JavaEE 学习

目录

第一部分：javaEE 基础-javaweb

HTML&CSS 知识点

表单标签

表单标签：所有需要提交到服务器端的表单项必须使用<form></form>括起来!
form 标签属性：
action,整个表单提交的位置(可以是一个页面，也可以是一个后台 java 代码)
method,表单提交的方式(get/post/delete……等 7 种)

Get 与 post 提交方式的区别? 【默认提交方式为 get】

Get 提交方式，所有的内容显示在地址栏， 不够安全，长度有限制。
Post 提交方式，所有的内容不会显示在地址栏，比较安全，长度没有限制

DIV 相关的技术

Div 它是一个 html 标签，一个块级元素(单独显示一行)。 它单独使用没有任何意义，必须结合 CSS 来使用。 它主要用于页面的布局。
Span 它是一个 html 标签，一个内联元素(显示一行)。 它单独使用没有任何意义， 必须结合 CSS 来使用。 它主要用于对括起来的内容进行样式的修饰。

CSS 语法和规范

选择器{
属性名 1:属性值 1;
属性名 2:属性值 2;
属性名 3:属性值 3;
}

- **CSS 的引入方式**

CSS 的引入方式分为三种
第一种： 行内引入

```
<div style="color:red;font-size: 100px;">
JavaEE0516 就业班
</div>
```

第二种：内部引入方式

```
<style type="text/css">
div{
color:red;
font-size: 100px;
}
</style>
```

第三种方式：外部引入
如果<style type="text/css"></style>
优先级问题:
谁离需要修饰的元素近，谁的样式生效，其它的被覆盖掉。 (就近原则)

CSS 的选择器

CSS 基本选择器有三种(元素选择器、类选择器、 id 选择器)

- **ID 选择器**

#id 属性名{
属性名 1:属性值 1;
属性名 2:属性值 2;
属性名 3:属性值 3;
} Id 保证唯一。

- **元素选择器**

元素名{
属性名 1:属性值 1;
属性名 2:属性值 2;
属性名 3:属性值 3;
}如果多个相同的元素设置相同的样式，使用此种方式最为合适

- **类选择器**

.类名{
属性名 1:属性值 1;
属性名 2:属性值 2;
属性名 3:属性值 3;
}
对多个元素设置相同的样式，此时使用类选择器比较合适。

- **CSS 的浮动**

那么此时，我们可以清除浮动来清除之前框 1 和框 2 使用浮动后造成的问题!
解决办法:
在框 3 的前面定义一个 div( )
定义 CSS 样式:
#three{
clear:both;
}

- **CSS 中如何让块级元素成为内联元素**

我们可以使用个 CSS 中的 display 属性(inline)进行设置

去掉超链接的下划线：

```
a{
text-decoration: none;
}
```

其它选择器

- **层级选择器**

可以使用层级选择器设置列表的样式

```
元素名 子元素名{
属性名 1:属性值 1;
属性名 2:属性值 2;
属性名 3:属性值 3;
}
```

- **属性选择器**

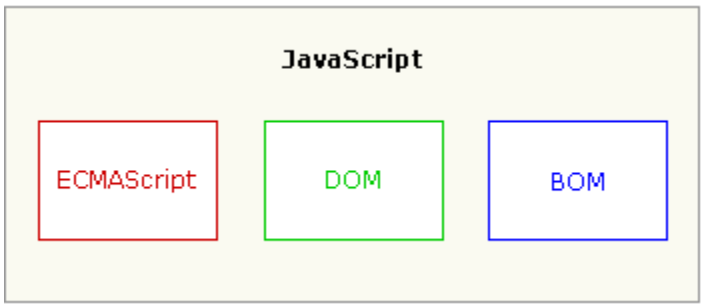
语法：

```
元素名[属性名="属性值"]{
属性名 1:属性值 1;
属性名 2:属性值 2;
属性名 3:属性值 3;
}
```

```
<html>
<head>
<meta charset="UTF-8">
<title></title>
<style>
input[type="text"]{
background-color: red;
}
input[type="password"]{
background-color: blue;
}
</style>
</head>
<body>
用户名：  <input type="text" name="username"/><br />
密码：  <input type="password" name="password" />
</body>
</html>
```

Javascript 知识点

javascript 的组成部分



ECMAScript:它是 javascript 的核心(语法、变量、数据类型、语句、函数……)
DOM:document object model 整个文档对象
BOM:浏览器对象

javascript 语法

区分大小写
变量是弱类型的(String str="aaa" ,var str="123";)
每行结尾的分号可有可无(建议大家写上)
注释与 java、php 等语言相同。

javascript 的变量

变量可以不用声明，变量是弱类型。统一使用 var 来定义！定义变量的时候不要使用关键字和保留字。

javascript 数据类型

Javascript 数据类型分为原始数据类型和引用数据类型
原始数据类型:
　　string、number、boolean、null、undefined（变量存在，但未赋值）
引用数据类型:
Array
Boolean
Date
Math
Number
String
RegExp

javascript 运算符

其它运算符与 java 大体一致，需要注意其等性运算符。

== 它在做比较的时候会进行自动转换。
=== 它在做比较的时候不会进行自动转换。

javascript 语句

所有语句与 java 大体一致。

获取元素内容

获取元素
document.getElementById("id 名称");
获取元素里面的值
document.getElementById("id 名称").value;

javascripte 的输出

警告框：alert();
向页面指定位置写入内容：innerHTML(属性)
向页面写入内容：document.write("");

javascript 的引入方式

- **内部引入方式**

  直接将 javascript 代码写到<script type="text/javascript"></script>

- **外部引入方式**

  需要创建一个.js 文件，在里面书写 javascript 代码，然后在 html 文件中通过 script 标签的 src 属性引入该外部的 js 文件

- **Window 对象**

Window 对象表示浏览器中打开的窗口。

Window 对象方法

方法	描述
alert()	显示带有一段消息和一个确认按钮的警告框。
blur()	把键盘焦点从顶层窗口移开。
clearInterval()	取消由 setInterval() 设置的 timeout。
clearTimeout()	取消由 setTimeout() 方法设置的 timeout。
close()	关闭浏览器窗口。
confirm()	显示带有一段消息以及确认按钮和取消按钮的对话框。
createPopup()	创建一个 pop-up 窗口。
focus()	把键盘焦点给予一个窗口。
moveBy()	可相对窗口的当前坐标把它移动指定的像素。
moveTo()	把窗口的左上角移动到一个指定的坐标。
open()	打开一个新的浏览器窗口或查找一个已命名的窗口。
print()	打印当前窗口的内容。
prompt()	显示可提示用户输入的对话框。
resizeBy()	按照指定的像素调整窗口的大小。
resizeTo()	把窗口的大小调整到指定的宽度和高度。
scrollBy()	按照指定的像素值来滚动内容。
scrollTo()	把内容滚动到指定的坐标。
setInterval()	按照指定的周期（以毫秒计）来调用函数或计算表达式。
setTimeout()	在指定的毫秒数后调用函数或计算表达式。

setInterval():它有一个返回值，主要是提供给 clearInterval 使用。
setTimeout():它有一个返回值，主要是提供给 clearTimeout 使用。
clearInterval():该方法只能清除由 setInterval 设置的定时操作
clearTimeout():该方法只能清除由 setTimeout 设置的定时操作

- **Location 对象**

Location 对象包含有关当前 URL 的信息。

Location 对象属性

属性	描述
hash	设置或返回从井号（#）开始的 URL（锚）。
host	设置或返回主机名和当前 URL 的端口号。
hostname	设置或返回当前 URL 的主机名。
href	设置或返回完整的 URL。
pathname	设置或返回当前 URL 的路径部分。
port	设置或返回当前 URL 的端口号。
protocol	设置或返回当前 URL 的协议。
search	设置或返回从问号（?）开始的 URL（查询部分）。

- **History 对象**

History 对象包含用户（在浏览器窗口中）访问过的 URL。

History 对象方法

方法	描述
back()	加载 history 列表中的前一个 URL。
forward()	加载 history 列表中的下一个 URL。
go()	加载 history 列表中的某个具体页面。

历史页面：使用 location 页面(把 href 属性值改为当前的 history)
go(参数)
参数：-1 返回上一个历史记录页面；-2 返回上上一个历史记录页面，1 进入下一个历史记录页面。

让按钮点击失效：
onclick="javascript:volid(0)"

- **Navigator 对象**

Navigator 对象包含有关浏览器的信息。(该对象开发中不怎么常用)

Navigator 对象属性

属性	描述
appCodeName	返回浏览器的代码名。
appMinorVersion	返回浏览器的次级版本。
appName	返回浏览器的名称。
appVersion	返回浏览器的平台和版本信息。
browserLanguage	返回当前浏览器的语言。
cookieEnabled	返回指明浏览器中是否启用 cookie 的布尔值。
cpuClass	返回浏览器系统的 CPU 等级。
onLine	返回指明系统是否处于脱机模式的布尔值。
platform	返回运行浏览器的操作系统平台。
systemLanguage	返回 OS 使用的默认语言。
userAgent	返回由客户机发送服务器的 user-agent 头部的值。
userLanguage	返回 OS 的自然语言设置。

- **Screen 对象**

Screen 对象包含有关客户端显示屏幕的信息。(该对象开发中不怎么常用)

Screen 对象属性

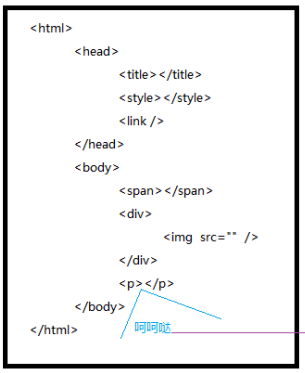
属性	描述
availHeight	返回显示屏幕的高度 (除 Windows 任务栏之外)。
availWidth	返回显示屏幕的宽度 (除 Windows 任务栏之外)。
bufferDepth	设置或返回调色板的比特深度。
colorDepth	返回目标设备或缓冲器上的调色板的比特深度。
deviceXDPI	返回显示屏幕的每英寸水平点数。
deviceYDPI	返回显示屏幕的每英寸垂直点数。
fontSmoothingEnabled	返回用户是否在显示控制面板中启用了字体平滑。
height	返回显示屏幕的高度。
logicalXDPI	返回显示屏幕每英寸的水平方向的常规点数。
logicalYDPI	返回显示屏幕每英寸的垂直方向的常规点数。
pixelDepth	返回显示屏幕的颜色分辨率（比特每像素）。
updateInterval	设置或返回屏幕的刷新率。
width	返回显示器屏幕的宽度。

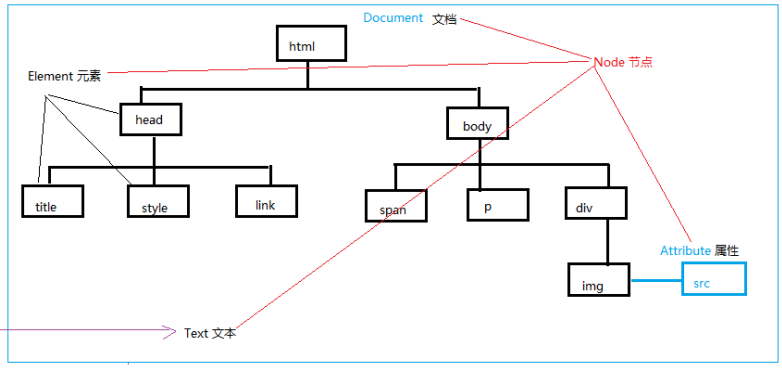
javascript 的事件

属性	当以下情况发生时，出现此事件	FF	N	IE
onabort	图像加载被中断	1	3	4
onblur	元素失去焦点	1	2	3
onchange	用户改变域的内容	1	2	3
onclick	鼠标点击某个对象	1	2	3
ondblclick	鼠标双击某个对象	1	4	4
onerror	当加载文档或图像时发生某个错误	1	3	4
onfocus	元素获得焦点	1	2	3
onkeydown	某个键盘的键被按下	1	4	3
onkeypress	某个键盘的键被按下或按住	1	4	3
onkeyup	某个键盘的键被松开	1	4	3
onload	某个页面或图像被完成加载	1	2	3
onmousedown	某个鼠标按键被按下	1	4	4
onmousemove	鼠标被移动	1	6	3
onmouseout	鼠标从某元素移开	1	4	4
onmouseover	鼠标被移到某元素之上	1	2	3
onmouseup	某个鼠标按键被松开	1	4	4
onreset	重置按钮被点击	1	3	4
onresize	窗口或框架被调整尺寸	1	4	4
onselect	文本被选定	1	2	3
onsubmit	提交按钮被点击	1	2	3
onunload	用户退出页面	1	2	3

onfocus/onblur:聚焦离焦事件，用于表单校验的时候比较合适。
onclick/ondblclick:鼠标单击和双击事件
onkeydown/onkeypress：搜索引擎使用较多
onload：页面加载事件，所有的其它操作(匿名方式)都可以放到这个绑定的函数里面去。如果是有名称，那么在 html 页面中只能写一个。
onmouseover/onmouseout/onmousemove：购物网站商品详情页。
onsubmit：表单提交事件 ，有返回值，控制表单是否提交。
onchange:当用户改变内容的时候使用这个事件(二级联动)

javascript 的 DOM 操作





Document:整个 html 文件都成为一个 document 文档
Element:所有的标签都是 Element 元素
Attribute：标签里面的属性
Text：标签中间夹着的内容为 text 文本
Node:document、element、attribute、text 统称为节点 node.

➢ Document 对象

每个载入浏览器的 HTML 文档都会成为 Document 对象。

方法	描述
close()	关闭用 document.open() 方法打开的输出流，并显示选定的数据。
getElementById()	返回对拥有指定 id 的第一个对象的引用。
getElementsByName()	返回带有指定名称的对象集合。
getElementsByTagName()	返回带有指定标签名的对象集合。
open()	打开一个流，以收集来自任何 document.write() 或 document.writeln() 方法的输出。
write()	向文档写 HTML 表达式 或 JavaScript 代码。
writeln()	等同于 write() 方法，不同的是在每个表达式之后写一个换行符。

后面两个方法获取之后需要遍历!

以下两个方法很重要，但是在手册中查不到!
创建文本节点：document.createTextNode()
创建元素节点：document.createElement()

➢ Element 对象

我们所认知的 html 页面中所有的标签都是 element 元素

element.appendChild() 向元素添加新的子节点，作为最后一个子节点。
element.firstChild 返回元素的首个子节点。
element.getAttribute() 返回元素节点的指定属性值。
element.innerHTML 设置或返回元素的内容。
element.insertBefore() 在指定的已有的子节点之前插入新节点。
element.lastChild 返回元素的最后一个子元素。
element.setAttribute() 把指定属性设置或更改为指定值。
element.removeChild() 从元素中移除子节点。
element.replaceChild() 替换元素中的子节点。

➢ Attribute 对象

我们所认知的 html 页面中所有标签里面的属性都是 attribute

属性 / 方法	描述
attr.isId	如果属性是 id 类型，则返回 true，否则返回 false。
attr.name	返回属性的名称。
attr.value	设置或返回属性的值。
attr.specified	如果已指定属性，则返回 true，否则返回 false。
nodemap.getNamedItem()	从 NamedNodeMap 返回指定的属性节点。
nodemap.item()	返回 NamedNodeMap 中位于指定下标的节点。
nodemap.length	返回 NamedNodeMap 中的节点数。
nodemap.removeNamedItem()	移除指定的属性节点。
nodemap.setNamedItem()	设置指定的属性节点（通过名称）。

javascript 内置对象

JavaScript 对象
JS Array
JS Boolean
JS Date
JS Math
JS Number
JS String
JS RegExp

- **Array 对象**

数组的创建:

Array 对象

Array 对象用于在单个的变量中存储多个值。

创建 Array 对象的语法：

```
new Array();
new Array(size);
new Array(element0, element1, ..., elementn);
```

数组的特点:

长度可变！数组的长度=最大角标+1

- **Boolean 对象**

对象创建:

创建 Boolean 对象的语法：

```
new Boolean(value);          //构造函数
Boolean(value);              //转换函数
```

如果 value 不写，那么默认创建的结果为 false

➢ Date 对象

getTime() 返回 1970 年 1 月 1 日至今的毫秒数。
解决浏览器缓存问题

➢ Math 和 number 对象

与 java 里面的基本一致。

➢ String 对象

match() 找到一个或多个正则表达式的匹配。
substr() 从起始索引号提取字符串中指定数目的字符。
substring() 提取字符串中两个指定的索引号之间的字符。

➢ RegExp 对象

正则表达式对象
test 检索字符串中指定的值。返回 true 或 false。

全局函数

全局属性和函数可用于所有内建的 JavaScript 对象

函数	描述
decodeURI()	解码某个编码的 URI。
decodeURIComponent()	解码一个编码的 URI 组件。
encodeURI()	把字符串编码为 URI。
encodeURIComponent()	把字符串编码为 URI 组件。
escape()	对字符串进行编码。
eval()	计算 JavaScript 字符串，并把它作为脚本代码来执行。
getClass()	返回一个 JavaObject 的 JavaClass。
isFinite()	检查某个值是否为有穷大的数。
isNaN()	检查某个值是否是数字。
Number()	把对象的值转换为数字。
parseFloat()	解析一个字符串并返回一个浮点数。
parseInt()	解析一个字符串并返回一个整数。
String()	把对象的值转换为字符串。
unescape()	对由 escape() 编码的字符串进行解码。

jQuery 知识点

➢ Jquery 的简单入门

所有的 jquery 代码写在页面加载函数
$(function(){
    Jquery 代码
});
传统的 JavaScript 页面加载函数是最后一个生效，它会覆盖之前的。它的加载顺序比 jQuery 的要慢。【它是整个文档加载完毕后才会执行】
jQuery 的页面加载函数可以存在多个(不会发生覆盖)，它会按照顺序进行执行。(dom 数加载完成)

