# 乱码处理

1. 开始编码前统一设置：workspace + resource + jsp

Servlet中：

request.setCharacterEncoding(“utf-8”)

response.setCharacterEncoding(“utf-8”);

1. 使用new String(byte[],charset)方法重组字符串
2. 过滤器统一过滤

# 请求转发与重定向

1. Request.getRequestDispatcher(“”).**forward**(request,response)：

页面地址不变化，默认使用post提交！！，request，response对象被直接传递，在服务端执行，对于浏览器（用户）是透明的。

ServletContext.getRequestDispatcher(“”)只能使用绝对路径，而ServletRequest.getRequestDispatcher(“”)可以使用相对路径

1. Response.**sendReDirect**(“”)：

页面地址变化，默认使用get提交，request，response对象被丢弃，重新生成。此时request和response的编码设置也丢弃，注意设置编码。

我们在JSP中让页面Redirect到一个某个A页面中，其实是让客户端再发一个请求到A页面，这个需要Redirect到的A页面的URL，其实就是通过**响应报文头的Location属性**告知客户端的，如下的报文头属性，将使客户端redirect到iteye的首页中，如：Location: http://www.iteye.com，这个响应报文属性将使客户端再次发送请求到指定url（即重定向）。

1. sendRedirect()会新建request对象，所以上一个request中的数据会丢失。 更具体来说就是这样的：   
   redirect 会首先发一个response给浏览器, 然后浏览器收到这个response后再发一个requeset给服务器, 然后服务器发新的response给浏览器. 这时页面收到的request是一个新从浏览器发来的。  
   forward 发生在服务器内部, 在浏览器完全不知情的情况下发给了浏览器另外一个页面的response. 这时页面收到的request不是从浏览器直接发来了,可能己经用request.setAttribute在request里放了数据.在转到的页面可直接用request.getAttribute获得数据。
2. forward只能跳转到同一个Web应用中的资源，而sendRedirect可以跳转到任何URL

[*http://www.cnblogs.com/phpzxh/archive/2010/02/01/1661137.html*](http://www.cnblogs.com/phpzxh/archive/2010/02/01/1661137.html)

1. form 的action：

form的action提交方式既不是请求转发也不是重定向，form提交由客户端发起（重定向和转发都属于服务端动作）。

# ? 号传参

只能传递字符串；request.getParameter(“”)

？后有中文字符时不能直接传递，需要将中文字符的编码转为 ISO-8859-1

String msg = new String("登陆失败".getBytes("utf-8"), "ISO-8859-1");

# HttpSession 会话

1. 保存上下文信息，实现会话跟踪，针对每一个用户的值，变量保存到服务器端，通过sessionID区分不同的客户
2. session何时被删除：

程序调用HttpSession.invalidate()；

距离上一次调用时间超过MaxInactiveInterval时间（默认时间30 minute）；

服务器进程被停止；

关闭浏览器只会使存储在客户端浏览器内存中的seesion cookie失效，不会使服务端的session对象失效。

1. isNew: 如果会话尚未与客户端发生任何联系或者客户端选择不加入会话时返回true；

比如，如果客户端只使用基于cookie的会话，而浏览器恰恰禁用了cookie，那么无论何时调用都返回true，因为会话对浏览器而言始终是新的。

isNew返回false，只能说明用户曾经访问过该web应用，并不一定代表其曾经访问过我们的具体哪个servlet或jsp页面。

正确的做法是判断session中是否有特定key存在，及其值是否正确。

1. 关闭浏览器不会使session销毁，服务器需要为session设置一个失效时间。

<http://www.cnblogs.com/zhaoYuQing-java2015/p/4621384.html>

1. session 无法跨浏览器获取数据。即更换浏览器会开启新的会话。

# ServletContext

1. 一个 java web应用只创建一个ServletContext对象，一般用于在多个客户端(浏览器)之间共享数据。
2. ServletContext被定位于 <http://localhost:8080/project/>开始的所有请求被发送到与此应用关联的Web应用程序。
3. ServletContext提供对应用作用域与对象选择：

request：跟当前操作相关

session：更跟用户相关

application：跟项目全局信息相关

ServletContext在服务器关闭Web应用时销毁。

是一个**全局的储存信息的空间**，服务器开始，其就存在，服务器关闭，其才释放。

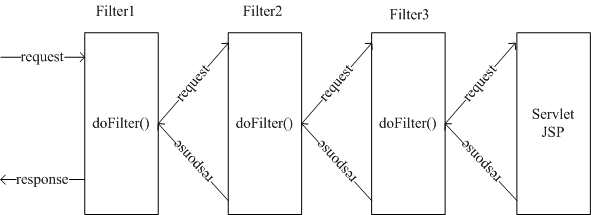
一个用户有可以多个request；一个用户的一次会话只有一个session；而servletContext是所有用户共用的。所以，为了节省空间，提高效率，ServletContext中，要**放必须的、重要的、所有用户需要共享的线程又是安全的一些信息**。

# 过滤器javax.servlet.Filter

1. 过滤器位于客户端和Web应用程序之间，用于**检查和修改两者之间流过的请求和响应**。
2. 在请求到达Servlet/Jsp之前，过滤器截获请求；

在响应到达客户端之前，过滤器截获响应。

1. 多个过滤器形成一个过滤器链，过滤器链中不同过滤器的先后顺序由部署文件web.xml中过滤器映射<filter-mapping>的顺序决定。
2. 最先截获客户端**请求**的过滤器将最后截获Servlet/JSP的**响应信息**。
3. 过滤器的链式结构：

为一个Web应用部署多个过滤器，这些过滤器组成一个过滤器链，每个过滤器只执行某个特定的操作或检查，这样请求在到达被访问的目标之前，需要经过这个过滤器链。

1. 在过滤器的doFilter()方法实现中，任何出现在FilterChain的doFilter方法之前地方，request是可用的；在doFilter()方法之后response是可用的。
2. url-pattern 与 dispatcher 是且的关系，只有同时满足两者，该Filter才会生效

# JSP（java service page）java 服务器页面

1. Servlet 缺陷：

写html标签可读性差；

html可以转化为js，servlet不能转换；

编码处理麻烦；

jsp弥补了上述不足，jsp通过在标准的HTMl页面中嵌套java代码，使其静态的部分无需java控制，只有那些需要从数据库或动态生成的页面内容，才使用java脚本控制。

1. 语法：

<% %> ：java脚本

1. 每一个jsp页面就是一个Servlet实例。Servlet容器会负责将jsp转为servlet类。
2. 工作原理：

（1）如果是第一次请求页面，或页面发生改动，则Servlet容器首先将jsp页面转化为Servlet实例，再将其编译为class文件，编译过程会耗费时间。

（2）jsp容器负责调用从jsp转换而来的Servlet，这些Servlet负责响应用户的请求（提供服务），容器会为Servlet建立一个线程，如果有多个请求，则建立多个线程。

（3）容器执行Servlet的字节码文件，返回的最终方式为html格式的文件流。

1. jsp 页面转换为Servlet时仅将html内容用JspWriter（out）输出。转化结果文件查看路径：D:\Duan\tomcat\apache-tomcat-8.5.16\work\Catalina
2. 自动转换的Servlet的继承结构：\*\*\*\_jsp extends HttpJspBase extends HttpServlet：jsp 页面中的静态内容和动态内容都由 \_jspService 方法使用JspWriter（apache的实现为JspWriterImpl）对象输出到客户端（\_jspService 方法调用封装在HttpJspBase 的service 方法中）。

# Servlet

1. Servlet只在第一次使用（请求）时加载和实例化，由容器管理和控制servlet的生命周期。
2. Servlet一旦被加载，一般不会从容器中删除，直至服务器关闭或重新启动，但当容器做内存回收动作时，servlet有可能被删除，也正是这个原因，第一次访问Servlet所用的时间要大大多于以后访问所用的时间。
3. Servlet在服务器中的运行：加载 -> 初始化 -> 调用 -> 销毁。
4. Servlet不是单例的，不过一般一个Servlet只会有一个实例。
5. Servlet的工作过程：
6. 加载和实例化

如果Servlet容器还没有实例化一个Servlet对象，此时容器装载和实例化Servlet，创建一个实例，如果已经存在则不创建新实例。

初始化

1. 在产生Servlet实例后，容器负责调用该Servlet实例的init()方法，在处理用户请求之前，完成初始化。
2. 处理请求，当Servlet容器接收到请求时，运行对应Servlet实例的service()方法，service方法根据请求类型调用相应的方法（doPost，doGet等）。
3. 销毁，当容器决定将一个Servlet从服务器移除时（如Servlet文件更新时），便调用Servlet的destroy方法。

其中（1）、（2）、（4）在整个生命周期中只会被执行一次。

JSP/Servlet工作原理 ：<http://blog.csdn.net/insistgogo/article/details/20788749>

Servlet单实例多线程模式：<http://kakajw.iteye.com/blog/920839>

# JSP 9大内置对象

## 1 Request

请求对象，类型 javax.servlet.ServletRequest，作用域 Request。

## 2 Response

响应对象类型 javax.servlet.SrvletResponse，作用域 Page。

## 3 PageContext

页面上下文对象，类型 javax.servlet.jsp.PageContext，作用域 Page。

* 1. javax.servlet.jsp.PageContext pageContext：代表jsp页面上下文，访问页面共享数据。
  2. 使用pageContext将属性设置到session范围中，仅PageContext 有该方法。pageContext.setAttribute(“page”,”first page”,PageContext.SESSION\_SCOPE);
  3. page代表当前页面，即自动生成的Servlet。
  4. pageContext.findAttribute(String name)：

该方法会依次在四个作用域中寻找，顺序为(由内而外)：page -> request -> session -> application，无需知道作用域就可以找到存在的数据。

## 4 Session

会话对象，类型 javax.servlet.http.HttpSession，作用域 Session。

## 5 Application

应用程序对象，类型 javax.servlet.ServletContext，作用域 Application。

javax.servlet.ServletContext application：

* 1. 全局作用范围，整个Web应用内共享，生命周期为应用启动到停止。服务器启动后就产生了一个application对象，当用户在访问Web应用的不同页面时application始终为同一个，直到Web应用停止。
  2. 存在于服务器的内存空间，在application对象的生命周期中，该应用的任何jsp页面都能获取到application对象绑定的参数。（不借助数据库就能实现聊天室功能）。

