全部向上抛出异常，方法向 上抛出异常throws Exception

@Override

**public** ItemsCustom findItemsById(**int** id) **throws** Exception {

Items items = itemsMapper.selectByPrimaryKey(id);

//如果查询的商品信息为空，抛出系统 自定义的异常

**if**(items==**null**){

**throw** **new** CustomException("修改商品信息不存在");

}

#### 图解：

http请求

DispatcherServlet

Handler(Controller)

service

mapper

左边的箭头就是请求处理的过程

有异常向上抛出

有异常向上抛出

有异常向上抛出

DispatcherServlet统一try/catch捕获异常调用异常处理器（全系统只有一个）

异常处理器需要实现HandlerExceptionResolver接口

要求dao、service、controller遇到异常全部向上抛出异常，方法向 上抛出异常throws Exception

课9：springmvc对RESTful的支持

## 什么是RESTful

RESTful软件开发理念，RESTful对http进行非常好的诠释。

RESTful即Representational State Transfer的缩写。

综合上面的解释，我们总结一下什么是RESTful架构：

　　（1）每一个URI（统一资源定位符）代表一种资源；

　　（2）客户端和服务器之间，传递这种资源的某种表现层；

　　（3）客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"。

## url的RESTful实现

非RESTful的http的url：http://localhost:8080/items/editItems.action?id=1&....

RESTful的url是简洁的：http:// localhost:8080/items/editItems/1

参数通过url传递，rest接口返回json数据

### 需求

根据id查看商品信息，商品信息查看的连接使用RESTful方式实现，商品信息以json返回。

### 第一步更改DispatcherServlet配置

<!-- restful的配置 -->

<servlet>

<servlet-name>springmvc\_rest</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 加载springmvc配置 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<!-- 配置文件的地址 如果不配置contextConfigLocation， 默认查找的配置文件名称classpath下的：servlet名称+"-serlvet.xml"即：springmvc-serlvet.xml -->

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc\_rest</servlet-name>

<!-- rest方式配置为/ -->

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

</servlet-mapping>

### 第二步参数通过url传递

//根据商品id查看商品信息rest接口

//@RequestMapping中指定restful方式的url中的参数，参数需要用{}包起来

//@PathVariable将url中的{}包起参数和形参进行绑定

@RequestMapping("/viewItems/{id}")

**public** @ResponseBody ItemsCustom viewItems(@PathVariable("id") Integer id) **throws** Exception{

//调用 service查询商品信息

ItemsCustom itemsCustom = itemsService.findItemsById(id);

**return** itemsCustom;

}

### 设置静态资源解析

当DispatcherServlet拦截/开头的所有请求，对静态资源的访问就报错：



需要通过设置对静态资源进行解析.

<!-- 静态资源 解析 -->

<mvc:resources location=*"/js/"* mapping=*"/js/\*\*"* />

<mvc:resources location=*"/img/"* mapping=*"/img/\*\*"* />

访问/js/\*\*的url从工程下/js/下解析。

课10：springmvc拦截器测试

## 拦截器的异常场合

用户请求到DispatherServlet中，DispatherServlet调用HandlerMapping查找Handler，HandlerMapping返回一个拦截的链儿（多个拦截），springmvc中的拦截器是通过HandlerMapping发起的。

在企业开发，使用拦截器实现用户认证（用户登陆后进行身份校验拦截），用户权限拦截。

## springmvc拦截器方法

**public** **class** HandlerInterceptor1 **implements** HandlerInterceptor {

//在执行handler之前来执行的

//用于用户认证校验、用户权限校验

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

System.out.println("HandlerInterceptor1...preHandle");

//如果返回false表示拦截不继续执行handler，如果返回true表示放行

**return** **false**;

}

//在执行handler返回modelAndView之前来执行

//如果需要向页面提供一些公用 的数据或配置一些视图信息，使用此方法实现 从modelAndView入手

@Override

**public** **void** postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

System.out.println("HandlerInterceptor1...postHandle");

}

//执行handler之后执行此方法

//作系统 统一异常处理，进行方法执行性能监控，在preHandle中设置一个时间点，在afterCompletion设置一个时间，两个时间点的差就是执行时长

//实现 系统 统一日志记录

@Override

**public** **void** afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

**throws** Exception {

System.out.println("HandlerInterceptor1...afterCompletion");