➢ 获取元素

JS:document.getElementById();
JQ:$("#id");

➢ Jquery 对象与 DOM 对象转换

```
function JSWrite(){
    //document.getElementById("span1").innerHTML="美美哒！";
    var spanEle = document.getElementById("span1");
    $(spanEle).html("美美哒！");
}
$(function(){
    /*document.getElementById("btn1").onclick = function(){
        document.getElementById("span1").innerHTML="帅帅哒！";
    }*/
    $("#btn1").click(function(){
        //JQ对象转换成DOM对象的第一种方式
        //$("#span1")[0].innerHTML="呵呵哒！";
        //JQ对象转换成DOM对象的第二种方式
        $("#span1").get(0).innerHTML="呵呵哒！";
    });
});
```

注意：JQ 对象只能操作 JQ 里面的属性和方法
JS 对象只能操作 JS 里面的属性和方法。

- **Jquery 的效果**

jquery 的选择器

- **基本选择器**

id 选择器：$("#id 名称");
元素选择器：$("元素名称");
类选择器：$(".类名");
通配符：*
多个选择器共用(并集)

- **层级选择器**

ancestor descendant: 在给定的祖先元素下匹配所有的后代元素(儿子、孙子、重孙子)
parent > child : 在给定的父元素下匹配所有的子元素(儿子)
prev + next: 匹配所有紧接在 prev 元素后的 next 元素(紧挨着的，同桌)
prev ~ siblings: 匹配 prev 元素之后的所有 siblings 元素(兄弟)

- **基本过滤选择器**

$('li').first() 等价于：$("li:first")

基本
:first
:last
:not
:even
:odd
:eq
:gt
:lt
:header
:animated
内容
:contains
:empty
:has
:parent
可见性
:hidden
:visible

- 属性选择器

属性
[attribute]
[attribute=value]
[attribute!=value]
[attribute^=value]
[attribute$=value]
[attribute*=value]
[attrSel1][attrSel2][attrSelN]

- 表单选择器

表单
:input
:text
:password
:radio
:checkbox
:submit
:image
:reset
:button
:file
:hidden
表单对象属性
:enabled
:disabled
:checked
:selected

数组的遍历操作

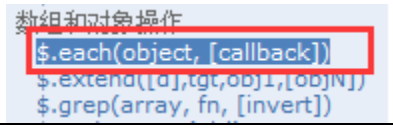
方式一:

jQuery 对象访问
each(callback)
size()
length

```
$(function(){
```

```
    // 全选/ 全不选
    $("#checkallbox").click(function(){
        var isChecked = this.checked;
        //使用对象访问的方式进行遍历，语法：$().each(function(){})
        $("input[name='hobby']").each(function(){
            this.checked = isChecked;
        });
    });
});
```

方式二：




```
$.each( [0,1,2], function(i, n){
   alert( "Item #" + i + ": " + n );
});
```

文档处理操作

追加内容

append: A.append(B) 将 B 追加到 A 的内容的末尾处

appendTo: A.appendTo(B) 将 A 加到 B 内容的末尾处

Bootstrap 知识点

响应式布局

- 响应式布局：一个网站能够兼容多个终端(手机、iPad 等)，而不需要为每个终端做一个特定的版本。此概念是为解决移动互联网浏览而诞生的。
- 响应式布局可以为不同终端的用户提供更加舒适的界面和更好的用户体验，而且随着目前大屏幕移动设备的普及，用“大势所趋”来形容也不为过。随着越来越多的设计师采用这个技术，我们不仅看到很多的创新，还看到了一些成形的模式。
- Bootstrap 就是响应式布局最成功的实现，为了兼容不同的浏览器采用 jQuery，为了适配不同的终端采用 CSS3 Media Query （媒体查询）

Bootstrap 基本模板：

```
<!DOCTYPE html>
<html lang="zh-CN">
    <head>
        <meta charset="utf-8">
        <!--声明文档兼容模式，表示使用 IE 浏览器的最新模式-->
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <!--设置视口的宽度(值为设备的理想宽度)，页面初始缩放值<理想宽度/可见宽度>-->
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!--  上述 3 个 meta 标签*必须*放在最前面，任何其他内容都*必须*跟随其后！  -->
```

```
        <title>Bootstrap 基本模板</title>
        <!-- 引入 Bootstrap 核心样式文件 -->
        <link href="../../css/bootstrap.min.css" rel="stylesheet">
    </head>
    <body>
        <h1>你好，世界！</h1>
        <!-- 引入 jQuery 核心 js 文件 -->
        <script src="../../js/jquery-1.11.0.min.js"></script>
        <!-- 引入 BootStrap 核心 js 文件 -->
        <script src="../../js/bootstrap.min.js"></script>
    </body>
</html>
```

Bootstrap 完整版模板：

```
<!DOCTYPE html>
<html lang="zh-CN">
    <head>
        <meta charset="utf-8">
        <!--声明文档兼容模式，表示使用 IE 浏览器的最新模式-->
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <!--设置视口的宽度(值为设备的理想宽度)，页面初始缩放值<理想宽度/可见宽度>-->
        <!--视口的作用：在移动浏览器中，当页面宽度超出设备，浏览器内部虚拟的一个页面容器，会将页面缩放到设备这么大来展示-->
        <!--width 设置 layout viewport 的宽度，为一个正整数，或字符串"width-device"(表示采用设备的宽度)
            initial-scale 设置页面的初始缩放值，为一个数字，可以带小数
            minimum-scale 允许用户的最小缩放值，为一个数字，可以带小数
            maximum-scale 允许用户的最大缩放值，为一个数字，可以带小数
            height 设置 layout viewport 的高度，这个属性对我们并不重要，很少使用
            user-scalable 是否允许用户进行缩放，值为"no"或"yes", no 代表不允许，yes 代表允许
            如果设置"user-scalable=no",那么"minimum-scale"和"maximum-scale"无效
        -->
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- 上述 3 个 meta 标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
        <title>Bootstrap 完整模板</title>

        <!-- 引入 Bootstrap 核心样式文件 -->
        <link href="../../css/bootstrap.min.css" rel="stylesheet">
        <!-- HTML5 Shim 和 Respond.js 用于让 IE8 支持 HTML5 元素和媒体查询 -->
        <!-- 注意： 如果通过 file:// 引入 Respond.js 文件，则该文件无法起效果，必须放置到 web 服务器中，暂时不必掌握 -->
        <!--[if lt IE 9]>
          <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
          <script src="https://oss.maxcdn.com/libs/respond.js/1.3.0/respond.min.js"></script>
        <![endif]-->
    </head>

    <body>
        <!--正文从此处开始-->
        <h1>你好，世界！</h1>
        <!--前端开发建议：网站优化时，除了立即需要工作的 js 存放在 head 外，将大部分 JS 文件放在页面的末尾-->
        <!-- 引入 jQuery 核心 js 文件，必须放置在 bootStrap.js 前面！ -->
        <script src="../../js/jquery-1.11.0.min.js"></script>
        <!-- 引入 BootStrap 核心 js 文件 -->
```

```
        <script src="../../js/bootstrap.min.js"></script>
    </body>
</html>
```

JDBC & 配置文件 & 连接池

导入 mysql 数据库的驱动 jar 包：

mysql-connector-java-5.1.39-bin.jar；

注册驱动

看清楚了，注册驱动就只有一句话：Class.forName(“com.mysql.jdbc.Driver”)，下面的内容都是对这句代码的解释。今后我们的代码中，与注册驱动相关的代码只有这一句。

DriverManager 类的 registerDriver()方法的参数是 java.sql.Driver，但 java.sql.Driver 是一个接口，实现类由 mysql 驱动来提供，mysql 驱动中的 java.sql.Driver 接口的实现类为 com.mysql.jdbc.Driver！那么注册驱动的代码如下：

DriverManager.registerDriver(new com.mysql.jdbc.Driver());

上面代码虽然可以注册驱动，但是出现硬编码（代码依赖 mysql 驱动 jar 包），如果将来想连接 Oracle 数据库，那么必须要修改代码的。并且其实这种注册驱动的方式是注册了两次驱动！

JDBC 中规定，驱动类在被加载时，需要自己“主动”把自己注册到 DriverManger 中，下面我们来看看 com.mysql.jdbc.Driver 类的源代码：

com.mysql.jdbc.Driver.java

```
public class Driver extends NonRegisteringDriver implements java.sql.Driver {
    static {
        try {
            java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
            throw new RuntimeException("Can't register driver!");
        }
    }
    ……
}
```

com.mysql.jdbc.Driver 类中的 static 块会创建本类对象，并注册到 DriverManager 中。这说明只要去加载 com.mysql.jdbc.Driver 类，那么就会执行这个 static 块，从而也就会把 com.mysql.jdbc.Driver 注册到 DriverManager 中，所以可以把**注册驱动类的代码修改为加载驱动类。**

Class.forName(“com.mysql.jdbc.Driver”);

获取连接

获取连接需要两步，一是使用 DriverManager 来注册驱动，二是使用 DriverManager 来获取 Connection 对象。

获取连接的也只有一句代码：

DriverManager.getConnection(url,username,password)，

其中 username 和 password 是登录数据库的用户名和密码，如果我没说错的话，你的 mysql 数据库的用户名和密码分别是：root、123。

url 查对复杂一点，它是用来找到要连接数据库“网址”，就好比你要浏览器中查找百度时，也需要提供一个 url。下面是 mysql 的 url：

jdbc:mysql://localhost:3306/mydb1

JDBC 规定 url 的格式由三部分组成，每个部分中间使用冒号分隔。

- 第一部分是 jdbc，这是固定的；
- 第二部分是数据库名称，那么连接 mysql 数据库，第二部分当然是 mysql 了；
- 第三部分是由数据库厂商规定的，我们需要了解每个数据库厂商的要求，mysql 的第三部分分别由数据库服务器的 IP 地址(localhost)、端口号(3306)，以及 DATABASE 名称(mydb1)组成。

下面是获取连接的语句：

```
Connection con = DriverManager.getConnection(“jdbc:mysql://localhost:3306/web08”,”root”,”root”);
```

还可以在 url 中提供参数：

jdbc:mysql://localhost:3306/web08**?useUnicode=true&characterEncoding=UTF8**

useUnicode 参数指定这个连接数据库的过程中，使用的字节集是 Unicode 字节集；

characherEncoding 参数指定穿上连接数据库的过程中，使用的字节集编码为 UTF-8 编码。请注意，mysql 中指定 UTF-8 编码是给出的是 UTF8，而不是 UTF-8。要小心了！

获取 Statement

在得到 Connectoin 之后，说明已经与数据库连接上了，下面是通过 Connection 获取 Statement 对象的代码：

```
Statement stmt = con.createStatement();
```

Statement 是用来向数据库发送要执行的 SQL 语句的！

发送 SQL 查询语句

```
String sql = “select * from user”;
ResultSet rs = stmt.executeQuery(sql);
```

请注意，执行查询使用的不是 executeUpdate()方法，而是 executeQuery()方法。executeQuery()方法返回的是 ResultSet，ResultSet 封装了查询结果，我们称之为结果集。

读取结果集中的数据

ResultSet 就是一张二维的表格，它内部有一个“行光标”，光标默认的位置在“第一行上方”，我们可以调用 rs 对象的 next()方法把“行光标”向下移动一行，当第一次调用 next()方法时，“行光标”就到了第一行记录的位置，这时就可以使用 ResultSet 提供的 getXXX(int col)方法来获取指定列的数据了：

```
rs.next();//光标移动到第一行
rs.getInt(1);//获取第一行第一列的数据
```

当你使用 rs.getInt(1)方法时，你必须可以肯定第 1 列的数据类型就是 int 类型，如果你不能肯定，那么最好使用 rs.getObject(1)。在 ResultSet 类中提供了一系列的 getXXX()方法，比较常用的方法有：

```
Object getObject(int col)
String getString(int col)
int getInt(int col)
double getDouble(int col)
```

关闭

与 IO 流一样，使用后的东西都需要关闭！关闭的顺序是先得到的后关闭，后得到的先关闭。

```
rs.close();
stmt.close();
con.close();
```

完成查询操作代码

```
@Test
public void query() {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
        con = getConnection();
        stmt = con.createStatement();
        String sql = "select * from user";
        rs = stmt.executeQuery(sql);
        while(rs.next()) {
            String username = rs.getString(1);
            String password = rs.getString(2);
            System.out.println(username + ", " + password);
        }
```

```
        } catch(Exception e) {
            throw new RuntimeException(e);
        } finally {
            try {
                if(rs != null) rs.close();
                if(stmt != null) stmt.close();
                if(con != null) con.close();
            } catch(SQLException e) {}
        }
    }
```

防止 SQL 攻击

- 过滤用户输入的数据中是否包含非法字符；
- 分步校验！先使用用户名来查询用户，如果查找到了，再比较密码；
- 使用 PreparedStatement。

PreparedStatement 是什么？

PreparedStatement 叫预编译声明！

PreparedStatement 是 Statement 的子接口，你可以使用 PreparedStatement 来替换 Statement。

PreparedStatement 的好处：

- 防止 SQL 攻击；
- 提高代码的可读性，以可维护性；
- 提高效率。

PreparedStatement 的使用

- 使用 Connection 的 prepareStatement(String sql)：即创建它时就让它与一条 SQL 模板绑定；
- 调用 PreparedStatement 的 setXXX()系列方法为问号设置值
- 调用 executeUpdate()或 executeQuery()方法，但要注意，调用没有参数的方法；

```
String sql = "select * from tab_student where s_number=?";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setString(1, "S_1001");
ResultSet rs = pstmt.executeQuery();
rs.close();
pstmt.clearParameters();
pstmt.setString(1, "S_1002");
rs = pstmt.executeQuery();
```

在使用 Connection 创建 PreparedStatement 对象时需要给出一个 SQL 模板，所谓 SQL 模板就是有“?”的 SQL 语句，其中“?”就是参数。

在得到 PreparedStatement 对象后，调用它的 setXXX()方法为“?”赋值，这样就可以得到把模板变成一条完整的 SQL 语句，然后再调用 PreparedStatement 对象的 executeQuery()方法获取 ResultSet 对象。

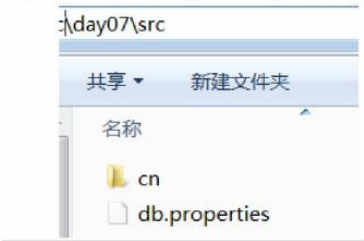
注意 PreparedStatement 对象独有的 executeQuery()方法是没有参数的，而 Statement 的 executeQuery()是需要参数（SQL 语句）的。因为在创建 PreparedStatement 对象时已经让它与一条 SQL 模板绑定在一起了，所以在调用它的 executeQuery()和 executeUpdate()方法时就不再需要参数了。

PreparedStatement 最大的好处就是在于重复使用同一模板，给予其不同的参数来重复的使用它。这才是真正提高效率的原因。

所以，建议大家在今后的开发中，无论什么情况，都去需要 PreparedStatement，而不是使用 Statement。

配置文件

右键/New/File，输入“db.properties”文件名。




```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/day07_db
jdbc.user=root
jdbc.password=1234
```

3.2.3.2 加载配置文件：ResourceBundle 对象

我们将在 v2 版本中使用 JDK 提供的工具类 ResourceBundle 加载 properties 文件，ResourceBundle 提供 getBundle()方法用于只提供 properties 文件即可，之后使用 getString(key)通过 key 获得 value 的值。

```
public class 配置文件连接 {
  public static String driver;
  public static String url;
  public static String user;
  public static String password;

  /**
   * 静态方法绑定配置文件
   */
  static {
      ResourceBundle bundle = ResourceBundle.getBundle("db");
      driver = bundle.getString("driver");
      url = bundle.getString("url");
      user = bundle.getString("username");
      password = bundle.getString("password");
  }

  /**
```

```
     * 连接mysql数据库的方法
     * @return
     */
    public static Connection getConnection() {
        Connection conn = null;
        try {
            Class.forName(driver);
            conn = DriverManager.getConnection(url, user, password);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return conn;
    }

    /**
     * 释放mysql数据库连接
     * @param conn
     * @param pstmt
     * @param rs
     */
    public static void releaseDb(Connection conn, PreparedStatement pstmt,
ResultSet rs) {
        if(rs != null) {
            try {
                rs.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        if(pstmt != null) {
            try {
                pstmt.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        if(conn != null) {
            try {
                conn.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

3.2.3.4 加载配置文件：Properties 对象（可选）

对应 properties 文件处理，开发中也会使用 Properties 对象进行。在 v3 版本中我们将采用加载 properties 文件获得流，然后使用 Properties 对象进行处理。

```
    /**
     * 静态代码块加载配置文件信息
     */
    static {
        try {
            // 1.通过当前类获取类加载器
            ClassLoader classLoader = JDBCUtils_V3.class.getClassLoader();
            // 2.通过类加载器的方法获得一个输入流
            InputStream is = classLoader.getResourceAsStream("db.properties");
            // 3.创建一个properties对象
            Properties props = new Properties();
            // 4.加载输入流
            props.load(is);
            // 5.获取相关参数的值
            driver = props.getProperty("driver");
            url = props.getProperty("url");
```

```
            username = props.getProperty("username");
            password = props.getProperty("password");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

开源的数据库连接池

➢ C3P0

第一步：引入 C3P0 连接池的 jar 包.