## 6 Out

输出对象，类型 javax.servlet.jsp.JspWriter，作用域 Page。

javax.servlet.jsp.JspWriter out：

实现类：org.apache.jasper.runtime.JspWriterImpl

JspWriter与PrintWriter异同：JspWriter可以在jsp页面中直接输出，亦可用 pageContext.getOut() （与内置对象 out 为同一个对象）方法获得；

同时用两者输出内容时，不管JspWriter与PrintWriter在程序中的顺序怎么样,始终先会输出PringWriter中的数据然后再输出JspWriter中的数据.这是因为out对象相当于插入到了PrintWriter前面的缓冲区中.out对象满足一定条件时,才会调用PrintWriter对象的print()方法,把out缓冲区中的内容输出到浏览器端。

<http://blog.csdn.net/zhd_superstar/article/details/6588222>

out.print(100)：将内容转化为字符串打印到页面

out.write(100)：将原有内容输出（ASCI码）

## 7 Config

配置对象，类型 javax.servlet.ServletConfig，作用域 Page。

## 8 Page

页面对象，类型 javax.lang.Object，作用域 Page。

## 9 Exception

例外对象，类型 javax.lang.Throwable，作用域 page。

# 6个操作指令

* Jsp:**include**

用来在页面被请求时引入一个文件，可通过jsp:param指令传递参数到被包含页面。

* Jsp:**useBean**

用来寻找或实例化一个javaBean。

* Jsp:**setProperty**
* 用来设置已实例化的bean对象的属性。
* Jsp:**getProperty**

用来获取已实例化的bean对象的属性值。

* Jsp:**forward**

用来把请求转发到指定url。

* Jsp**:plugin**

用于在浏览器中播放或显示一个对象， 需和java applet一起使用。

# 静态导入和动态导入

<%@include file=”index2.jsp”%>与<jsp:inclde page=”index3.jsp”></…> 异同

1. %@include - 静态包含：

（1）被包含页面中定义的java变量被修改声明为**包内权限的全局变量**（不静态包含时为\_JspService方法（HttpJspBase中的service调用）的局部变量）。方法原样包含。

（2）只对应生成一个Servlet文件（容器只转换生成一个java文件），两部分的内容被合并。

1. <jsp:include – 动态包含：内部实现为，request.getRequestDispatcher(resourcePath).include(request,new ServletResponseWrapperInclude(response,out));

其中RequestDispatcher的 apache实现为ApplicationDispatcher，最终调用方法为ApplicationDispatcher的doInclude方法。

RequestDispatcher接口的include方法定义注释为：将servlet、jsp page、html的内容包含进response，目标jsp和动态包含的jsp都会被转换为Servlet并编译。

需要传递参数时可使用<jsp:param name=”” value=””>标签，底层实现为？号传参。见**20.RequestDispatcher的include与forward的区别**

# Servlet自启动

web.xml

<servlet>

// 初始化参数

<init-param>

<param-name>isAdmin</param-name>

<param-value>true</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

1. load-on-startup：

（1）容器启动时自动加载Servlet（实例化并调用其init方法），值必须是一个整数，表示servlet的加载顺序（大于等于0）

（2）当值小于0或不指定时表示加载时机由容器自行决定。

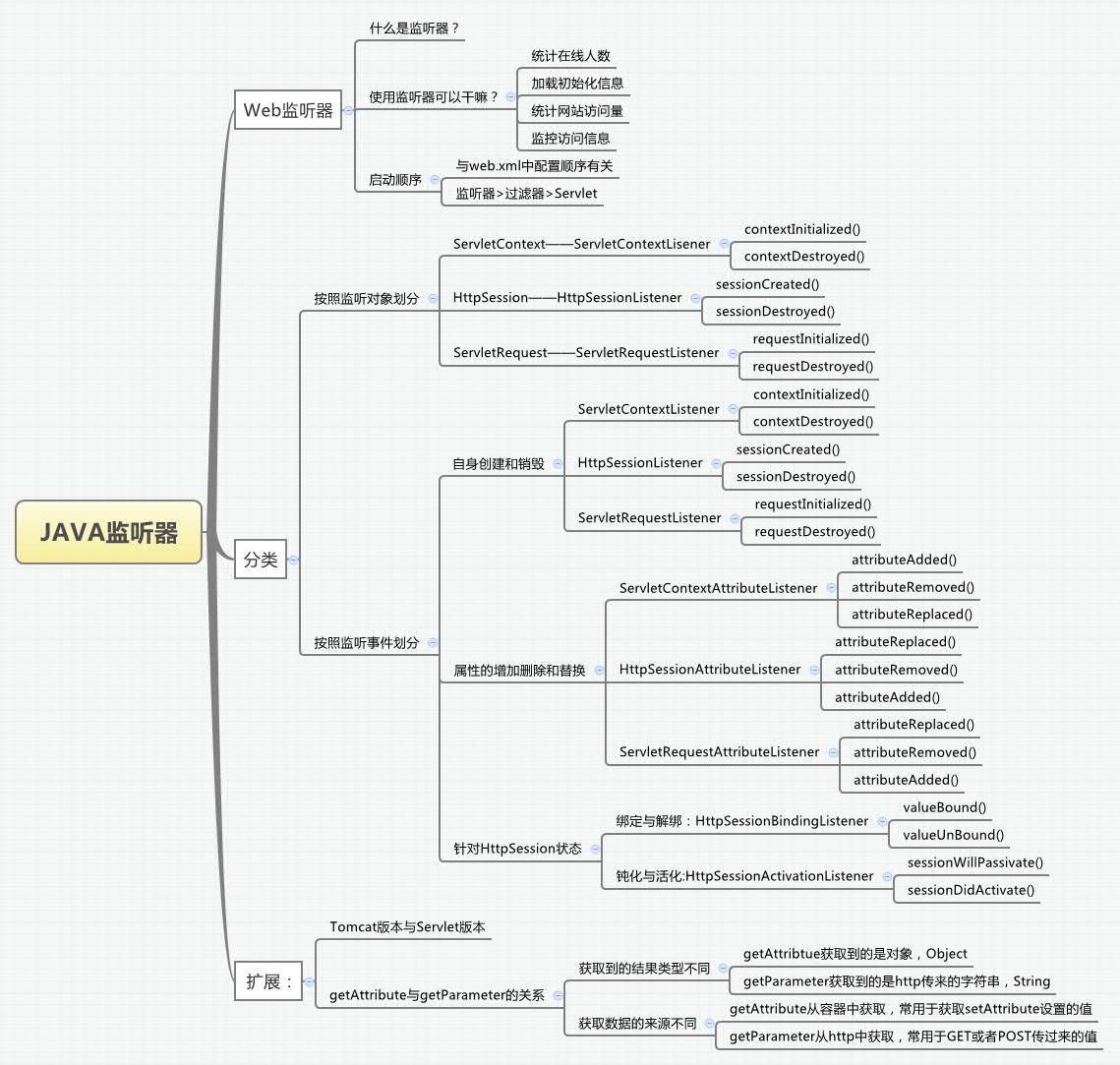
（3）正数的值越小，该servlet的优先级越高，应用启动时就越先加载，当值相等时，容器会自己决定加载顺序。

# 监听器

1. 用于监听ServletContext、HttpSession、ServletRequest等域对象的创建和销毁事件，还可以监听域对象的**属性**发生修改的事件，可以在事件发生前或者后做一些必要的处理。

2. 在web.xml中注册相应的监听即可，如果一个web.xml中注册了多个监听器，则监听器的启动顺序按照web.xml中的注册顺序启动。

3. 当web.xml同时有servlet、filter、listener时加载顺序为：listener -> filter -> servlet



## ServletContextListener

它能够监听 ServletContext 对象的生命周期，实际上就是监听 Web 应用的生命周期。

当Servlet 容器启动或终止Web 应用时，会触发contextInitialized和contextDestroyed事件。

# Session的创建时机

<http://www.360doc.com/content/12/0511/12/1542811_210284774.shtml>

session不是一打开网站就立刻建立，它的建立需要基于下面两个条件中的任意一个：

（1）在servlet中手动调用getSession()（有就返回，没有就创建一个再返回），或getSession(true)（为true时同getSession方法，为false时有就返回，没有返回null）。jsp页面被容器自动转换为servlet时会在其\_JspService方法（该方法在有请求发生时才会调用）中调用getSession方法为9大内置对象之一的session赋值（前提为<%@page session=”true”%>默认为true）。而\_JspService调用的前提为有请求发送到Servlet。

（2）jsp页面中设置<%@page session=”true”%>，默认情况下为true，可手动设置为false。

# Servlet中用到的外观模式（Facade）

外观模式是为了**解决类与类之间的依赖关系**，像spring一样，可以将类和类之间的关系配置到配置文件中，而外观模式就是将他们的关系放在一个Facade类中，间接调用，降低了类之间的耦合度。

通过外观的包装，使应用程序只能看到外观现象，而不会看到具体的细节对象，这样会**降低程序的复杂度，提高程序的可维护性**。

用户只需直接与外观角色交互，用户与子系统之间的复杂关系由外观角色充当，降低耦合度。

如jsp 9大内置对象之：session、application、config

1. session：org.apache.catalina.session.**StandardSessionFacade**

实现HttpSession，同时内部组合有正真的HttpSession。

2. application：org.apache.catalina.core.**ApplicationContextFacade**

实现ServletContext，内部组合ApplicationContext（实现ServletContext），

3. config：org.apache.catalina.core.**StandardWrapperFacade**

实现ServletConfig，内部组合StandardWrapper（实现ServletConfig）；

# RequestDispatcher的include与forward的区别

（1）forward方法可以将http请求转发给其它的Web资源（jsp，servlet，html），并产生响应。

（2）include方法与forward非常类似，不同的是，include方法将请求转发给其它Servlet后，被调用的servlet虽然可以处理这个http请求，但最后的主导权仍然是原来的servlet。

（3）RequestDispatcher是Web资源包装器，可以把request传递到任意一个目标资源。

（4）include将资源包含到当前的输出中。

（5）注意：只有在尚未向客户端输出响应时才可以调用forward方法，如果页面缓存不为空在重定向前将自动清空缓存，否则将抛出一个IllegalStateException。