}

}

统一日志怎么做的？

首先，做一个统一拦截器，在里面统一记录系统日志。

## 测试拦截器

### 定义两个拦截器



### 配置拦截器

配置全局的拦截器，DispatcherServlet将配置的全局拦截器加载到所有的HandlerMapping。

在springmvc.xml中配置：

<!--拦截器 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<!-- <mvc:interceptor>

<mvc:mapping path="/\*\*" />

<bean class="cn.itcast.ssm.controller.interceptor.HandlerInterceptor1"></bean>

</mvc:interceptor>

<mvc:interceptor>

<mvc:mapping path="/\*\*" />

<bean class="cn.itcast.ssm.controller.interceptor.HandlerInterceptor2"></bean>

</mvc:interceptor> -->

### 测试1 （1 号和2号都放行）

测试结果：

HandlerInterceptor1...preHandle

HandlerInterceptor2...preHandle

HandlerInterceptor2...postHandle

HandlerInterceptor1...postHandle

HandlerInterceptor2...afterCompletion

HandlerInterceptor1...afterCompletion

总结：

执行preHandle是顺序执行。

执行postHandle、afterCompletion是倒序执行

### 测试2 （1 号放行和2号不放行）

测试结果：

HandlerInterceptor1...preHandle

HandlerInterceptor2...preHandle

HandlerInterceptor1...afterCompletion

总结：

如果preHandle不放行，postHandle、afterCompletion都不执行。

只要有一个拦截器不放行，controller不能执行完成

### 测试3 （1 号不放行和2号不放行）

测试结果：

HandlerInterceptor1...preHandle

总结：

只有前边的拦截器preHandle方法放行，下边的拦截器的preHandle才执行。

### 日志拦截器或异常拦截器要求

将日志拦截器或异常拦截器放在拦截器链儿中第一个位置，且preHandle方法放行

课11：springmvc拦截器应用-用户认证拦截

## 拦截器应用（用户认证拦截）

### 需求

用户访问系统的资源(url)，如果用户没有进行身份认证，进行拦截，系统跳转登陆页面，如果用户已经认证通过，用户可以继续访问系统 的资源。

### 用户登陆及退出功能开发

**public** **class** LoginController {

//用户登陆提交方法

@RequestMapping("/login")

**public** String login(HttpSession session, String usercode,String password)**throws** Exception{

//调用service校验用户账号和密码的正确性

//..

//如果service校验通过，将用户身份记录到session

session.setAttribute("usercode", usercode);

//重定向到商品查询页面

**return** "redirect:/items/queryItems.action";

}

//用户退出

@RequestMapping("/logout")

**public** String logout(HttpSession session)**throws** Exception{

//session失效

session.invalidate();

//重定向到商品查询页面

**return** "redirect:/items/queryItems.action";

}

}

拦截实现思路：

http请求url

如果url是公开地址

放行

如果用户在session中存在

放行

公开地址：

不需要认证即可访问的url

如果用户在session中不存在

登陆页面

### 拦截器

//得到请求的url

String url = request.getRequestURI();

//判断是否是公开 地址

//实际开发中需要公开 地址配置在配置文件中

//...

**if**(url.indexOf("login.action")>=0){

//如果是公开 地址则放行

**return** **true**;

}

//判断用户身份在session中是否存在

HttpSession session = request.getSession();

String usercode = (String) session.getAttribute("usercode");

//如果用户身份在session中存在放行

**if**(usercode!=**null**){

**return** **true**;

}

//执行到这里拦截，跳转到登陆页面，用户进行身份认证

request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request, response);

//如果返回false表示拦截不继续执行handler，如果返回true表示放行

**return** **false**;

在springmvc.xml中配置拦截器：

<mvc:interceptor>

<!-- /\*\*可以拦截路径不管多少层 -->

<mvc:mapping path=*"/\*\*"* />

<bean class=*"cn.itcast.ssm.controller.interceptor.LoginInterceptor"*></bean>

</mvc:interceptor>

</mvc:interceptors>

## 学生练习

商品查询添加查询条件：商品名称、价格范围

添加商品功能开发

删除商品功能开发