第二步：编写代码：

* 手动设置参数：default-config 默认，named-config 自定义可以调用

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>
    <default-config>
      <property name="driverClass">com.mysql.cj.jdbc.Driver</property>
      <property name="jdbcUrl">jdbc:mysql://47.100.188.155/airquality</property>
      <property name="user">root</property>
      <property name="password">root</property>
      <property name="initialPoolSize">5</property>
      <property name="maxPoolSize">20</property>
    </default-config>
    <named-config name="oracle">
      <property name="driverClass">com.mysql.jdbc.Driver</property>
      <property name="jdbcUrl">jdbc:mysql:///web_07</property>
      <property name="user">root</property>
      <property name="password">123</property>
    </named-config>
</c3p0-config>
```

* 代码调用：

```
public void testUser() {
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    ComboPooledDataSource dataSource = new ComboPooledDataSource();//加载默认配置文件的数据库配置
    //ComboPooledDataSource datasource1 = new ComboPooledDataSource("oracle");  //加载配置文件
                                                                                  中名字为“oracle”的数据库配置
    try {
        conn = dataSource.getConnection();
        String sql = "select * from air_quality where id<?";
        pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, 4050);
```

```
            rs = pstmt.executeQuery();
            while(rs.next()) {
                System.out.println(rs.getString(2));
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

C3P0 改造工具类

```
public class JDBCUtils2 {
    private static final ComboPooledDataSource DATA_SOURCE =new ComboPooledDataSource();
    /**
     * 获得连接的方法
     */
    public static Connection getConnection(){
        Connection conn = null;
        try {
            conn = DATA_SOURCE.getConnection();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return conn;
    }
```

➢ DBCP

第一步：引入 DBCP 连接池的 jar 包.

第二步：编写 DBCP 代码：

* 手动设置参数（properties 文件）：

```
driver=com.mysql.cj.jdbc.Driver
url=jdbc:mysql://47.100.188.155:3306/airquality
username=root
password=root
```

【DBCP 连接池的使用】

第一步：工具类封装

```
public class testDbcp {
    private static DataSource datasource;
    static {

        try {
            //加载properties文件
            InputStream is = testDbcp.class.getClassLoader().getResourceAsStream("db.properties");
```

```
                Properties props = new Properties();
                props.load(is);
                datasource = BasicDataSourceFactory.createDataSource(props);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
        public static DataSource getDataSource() {
            return datasource;
        }
        public static Connection getConnection() {
            try {
                return datasource.getConnection();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
```

第二步：调用实现

```
public class test {
    @Test
    public void testdbcp() {
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            conn = testDbcp.getConnection();
            String sql = "select * from air_quality where id<?";
            pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, 4050);
            rs = pstmt.executeQuery();
            while(rs.next()) {
                System.out.println(rs.getString(2));
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

DBUtils

JavaBean 就是一个类，在开发中常用语封装数据。具有如下特性

1. 需要实现接口：java.io.Serializable ，通常偷懒省略了。
2. 提供私有字段：private 类型 字段名;
3. 提供 getter/setter 方法：
4. 提供无参构造

```
public class javaBeantest {
    private String stationName;
    private String uniqueCode;
    public String getStationName() {
        return stationName;
    }
    public void setStationName(String stationName) {
        this.stationName = stationName;
    }
    public String getUniqueCode() {
        return uniqueCode;
    }
    public void setUniqueCode(String uniqueCode) {
        this.uniqueCode = uniqueCode;
    }
    public javaBeantest() {
    }
}
```

DBUtils 是 java 编程中的数据库操作实用工具，小巧简单实用。
DBUtils 封装了对 JDBC 的操作，简化了 JDBC 操作，可以少写代码。
Dbutils 三个核心功能介绍

- QueryRunner 中提供对 sql 语句操作的 API.
- ResultSetHandler 接口，用于定义 select 操作后，怎样封装结果集.
- DbUtils 类，它就是一个工具类,定义了关闭资源与事务处理的方法

2.3.2 QueryRunner 核心类

- QueryRunner(DataSource ds) ,提供数据源（连接池），DBUtils 底层自动维护连接 connection
- update(String sql, Object... params) ，执行更新数据
- query(String sql, ResultSetHandler<T> rsh, Object... params) ，执行查询

➢ ResultSetHandler

我们知道在执行 select 语句之后得到的是 ResultSet，然后我们还需要对 ResultSet 进行转换，得到最终我们想要的数据。你可以希望把 ResultSet 的数据放到一个 List 中，也可能想把数据放到一个 Map 中，或是一个 Bean 中。

DBUtils 提供了一个接口 ResultSetHandler，它就是用来 ResultSet 转换成目标类型的工具。你可以自己去实现这个接口，把 ResultSet 转换成你想要的类型。

DBUtils 提供了很多个 ResultSetHandler 接口的实现，这些实现已经基本够用了，我们通常不用自己去实现 ResultSet 接口了。

MapHandler：单行处理器！把结果集转换成 Map<String,Object>，其中列名为键！

MapListHandler：多行处理器！把结果集转换成 List<Map<String,Object>>；
BeanHandler：单行处理器！把结果集转换成 Bean，该处理器需要 Class 参数，即 Bean 的类型；
BeanListHandler：多行处理器！把结果集转换成 List<Bean>；
ColumnListHandler：多行单列处理器！把结果集转换成 List<Object>，使用 ColumnListHandler 时需要指定某一列的名称或编号，例如：new ColumListHandler(“name”)表示把 name 列的数据放到 List 中。
ScalarHandler：单行单列处理器！把结果集转换成 Object。一般用于聚集查询，例如 select count(*) from tab_student。

➢ 利用 DBUtils 进行增删改

```
/**
     * 添加所有用户方法
     */
    @Test
    public void testAddUser() {
        try {
            // 1.创建核心类 QueryRunner
            QueryRunner qr = new QueryRunner(C3P0Utils.getDataSource());
            // 2.编写 SQL 语句
            String sql = "insert into tbl_user values(null,?,?)";
            // 3.为站位符设置值
            Object[] params = { "余淮", "耿耿" };
            // 4.执行添加操作
            int rows = qr.update(sql, params);
            if (rows > 0) {
                System.out.println("添加成功!");
            } else {
                System.out.println("添加失败!");
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    /**
     * 根据 id 修改用户方法
     *
     */
    @Test
    public void testUpdateUserById() {
        try {
            // 1.创建核心类 QueryRunner
```

```
            QueryRunner qr = new QueryRunner(C3P0Utils.getDataSource());
            // 2.编写 SQL 语句
            String sql = "update tbl_user set upassword=? where uid=?";
            // 3.为站位符设置值
            Object[] params = { "xxx", 21 };
            // 4.执行添加操作
            int rows = qr.update(sql, params);
            if (rows > 0) {
                System.out.println("修改成功!");
            } else {
                System.out.println("修改失败!");
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    /**
     * 根据 id 删除用户方法
     */
    @Test
    public void testDeleteUserById() {
        try {
            // 1.创建核心类 QueryRunner
            QueryRunner qr = new QueryRunner(C3P0Utils.getDataSource());
            // 2.编写 SQL 语句
            String sql = "delete from tbl_user where uid=?";
            // 3.为站位符设置值
            Object[] params = {19};
            // 4.执行添加操作
            int rows = qr.update(sql, params);
            if (rows > 0) {
                System.out.println("删除成功!");
            } else {
                System.out.println("删除失败!");
            }
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

➢ 利用 DBUtils 进行查询

```
@Test
    public void dbQuery() {
        try {
            //1.创建核心类 QueryRunner
            QueryRunner qr = new QueryRunner(testDbcp.getDataSource());
            //2.编写 sql 语句
            String sql = "select * from air_quality where id<?";
            //3.为占位符设置值
            Object[] params = {4050};
            //4.执行操作
            List<javaBeantest> infos = qr.query(sql, params, new
BeanListHandler<javaBeantest>(javaBeantest.class));
            //5.查询结果遍历
            for(javaBeantest i : infos) {
                System.out.println(i.getStationName() + i.getUniqueCode());
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
```

Xml 文档及解析

- 开发中比较常见的解析方式有三种，如下：

1. DOM：要求解析器把整个 XML 文档装载到内存，并解析成一个 Document 对象。
   a) 优点：元素与元素之间保留结构关系，故可以进行增删改查操作。
   b) 缺点：XML 文档过大，可能出现内存溢出显现。
2. SAX：是一种速度更快，更有效的方法。它逐行扫描文档，一边扫描一边解析。并以事件驱动的方式进行具体解析，每执行一行，都将触发对应的事件。（了解）
   a) 优点：处理速度快，可以处理大文件
   b) 缺点：只能读，逐行后将释放资源。
3. PULL：Android 内置的 XML 解析方式，类似 SAX。（了解）

- 解析器：就是根据不同的解析方式提供的具体实现。有的解析器操作过于繁琐，为了方便开发人员，有提供易于操作的解析开发包。

- 常见的解析开发包：
  - JAXP：sun 公司提供支持 DOM 和 SAX 开发包
  - JDom：dom4j 兄弟
  - jsoup：一种处理 HTML 特定解析开发包
  - dom4j：比较常用的解析开发包，hibernate 底层采用。

如果需要使用 dom4j，必须导入 jar 包。

dom4j 必须使用核心类 SaxReader 加载 xml 文档获得 Document，通过 Document 对象获得文档的根元素，然后就可以操作了。

常用 API 如下：

1. SaxReader 对象
   a) read(...) 加载执行 xml 文档
2. Document 对象
   a) getRootElement() 获得根元素
3. Element 对象
   a) elements(...) 获得指定名称的所有子元素。可以不指定名称
   b) element(...) 获得指定名称第一个子元素。可以不指定名称
   c) getName() 获得当前元素的元素名
   d) attributeValue(...) 获得指定属性名的属性值
   e) elementText(...) 获得指定名称子元素的文本值
   f) getText() 获得当前元素的文本内容

```
// 1.获取解析器
SAXReader saxReader = new SAXReader();
// 2.获得document文档对象
Document doc = saxReader.read("src/cn/itheima/xml/schema/web.xml");
// 3.获取根元素
Element rootElement = doc.getRootElement();
// System.out.println(rootElement.getName());//获取根元素的名称
// System.out.println(rootElement.attributeValue("version"));//获取根元素中的属性值
// 4.获取根元素下的子元素
List<Element> childElements = rootElement.elements();
// 5.遍历子元素
for (Element element : childElements) {
    //6.判断元素名称为servlet的元素
    if ("servlet".equals(element.getName())) {
        //7.获取servlet-name元素
        Element servletName = element.element("servlet-name");
        //8.获取servlet-class元素
        Element servletClass = element.element("servlet-class");
        System.out.println(servletName.getText());
        System.out.println(servletClass.getText());
    }
}
```

反射

什么是反射技术?

动态获取指定类以及类中的内容(成员)，并运行其内容。

应用程序已经运行，无法在其中进行 new 对象的建立，就无法使用对象。这时可以根据配置文件的类全名去找对应的字节码文件，并加载进内存，并创建该类对象实例。这就需要使用反射技术完成

获取 class 对象的三种方式

获取 Class 对象的方式一：

通过对象具备的getClass方法(源于Object类的方法)。有点不方便，需要用到该类，并创建该类的对象，再调用getClass方法完成。

```
Person p = new Person();//创建Peron对象
Class clazz = p.getClass();//通过object继承来的方法(getClass)获取Person对应的字节码文件对象
```

获取Class对象的方式二：

每一个类型都具备一个class静态属性，通过该属性即可获取该类的字节码文件对象。比第一种简单了一些，仅用一个静态属性就搞定了。但是，还是有一点不方便，还必须要使用到该类。

```
Class clazz = Person.class;
```

获取Class对象方式三：

```
* 去找找Class类中是否有提供获取的方法呢?
* 找到了，static Class forName(className);
* 相对方便的多，不需要直接使用具体的类，只要知道该类的名字即可。
* 而名字完成可以作为参数进行传递 ,这样就可以提高扩展性。
* 所以为了动态获取一个类，第三种方式最为常用。
Class clazz = Class.forName("cn.itcast.bean.Person");//必须类全名
```

创建 Person 对象的方式

以前：1，先加载 cn.itcast.bean.Person 类进内存。

2,将该类封装成Class对象。

3,根据Class对象，用new操作符创建cn.itcast.bean.Person对象。

4,调用构造函数对该对象进行初始化。

```
cn.itcast.bean.Person p = new cn.itcast.bean.Person();
```

通过方式三：(此外还可以使用构造，构造可以指定参数---如 String.class)

```
String className = "cn.itcast.bean.Person";
//1，根据名称获取其对应的字节码文件对象
```

1，通过forName()根据指定的类名称去查找对应的字节码文件，并加载进内存。

2，并将该字节码文件封装成了Class对象。

3，直接通过newIntstance方法，完成该对象的创建。

4，newInstance方法调用就是该类中的空参数构造函数完成对象的初始化。

Class clazz = Class.forName(className);

//2，通过 Class 的方法完成该指定类的对象创建。

Object object = clazz.newInstance();//该方法用的是指定类中默认的空参数构造函数完成的初始化。

清单 1，获取字节码文件中的字段。

```
Class clazz = Class.forName("cn.itcast.bean.Person");
    //获取该类中的指定字段。比如age
Field field = clazz.getDeclaredField("age");//clazz.getField("age");
    //为了对该字段进行操作，必须要先有指定类的对象。
Object obj = clazz.newInstance();
    //对私有访问，必须取消对其的访问控制检查，使用AccessibleObject父类中的setAccessible的方法
field.setAccessible(true);//暴力访问。建议大家尽量不要访问私有
field.set(obj, 789);
    //获取该字段的值。
Object o = field.get(obj);
System.out.println(o);
```

备注：getDeclaredField：获取所有属性，包括私有。

getField:获取公开属性，包括从父类继承过来的，不包括非公开方法。

清单 2，获取字节码文件中的方法。

```
//根据名称获取其对应的字节码文件对象
Class clazz = Class.forName("cn.itcast.bean.Person");
//调用字节码文件对象的方法getMethod获取class对象所表示的类的公共成员方法(指定方法)，参数为方法名和当前方法的参数，无需创建对象，它是静态方法
Method method = clazz.getMethod("staticShow", null);
//调用class对象所表示的类的公共成员方法，需要指定对象和方法中的参数列表
method.invoke(null, null);
```

……

```
Class clazz = Class.forName("cn.itcast.bean.Person");
    //获取指定方法。
Method method = clazz.getMethod("publicShow", null);
    //获取指定的类对象。
Object obj = clazz.newInstance();
method.invoke(obj, null);//对哪个对象调用方法，是参数组
```

好处：大大的提高了程序的扩展性。

例子：

```
//8.创建一个map集合
    private HashMap<String, String> data = new HashMap<String,String>();
```