# AJax （asynchronous javascript and xml）

<http://www.cnblogs.com/mingmingruyuedlut/archive/2011/10/18/2216553.html>

ajax，cookie和session的关系，session创建时机e.g

<http://www.cnblogs.com/rush/archive/2012/05/15/2502264.html>

1. 是一种用于创建快速动态网页的技术，时一种无需重新加载整个网页的情况下，能够更新部分网页的技术，Ajax原则——**按需取数据。**

2. 通过在后台与服务器进行少量数据交换，Ajax可以实现异步更新，这意味着可以在不重新加载整个网页的情况下，对网页的部分进行更新。（传统的网页，如果不使用Ajax，更新网页内容需要重载整个页面）。

3. web的运作原理：一次HTTP请求对应一个页面，如果要让用户停留在当前页面，同时发送http请求，就必须用javaScript发送这个请求，接收到数据后，再用JavaScript更新页面，这样，用户就感觉自己仍然停留在当前页面但数据却可以不断更新。

## Ajax包含的技术：

（1）使用css和xhtml来表示；

（2）使用DOM模型来交互和动态显示；

（3）使用XMLHttpRequest来和服务器进行异步通信；

（4）使用javaScript来绑定和调用。

5. 完全客户端的技术，而XMLHttpRequest正是处理了服务端与客户端通信的问题。

## Ajax原理和XmlHttpRequest对象

通过XMLHttpRequest对象来向服务器发起异步请求，从服务端获得数据后使用javaScript来操作DOM更新页面。

javascript可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

1. XmlHttpRequest属性：

* onreadystatechange  每次状态改变所触发事件的事件处理程序。
* responseText     从服务器进程返回数据的字符串形式。
* responseXML    从服务器进程返回的DOM兼容的文档数据对象。
* status           从服务器返回的数字代码，比如常见的404（未找到）和200（已就绪）
* statusText       伴随状态码的字符串信息
* readyState       对象状态值

常见readyState状态码：

0 (未初始化) 对象已建立，但是尚未初始化（尚未调用open方法）

1 (初始化) 对象已建立，尚未调用send方法

2 (发送数据) send方法已调用，但是当前的状态及http头未知

3 (数据传送中) 已接收部分数据，因为响应及http头不全，这时通过responseBody和responseText获取部分数据会出现错误，

4 (完成) 数据接收完毕,此时可以通过通过responseXml和responseText获取完整的回应数据

## Ajax浏览器差异（兼容）处理

差异主要体现在IE和其它浏览器之间：

function CreateXmlHttp() {  
  
 //非IE浏览器创建XmlHttpRequest对象  
 if (window.XmlHttpRequest) {  
 xmlhttp = new XmlHttpRequest();  
 }  
  
 //IE浏览器创建XmlHttpRequest对象  
 if (window.ActiveXObject) {  
 try {  
 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
 }  
 catch (e) {  
 try {  
 xmlhttp = new ActiveXObject("msxml2.XMLHTTP");  
 }  
 catch (ex) { }  
 }  
 }

return xmlhttp;  
}

## ajax的优/缺点

<http://www.cnblogs.com/mingmingruyuedlut/archive/2011/10/18/2216553.html>

# Cookie

[**http://blog.csdn.net/fangaoxin/article/details/6952954/**](http://blog.csdn.net/fangaoxin/article/details/6952954/)

1. Cookie通过在**客户端**记录信息确定用户身份，Session通过在**服务器**记录信息确定用户身份。

2. HTTP协议是无状态协议，一旦数据交换完毕，客户端与服务端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上**跟踪**会话。Cookie弥补了HTTP协议的无状态的不足，在Session出现之前，Cookie被大量使用。

3. 由于HTTP协议是一种无状态协议，服务器单从网络连接上无从知道客户机身份。怎么办呢？就给客户端颁发一个通行证吧，每人一个，无论谁都必须携带自己的通行证。这样服务器就能从通行证上确认客户身份。这就是cookie的工作原理。

4. Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

5. 在浏览器输入 javascript:alert(document.cookie) 就可以查看一个网站包含的cookie，需要有网。

6. Cookie功能需要浏览器的支持，如果浏览器不支持（大部分的手机浏览器），或者禁用了，Cookie就会失效，不同浏览器采用不同的方式保存Cookie，IE会保存在：C:\Documents and Settings\你的用户名\Cookies 目录下，以文件形式保存。

7. Cookie的不可跨域名性

Biadu和Google都会给浏览器颁发Cookie，那么浏览器访问Baidu时会将Google的Cookie也提交吗？**不会的。**

Cookie具有不可跨域名性，根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。浏览器区分的依据是域名。二者不能互相操作彼此的Cookie。

8. 单个Cookie的客户端限制是3k，即一个站点在客户端存放的Cookie不能超过 3 k；将登陆信息等重要信息存放在session，其它信息如果需要保留，可以存放在Cookie中。

<http://blog.csdn.net/axin66ok/article/details/6175522>

# 文件上传，下载

# js getElementById().value 与innerHTML的区别

表单用value，插入字段用innerHTML。有value属性的标签才能使用 .value。

# EL表达式

[**http://www.jb51.net/article/20042.htm**](http://www.jb51.net/article/20042.htm)

[**http://www.jb51.net/article/105314.htm**](http://www.jb51.net/article/105314.htm)

<%@page isELIgnored="false"%>

Servlet容器内部会直接调用org.apache.jspager.runtime.PageContextImpl.proprietaryEvaluate(…)方法解析 el 表达式（内部为 ELContext 对象）

1它是一种简单的语言，基于可用的命名空间（PageContext 属性）、嵌套属性和对集合、操作符（算术型、关系型和逻辑型）的访问符、映射到 Java 类中静态方法的可扩展函数以及一组隐式对象。EL 提供了在 JSP 脚本编制元素范围外使用运行时表达式的功能。脚本编制元素是指页面中能够用于在 JSP 文件中嵌入 Java 代码的元素。它们通常用于对象操作以及执行那些影响所生成内容的计算。JSP 2.0 将 EL 表达式添加为一种脚本编制元素。

2 语法结构 ${expression}

3 [] 与 . 运算符

当要存取的属性名称中包含一些特殊字符，如 . 、- 或 ？ 等并非字母或数字的符号，就一定要使用 []。如：

${user.nick-name} 应改为 ${user[“nick-name”]}。

如果要动态取值，可以用 [] 来做，而 . 无法做到动态取值，如：

${sessionScope.user[data]} 中data是一个变量。

4 ${user}: 取出某一范围中名称为user的变量，因为并没有指定user变量的范围，所以它会依次从**Page、Request、Session、Application范围（由内而外）中查找**。如果中途找到，直接回传并中断查找，如果所有范围中都没有则回传null（打印时为””）。

5 el 中的有效表达式

文字：可以用位子表示特定数据类型的值，如true、false、24、-13、23.3、-1.8e4、’name’、”user”、null 等。

操作符：

算数：+、-、\*、/、%、div、mod、

逻辑：and、&&、or、||、!、not、

关系：==、eq、!=、ne、<=、le、>=、ge、

空：、、

条件：A ? B : C

5 隐式对象

JSP页面上下文：pageContext

请求参数的值映射到单个字符串（ServletRequest.getParameter(name)）：param

所有请求参数的值映射到一个字符串数组（ServletRequset）：paramValues

…

header

headerValues

cookie

initParam

…

pageScope

requestScope

sessionScope

applicationScope

6 ${pageContext} 将会输出 pageContext （调用其 toString）方法

注意 <%@ page isELIgnored="true" %> 表示是否禁用EL语言,TRUE表示禁止.FALSE表示不禁止.JSP2.0中默认的启用EL语言。

# JSTL

## C标签

<http://blog.csdn.net/leolu007/article/details/45029627>

<%@ taglib prefix=”c” uri=<http://java.sun.com/jsp/jstl/core>” %>

1 <c:out value=”value” [escapeXml=”{true|false}”] [default=”defaultValue”]/>

2 <c:set value="value" var="varName" [scope="{page|request|session|application}"]/>

3 <c:remove var="varName" [scope="{page|request|session|application}"]/>

4 <c:catch [var=”varName”]>

可能抛异常的部分

<c:catch>

5 <c:if test=”booleanexpressopn” var=”var” [scope=”page|request|session|application”] />

6 <c:choose>

<c:when test="${product.onSale}">

${product.salesPrice} On sale!

</c:when>

<c:otherwise>

${product.price}

</c:otherwise>

</c:choose>

7 <c:forEach items="collection" [var="var"] [varStatus="varStatus"] [begin="startIndex"] [end="stopIndex"] [step="increment"]>

JSP elements

</c:forEach>

8 <c:forTokens items="stringFoTokens" delims="delimmmmiters" [var="varName"]

            [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>

 ...body内容....

</c:forTokens>

9 <c:import url="url" [context="context"]

[var="varName"] [scope="{page|request|session|application}"] [charEncoding="charEncoding"]>

内容

</c:import>

10 <c:url value="value" [context="context"] [var="varName"]

[scope="{page|request|session|application}"] />

语法2：本体内容代表查询字符串(Query String)参数

<c:url value="value" [context="context"] [var="varName"]

[scope="{page|request|session|application}"] >

<c:param> 标签

</c:url>

11 <c:redirect url="url" [context="context"] />

语法2：本体内容代表查询字符串(Query String)参数

<c:redirect url="url" [context="context"] >

<c:param>

</c:redirect >

## Fn标签

<%@ taglib prefix=”fn” uri=<http://java.sun.com/jsp/jstl/functions>%>

1. <fn:contains(str,substr)> boolean
2. containsIfnoreCase(str,substr) boolean
3. endsWith(str,suffix) boolean
4. escapeXml(str) String
5. indexOf(str,substr) int
6. join(arry,separator) String 将数组用指定连接符拼接并返回
7. length(array|Collection) int
8. replace(str,before,after) String
9. split(str,separator) Array
10. startsWith(str,prefix) boolean
11. subString(str,begin,end) String
12. subStringAfter(str,substr) String
13. subStringBefore(str,substr) String
14. to[Lower | Upper]case(str) String
15. trim(str) String 去除首尾空格

# jQuery

[**http://jquery.cuishifeng.cn/**](http://jquery.cuishifeng.cn/)

1 根据不同选择器找到对象，类、id，所有的div

* 类选择器：var es = $(‘button.className,#idName,div’)[0]; // dom对象

var ess = $(es); // jQuery对象

Object.prototype.toString.call(ess); // 精确输出**对象类型**

* id选择器：$(‘#idName’)
* 所有的div：$(‘div’)
* 所有的有id属性的input标签：$(‘input[id]’)
* 有name属性的input标签，且name为nm：$(‘input[name=”nm”]’)
* ul 下的指定列、偶数列、奇数列、3倍数加一的列（4,7,10…）：$(‘ul li-nth-child([2 | even | odd | 3n+1])’)
* ul 下的第一（最后）行：$(‘ul li-[first | last]-child’)
* $(select option:selected)