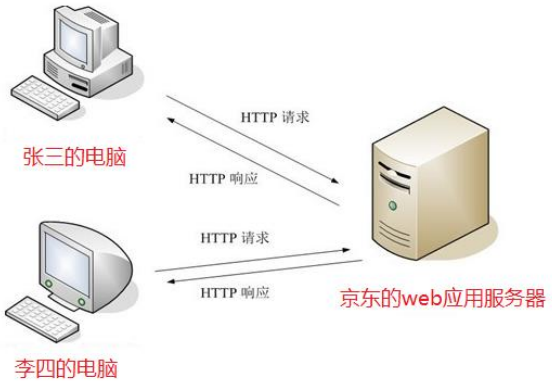
```
@Before
public void testReadWEBXml(){
    try {
        //1.创建解析器对象
        SAXReader saxReader = new SAXReader();
        //2.使用解析器加载web.xml文件得到document对象
        Document document = saxReader.read("src/cn/itheima/web/servlet1/web.xml");
        //3.获取根元素节点
        Element rootElement = document.getRootElement();
        //4.获取子节点(servlet和servlet-mapping)
        List<Element> childElements = rootElement.elements();
        //5.遍历
        for (Element element : childElements) {
            //6.判断元素的名称为servlet的元素节点
            if("servlet".equals(element.getName())){
                //7.分别获取servlet元素节点的servlet-name和servlet-class的值
                String servletName = element.element("servlet-name").getText();
                String servletClass = element.element("servlet-class").getText();
                /*System.out.println(servletName);
                System.out.println(servletClass);*/
                data.put(servletName, servletClass);
            }
            //9.判断元素的名称为servlet-mapping的元素节点
            if("servlet-mapping".equals(element.getName())){
                //10.分别获取servlet元素节点的servlet-name和servlet-class的值
                String servletName = element.element("servlet-name").getText();
                String urlPattern = element.element("url-pattern").getText();
                //11.将servletName作为key来获取servletClass的值
                String servletClass = data.get(servletName);
                //12.将url-pattern作为key,servletClass作为value存到map中去
                data.put(urlPattern, servletClass);
                //13.移除servletName
                data.remove(servletName);
            }
        }
        //System.out.println(data);
    } catch (DocumentException e) {
        e.printStackTrace();
    }
}
@Test
public void testMyServlet(){
    try {
        //1.模拟在浏览器输入一个url
```

```
            String url1 = "/myServlet2";
            //2.将urlPattern作为key来获取servletClass
            String className = data.get(url1);
            //3.通过servletClass获取字节码文件
            Class clazz = Class.forName(className);
            //4.通过字节码文件创建实例对象
            Object obj = clazz.newInstance();
            //5.通过字节码文件获取方法(两个参数：第一个是方法名称；第二个参数是方法的参数)
            Method method = clazz.getMethod("service", null);
            //6.调用invoke方法执行实例对象里面的方法(前面写的方法init)【两个参数：第一个是调用方法的实例对象，第二个是方法的实参】
            method.invoke(obj, null);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Http 协议和 Tomcat 服务器

Http 协议的组成

Http 协议由 Http 请求和 Http 响应组成，当在浏览器中输入网址访问某个网站时，你的浏览器会将你的请求封装成一个 Http 请求发送给服务器站点，服务器接收到请求后会组织响应数据封装成一个 Http 响应返回给浏览器。即没有请求就没有响应。



点击提交按钮，抓包如下：

Http 请求

1）请求行

请求方式：POST、GET

请求的资源：/DemoEE/form.html

协议版本：HTTP/1.1

HTTP/1.0，发送请求，创建一次连接，获得一个 web 资源，连接断开。

HTTP/1.1，发送请求，创建一次连接，获得多个 web 资源，保持连接。

2）请求头

请求头是客户端发送给服务器端的一些信息，使用键值对表示 key：value

常见请求头	描述 （红色掌握，其他了解）
Referer	浏览器通知服务器，当前请求来自何处。如果是直接访问，则不会有这个头。常用于：防盗链
If-Modified-Since	浏览器通知服务器，本地缓存的最后变更时间。与另一个响应头组合控制浏览器页面的缓存。
Cookie	与会话有关技术，用于存放浏览器缓存的 cookie 信息。
User-Agent	浏览器通知服务器，客户端浏览器与操作系统相关信息
Connection	保持连接状态。Keep-Alive 连接中，close 已关闭
Host	请求的服务器主机名
Content-Length	请求体的长度
Content-Type	如果是 POST 请求，会有这个头，默认值为 application/x-www-form-urlencoded，表示请求体内容使用 url 编码
Accept:	浏览器可支持的 MIME 类型。文件类型的一种描述方式。 MIME 格式：大类型/小类型[;参数] 例如： text/html ， html 文件 text/css，css 文件 text/javascript，js 文件 image/*，所有图片文件
Accept-Encoding	浏览器通知服务器，浏览器支持的数据压缩格式。如：GZIP 压缩
Accept-Language	浏览器通知服务器，浏览器支持的语言。各国语言（国际化 i18n）

3）请求体

当请求方式是 post 的时，请求体会有请求的参数，格式如下：

username=zhangsan&password=123

如果请求方式为 get，那么请求参数不会出现在请求体中，会拼接在 url 地址后面

http://localhost:8080...?username=zhangsan&password=123

Http 响应

1）响应行

Http 协议

状态码：

常用的状态码如下：

200 ： 请求成功。

302 ： 请求重定向。

304 ： 请求资源没有改变，访问本地缓存。

404 ： 请求资源不存在。通常是用户路径编写错误，也可能是服务器资源已删除。

500 ： 服务器内部错误。通常程序抛异常。

状态信息：状态信息是根据状态码变化而变化的

2）响应头

响应也都是键值对形式，服务器端将信息以键值对的形式返回给客户端

常见请求头	描述
Location	指定响应的路径，需要与状态码 302 配合使用，完成跳转。
Content-Type	响应正文的类型（MIME 类型） 取值：text/html;charset=UTF-8
Content-Disposition	通过浏览器以下载方式解析正文 取值：attachment;filename=xx.zip
Set-Cookie	与会话相关技术。服务器向浏览器写入 cookie
Content-Encoding	服务器使用的压缩格式 取值：gzip
Content-length	响应正文的长度
Refresh	定时刷新，格式：秒数;url=路径。url 可省略，默认值为当前页。 取值：3;url=www.itcast.cn　　//三秒刷新页面到 www.itcast.cn
Server	指的是服务器名称，默认值：Apache-Coyote/1.1。可以通过 conf/server.xml 配置进行修改。<Connector port="8080" ... server="itcast"/>
Last-Modified	服务器通知浏览器，文件的最后修改时间。与 **If-Modified-Since** 一起使用。

3）响应体

响应体是服务器回写给客户端的页面正文，浏览器将正文加载到内存，然后解析渲染显示页面内容

Tomcat 服务器

➢ Web 开发中的常见概念

B/S 系统和 C/S 系统

Brower/Server：浏览器 服务器 系统 ----- 网站

Client/Server：客户端 服务器 系统 ----- QQ、飞秋、大型游戏

web 应用服务器

供向外部发布 web 资源的服务器软件

web 资源

存在于 web 应用服务器可供外界访问的资源就是 web 资源

例如：存在于 web 应用服务器内部的 Html、Css、js、图片、视频等

1）静态资源：指 web 页面中供人们浏览的数据始终是不变。比如：HTML、CSS、 JS、图片、多媒体。

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

javaWEB 领域：动态资源认为通过 java 代码去动态生成 html

➢ Web 开发中常用的 web 应用服务器

1）weblogic：oracle 公司的大型收费 web 服务器 支持全部 javaEE 规范

2）websphere：IBM 公司的大型收费 web 服务器 支持全部的 javaEE 规范

3）Tomcat：Apache 开源组织下的开源免费的中小型的 web 应用服务器支持 javaEE 中的 servlet 和 jsp 规范

Tomcat 启动不成功的原因分析：

如果没有配置 JAVA_HOME 环境变量，在双击“startup.bat”文件运行 tomcat 时，将一闪立即关闭。且必须配置正确，及 JAVA_HOME 指向 JDK 的安装目录

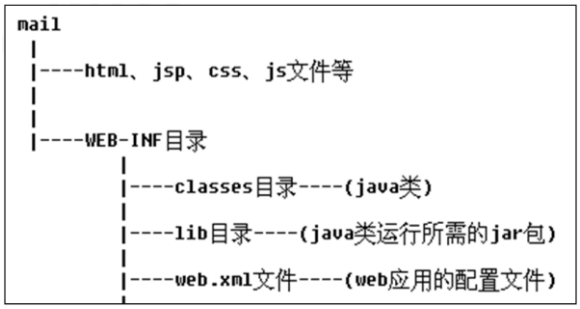
端口冲突

java.net.BindException: Address already in use: JVM_Bind <null>:8080

修改 Tomcat/conf/server.xml

```
-->
<Connector port="9999" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
```

➢ Web 应用的目录结构



注意：WEB-INF 目录是受保护的，外界不能直接访问

JavaWeb 核心之 Servlet

什么是 Servlet

Servlet 运行在服务端的 Java 小程序，是 sun 公司提供一套规范（接口），用来处理客户端请求、响应给浏览器的动态资源。但 servlet 的实质就是 java 代码，通过 java 的 API 动态的向客户端输出内容

servlet 规范：包含三个技术点

1）servlet 技术

2）filter 技术---过滤器

3）listener 技术---监听器

Servlet 快速入门

实现步骤：

1）创建类实现 Servlet 接口

2）覆盖尚未实现的方法---service 方法

3）在 web.xml 进行 servlet 的配置

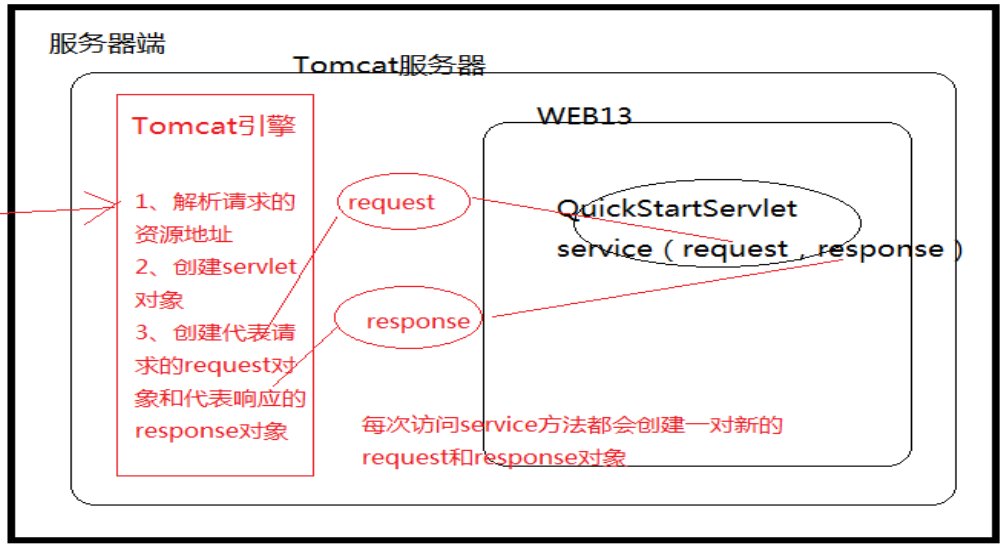
但在实际开发中，我们不会直接去实现 Servlet 接口，因为那样需要覆盖的方法太多，我们一般创建类继承 HttpServlet

实现步骤：

1）创建类继承 HttpServlet 类

2）覆盖 doGet 和 doPost

3）在 web.xml 中进行 servlet 的配置



Servlet 的 API（生命周期）

➢ Servlet 接口中的方法

1）init(ServletConfig config)

何时执行：servlet 对象创建的时候执行

ServletConfig ： 代表的是该 servlet 对象的配置信息

2）service（ServletRequest request,ServletResponse response）

何时执行：每次请求都会执行

ServletRequest ： 代表请求 认为 ServletRequest 内部封装的是 http 请求的信息

ServletResponse ： 代表响应 认为要封装的是响应的信息

3）destroy()

何时执行：servlet 销毁的时候执行

➢ HttpServlet 类的方法

1）init()

2）doGet(HttpServletRequest request,HttpServletResponse response)

3）doPost(HttpServletRequest request,HttpServletResponse response)

4）destroy()

➢ Servlet 的生命周期（面试题）

1）Servlet 何时创建

默认第一次访问 servlet 时创建该对象

2）Servlet 何时销毁

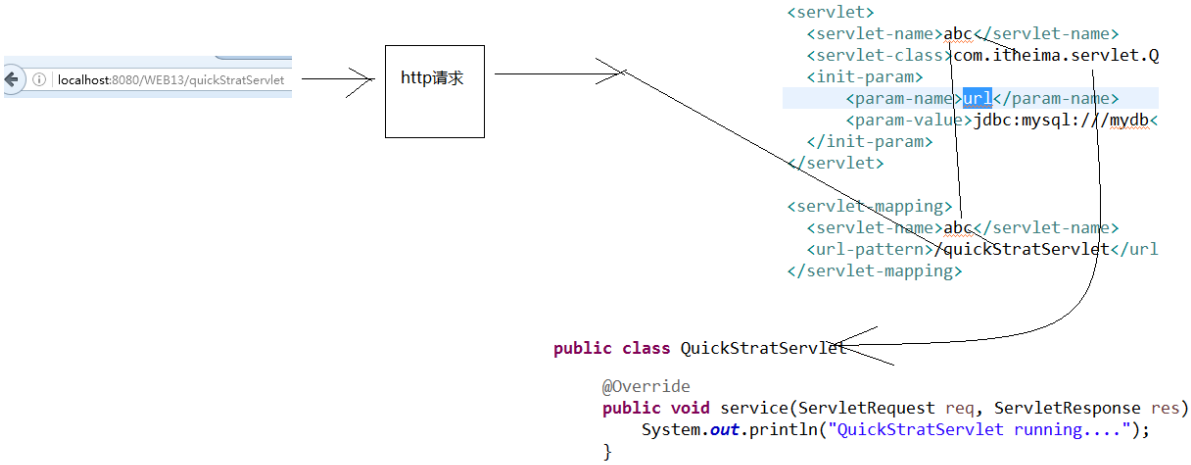
服务器关闭 servlet 就销毁了

3）每次访问必然执行的方法

service(ServletRequest req, ServletResponse res)方法

问题：对 XXXServlet 进行了 10 次访问，init()，destory()，service()，doGet()，doPost() 一共执行力多少次？request 对象创建几个？response 创建几个？

网页中 http 请求的过程：



Servlet 的配置

- **基本配置**

```
<!-- servlet的类的配置 -->
<servlet>
  <servlet-name>abc</servlet-name>
  <servlet-class>com.itheima.servlet.QuickStratServlet</se
  <init-param>
      <param-name>url</param-name>
      <param-value>jdbc:mysql:///mydb</param-value>
  </init-param>
</servlet>
<!-- servlet的虚拟路径的配置 -->
<servlet-mapping>
  <servlet-name>abc</servlet-name>
  <url-pattern>/quickStratServlet</url-pattern>
</servlet-mapping>
```

其中 url-pattern 的配置方式：

1）完全匹配 访问的资源与配置的资源完全相同才能访问到

```
<url-pattern>/quickStratServlet</url-pattern>
```

2）目录匹配 格式：/虚拟的目录../* *代表任意

```
<url-pattern>/aaa/bbb/ccc/*</url-pattern>
```

3）扩展名匹配 格式：*.扩展名

```
<url-pattern>*.abcd</url-pattern>
```

注意：第二种与第三种不要混用 **/aaa/bbb/*.abcd（错误的）**

- **服务器启动实例化 Servlet 配置**

Servlet 的何时创建：**默认**第一次访问时创建
为什么是默认？
当在 servlet 的配置时 加上一个配置 <load-on-startup> servlet 对象在服务器启动时就创建

- **缺省 Servlet**

可以将 url-pattern 配置一个/，代表该 servlet 是缺省的 servlet
什么是缺省的 servlet?
当你访问资源地址所有的 servlet 都不匹配时 ， 缺省的 servlet 负责处理
其实，web 应用中所有的资源的响应都是 servlet 负责，包括静态资源

- **欢迎页面**

```
<welcome-file-list>
  <welcome-file>1.html</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>default.html</welcome-file>
  <welcome-file>default.htm</welcome-file>
  <welcome-file>default.jsp</welcome-file>
</welcome-file-list>
```

ServletContext 对象

- **什么是 ServletContext 对象**

ServletContext 代表是一个 web 应用的环境（上下文）对象，ServletContext 对象内部封装是该 web 应用的信息，ServletContext 对象一个 web 应用只有一个
问题：一个 web 应用有几个 servlet 对象？----多个
ServletContext 对象的生命周期?
创建：该 web 应用被加载（服务器启动或发布 web 应用（前提，服务器启动状态））

销毁：web 应用被卸载（服务器关闭，移除该 web 应用）

- **怎样获得 ServletContext 对象**

1）ServletContext servletContext = config.getServletContext();（一般不用）

2）ServletContext servletContext = this.getServletContext();

- **ServletContext 的作用**

获得 web 应用全局的初始化参数

```
<!-- 配置全局的初始化参数 -->
<context-param>
  <param-name>driver</param-name>
  <param-value>com.mysql.jdbc.Driver</param-value>
</context-param>
```

web.xml 中配置初始化参数

```
//获得ServletContext对象
ServletContext context = getServletContext();
//1、获得初始化参数
String initParameter = context.getInitParameter("driver");
System.out.println(initParameter);
```

通过 context 对象获得参数

获得 web 应用中任何资源的绝对路径（重要 重要 重要）

方法：String path = context.getRealPath(相对于该 web 应用的相对地址);

ServletContext 是一个域对象（重要 重要 重要）

什么是域对象？什么是域？
存储数据的区域就是域对象
ServletContext 域对象的作用范围：整个 web 应（所有的 web 资源都可以随意向 servletcontext 域中存取数据，数据可以共享）
域对象的通用的方法：
setAtrribute(String name,Object obj);
getAttribute(String name);
removeAttribute(String name);

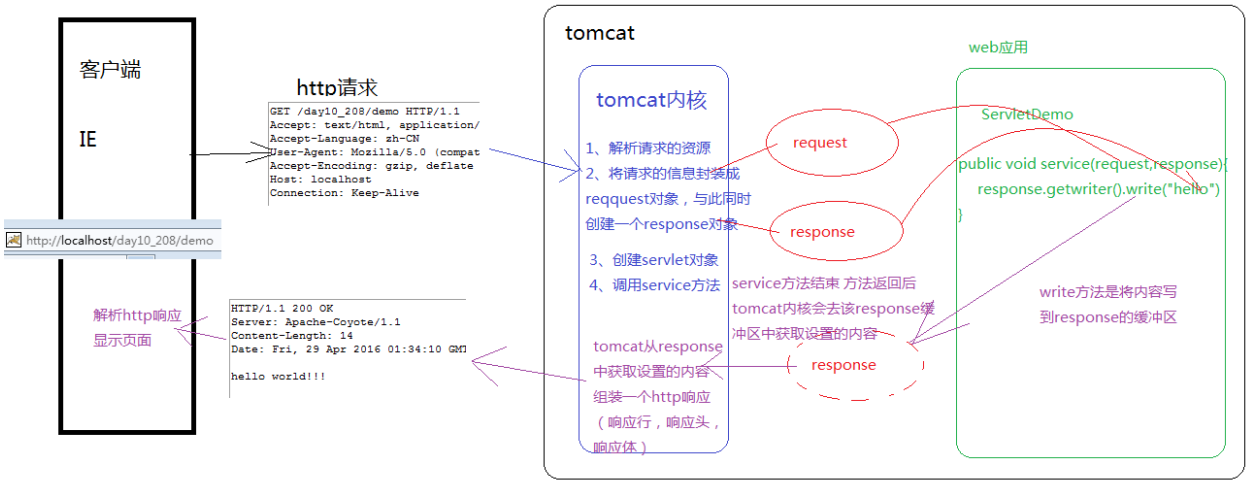
HttpServletResponse

HttpServletResponse 概述

我们在创建 Servlet 时会覆盖 service()方法，或 doGet()/doPost(),这些方法都有两个参数，一个为代表请求的 request 和代表响应 response。

service 方法中的 response 的类型是 ServletResponse，而 doGet/doPost 方法的 response 的类型是 HttpServletResponse，HttpServletResponse 是 ServletResponse 的子接口，功能和方法更加强大，今天我们学习 HttpServletResponse。

response 的运行流程



通过抓包工具抓取 Http 响应

Http响应

Http响应行

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
ETag: W/"312-1467289802502"
Last-Modified: Thu, 30 Jun 2016 12:30:02 GMT
Content-Type: text/html
Content-Length: 312
Date: Thu, 30 Jun 2016 12:31:12 GMT
```

Http响应头

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <form action="#" method="post">
        <input type="text" name="username"><br>
        <input type="password" name="password"><br>
        <input type="submit" value="娴嬭瘯http璇锋眰"><br>
    </form>
</body>
</html>
```

Http响应体

因为 response 代表响应，所以我们可以通过该对象分别设置 Http 响应的响应行，响应头和响应体

通过 response 设置响应行

设置响应行的状态码

setStatus(int sc)

通过 response 设置响应头

addHeader(String name, String value)
addIntHeader(String name, int value)
addDateHeader(String name, long date)

setHeader(String name, String value)

setDateHeader(String name, long date)
setIntHeader(String name, int value)
其中，add 表示添加，而 set 表示设置

- **重定向：**

状态啊：302　　响应头：location 代表重定向的地址

```
//没有响应头，告知客户端去重定向到ServletCDX2
//1.设置状态码302
response.setStatus(302);
//2.设置响应头loction
response.setHeader("Location", "/HttpResponseStudy/ServletCDX2");
```

java 自己封装了重定向方法：sebdRedirect（url）

```
response.sendRedirect("/HttpResponseStudy/ServletCDX2");
```

定时刷新的头（重定向）：refresh，5 代表 5 秒后跳转

```
response.setHeader(“refresh”,“5;url=http://www.baidu.com”)
```

- **例子：页面跳转定时器：**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
<script type="text/javascript">
    window.onload=function(){
```

```
            var time = 5
            var secondEle = document.getElementById("second");
            var timer = setInterval(() => {
                secondEle.innerHTML = time;
                time--;
                if(time==0){
                    clearInterval(timer);
                    location.href="http://www.baidu.com";
                }
            }, 1000);
        }
    </script>
    </head>
    <body>
        恭喜你，注册成功，<span style="color:red" id="second">5</span>秒后跳转，如不跳转点击<a href="http://www.baidu.com">这里！</a>
    </body>
    </html>
```

通过 response 设置响应体

- **响应体设置文本**

PrintWriter getWriter()

获得字符流，通过字符流的 write(String s)方法可以将字符串设置到 response 缓冲区中，随后 Tomcat 会将 response 缓冲区中的内容组装成 Http 响应返回给浏览器端。

关于设置中文的乱码问题

原因：response 缓冲区的默认编码是 iso8859-1，此码表中没有中文，可以通过 response 的 setCharacterEncoding(String charset) 设置 response 的编码

但我们发现客户端还是不能正常显示文字

原因：我们将 response 缓冲区的编码设置成 UTF-8，但浏览器的默认编码是本地系统的编码，因为我们都是中文系统，所以客户端浏览器的默认编码是 GBK，我们可以手动修改浏览器的编码是 UTF-8。

我们还可以在代码中指定浏览器解析页面的编码方式，

通过 response 的 setContentType(String type)方法指定页面解析时的编码是 UTF-8

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

上面的代码不仅可以指定浏览器解析页面时的编码，同时也内含 setCharacterEncoding 的功能，所以在实际开发中只要编

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

就可以解决页面输出中文乱码问题。

- **响应头设置字节**

  ServletOutputStream getOutputStream()
  获得字节流，通过该字节流的 write(byte[] bytes)可以向 response 缓冲区中写入字节，在由 Tomcat 服务器将字节内容组成 Http 响应返回给浏览器。

代码例子:

```
//使用response获得字节输出流
ServletOutputStream outputStream = response.getOutputStream();

//获取服务器上的图片存为字节流
String realPath = this.getServletContext().getRealPath("a.JPG");
InputStream in = new FileInputStream(realPath);

int len = 0;
byte[] buffer = new byte[1024];
while((len=in.read(buffer))>0) {
    outputStream.write(buffer, 0, len);
}
in.close();
outputStream.close();
```

案例-完成文件下载

问题:

1) 什么情况下会文件下载?
   浏览器不能解析的文件就会下载
2) 什么情况下需要在服务器端编写文件下载的代码?
   理论上，浏览器可以解析的文件需要编写文件下载代码
   实际开发中，只要是下载的文件都编写文件下载代码

文件下载的实质就是文件拷贝，将文件从服务器端拷贝到浏览器端。所以文件下载需要 IO 技术将服务器端的文件使用 InputStream 读取到，在使用 ServletOutputStream 写到 response 缓冲区中

代码如下：

```
//1、获得download文件夹下的a.mp3文件的绝对路径
String path = getServletContext().getRealPath("download/a.jpg");
//2、使用io读取到该文件
InputStream in = new FileInputStream(path);
//3、将该文件写到response的缓冲区中
OutputStream out = response.getOutputStream();
int len = 0;
byte[] buffer = new byte[1024];
while((len=in.read(buffer))>0){
    out.write(buffer, 0, len);
}
//4、关闭资源
in.close();
//out.close();//response获得流不用手动关闭 会自动关闭
```

上述代码可以将图片从服务器端传输到浏览器，但浏览器直接解析图片显示在页面上，而不是提供下载，我们需要设置两个响应头，告知浏览器文件的类型和文件的打开方式。

1）告知浏览器文件的类型：response.setContentType(文件的 MIME 类型);

2）告示浏览器文件的打开方式是下载：

response.setHeader("Content-Disposition","attachment;filename=文件名称");

代码如下：

```
String filename = request.getParameter("filename");

//告诉客户端要下载的这个文件的类型-----客户端通过文件的MIME类型区分类型
response.setContentType(this.getServletContext().getMimeType(filename));

//告诉客户端该文件不是直接解析 而是以附件形式打开----下载
response.setHeader("Content-Disposition", "attachment;filename="+filename);

String realPath =
            this.getServletContext().getRealPath("download/"+filename);
InputStream in = new FileInputStream(realPath);

//获得输出流-----通过response获得输出流用于向客户端写内容
ServletOutputStream out = response.getOutputStream();

int len = 0;
byte[] buffer = new byte[1024];
while(((len=in.read(buffer))>0)) {
    out.write(buffer, 0, len);
}
in.close();

//out.close();
```

但是，如果下载中文文件，页面在下载时会出现中文乱码或不能显示文件名的情况，原因是不同的浏览器默认对下载文件的编码方式不同，ie 是 UTF-8 编码方式，而火狐浏览器是 Base64 编码方式。所里这里需要解决浏览器兼容性问题，解决浏览器兼容性问题的首要任务是要辨别访问者是 ie 还是火狐（其他），通过 Http 请求体中的一个属性可以辨别

```
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;
WOW64; Trident/5.0)
```

```
User-Agent: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0"
```

解决乱码方法如下（不要记忆--了解）：其中 agent 就是请求头 User-Agent 的值

```
//获得要下载的文件的名称
String filename = request.getParameter("filename");                    //中文.jpg
//解决获得中文参数的乱码
filename = new String(filename.getBytes("ISO8859-1"),"UTF-8");    //中文.jpg
```

```
String filename = "美女.jpg";

//获得浏览器的User-Agent
String agent = request.getHeader("User-Agent");
if (agent.contains("MSIE")) {
    // IE浏览器
    filename = URLEncoder.encode(filename, "utf-8");
    filename = filename.replace("+", " ");
} else if (agent.contains("Firefox")) {
    // 火狐浏览器
    BASE64Encoder base64Encoder = new BASE64Encoder();
    filename = "=?utf-8?B?"
            + base64Encoder.encode(filename.getBytes("utf-8")) + "?=";
} else {
    // 其它浏览器
    filename = URLEncoder.encode(filename, "utf-8");
}

//0、设置两个文件下载的头
response.setContentType(getServletContext().getMimeType("美女.jpg"));
response.setHeader("Content-Disposition", "attachment;filename="+filename);
```

response 细节点

1、 response 获得的流不需要手动关闭，web 容器（tomcat）会帮我们关闭
2、 getWriter 和 getOutPutStream 不能同时调用

HttpServletRequest

HttpServletRequest 概述

我们在创建 Servlet 时会覆盖 service()方法，或 doGet()/doPost(),这些方法都有两

个参数，一个为代表请求的 request 和代表响应 response。

service 方法中的 request 的类型是 ServletRequest，而 doGet/doPost 方法的 request 的类型是 HttpServletRequest，HttpServletRequest 是 ServletRequest 的子接口，功能和方法更加强大，今天我们学习 HttpServletRequest。

通过抓包工具抓取 Http 请求

因为 request 代表请求，所以我们可以通过该对象分别获得 Http 请求的请求行，请求头和请求体

通过 request 获得请求行

获得客户端的请求方式：String getMethod()

获得请求的资源：

String getRequestURI()

StringBuffer getRequestURL()

String getContextPath() ---web 应用的名称

String getQueryString() ---- get 提交 url 地址后的参数字符串

username=zhangsan&password=123

注意：request 获得客户机（客户端）的一些信息

request.getRemoteAddr() --- 获得访问的客户端 IP 地址

通过 request 获得请求头

long getDateHeader(String name)

String getHeader(String name)

Enumeration getHeaderNames()
Enumeration getHeaders(String name)
int getIntHeader(String name)

referer 头的作用：执行此次访问的的来源做防盗链

通过 request 获得请求体

请求体中的内容是通过 post 提交的请求参数，格式是：
username=zhangsan&password=123&hobby=football&hobby=basketball

key	value
username	[zhangsan]
password	[123]
hobby	[football，basketball]

以上面参数为例，通过一下方法获得请求参数：

String getParameter(String name)
String[] getParameterValues(String name)
Enumeration getParameterNames()
Map<String,String[]> getParameterMap()

注意：get 请求方式的请求参数 上述的方法一样可以获得
解决 post 提交方式的乱码：request.setCharacterEncoding("UTF-8");
解决 get 提交的方式的乱码：

```
parameter = new String(parameter.getbytes("iso8859-1"),"utf-8");
```

request 的其他功能

- **request 是一个域对象**

request 对象也是一个存储数据的区域对象，所以也具有如下方法：
setAttribute(String name, Object o)
getAttribute(String name)
removeAttribute(String name)
注意：request 域的作用范围：一次请求中

- **request 完成请求转发**

获得请求转发器----path 是转发的地址

RequestDispatcher getRequestDispatcher(String path)

通过转发器对象转发

requestDispathcer.forward(ServletRequest request, ServletResponse response)

注意：ServletContext 域与 Request 域的生命周期比较?

ServletContext：

创建：服务器启动
销毁：服务器关闭
域的作用范围：整个 web 应用

request：

创建：访问时创建 request
销毁：响应结束 request 销毁
域的作用范围：一次请求中

注意：转发与重定向的区别?

1）重定向两次请求，转发一次请求
2）重定向地址栏的地址变化，转发地址不变
3）重新定向可以访问外部网站 转发只能访问内部资源
4）转发的性能要优于重定向

注意：客户端地址与服务器端地址的写法?

客户端地址：是客户端去访问服务器的地址，服务器外部的地址，特点：写上 web 应用名称，直接输入地址，重定向

服务器端地址：服务器内部资源的跳转的地址，特点：不需要写 web 应用的名称，转发

总结：

request 获得行的内容
request.getMethod()
request.getRequestURI()
request.getRequestURL()
request.getContextPath()
request.getRemoteAddr()
request 获得头的内容
request.getHeader(name)
request 获得体（请求参数）
String request.getParameter(name)
Map<String,String[]> request.getParameterMap();
String[] request.getParameterValues(name);
注意：客户端发送的参数 到服务器端都是字符串
获得中文乱码的解决：
post:request.setCharacterEncoding(“UTF-8”);

get:
parameter = new String(parameter.getBytes(“iso8859-1”),”UTF-8”);
request 转发和域
request.getRequestDispatcher(转发的地址).forward(req,resp);
request.setAttribute(name,value)
request.getAttribute(name)
乱码问题:

```
//设置request的编码防止乱码------只适合  post 方式
request.setCharacterEncoding("UTF-8");
//get方式的乱码解决方法      先用iso8859-1编码，再用utf-8解码
String username = request.getParameter("user");
username = new String(username.getBytes("iso8859-1"),"UTF-8");
```

会话技术 Cookie&Session

存储客户端的状态

由一个问题引出今天的内容，例如网站的购物系统，用户将购买的商品信息存储到哪里？因为 Http 协议是无状态的，也就是说每个客户访问服务器端资源时，服务器并不知道该客户端是谁，所以需要会话技术识别客户端的状态。会话技术是帮助服务器记住客户端状态（区分客户端）

会话技术

从打开一个浏览器访问某个站点，到关闭这个浏览器的整个过程，成为一次会话。会　话技术就是记录这次会话中客户端的状态与数据的。

会话技术分为 Cookie 和 Session:

Cookie：数据存储在客户端本地，减少服务器端的存储的压力，安全性不好，客户端可以清除 cookie

Session：将数据存储到服务器端，安全性相对好，增加服务器的压力

Cookie 技术

Cookie 技术是将用户的数据存储到客户端的技术，我们分为两方面学习:

第一，服务器端怎样将一个 Cookie 发送到客户端

第二，服务器端怎样接受客户端携带的 Cookie

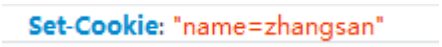
服务器端向客户端发送一个 Cookie

- **创建 Cookie:**

  Cookie cookie = new Cookie(String cookieName,String cookieValue);

  示例:

  Cookie cookie = new Cookie("username", "zhangsan");

  那么该 cookie 会以响应头的形式发送给客户端:

  注意：Cookie 中不能存储中文

- **设置 Cookie 在客户端的持久化时间:**

  cookie.setMaxAge(int seconds); ---时间秒
  注意：如果不设置持久化时间，cookie 会存储在浏览器的内存中，浏览器关闭 cookie 信息销毁（会话级别的 cookie），如果设置持久化时间，cookie 信息会 被持久化到浏览器的磁盘文件里
  示例:
  cookie.setMaxAge(10*60);
  设置 cookie 信息在浏览器的磁盘文件中存储的时间是 10 分钟，过期浏览器自动删除该 cookie 信息

- **设置 Cookie 的携带路径:**

  cookie.setPath(String path);
  注意：如果不设置携带路径，那么该 cookie 信息会在访问产生该 cookie 的 web 资源所在的路径都携带 cookie 信息
  示例:
  cookie.setPath("/WEB16");
  代表访问 WEB16 应用中的任何资源都携带 cookie
  cookie.setPath("/WEB16/cookieServlet");
  代表访问 WEB16 中的 cookieServlet 时才携带 cookie 信息

- 向客户端发送 cookie：

```
response.addCookie(Cookie cookie);
```

- 删除客户端的 cookie：

如果想删除客户端的已经存储的 cookie 信息，那么就使用**同名同路径的持久化时间为 0** 的 cookie 进行覆盖即可

持久化时　　间为 0 的 cookie 进行覆盖即可

服务器端怎么接受客户端携带的 Cookie

cookie 信息是以请求头的方式发送到服务器端的：

```
Cookie: "name=zhangsan"
```

1）通过 request 获得所有的 Cookie：

```
Cookie[] cookies = request.getCookies();
```

2）遍历 Cookie 数组，通过 Cookie 的名称获得我们想要的 Cookie

```
for(Cookie cookie : cookies){
    if(cookie.getName().equal(cookieName)){
        String cookieValue = cookie.getValue();
    }
}
```

Session 技术

Session 技术是将数据存储在服务器端的技术，会为每个客户端都创建一块内存空间　存储客户的数据，但客户端需要每次都携带一个标识 ID 去服务器中寻找属于自己的内存空间。所以说 Session 的实现是基于 Cookie，Session 需要借助于 Cookie 存储客户的唯一性标识 JSESSIONID

在 Session 这我们需要学习如下三个问题：

怎样获得属于本客户端的 session 对象（内存区域）？

怎样向 session 中存取数据（session 也是一个域对象）？

session 对象的生命周期?