2 jQuery转为HTML\*Element：$(‘div’).get(0);

## attr（jQuery 1.0）和prop（jQuery 1.6）

<http://www.jb51.net/article/89038.htm> 尽量使用prop()函数

1操作对象不同：attribute表示HTML文档节点的属性，property表示JS对象的属性。

attr方法依赖于Element的getAttribute()和setAttribute()方法，prop函数依赖的则是JS中原生的对象属性获取和设置方式。

2属性值类型不同：attr函数操作的是文档节点的属性，因此设置的属性值只能是字符串类型，如果不是也会调用其toString方法；prop函数设置的属性值可以包括数组对象在内的任意类型。

此外，对于表单元素的“checked”、“selected”、“disabled”等属性，在jQuery 1.6之前，attr()获取这些属性的返回值为Boolean类型：如果被选中(或禁用)就返回true，否则返回false。

但是从1.6开始，使用attr()获取这些属性的返回值为String类型，如果被选中(或禁用)就返回“checked”、“selected”或“disabled”，否则(即元素节点没有该属性)返回undefined。并且，在某些版本中，这些属性值表示文档加载时的初始状态值，即使之后更改了这些元素的选中(或禁用)状态，对应的属性值也不会发生改变。

<http://jquery.cuishifeng.cn/>

添加类：$(…).addClass()

判断是否有类：$(…).hasClass()

移除类：$(…).removeClass()

## jQuery attr、prop、css

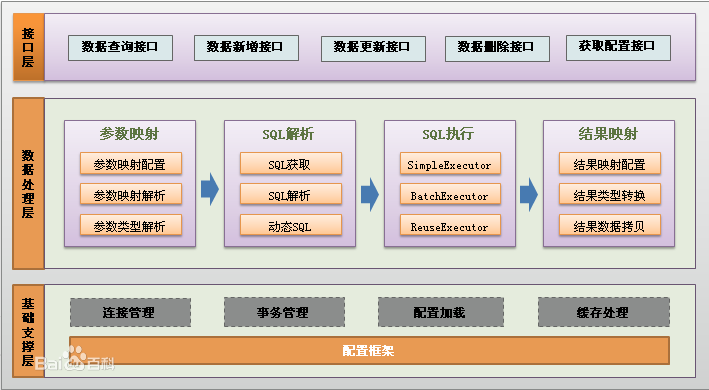
1. w3c对于属性的定义包括：**标准属性**（id，class，style…），**语言属性**（lang（语言类型 en），dir（文本方向ltr|rtl）），元素**固有属性**（a的href，input的type等），不存在的属性叫**新增属性**。
2. attr( )：可以设置元素的属性值（固有属性，新增属性），可以获取元素本来就有的属性以及新增属性。如果要获取的属性**没有设置**，那么获取到的结果是 undefined；
3. prop( )：可以设置元素的属性值（固有属性，新增属性），可以获取元素本来就有的属性值，**如果是额外设置的其他属性，则无法通过prop（）获取。**
4. 对于**固有的HTMl属性**，如果在元素中没有设置（html中没有指定），那么 attr 获取的结果是undefined 而prop（js对象的属性）可以获取那个属性的值（默认值）。
5. css（）设置和获取样式属性（即style中设置的值）。

# MyBatis

官方文档：<http://www.mybatis.org/mybatis-3/zh/getting-started.html>

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。

架构图



## 处理流程

1. 加载配置并初始化。触发条件：加载配置文件

处理过程：将SQL的配置信息加载成为一个个MappedStatement对象（包括了传入参数映射配置、执行的SQL语句、结果映射配置），存储在内存中。

1. 接收调用请求。触发条件：调用Mybatis提供的API

传入参数：为SQL的ID和传入参数对象处理过程：将请求传递给下层的请求处理层进行处理。处理操作请求。触发条件：API接口层传递请求过来

传入参数：为SQL的ID和传入参数对象

处理过程：根据SQL的ID查找对应的**MappedStatement**对象。

根据传入参数对象解析MappedStatement对象，得到最终**要执行的SQL和执行传入参数**。

**获取数据库连接**，根据得到的最终SQL语句和执行传入参数到数据库执行，并**得到执行结果**。

根据MappedStatement对象中的结果**映射配置对得到的执行结果进行转换处理**，并得到最终的处理结果。

1. 释放连接资源。
2. 返回处理结果将最终的处理结果返回。

## 使用步骤

1 添加mybatis jar包

2 配置mybatis-config.xml文件：mybatis框架工作的设置和属性，可以使用 ${} 来直接引用properties标签中指定属性文件中的属性。

3 配置bean 对应的xml配置文件

1. mybatis中 # 和 $ 的区别

* #：是将传入的值当字符串处理，使用#可以很大程度上防止SQL注入（更常用），JDBC解析为一个占位符 ? 等。
* $：是将传入的数据**直接显示**，生成语句，在order by中就需要使用 $，仅为纯粹的字符串替换。

#{id}：创建一个预处理语句，相当于JDBC的？。

#与$的区别最大在于： #{} 传入值时，sql解析时，参数是带引号的，而${}传入值，sql解析时，参数是不带引号的。

\* 注意：用 ’ ’ 包住 ${} ，同时参数名为value（只有一个参数时）

SELECT \* FROM user WHERE name LIKE '%${**value**}%'

#{beanFiledName} beanFiledName 为实体类（如User）的字段名。（insert操作，且parameterType为bean）

INSERT INTO user(name,password,state,age) VALUES(#{**userName**},#{**userPassword**},#{**userState**},#{**userAge**})

# {mapKeyName} mapKeyName 为传的数据（Map）的键名。（insert操作，且parameterType为map/hashmap）

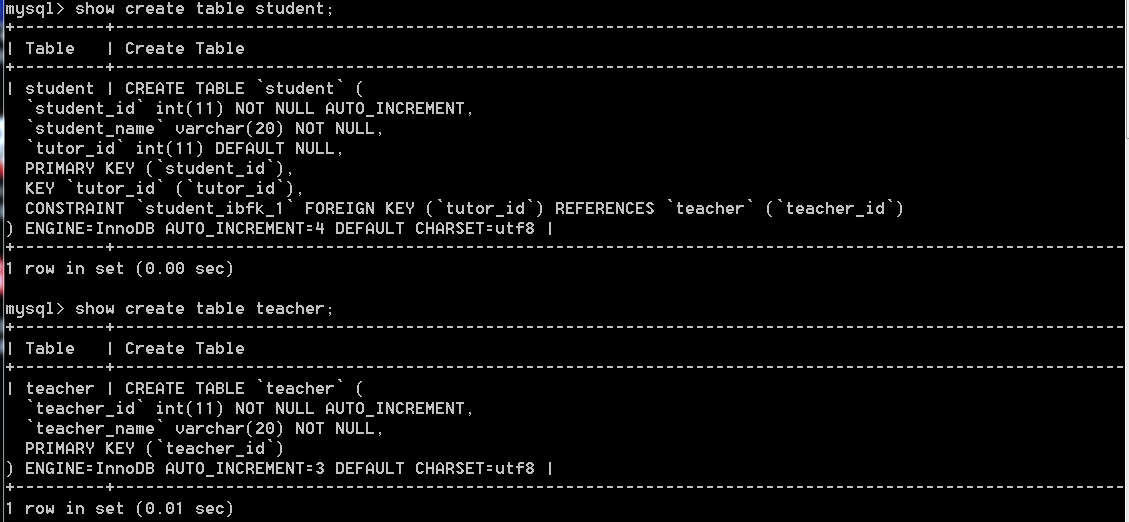
INSERT INTO user(name,password,state,age) VALUES(#{n},#{p},#{s},#{a})

Mapper接口：在mybatis框架中，写dao层的mapper接口时，是不可以进行方法的重载的

## myBatis两种方式完成一对一和一对多查询

mybatis官网文档：<http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html>

数据库中表：



实体类（**必须包含空参一个空参构造函数**，数据库字段对应成员变量的getter和setter）：

Student.java：

int id; String name; Teacher tutor;

Teacher.java：

int id; String name; List<Student> students;

### xml配置

需要在mybatis配置文件的mappers中注册XXXMapper.xml（resource），可将实体类在typeAliases中声明别名以简化xml配置。

XXXMapper.xml的数据库查询语句中使用 \* 可查询出resultMap中所有需要的column，不使用 \* 则需要将resultMap中用到的column都在select中声明：

select \* from …

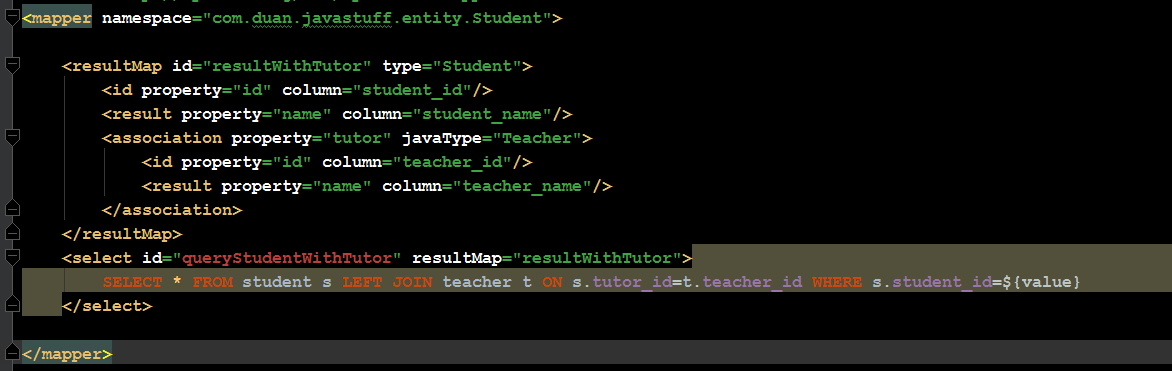
select s.student\_id,s.student\_name,s.tutor\_id,t.tercher\_id,t.teacher\_name …

配置在resultMap标签中：

1. property为实体类对应的成员变量名
2. column为数据库对应字段的字段名
3. id为主键专用。

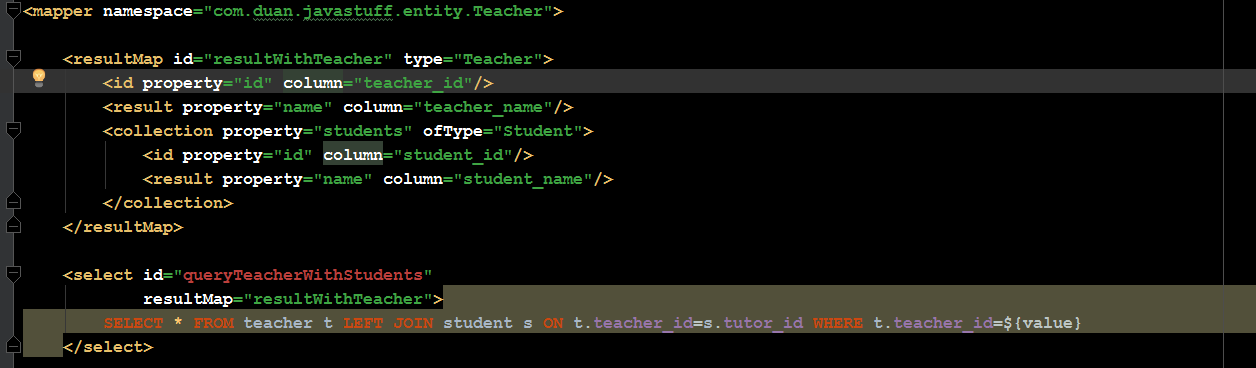
* 一对一

**association**为关联对象的映射，javaType为对应的类名（实体类或其他对应java类）



* 一对多

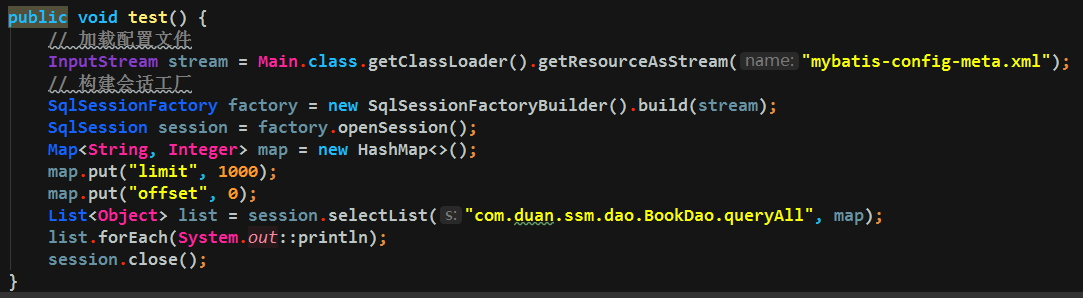
collection标签为关联集合的映射，ofType为集合中元素对应的类名。

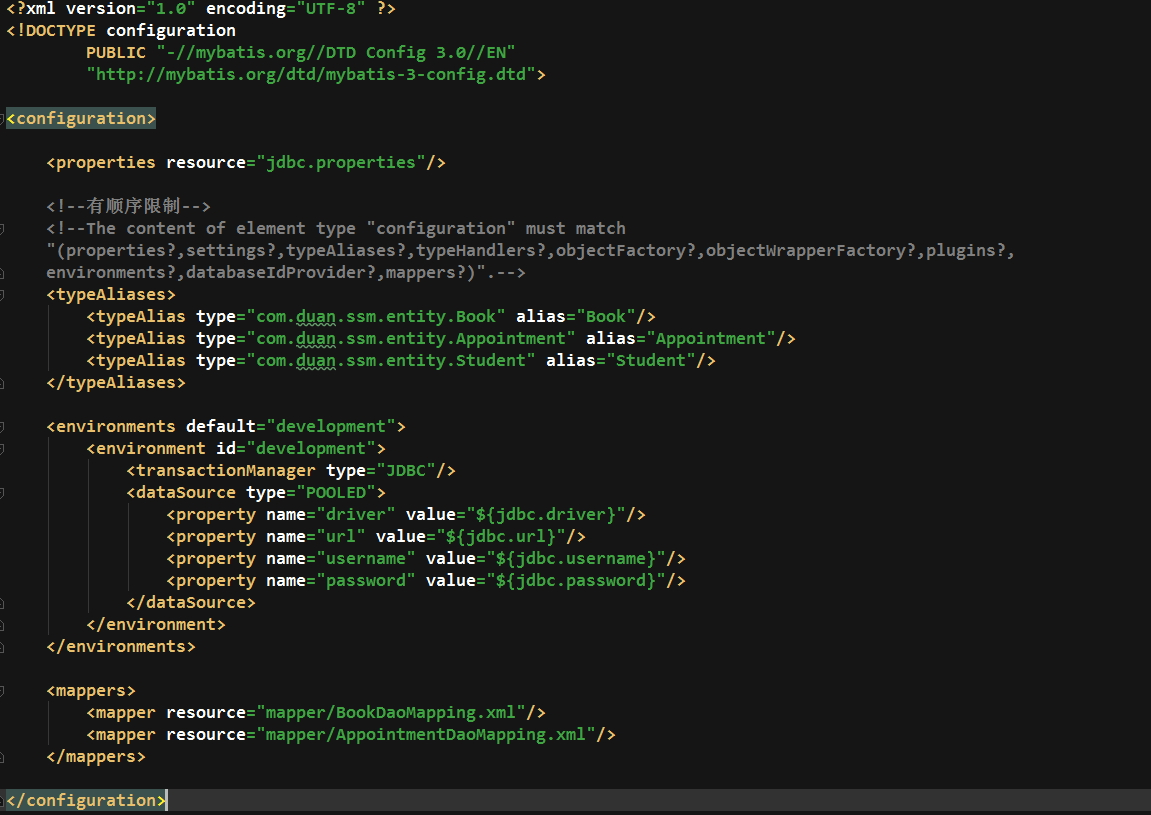


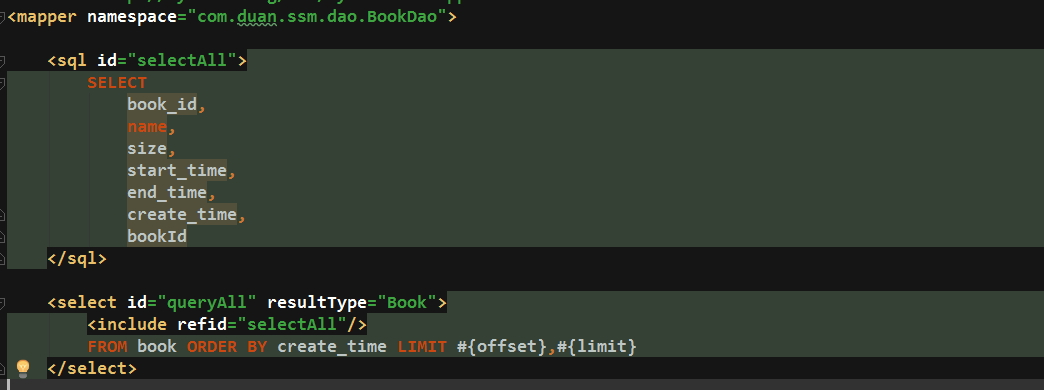
调用示例：

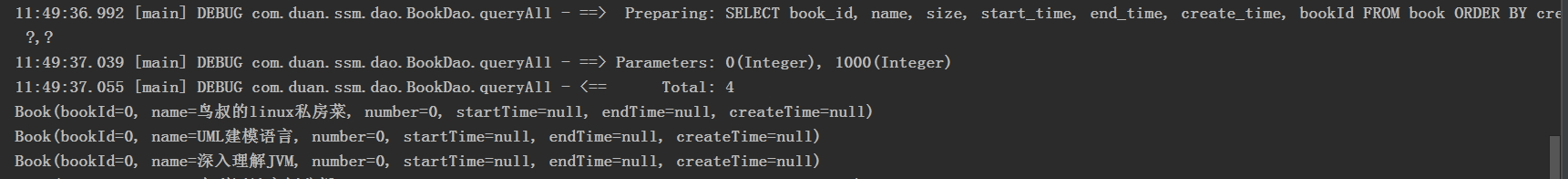


完整配置









### java注解配置

创建对应的接口，XXXMapper.java，并在mybatis配置文件的mappers标签中注册（clas

）。

@Select：书写sql语句

@Results：相当于xml中的resultMap标签

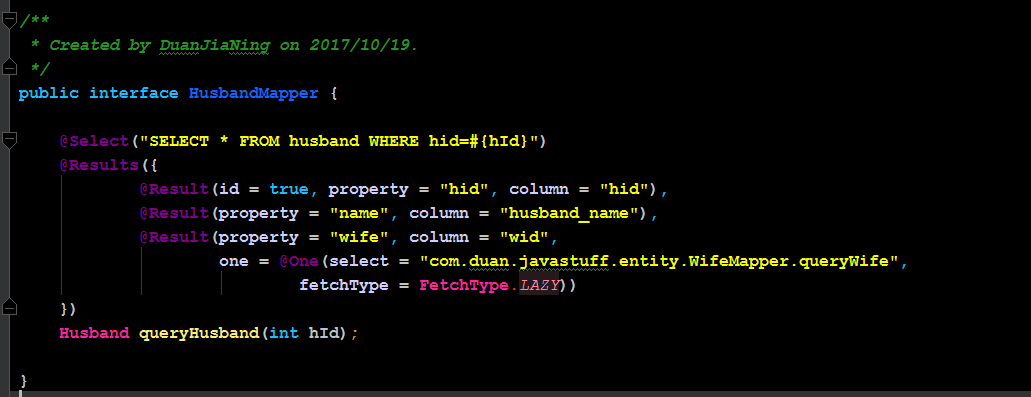
property：实体类成员变量名

column：数据库对应字段名

id：true/false，是否为主键

one：一对一关系，用 @One 注解：指向另一个方法调用（select属性给定方法名）来获取值，fetchType指定获取方式，LAZY、EAGER、DEFAULT。