获得 Session 对象

```
HttpSession session = request.getSession();
```

此方法会获得专属于当前会话的 Session 对象，如果服务器端没有该会话的 Session 对象会创建一个新的 Session 返回，如果已经有了属于该会话的 Session 直接将已有的 Session 返回（实质就是根据 JSESSIONID 判断该客户端是否在服务器上已经存在 session 了）

怎样向 session 中存取数据（session 也是一个域对象）

Session 也是存储数据的区域对象，所以 session 对象也具有如下三个方法：

session.setAttribute(String name,Object obj);

session.getAttribute(String name);

session.removeAttribute(String name);

Session 对象的生命周期（面试题/笔试题）

创建：第一次执行 request.getSession()时创建

销毁：

1）服务器（非正常）关闭时

2）session 过期/失效（默认 30 分钟）

问题：时间的起算点 从何时开始计算 30 分钟？

从不操作服务器端的资源开始计时

可以在工程的 web.xml 中进行配置

<session-config>

<session-timeout>30</session-timeout>

</session-config>

3）手动销毁 session

session.invalidate();

作用范围：

默认在一次会话中，也就是说在，一次会话中任何资源公用一个 session 对象

面试题：浏览器关闭，session 就销毁了？ 不对

为 session 设置持久化时间

通过cookie来设置

```
Cookie cookie = new Cookie("JSESSIONID", id);
cookie.setPath("/CookieAndSession/");
cookie.setMaxAge(60*10);
response.addCookie(cookie);
```

总结：

- **Cookie 技术：存到客户端**

发送 cookie
    Cookie cookie = new Cookie(name,value)
    cookie.setMaxAge(秒)
    cookie.setPath()
    response.addCookie(cookie)
获得 cookie
    Cookie[] cookies = request.getCookies();
    cookie.getName();
    cookie.getValue();

- **Session 技术：存到服务器端 借助 cookie 存储 JSESSIONID**

HttpSession session = request.getSession();
setAttribute(name,value);
getAttribute(name);
session 生命周期
    创建：第一次指定 request.getSession();
    销毁：服务器关闭、session 失效/过期、手动 session.invalidate();
    session 作用范围：默认一会话中

动态页面技术（JSP/EL/JSTL）

JSP 技术

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

jsp 相对于 html 多的一行

- **jsp 脚本和注释**

jsp 脚本：

1）<%java 代码%> ----- 内部的 java 代码翻译到 **service 方法**的内部

2）<%=java 变量或表达式> ----- 会被翻译成 **service 方法内部 out.print()**

3）<%!java 代码%> ---- 会被翻译成 **servlet 的成员**的内容，里面可以写方法

jsp 注释： 不同的注释可见范围是不同

1）Html 注释：<!--注释内容--> ---可见范围 jsp 源码、翻译后的 servlet、页面显示 html 源码

2）java 注释：//单行注释 /*多行注释*/ --可见范围 jsp 源码 翻译后的 servlet

3）jsp 注释：<%--注释内容--%> ----- 可见范围 jsp 源码可见

➢ jsp 运行原理-----jsp 本质就是 servlet（面试）

jsp 在第一次被访问时会被 Web 容器翻译成 servlet，再执行

过程：

第一次访问---->helloServlet.jsp---->helloServlet_jsp.java---->编译运行

PS：被翻译后的 servlet 在 Tomcat 的 work 目录中可以找到

➢ jsp 指令（3 个）

jsp 的指令是指导 jsp 翻译和运行的命令，jsp 包括三大指令：

1）page 指令 --- 属性最多的指令（实际开发中 page 指令默认）

属性最多的一个指令，根据不同的属性，指导整个页面特性

格式：<%@ page 属性名 1= "属性值 1" 属性名 2= "属性值 2" ...%>

常用属性如下：

language：jsp 脚本中可以嵌入的语言种类

pageEncoding：当前 jsp 文件的本身编码---内部可以包含 contentType

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

session：是否 jsp 在翻译时自动创建 session

import：导入 java 的包

errorPage：当当前页面出错后跳转到哪个页面

isErrorPage：当前页面是一个处理错误的页面

2）include 指令

页面包含（静态包含）指令，可以将一个 jsp 页面包含到另一个 jsp 页面中

格式：<%@ include file="被包含的文件地址"%>

3）taglib 指令

在 jsp 页面中引入标签库（jstl 标签库、struts2 标签库）

格式：<%@ taglib uri="标签库地址" prefix="前缀"%>

jsp 内置/隐式对象（9 个）----- 笔试

jsp 被翻译成 servlet 之后，service 方法中有 9 个对象定义并初始化完毕，我们在 jsp 脚本中可以直接使用这 9 个对象

名称	类型	描述
out	javax.servlet.jsp.JspWriter	用于页面输出
request	javax.servlet.http.HttpServletRequest	得到用户请求信息，
response	javax.servlet.http.HttpServletResponse	服务器向客户端的回应信息
config	javax.servlet.ServletConfig	服务器配置，可以取得初始化参数
session	javax.servlet.http.HttpSession	用来保存用户的信息
application	javax.servlet.ServletContext	所有用户的共享信息
page	java.lang.Object	指当前页面转换后的 Servlet 类的实例
pageContext	javax.servlet.jsp.PageContext	JSP 的页面容器
exception	java.lang.Throwable	表示 JSP 页面所发生的异常，在错误页中才起作用

➢ out 对象

out 的类型：JspWriter
out 作用就是想客户端输出内容----out.write()
out 缓冲区默认 8kb 可以设置成 0 代表关闭 out 缓冲区 内容直接写到 respons 缓冲器

➢ pageContext 对象

jsp 页面的上下文对象，作用如下：
page 对象与 pageContext 对象不是一回事

1）pageContext 是一个域对象

setAttribute(String name,Object obj)

getAttribute(String name)

removeAttrbute(String name)

pageContext 可以向指定的其他域中存取数据

setAttribute(String name,Object obj,int scope)

getAttribute(String name,int scope)

removeAttrbute(String name,int scope)

findAttribute(String name)

---依次从 pageContext 域，request 域，session 域，application 域中获取属性，在某个域中获取后将不在向后寻找

四大作用域的总结：

page 域：当前 jsp 页面范围

request 域：一次请求

session 域：一次会话

application 域：整个 web 应用

2）可以获得其他 8 大隐式对象

例如： pageContext.getRequest()

pageContext.getSession()

jsp 标签（动作）

1）页面包含（动态包含）：<jsp:include page="被包含的页面"/>

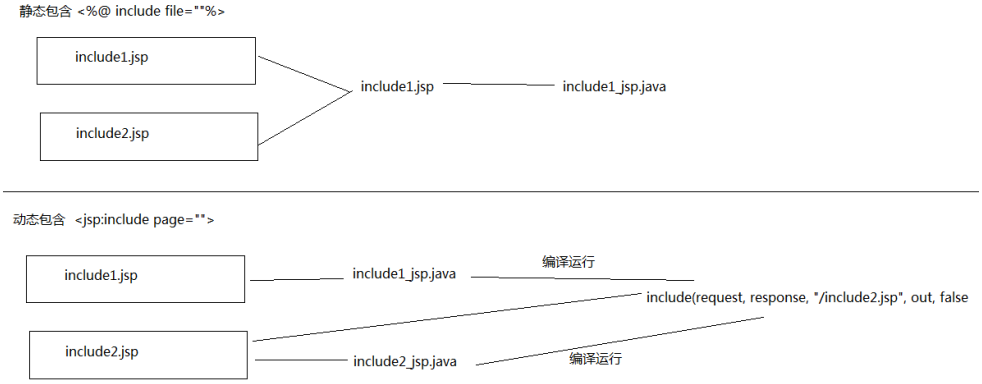
页面包含（静态包含）：<%@ include file="被包含的文件地址"%>

2）请求转发：<jsp:forward page="要转发的资源" />

静态包含与动态包含的区别?

静态：合并后一起编译

动态：边运行边编译，一步一步的



EL 技术

➢ EL 表达式概述

EL（Express Lanuage）表达式可以嵌入在 jsp 页面内部，减少 jsp 脚本的编写，EL 出现的目的是要替代 jsp 页面中脚本的编写。

- **EL 从域中取出数据（最重要作用）**

  jsp 脚本：<%=request.getAttribute(name)%>
  EL 表达式替代上面的脚本：${requestScope.name}

  EL 最主要的作用是获得四大域中的数据，格式**${EL 表达式}**
  EL 获得 pageContext 域中的值：${pageScope.key};
  EL 获得 request 域中的值：${requestScope.key};
  EL 获得 session 域中的值：${sessionScope.key};
  EL 获得 application 域中的值：${applicationScope.key};
  EL 从四个域中获得某个值${key};
  ---同样是依次从 pageContext 域，request 域，session 域，application 域中获取属性，在某个域中获取后将不在向后寻找

- **EL 的内置对象　11 个**

pageScope,requestScope,sessionScope,applicationScope
　　---- 获取 JSP 中域中的数据
param,paramValues　- 接收参数.
　　相当于 request.getParameter()　rrquest.getParameterValues()
header,headerValues　- 获取请求头信息
　　相当于 request.getHeader(name)
initParam　　- 获取全局初始化参数
　　相当于 this.getServletContext().getInitParameter(name)
cookie　　- WEB 开发中 cookie
　　相当于 request.getCookies()---cookie.getName()---cookie.getValue()
pageContext　　- WEB 开发中的 pageContext.
　　pageContext 获得其他八大对象

${pageContext.request.contextPath}
相当于
<%=pageContext.getRequest().getContextPath%>　这句代码不能实现
获得 WEB 应用的名称

- **EL 执行表达式**

  例如：
  ${1+1}
  ${empty user}
  ${user==null?true:false}

JSTL 技术

➢ JSTL 概述

JSTL（JSP Standard Tag Library），JSP 标准标签库，可以嵌入在 jsp 页面中使用标签的形式完成业务逻辑等功能。jstl 出现的目的同 el 一样也是要代替 jsp 页面中的脚本代码。JSTL 标准标准标签库有 5 个子库，但随着发展，目前常使用的是他的核心库

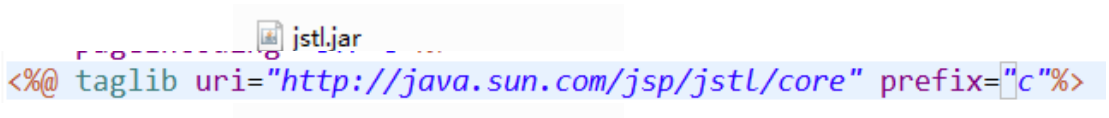
标签库	标签库的 URI	前缀
Core	http://java.sun.com/jsp/jstl/core	c
I18N	http://java.sun.com/jsp/jstl/fmt	fmt
SQL	http://java.sun.com/jsp/jstl/sql	sql
XML	http://java.sun.com/jsp/jstl/xml	x
Functions	http://java.sun.com/jsp/jstl/functions	fn

➢ JSTL 下载与导入

JSTL 下载:

从 Apache 的网站下载 JSTL 的 JAR 包。进入“http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/”网址下载 JSTL 的安装包。jakarta-taglibs-standard-1.1.2.zip，然后将下载好的 JSTL 安装包进行解压，此时，在 lib 目录下可以看到两个 JAR 文件，分别为 jstl.jar 和 standard.jar。其中，jstl.jar 文件包含 JSTL 规范中定义的接口和相关类，standard.jar 文件包含用于实现 JSTL 的.class 文件以及 JSTL 中 5 个标签库描述符文件（TLD）

将两个 jar 包导入我们工程的 lib 中



使用 jsp 的 taglib 指令导入核心标签库

➢ JSTL 核心库的常用标签

1）<c:if test="">标签

其中 test 是返回 boolean 的条件

2）<c:forEach>标签

使用方式有两种组合形式：

```
<!-- forEach模拟
    for(int i=0;i<=5;i++){
        syso(i)
    }
 -->
 <c:forEach begin="0" end="5" var="i">
    ${i }<br/>
 </c:forEach>
```

```
<!-- 模拟增强for    productList---List<Product>
    for(Product product : productList){
        syso(product.getPname());
    }
 -->
 <!-- items:一个集合或数组 var:代表集合中的某一个元素-->
 <c:forEach items="${productList }" var="pro">
    ${pro.pname }
 </c:forEach>
```

示例：

1） 遍历 List<String>的值

```
//模拟List<String> strList
List<String> strList = new ArrayList<String>();
strList.add("itcast");
strList.add("itheima");
strList.add("boxuegu");
strList.add("shandingyu");
request.setAttribute("strList", strList);
```

```
<c:forEach items="${strList }" var="str">
    ${str }<br/>
</c:forEach>
```

2） 遍历 List<User>的值

```
<h1>取出userList的数据</h1>
<c:forEach items="${userList}" var="user">
    user的name：${user.name }------user的password：${user.password }<br/>
</c:forEach>
```

3）遍历 Map<String,String>的值

```
<h1>取出strMap的数据</h1>
<c:forEach items="${strMap }" var="entry">
    ${entry.key }====${entry.value }<br/>
</c:forEach>
```

4）遍历 Map<String,User>的值

```
<h1>取出userMap的数据</h1>
<c:forEach items="${userMap }" var="entry">
    ${entry.key }:${entry.value.name }--${entry.value.password }
</c:forEach>
```

5）遍历 Map<User,Map<String,User>>的值

entry.key-----User

entry.value------List<String,User>

javaEE 的开发模式

什么是模式

模式在开发过程中总结出的“套路”，总结出的一套约定俗成的设计模式

javaEE 经历的模式

model1 模式：

技术组成：jsp+javaBean

model1 的弊端：随着业务复杂性 导致 jsp 页面比较混乱

model2 模式

技术组成：jsp+servlet+javaBean

model2 的优点：开发中 使用各个技术擅长的方面

servlet：擅长处理 java 业务代码

jsp：擅长页面的现实

MVC：---- web 开发的设计模式

M：Model---模型 javaBean：封装数据

V：View-----视图 jsp：单纯进行页面的显示

C：Controller----控制器 ServeIt：获取数据--对数据进行封装--传递数据--指派显示的 jsp 页面

javaEE 的三层架构

服务器开发时 分为三层

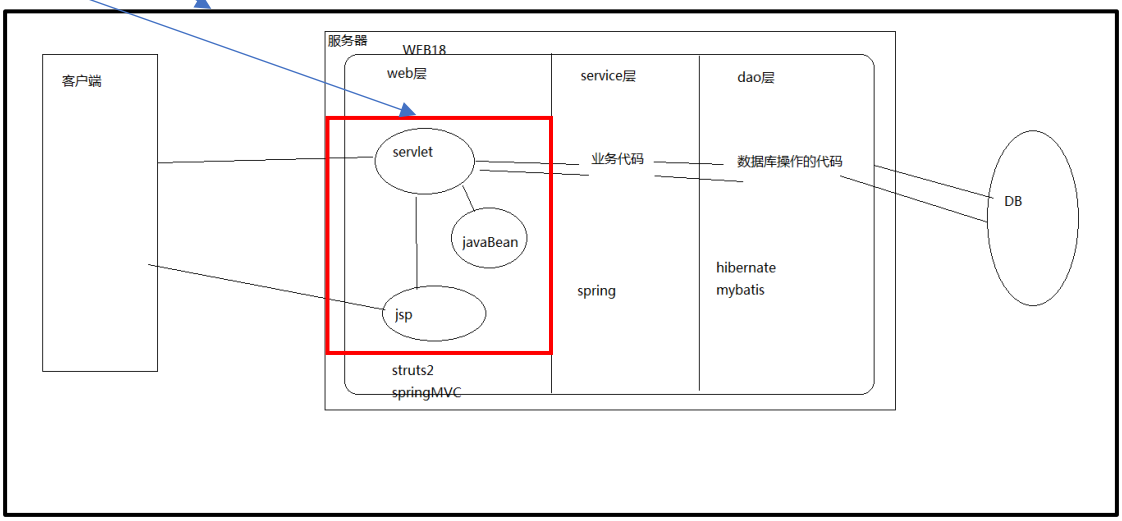
web 层：与客户端交互

service 层：复杂业务处理

dao 层：与数据库进行交互

开发实践时 三层架构通过包结构体现

MVC 与三层架构有什么关系？



总结：

EL 表达式

从域中取出数据 ${域中存储的数据的 name}

${pageContext.request.contextPath}

JSTL 标签(核心库)

<%@ taglib uri="" prefix="c"%>

<c:if test="">

<c:forEach items="数组或集合" var="数组或集合中的每一个元素">

javaEE 三层架构+MVC

web 层：收集页面数据，封装数据，传递数据，指定响应 jsp 页面---般 MVC 写在 web 层。

service 层：逻辑业务代码的编写

dao 层：数据库的访问代码的编写

事务（JDBC）

事务概述

➢ 什么是事务

一件事情有 n 个组成单元 要不这 n 个组成单元同时成功 要不 n 个单元就同时失败

就是将 n 个组成单元放到一个事务中

mysql 的事务

默认的事务：一条 sql 语句就是一个事务 默认就开启事务并提交事务

手动事务：

1）显示的开启一个事务：start transaction

2）事务提交：commit 代表从开启事务到事务提交 中间的所有的 sql 都认为有效 真正的更新数据库

3）事务的回滚：rollback 代表事务的回滚 从开启事务到事务回滚 中间的所有的 sql 操作都认为无效数据库没有被更新

JDBC 事务操作

默认是自动事务：

执行 sql 语句：executeUpdate() ---- 每执行一次 executeUpdate 方法 代表

事务自动提交

通过 jdbc 的 API 手动事务：

开启事务：conn.setAutoComnmit(false);

提交事务：conn.commit();

回滚事务：conn.rollback();

注意：控制事务的 connnection 必须是同一个

执行 sql 的 connection 与开启事务的 connnection 必须是同一个才能对事务进行控制

DBUtils 事务操作

➢ QueryRunner

有参构造：QueryRunner runner = new QueryRunner(DataSource dataSource);

有参构造将数据源(连接池)作为参数传入 QueryRunner，QueryRunner 会从连接池中获得一个数据库连接资源操作数据库，所以直接使用无 Connection 参数的 update 方法即可操作数据库

无参构造：QueryRunner runner = new QueryRunner();

无参的构造没有将数据源（连接池）作为参数传入 QueryRunner，那么我们在使用 QueryRunner 对象操作数据库时要使用有 Connection 参数的方法

使用 ThreadLocal 绑定连接资源

事务的特性和隔离级别（概念性问题---面试）

➢ 事务的特性 ACID

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

2）一致性（Consistency）一个事务中，事务前后数据的完整性必须保持一致。

3）隔离性（Isolation）多个事务，事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离。

4）持久性（Durability）持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

➢ 并发访问问题----由隔离性引起

如果不考虑隔离性，事务存在 3 中并发访问问题。

1）**脏读**：B 事务读取到了 A 事务尚未提交的数据------要求 B 事务要读取 A 事务提交的数据

2）**不可重复读**：一个事务中 两次读取的数据的内容不一致-----要求的是一个事务中多次读取时数据是一致的--- unpdate

3）**幻读/虚读**：一个事务中 两次读取的数据的数量不一致-----要求在一个事务多次读取的数据的数量是一致的--insert delete

➢ 事务的隔离级别

1）read uncommitted：读取尚未提交的数据 ： 哪个问题都不能解决

2）read committed：读取已经提交的数据 ： 可以解决脏读 ---- oracle 默认的

3）repeatable read：重复读取：可以解决脏读 和 不可重复读 ---mysql 默认的

4）serializable：串行化：可以解决 脏读 不可重复读 和 虚读---相当于锁表，性能低，应用的少

注意：mysql 数据库默认的隔离级别

查看 mysql 数据库默认的隔离级别：select @@tx_isolation

```
mysql> select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00 sec)
```

设置 mysql 的隔离级别：set session transaction isolation level 设置事务隔离级别

```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00 sec)

mysql> select @@tx_isolation;
+------------------+
| @@tx_isolation   |
+------------------+
| READ-UNCOMMITTED |
+------------------+
1 row in set (0.00 sec)
```

总结：

mysql 的事务控制：

开启事务：start transaction;

提交：commit;

回滚：rollback;

JDBC 事务控制：

开启事务：conn.setAutoCommit(false);

提交：conn.commit();

回滚：conn.rollback();

DBUtils 的事务控制也是通过 jdbc
ThreadLocal：实现的是通过线程绑定的方式传递参数
概念：
事务的特性 ACID
并发问题：脏读、不可重读、虚读\幻读
解决并发：设置隔离级别
read uncommitted
read committed
repeatable read （mysql 默认）
serialazable
隔离级别的性能：
read uncommitted>read committed>repeatable read>serialazable
安全性：
read uncommitted<read committed<repeatable read<serialazable

Json 数据格式（重要）

json 是一种与语言无关的数据交换的格式，作用：
使用 ajax 进行前后台数据交换
移动端与服务端的数据交换

Json 的格式与解析

json 有两种格式：
1）对象格式：{"key1":obj,"key2":obj,"key3":obj...}
2）数组/集合格式：[obj,obj,obj...]
例如：user 对象 用 json 数据格式表示
{"username":"zhangsan","age":28,"password":"123","addr":"北京"}
List<Product> 用 json 数据格式表示
[{"pid":"10","pname":"小米 4C"},{},{}]
注意：对象格式和数组格式可以互相嵌套
注意：json 的 key 是字符串 jaon 的 value 是 Object
json 的解析：
json 是 js 的原生内容，也就意味着 js 可以直接取出 json 对象中的数据

Json 的转换插件

将 java 的对象或集合转成 json 形式字符串

json 的转换插件是通过 java 的一些工具，直接将 java 对象或集合转换成 json 字符串。
常用的 json 转换工具有如下几种:

1）jsonlib

2）Gson：google

3）fastjson：阿里巴巴

Js 原生 Ajax 和 Jquery 的 Ajax

Ajax 概述

➢ 什么是同步，什么是异步

同步现象：客户端发送请求到服务器端，当服务器返回响应之前，客户端都处于等待 卡死状态

异步现象：客户端发送请求到服务器端，无论服务器是否返回响应，客户端都可以随 意做其他事情，不会被卡死

Ajax 的运行原理

页面发起请求，会将请求发送给浏览器内核中的 Ajax 引擎，Ajax 引擎会提交请求到 服务器端，在这段时间里，客户端可以任意进行任意操作，直到服务器端将数据返回 给 Ajax 引擎后，会触发你设置的事件，从而执行自定义的 js 逻辑代码完成某种页面 1 功能。

js 原生的 Ajax 技术（了解）

js 原生的 Ajax 其实就是围绕浏览器内内置的 Ajax 引擎对象进行学习的，要使用 js 原 生的 Ajax 完成异步操作，有如下几个步骤:

1）创建 Ajax 引擎对象

2）为 Ajax 引擎对象绑定监听（监听服务器已将数据响应给引擎）

3）绑定提交地址

4）发送请求

5）接受响应数据

```
//1、创建引擎对象
var xmlhttp = new XMLHttpRequest();
//2、绑定监听
xmlhttp.onreadystatechange = function(){
    //5、接受响应数据
    if(xmlhttp.readyState==4&&xmlhttp.status==200){
        var res = xmlhttp.responseText;
        alert(res);
    }
}
//3、绑定地址
xmlhttp.open("GET","${pageContext.request.contextPath}/ajax",true);
//4、发送请求
xmlhttp.send();
```

注意：如果是 post 提交
在发送请求之前设置一个头

```
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
```

总结：所用异步访问都是 ajax 引擎

GET 还是 POST

与 POST 相比，GET 更简单也更快，并且在大部分情况下都能用。

然而，在以下情况中，请使用 POST 请求：

- 无法使用缓存文件（更新服务器上的文件或数据库）
- 向服务器发送大量数据（POST 没有数据量限制）
- 发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

➢ GET 请求：

```
xmlhttp.open("GET","demo_get.jsp?t=" + Math.random(),true);
xmlhttp.send();
```

➢ POST 请求：

```
xmlhttp.open("POST","ajax_test.jsp",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");(多加)
xmlhttp.send("fname=Bill&lname=Gates");
```

Jquery 的 Ajax 技术（重点）

jquery 是一个优秀的 js 框架，自然对 js 原生的 ajax 进行了封装，封装后的 ajax 的操作方法更简洁，功能更强大，与 ajax 操作相关的 jquery 方法有如下几种，但开发中经常使用的有三种

```
$.ajax([options])
load(url, [data], [callback])
$.get(url, [data], [fn], [type])
$.getJSON(url, [data], [fn])
$.getScript(url, [callback])
$.post(url, [data], [fn], [type])
```

- **$.get(url, [data], [callback], [type])**

- **$.post(url, [data], [callback], [type])**

其中：

url：代表请求的服务器端地址

data：代表请求服务器端的数据（可以是 key=value 形式也可以是 json 格式）

callback：表示服务器端成功响应所触发的函数（只有正常成功返回才执行）

type：表示服务器端返回的数据类型 (jquery 会根据指定的类型自动类型转换)

常用的返回类型：text、json、html 等

- **$.ajax({ option1:value1,option2:value2... }); ---- 重要**

常用的 option 有如下：

async：是否异步，默认是 true 代表异步

data：发送到服务器的参数，建议使用 json 格式

dataType：服务器端返回的数据类型，常用 text 和 json

success：成功响应执行的函数，对应的类型是 function 类型

type：请求方式，POST/GET

url：请求服务器端地址

```
function fun1(){
        //get异步访问服务器
        $.get(
            "/AJAX/ajaxServlet2",                       //访问地址
            {"name":"haolian","age":23},                //访问的数据-------一般为json格式
            function(data){                             //data为服务器返回的数据
                alert("访问成功！返回数据：name=" + data.name);       //返回成功后执行的函数
```

```
            },
            "json"                          //服务器返回的数据类型
        );
    }
    function fun2(){
        //post异步访问服务器
        $.post(
            "/AJAX/ajaxServlet2",                           //访问地址
            {"name":"zhouman","age":22},            //访问的数据-------一般为json格式
            function(data){                                 //data为服务器返回的数据
                alert("访问成功！返回数据：name=" + data.name);  //返回成功后执行的函数
            },
            "json"                                  //服务器返回的数据类型
        );
    }
    function fun3(){
        //$ajax方法访问服务器
        $.ajax({
            url:"/AJAX/ajaxServlet2",                       //访问服务器的地址
            async:true,                                     //是否异步，true异步，false同步
            type:"POST",                                    //访问服务器的类型---GET,POST
            data:{"name":"goodloving","age":21},            //访问服务器的数据，json格式
            success:function(resultdata){           //如果访问成功就执行的函数，resultdata返回的数据
                alert("访问成功！返回数据：name=" + resultdata.name);
            },
            error:function(){                               //如果访问失败执行的函数
                alert("访问失败")
            },
            dataType:"json"             //服务器返回数据的类型
        });
    }
```

监听器 Listener

监听器 Listener

javaEE 包括 13 门规范 在课程中主要学习 servlet 技术 和 jsp 技术
其中 servlet 规范包括三个技术点：servlet listener filter

什么是监听器?

监听器就是监听某个对象的的状态变化的组件
监听器的相关概念:
事件源: 被监听的对象 ----- 三个域对象 request session servletContext
监听器: 监听事件源对象 事件源对象的状态的变化都会触发监听器 ---- 6+2
注册监听器: 将监听器与事件源进行绑定
响应行为: 监听器监听到事件源的状态变化时所涉及的功能代码 ---- 程序员编写代码

监听器有哪些?

第一维度: 按照被监听的对象划分:ServletRequest 域 HttpSession 域 ServletContext 域
第二维度: 监听的内容分:监听域对象的创建与销毁的监听域对象的属性变化的

	ServletContext域	HttpSession域	ServletRequest域
域对象的创建与销毁	ServletContextListener	HttpSessionListener	ServlerRequestListener
域对象内的属性的变化	ServletContextAttributeListener	HttpSessionAttributeListener	ServlerRequestAttributeListener

监听三大域对象的创建与销毁的监听器

➢ 监听 ServletContext 域的创建与销毁的监听器-ServletContextListener

1) Servlet 域的生命周期
何时创建: 服务器启动创建
何时销毁: 服务器关闭销毁
2) 监听器的编写步骤 (重点):
a、编写一个监听器类去实现监听器接口
b、覆盖监听器的方法
c、需要在 web.xml 中进行配置---注册
3) 监听的方法:

ServletContextListener监听器

```
@Override
public void contextInitialized(ServletContextEvent sce) {
    System.out.println("context init....");
}

@Override
public void contextDestroyed(ServletContextEvent sce) {
    System.out.println("context destory....");
}
```

创建ServletContext时执行
事件源对象
被监听的对象
servletContext销毁时执行

4) 配置文件:

```xml
<!-- 监听servletContext创建于销毁的监听器 -->
<listener>
  <listener-class>cn.itcast.listener.create.MyServletContextListener</list
</listener>
```

5）ServletContextListener 监听器的主要作用

a、初始化的工作：初始化对象 初始化数据 ---- 加载数据库驱动连接池的初始化

b、加载一些初始化的配置文件 --- spring 的配置文件

c、任务调度----定时器----Timer/TimerTask

```java
SimpleDateFormat fromat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
Date parse = null;
try {
    parse = fromat.parse("2016-03-27 24:00:00");
} catch (ParseException e) {
    e.printStackTrace();
}

//web应用一起动 就开启任务调度
Timer timer = new Timer();
timer.schedule(new TimerTask() {
    @Override
    public void run() {
        //定时执行的任务代码
        System.out.println("runner ......");
    }
}, parse, 3000);//24*60*60*1000
```

➢ 监听 Httpsession 域的创建于销毁的监听器 HttpSessionListener

1）HttpSession 对象的生命周期

何时创建：第一次调用 request.getSession 时创建

何时销毁：服务器关闭销毁 session 过期 手动销毁

2）HttpSessionListener 的方法

```java
@Override
public void sessionCreated(HttpSessionEvent se) {
    HttpSession session = se.getSession();
    System.out.println("session创建："+session.getId());
}

@Override
public void sessionDestroyed(HttpSessionEvent se) {
    HttpSession session = se.getSession();
    System.out.println("session销毁："+session.getId());
}
```

➢ 监听 ServletRequest 域创建与销毁的监听器 ServletRequestListener

1）ServletRequest 的生命周期

创建：每一次请求都会创建 request

销毁：请求结束

2）ServletRequestListener 的方法

```
@Override
public void requestDestroyed(ServletRequestEvent sre) {
    sre.getServletRequest();
    System.out.println("request销毁");
}

@Override
public void requestInitialized(ServletRequestEvent sre) {
    System.out.println("request创建");
}
```

监听三大域对象的属性变化的

➢ 域对象的通用的方法：

setAttribute(name,value)

--- 触发添加属性的监听器的方法

--- 触发修改属性的监听器的方法

getAttribute(name)

removeAttribute(name)

--- 触发删除属性的监听器的方法

ServletContextAttibuteListener 监听器

```
public class MyServletContextAttributeListener implements ServletContextAttributeListener{

    @Override
    public void attributeAdded(ServletContextAttributeEvent scab) {
        //放到域中的属性
        System.out.println(scab.getName());//放到域中的name
        System.out.println(scab.getValue());//放到域中的value
    }

    @Override
    public void attributeRemoved(ServletContextAttributeEvent scab) {
        System.out.println(scab.getName());//删除的域中的name
        System.out.println(scab.getValue());//删除的域中的value
    }

    @Override
    public void attributeReplaced(ServletContextAttributeEvent scab) {
        System.out.println(scab.getName());//获得修改前的name
        System.out.println(scab.getValue());//获得修改前的value
    }
}
```

HttpSessionAttributeListener 监听器（同上）

ServletRequestAriibuteListenr 监听器（同上）

与 session 中的绑定的对象相关的监听器（对象感知监听器）

➢ 即将要被绑定到 session 中的对象有几种状态

绑定状态：就一个**对象**被放到 session 域中

解绑状态：就是这个对象从 session 域中移除了
钝化状态：是将 session 内存中的对象持久化（序列化）到磁盘
活化状态：就是将磁盘上的对象再次恢复到 session 内存中
面试题：当用户很多时，怎样对服务器进行优化？

- **绑定与解绑的监听器 HttpSessionBindingListener—不用注册**

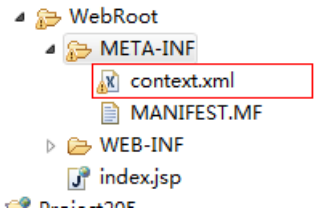
```
public class Person implements HttpSessionBindingListener{

    private String id;
    private String name;
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    //感知user被绑定到session中的方法
    @Override
    public void valueBound(HttpSessionBindingEvent event) {
        System.out.println("user被绑定到session域中了");
        System.out.println(event.getName());
    }

    //感知user从session中解绑的方法
    @Override
    public void valueUnbound(HttpSessionBindingEvent event) {
        System.out.println("user从session域中解绑了");
        System.out.println(event.getName());
    }
```

- **钝化与活化的监听器 HttpSessionActivationListener**

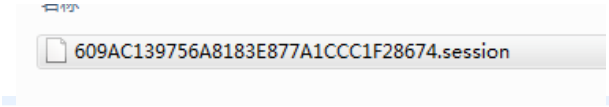
可以通过配置文件 指定对象钝化时间 --- 对象多长时间不用被钝化
在 META-INF 下创建一个 context.xml




```
<Context>
 <!-- maxIdleSwap:session 中的对象多长时间不使用就钝化 -->
 <!-- directory:钝化后的对象的文件写到磁盘的哪个目录下  配置钝化的对象文件在  work/catalina/localhost/钝化文件 -->
        <Manager className="org.apache.catalina.session.PersistentManager" maxIdleSwap="1">
            <Store className="org.apache.catalina.session.FileStore" directory="itcast205" />
```

```
        </Manager>
</Context>
```

被钝化到 work/catalina/localhost/的文件

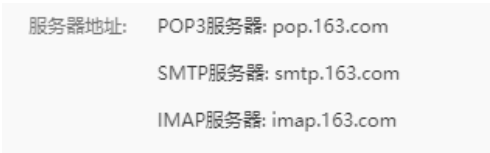


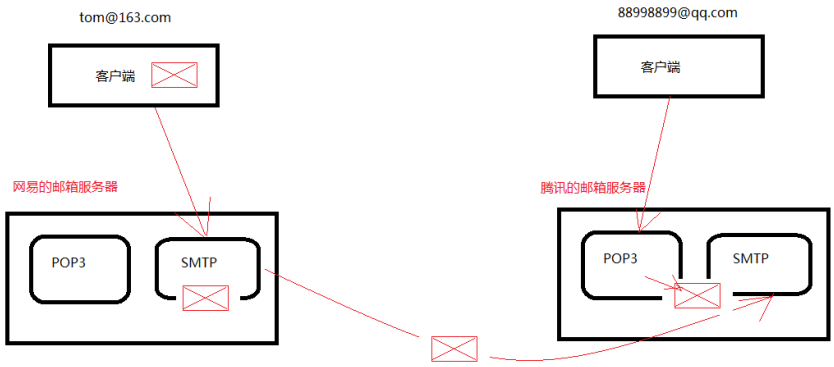

```java
//钝化
@Override
public void sessionWillPassivate(HttpSessionEvent se) {
    System.out.println("costomer被钝化了....");
}
//活化
@Override
public void sessionDidActivate(HttpSessionEvent se) {
    System.out.println("costomer被活化了....");
}
```

邮箱服务器

邮箱服务器的基本概念

邮件的客户端：可以只安装在电脑上的也可以是网页形式的
邮件服务器：起到邮件的接受与推送的作用
邮件发送的协议：
协议：就是数据传输的约束
接受邮件的协议：POP3　IMAP
发送邮件的协议：SMTP