示例：



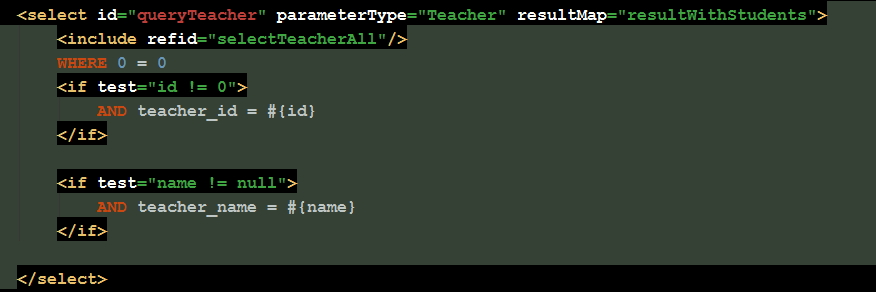
使用示例：



## 动态SQL（myBatis）

### if语句

test中放置判断语句，从前往后逐个判断。



示例解析：

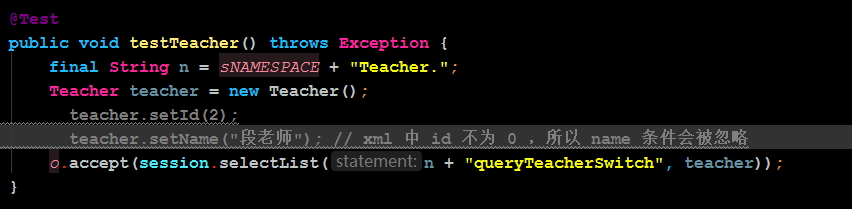
1如果传入的Teacher对象的id属性不等于0，sql语句拼接AND teacher\_id = #{id}，这里的id为Teacher对象传入的值（虽然mybatis会通过getXXX将结果自动装箱，但由于Teacher的id为int，默认值为0，所以不能用null判断），之后再判断Teacher对象的name属性（String类型）是否为null，不是则sql拼接AND teacher\_name = #{name}。

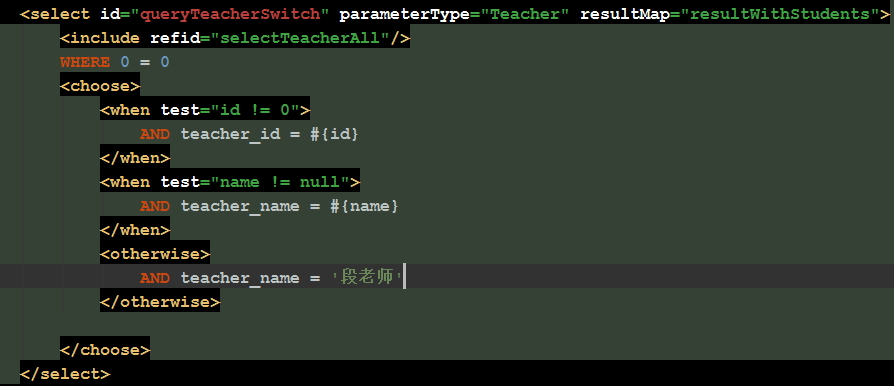
2 若id=0或name为null，就不拼接语句，即默认查询所有。

3 传参时的Teacher传null，或直接调用无参方法时，不是调用其bean的无参构造器创建对象后传递，处理方法也不同…结论是动态sql中用到的参数都用null替代。

### choose语句

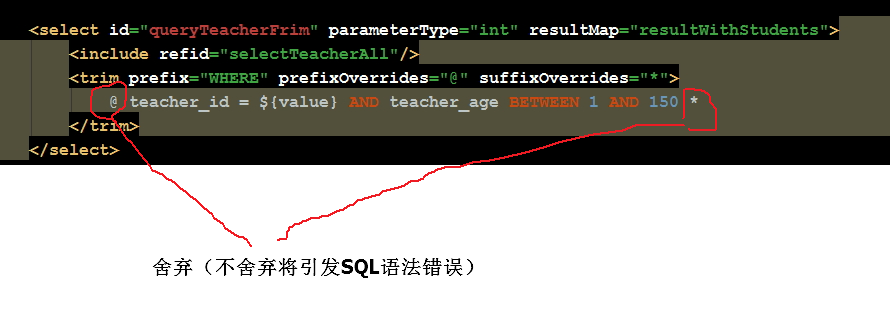
相当于java的switch（choose）、case（when）、default（otherwise）





### trim

对包含的内容加上prefix（前缀），suffix（后缀），prefixOverrides（忽略首部的某些内容），suffixOverrides（忽略尾部的某些内容）。



解析后SQL：

SELECT \* FROM teacher WHERE teacher\_id = 1 AND teacher\_age BETWEEN 1 AND 150

### where

简化SQL的where判断，能智能处理and和or。

1 where元素在写入where元素的地方拼接一个WHERE，如果where中的条件都不满足会查出所有（丢弃WHERE条件）。

2 如果AND/OR开头会被智能的舍弃

3 不用考虑空格的问题

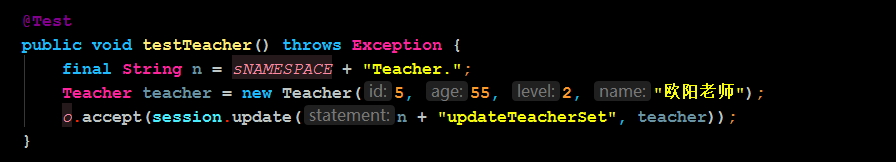


SELECT \* FROM teacher WHERE teacher\_id = ? AND teacher\_age > 10

### set

主要用于更新操作。

功能类似于where元素，在sql语句前拼接SET，如果语句以逗号结束的话把逗号丢弃。





UPDATE teacher **SET** teacher\_name = ?, teacher\_age = ?, teacher\_level = ? **WHERE** teacher\_id = ?

SQL语法中有些自动如果值为null的话会出错，有了set元素我们就可以只更新那些修改了的字段。

### foreach

主要用在构建**IN**条件。

item：迭代到的元素

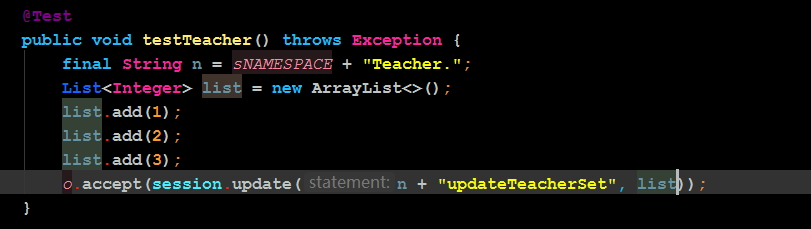
index：迭代时的下标（规则同java中Collection下标）

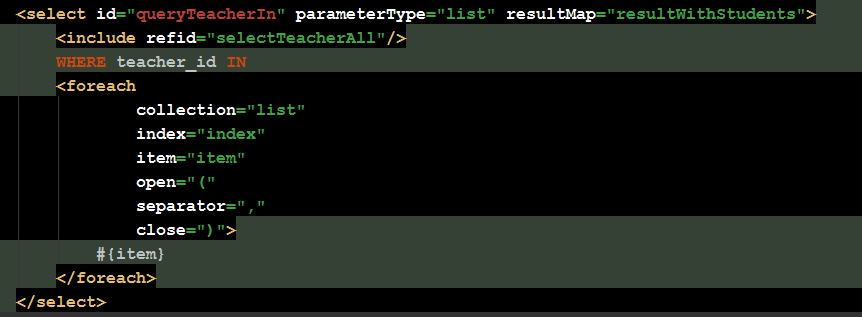
collection：list、array、或map（与paramerType有关，可以不指定，让myBatis自己推断）

open：表示SQl语句以什么字符串作为开始标记

separator：每一次迭代到的元素拼接到SQL语句中时用什么字符串作为分隔符

colse：表示SQl语句以什么字符串作为结束标记





## 缓存机制-cache

<http://blog.csdn.net/luanlouis/article/details/41390801>

1. 一级缓存：基于PerpertualCache的HashMap本地缓存，作用域为session，当session被flush或close时，session中的所有Cache将被清空。

深入理解实现：<http://blog.csdn.net/luanlouis/article/details/41280959>

1. 二级缓存：和一级缓存机制一样，基于PerpetualCache的HashMap本地缓存，作用域为Mapper（NameSpace）.

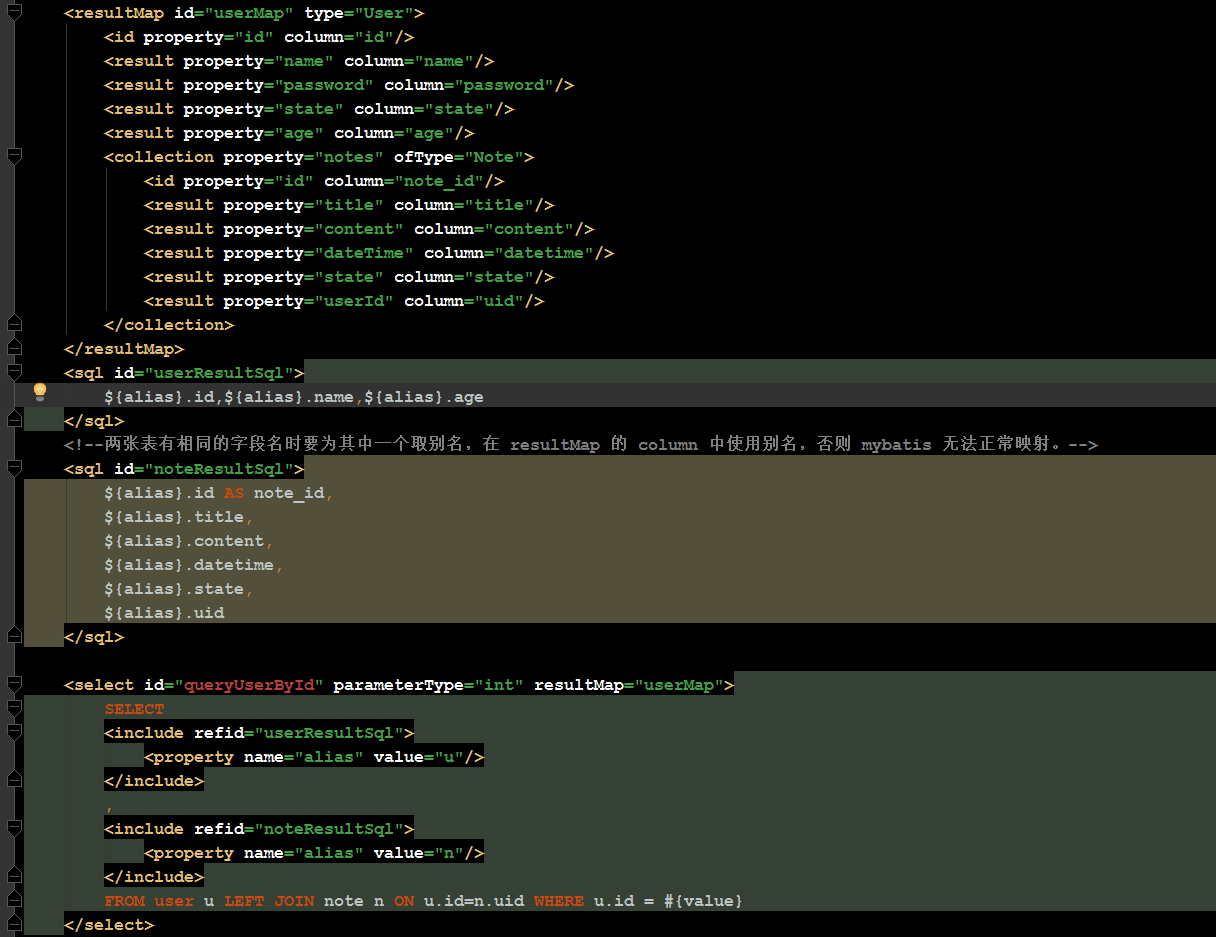
深入理解实现：<http://blog.csdn.net/luanlouis/article/details/41408341>