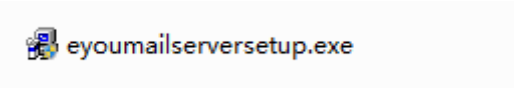




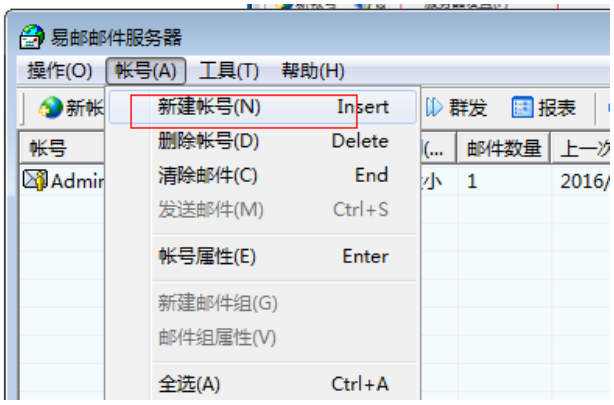
局域网邮箱服务器的安装及使用（了解）

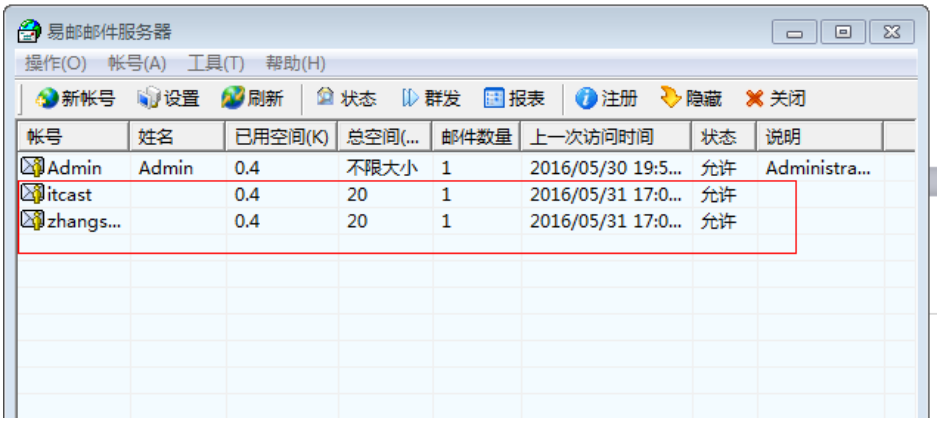
➢ 邮箱服务器的安装

1）双击邮箱服务器软件

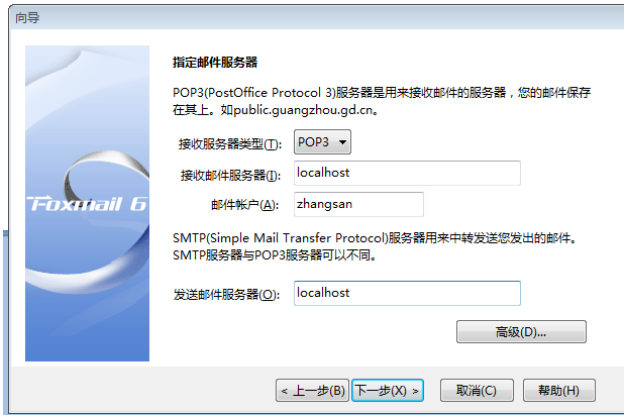


2）对邮箱服务器进行配置





- **邮箱客户端的安装**



- **邮件发送代码**

过滤器 Filter

filter 的简介

filter 是对客户端访问资源的过滤，符合条件放行，不符合条件不放行，并且可以对目 标资源访问前后进行逻辑处理

快速入门

步骤：

1）编写一个过滤器的类实现 Filter 接口

2）实现接口中尚未实现的方法(着重实现 doFilter 方法)

3）在 web.xml 中进行配置(主要是配置要对哪些资源进行过滤)

Filter 的 API 详解

- **`filter` 生命周期及其与生命周期相关的方法**

Filter 接口有三个方法，并且这个三个都是与 Filter 的生命相关的方法

init(Filterconfig)：代表 filter 对象初始化方法 filter 对象创建时执行

doFilter(ServletRequest,ServletResponse,FilterCha)：代表 filter 执行过滤的核心方法，如果某资源在已经被配置到这个 filter 进行过滤的话，那么每次访问这个资源都会执行 doFilter 方法

destory()：代表是 filter 销毁方法 当 filter 对象销毁时执行该方法

Filter 对象的生命周期：

Filter 何时创建：服务器启动时就创建该 filter 对象

Filter 何时销毁：服务器关闭时 filter 销毁

➢ Filter 的 AP 详解

1）init(FilterConfig)

其中参数 config 代表 该 Filter 对象的配置信息的对象，内部封装是该 filter 的配置信息。

```
@Override
public void init(FilterConfig filterConfig) throws ServletExcepti
    String filterName = filterConfig.getFilterName();
    System.out.println(filterName);
    ServletContext servletContext = filterConfig.getServletContex
    String initParameter = filterConfig.getInitParameter("aaa");
    System.out.println(initParameter);

    System.out.println("filter1创建....");
}
```

2）destory()方法

filter 对象销毁时执行

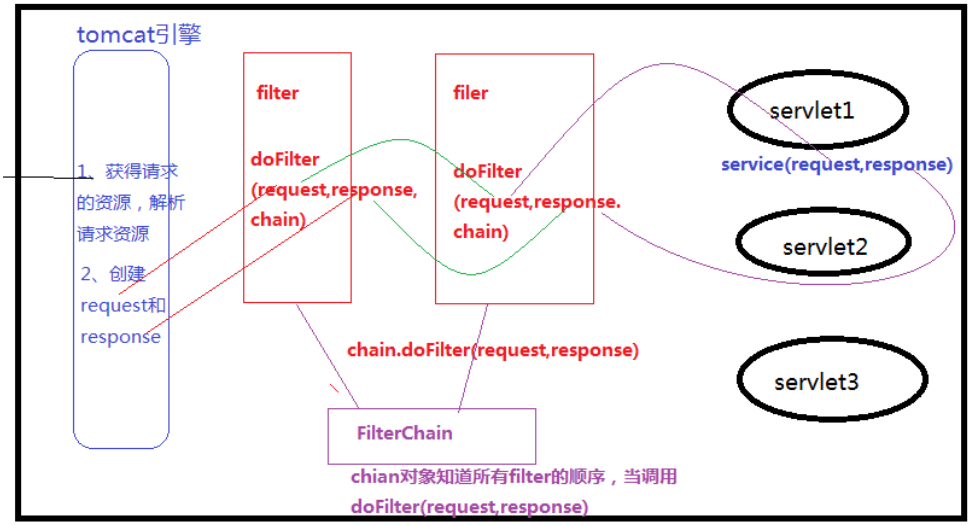
3）doFilter 方法

doFilter(ServletRequest,ServletResponse,FilterChain)

其中的参数：

ServletRequest/ServletResponse：每次在执行 doFilter 方法时 web 容器负责创建一个 request 和一个 response 对象作为 doFilter 的参数传递进来。该 request 和该 response 就是在访问目标资源的 service 方法时的 request 和 response。

FilterChain：过滤器链对象，通过该对象的 doFilter 方法可以放行该请求



➢ Filter 的配置

url-pattern 配置时

1）完全匹配　/sertvle1

```
<filter>
  <filter-name>Filter2</filter-name>
  <filter-class>cn.itcast.filter.Filter2</filter-class>
</filter>
<filter-mapping>
  <filter-name>Filter2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

2）目录匹配　/aaa/bbb/* ----最多的

/user/*：访问前台的资源进入此过滤器

/admin/*：访问后台的资源时执行此过滤器

3）扩展名匹配　*.abc　*.jsp

注意：url-pattern 可以使用 servlet-name 替代，也可以混用

dispatcher：访问的方式(了解)

REQUEST：默认值，代表直接访问某个资源时执 filter

FORWARD：转发时才执行 filter

INCLUDE：包含资源时执行 filter

ERROR：发生错误时 进行跳转是执行 filter

➢ 总结 Filter 的作用

1）公共代码的提取

2）可以对 request 和 response 中的方法进行增强(装饰者模式/动态代理)

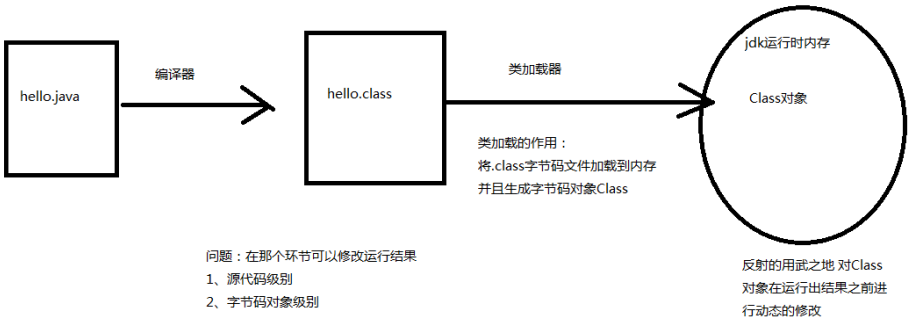
3）进行权限控制

基础加强

类加载器

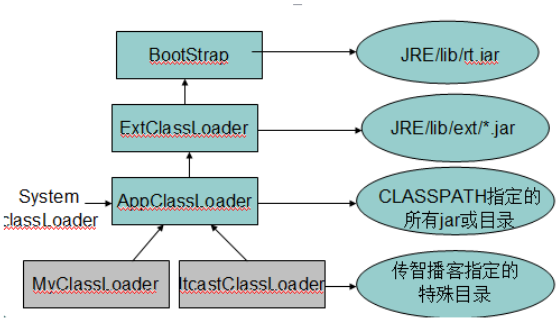
➢ 什么是类加载器，作用是什么?

类加载器就加载字节码文件(.class)



➢ 类加载器的种类

类加载器有三种，不同类加载器加载不同的



1）BootStrap：引导类加载器：加载都是最基础的文件

2）ExtClassLoader：扩展类加载器：加载都是基础的文件

3）AppClassLoader：应用类加载器：三方 jar 包和自己编写 java 文件

怎么获得类加载器？(重点)

ClassLoader 字节码对象.getClassLoader();

```
lic class Demo {

 public static void main(String[] args) {

     //获得Demo字节码文件的类加载器
     Class clazz = Demo.class;//获得Demo的字节码对象
     ClassLoader classLoader = clazz.getClassLoader();//获得类加载器
     //getResource的参数路径相对classes（src）
     //获得classes(src)下的任何的资源
     String path = classLoader.getResource("com/itheima/classloader/jdbc.proper
     System.out.println(path);

 }
```

注解 @xxx

➢ 什么是注解，注解作用

注解就是符合一定格式的语法 @xxxx

注解作用：

注释：在阅读程序时清楚----给程序员看的

注解：给 jvm 看的，给机器看的

注解在目前而言最主流的应用：代替配置文件

关于配置文件与注解开发的优缺点：

注解优点：开发效率高 成本低

注解缺点：耦合性大 并且不利于后期维护

➢ jdk5 提供的注解

@Override：告知编译器此方法是覆盖父类的

@Deprecated：标注过时

@SuppressWarnings：压制警告

发现的问题：

不同的注解只能在不同的位置使用(方法上、字段上、类上)

➢ 自定义注解（了解）

1）怎样去编写一个自定义的注解

2）怎样去使用注解

3）怎样去解析注解-----使用反射知识

➢ 编写一个注解

关键字：@interface

注解的属性：

```
public @interface MyAnno {

    String value();

    //String addr();

    int age() default 28;
}
```

语法：返回值类型 名称();

注意：如果属性的名字是 value，并且注解的属性值有一个 那么在使用注解时可以省略 value

注解属性类型只能是以下几种

1.基本类型

2.String

3.枚举类型

4.注解类型

5.Class 类型

6.以上类型的一维数组类型

➢ 使用注解

```
@MyAnno("zhangsan")
public class TestAnno {

}
```

在类/方法/字段 上面是@XXX

➢ 解析使用了注解的类

介入一个概念：元注解：代表修饰注解的注解，作用：限制定义的注解的特性

@Retention

SOURCE： 注解在源码级别可见

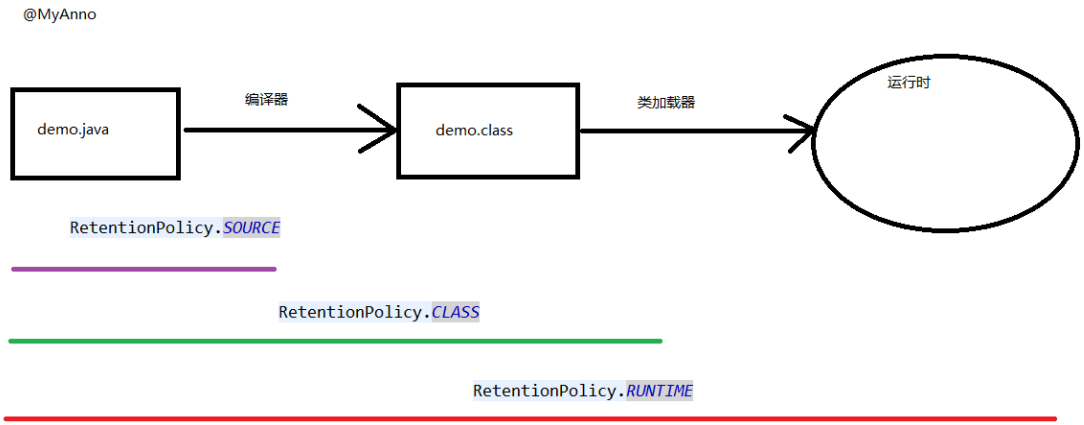
CLASS：注解在字节码文件级别可见

RUNTIME：注解在整个运行阶段都可见

@Target

代表注解修饰的范围：类上使用，方法上使用，字段上使用

FIELD:字段上可用此注解
METHOD:方法上可以用此注解
TYPE:类/接口上可以使用此注解



注意：要想解析使用了注解的类 那么该注解的 Retention 必须设置成 Runtime

关于注解解析的实质：从注解中解析出属性值

字节码对象存在于获得注解相关的方法

isAnnotationPresent(Class<? extends Annotation> annotationClass) ： 判断该字节码对象身上是否使用该注解了

getAnnotation(Class<A> annotationClass) ：获得该字节码对象身上的注解对象

第一步：创建注解

```
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ElementType.METHOD,ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnno {

    //注解的属性
    String name();

    int age() default 28;

    //String value();
```

第二步：使用注解

```
public class MyAnnoTest {

    @SuppressWarnings("all")
    @MyAnno(name = "zhangsan")
    //@MyAnno({ "aaa","bbb","ccc"})
    public void show(String str){
        System.out.println("show running...");
    }

}
```

第三步：解析注解

```
public static void main(String[] args) throws NoSuchMethodExcept

    //解析show方法上面的@MyAnno
    //直接的目的是 获得show方法上的@MyAnno中的参数

    //获得show方法的字节码对象
    Class clazz = MyAnnoTest.class;
    Method method = clazz.getMethod("show", String.class);
    //获得show方法上的@MyAnno
    MyAnno annotation = method.getAnnotation(MyAnno.class);
    //获得@MyAnno上的属性值
    System.out.println(annotation.name());//zhangsan
    System.out.println(annotation.age());//28

    //根据业务需求写逻辑代码

}
```

动态代理

➢ 什么是代理(中介)

目标对象/被代理对象 ------ 房主：真正的租房的方法
代理对象 ------- 黑中介：有租房子的方法（调用房主的租房的方法）
执行代理对象方法的对象 ---- 租房的人
流程：我们要租房----->中介（租房的方法）------>房主（租房的方法）
抽象：调用对象----->代理对象------>目标对象

➢ 动态代理

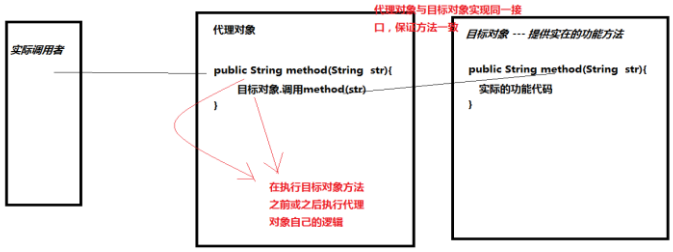
动态代理：不用手动编写一个代理对象，不需要一一编写与目标对象相同的方法，这个过程，在运行时 的内存中动态生成代理对象。------字节码对象级别的代理对象动态代理的 API：

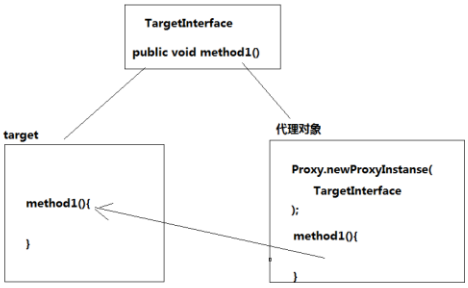
在 jdk 的 API 中存在一个 Proxy 中存在一个生成动态代理的的方法 newProxyInstance

static Object	**newProxyInstance**(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)

返回值：Object 就是代理对象
参数：loader：代表与目标对象相同的类加载器-------目标象.getClass().getClassLoader()
interfaces：代表与目标对象实现的所有的接口字节码对象数组
h：具体的代理的操作，InvocationHandler 接口

注意：JDK 的 Proxy 方式实现的动态代理 目标对象必须有接口 没有接口不能实现 jdk 版动态代理





➢ 代码实例：解决全局乱码问题（装饰者模式&动态代理）

1）装饰者模式

```
在Filter的doFilter方法中编写:
        //强转为httpservlet接口
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;
        /*
         * 装饰着模式
         */
        EnhanceRequest enhanceRequest = new EnhanceRequest(req);
        //放行
        chain.doFilter(enhanceRequest, resp);


/*
 * 装饰着模式增强request:1.继承同一个父类  2.对父类中需要增强的方法进行重写
 */
class EnhanceRequest extends HttpServletRequestWrapper{
    private HttpServletRequest request;
    public EnhanceRequest(HttpServletRequest request) {
        super(request);
        this.request = request;
    }
    @Override
    public String getParameter(String name) {
```

```
        String parameter = request.getParameter(name);            //未处理的乱码的字符
        if(parameter!=null) {
            try {
                parameter = new String(parameter.getBytes("ISO8859-1"),"UTF-8");
            } catch (UnsupportedEncodingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        return parameter;
    }
}
```

2)动态代理

```
        //强转为http的servlet对象
        final HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;
        //创建动态代理方法
        HttpServletRequest enhanceRequest = (HttpServletRequest) Proxy.newProxyInstance(
                req.getClass().getClassLoader(),
                req.getClass().getInterfaces(),
                new InvocationHandler() {
                    @Override
                    public Object invoke(Object proxy, Method method, Object[] args)
                                                        throws Throwable {
                        //获取 目标对象 的方法名
                        String name = method.getName();
                        //得到需要进行操作的方法
                        if(name.equals("getParameter")) {
                            //获取乱码的数据
                            String invoke = (String) method.invoke(req, args);
                            //对乱码数据进行转码
                            invoke = new String(invoke.getBytes("iso8895-1"),"UTF-8");
                            return invoke;
                        }
                        //最后将整个方法原封不动的返回
                        return method.invoke(req, args);
                    }
                }
            );
        //放行
        chain.doFilter(enhanceRequest, resp);
```

总结：装饰者模式多用于功能增强，动态代理多用于拦截