1. <http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html#cache>

## mybatis的resultMap两张表字段名相同时的特殊处理

进行一次查询时，如果查询结果依赖的多张表拥有相同的字段名，此时需要进行特殊处理，否则mybatis无法正常映射：

1. 在SQL语句中对重复的字段取别名（AS），使之不重复。
2. resultMap中column使用SQL语句中取的别名



# properties文件

<http://blog.csdn.net/easy_gemini/article/details/8229811>

配置文件，使具有通用性。各种语言都有自己的配置文件类型，如Python的.ini，java的.properties 等。

格式语法：

* #为注释标记
* 以key和value键-值得方式保存数据，用=连接

如：

# 以下为数据库连接信息

driver=com.mysql.jdbc.driver

# 以下为…

name=tom

=两端不要有空格，需要用空格时使用\u0020（中文空格为\u3000）。

java.util包中的Properties类可处理.properties文件，getPropertiy(key)，setPropertiy(key,value)，load，clear等。

# Web.xml中的classpath

<http://blog.csdn.net/gjswxhb/article/details/21330225>

classpath代表 /WEB-INF/classes/这个路径，将一个java web工程发布为war（web Application Archive），解压后查看就能看到。

使用场景，SSM架构中，配置Spring的上下文环境：

<servlet>

<servlet-name>springMVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<!--Sources标注的文件夹下需要新建一个spring文件夹-->

<param-name>contextConfigLocation</param-name>

<param-value>**classpath:**spring/spring-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

<async-supported>true</async-supported>

</servlet>

classpath:applicationContext.xml

也可以使用  /WEB-INF /classes/ applicationContext.xml  代替

classpath: 只会到class路径中查找文件

classpath\*: 不仅包含class路径，还包括jar文件中的class路径。

# XSD（xml schema definition）

1．  定义一个Xml文档中都有什么元素

2．  定义一个Xml文档中都会有什么属性

3．  定义某个节点的都有什么样的子节点，可以有多少个子节点，子节点出现的顺序

4．  定义元素或者属性的数据类型

5．  定义元素或者属性的默认值或者固定值

# javaee项目开发流程

<https://zhidao.baidu.com/question/433244940.html>

# 整合最优雅SSM框架

<https://www.2cto.com/kf/201606/518341.html>

D:\JetBrains\IdeaProjects\SsmTest

# 企业开发中为什么选择logback来记录日志，而不是log4j:

<http://www.cnblogs.com/rollenholt/p/3525822.html>

保证logback的组件更加快速的同时，同时所需的内存更加少。

crul：

curl是利用URL语法在命令行方式下工作的开源文件传输工具。它被广泛应用在Unix、多种Linux发行版中，并且有DOS和Win32、Win64下的移植版本。

# mybatis中的CDATA标签的用法

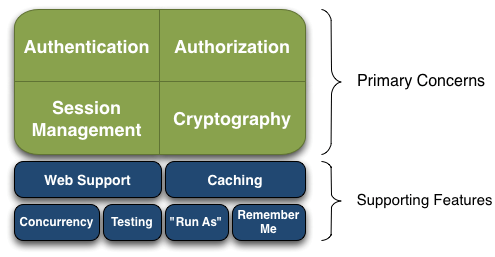
<http://kisuntech.blog.51cto.com/8003773/1313797>

术语 CDATA 指的是不应由 XML 解析器进行解析的文本数据（Unparsed Character Data）。CDATA 部分由 "<![CDATA[" 开始，由 "]]>" 结束。

# Shiro简介—《跟我学Shiro》

<http://jinnianshilongnian.iteye.com/blog/2018936/>

认证、授权、加密、会话管理、与Web集成、缓存等



Authentication：身份认证，登陆

Authorization：权限验证，授权

Session Manager：会话管理

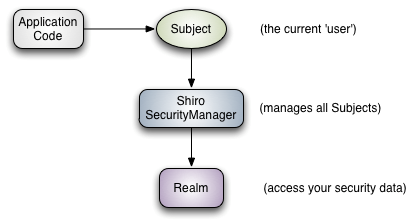
Cryptography：加密，数据安全性

Web Support：Web支持，易于集成到Web环境

Caching：缓存

Concurrency：多线程并发验证

Testing，Run As，Remember Me。



Subject：主体，任何可以与应用交互的“用户”

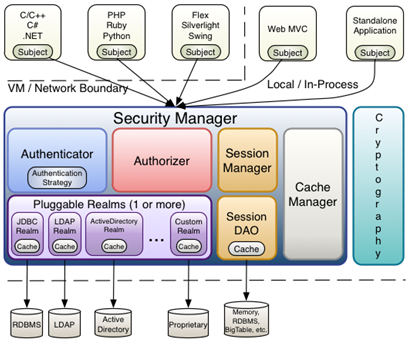
SecurityManager：Subject是门面，SercurityManager才是正真做事的人

Realm：域，安全数据源

也就是说对于我们而言，最简单的一个Shiro应用：

1、应用代码通过Subject来进行认证和授权，而Subject又委托给SecurityManager；

2、我们需要给Shiro的SecurityManager注入Realm，从而让SecurityManager能得到合法的用户及其权限进行判断。



SessionDao：用户会话的dao（CRUD）

CacheManager：缓存管理器

# ${pageContext.request.contextPath}

Jsp中使用${pageContext.request.contextPath}获取绝对路径。

# lombok

<http://blog.csdn.net/zhglance/article/details/54931430>

<https://projectlombok.org/features/all>

项目中使用的bean，entity等类，绝大部分需要getter，setter，toString，equals和hashcode方法，虽然IDE能快速生成方法，但一旦类有所改动，就需要对这些代码进行大量修改，使用了lombok提供的注解就可以解决这些问题。

@Setter，@getter，@ToString，@RequiredArgsConstructor，@EqualsAndHashCode，@NoArgsConstructor或@Data。

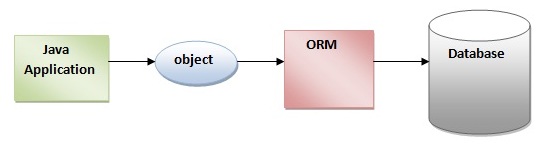
@Data相当于其它6个注解的合集

# REST（Representational State Transfer）表述性状态转移

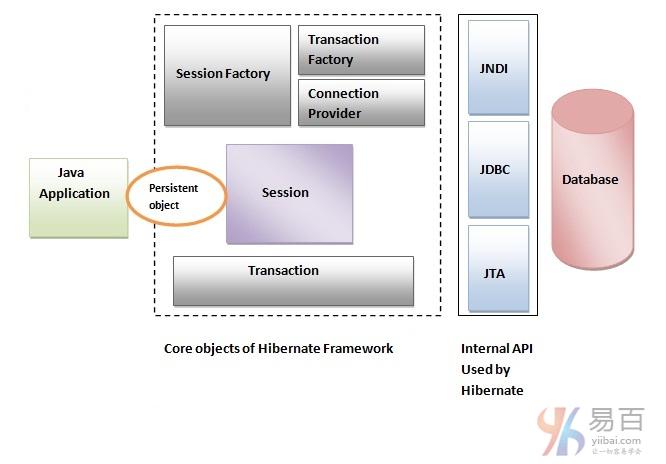
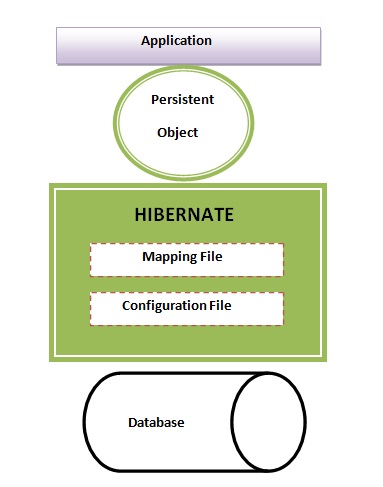
# Hibernate ORM（对象关系映射框架）

<http://hibernate.org/>

java应用程序与数据库交互



开源和轻量级、快速、数据库独立查询（HQL），自动创建表、简化复杂连接、提供查询统计和数据库状态。



JNDI：java命名目录接口

JTA：java事务API

SessionFactory是ConnectionProvider（连接提供者）的会话和客户端工厂，SessionFactory提供了方法来获得Session。

连接提供者(ConnectionProvider)：它是一个JDBC连接工厂。 它从DriverManager或DataSource抽象出来的应用程序。 它是一个可选项。

## Hibernate使用步骤

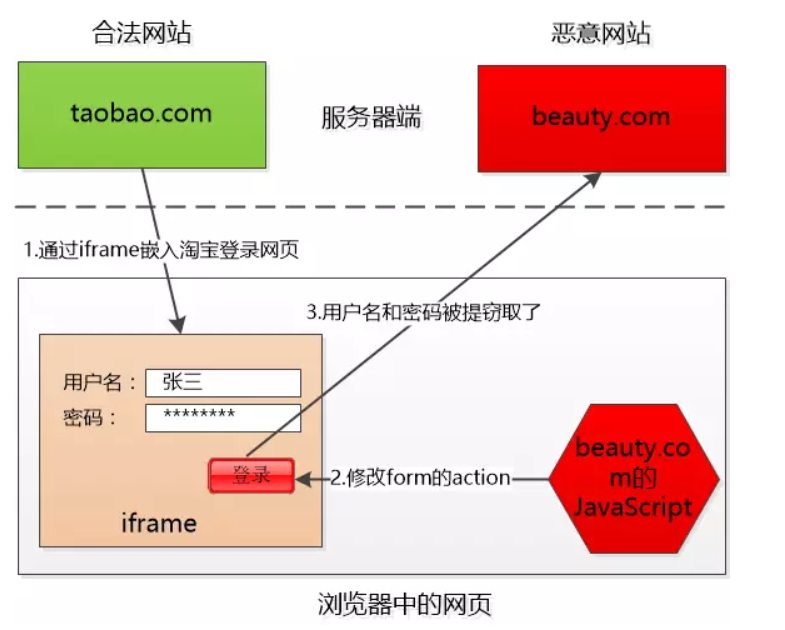
创建java项目，添加jar依赖，创建持久类，创建持久类的映射文件，创建配置文件，创建检索或存储持久对象的类，运行程序。

# 同源策略&跨域请求问题 : Jsonp & Cors

<http://mp.weixin.qq.com/s?__biz=MzAxOTc0NzExNg==&mid=2665514143&idx=1&sn=28ea209c00309e6b93d8d1f76032d7a4&chksm=80d67cdcb7a1f5ca81d8d454a98af56d58b22f6058f100e21ff30e70867ea6e3e922a4f000bf&mpshare=1&scene=23&srcid=1219hjtmL7SXftuOoe4cEI3A#rd>

## 同源策略

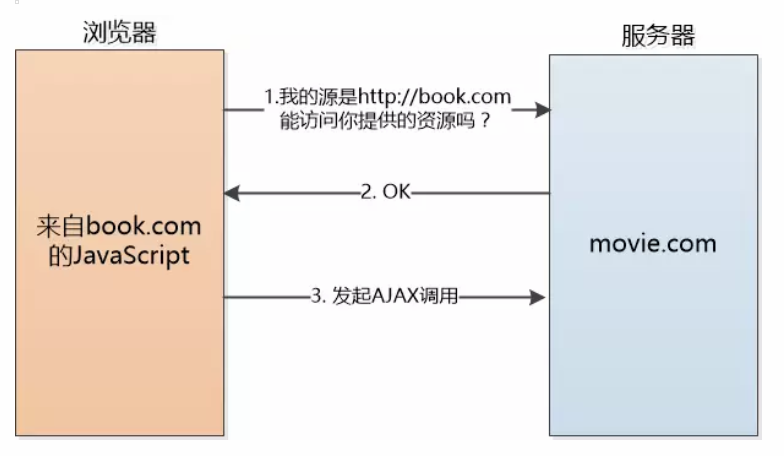
原因：Cookie失窃，非法网页通过javascript访问其他应用的服务器，获取到Cookie后伪装成用户进行非法操作。form的action指向其他的应用，将隐私数据提交到了不法网站，信息被盗取。



同源是指{protocol,host,port} 这三个东西必须得一样！ 例如有这么一个网页： http://www.store.com/product/page.html， 下面的表格列出了各种不同情况：



1. 有**src属性的标签不受同源策略**的影响。比如script、img、iframe、link。相当于浏览器发送**get**请求。原页面的js不能访问通过src加载的资源。
2. 如果两个网页的**一级域名是相同的，他们可以共享cookie**, 不过cookie的domain一定要设置为那个一级域名才可以，例如：”document.cookie = 'test=true;path=/;**domain=store.com'** ”
3. 对于浏览器来说：除了DOM、Cookie、LocalStorage、XMLHttprequest（Ajax）会受到同源策略的限制外，浏览器加载的第三方插件也有各自的同源策略。例如：flash,java applet,silverlight等。
4. 既然服务器（domain）之间是互信的，那一个服务器(domain)可以**设置一个白名单，里边列出它允许哪些服务器(domain)的AJAX请求**。假设movie.com的白名单中有book.com， 那当属于book.com的JavaScript试图访问movie.com的时候，book.com先询问movie的白名单中是否有自己（是否允许自己访问），有就继续访问。



把这个方法叫做Cross Origin Resource Sharing，简称CORS，只不过这个方法需要服务器的配合。

## json jsonp 区别

[**http://www.cnblogs.com/iovec/p/5312464.html**](http://www.cnblogs.com/iovec/p/5312464.html)

1. 同源策略（Same-Origin Policy）：它是由Netscape（网景）提出的一个著名的安全策略。现在所有支持JavaScript 的浏览器都会使用这个策略。所谓同源，就是必须**协议、域名、端口**都一致的，才叫做同源。<http://blog.csdn.net/xiaoxian8023/article/details/27817861>

如：<http://www.12306.cn> 和 <https://www.12306.cn> 协议不一致，不是同源。

直接访问浏览器会提示Access-Control-Allow-Origin，只要在服务端设置响应头即可，但存在安全隐患，因为存在通配符“\*”：response.setHeader(“Access-Control-Allow-Origin”,”\*”)。

1. jQuery的$.ajax支持跨域访问，只需将dataType设置为jsonp（JSON with padding），是json的一种“使用模式”， 可以让网页从别的网域获取资料。jsonp是采用的**js的回调机制**来实现的。使用方式也很简单，简略方式如下：

$.getJSON([http://\*\*\*\*.\*\*/\*\*/&callback=?,function(json){...})](http://****.**/**/&callback=?,function(json)%7b...%7d))；服务端需要获取callback的值并以如下方式返回数据：callback+”(“+jsondata+”)”; 格式是回调函数名+(json数组)。

1. jsonp就是简单的函数（javaScript）调用，数据被包裹在了参数中。
2. json是理想的数据交换格式，但没办法跨域直接获取，于是就将json包裹(padding)在一个合法的js语句中作为js文件传过去。这就是json和jsonp的区别，json是想要的东西，jsonp是达到这个目的而普遍采用的一种方法，当然最终获得和处理的还是json。所以说json是目的，jsonp只是手段。json总会用到，而jsonp只有在跨域获取数据才会用到。

junit 4 需要两个jar： junit-4.12.jar hamcrest-all-1.3.jar

## JSONP

<http://justcoding.iteye.com/blog/1366102>

由于同源策略的限制，XmlHttpRequest只允许请求**当前源（域名、协议、端口）的资源**，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并**执行回调函数**，从而解决了跨域的数据请求。

jsonp 的原理很简单，利用了【前端请求静态资源的时候不存在跨域问题】这个思路，但是**只支持 get**。既然这个方法叫 jsonp，后端数据一定要使用 json 数据。

### 前端js写法

1. **<script** type="text/javascript"**>**
2. function **jsonpCallback**(result) {
3. alert(result.a);
4. alert(result.b);
5. alert(result.c);
6. for(var i in result) {
7. alert(i+":"+result[i]);//循环输出a:1,b:2,etc.
8. }
9. }
10. </script>
11. **<script** type="text/javascript" src="http://crossdomain.com/services.php?**callback**=jsonpCallback"**></script>**

### 前端 jQuery 写法

1. $.ajax({
2. url:"http://crossdomain.com/services.php",
3. dataType:'jsonp',
4. data:'',
5. jsonp:'callback',
6. success:**function**(result) {
7. **for**(**var** i **in** result) {
8. alert(i+":"+result[i]);//循环输出a:1,b:2,etc.
9. }
10. },
11. timeout:3000
12. });

dataType: “jsonp”。除了这个，其他配置和普通的请求是一样的。

### 后端 SpringMVC 配置

如果你也使用 SpringMVC，那么配置一个 jsonp 的 Advice 就可以了，这样我们写的每一个 Controller 方法就完全不需要考虑客户端到底是不是 jsonp 请求了，Spring 会自动做相应的处理。

@ControllerAdvice

public class JsonpAdvice extends **AbstractJsonpResponseBodyAdvice** {

    public JsonpAdvice(){

        // 这样如果请求中带 callback 参数，Spring 就知道这个是 jsonp 的请求了

**super("callback");**

    }

}

以上写法要求 SpringMVC 版本不低于 3.2，低于 3.2 的我只能说，你们该升级了。

### Jsonp原理

首先在客户端注册一个callback, 然后把callback的名字传给服务器。此时，服务器先生成 json 数据。

然后以 javascript 语法的方式，生成一个function , function 名字就是传递上来的参数 jsonp.

最后将 json 数据直接以入参的方式，放置到 function 中，这样就生成了一段 js 语法的文档，返回给客户端。

客户端浏览器，解析script标签，并执行返回的 javascript 文档，此时数据作为参数，传入到了客户端预先定义好的 callback 函数里.（动态执行回调函数）

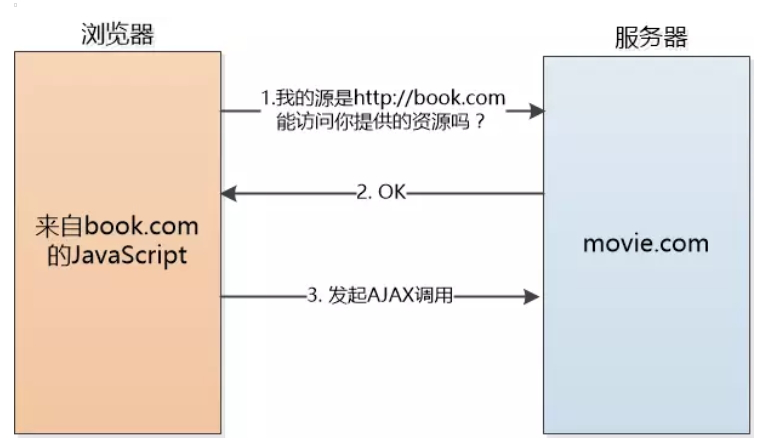
## CORS

Cross-Origin Resource Sharing

毕竟 jsonp 只支持 get 请求，肯定不能满足我们的所有的请求需要，所以才需要搬出 CORS。

通过服务器端中转，例如你是来自book.com的， 现在想访问movie.com，那可以让那个book.com把请求转发给movie.com嘛！利用了代理模式，book.com就是代理人。

跨域资源共享标准新增了一组 HTTP 首部字段，允许服务器声明哪些网站有权限访问哪些资源。另外，规范要求，对那些可能对服务器数据产生副作用的 HTTP 请求方法（特别是 GET 以外的 HTTP 请求，或者搭配某些 MIME 类型的 POST 请求），浏览器必须首先使用 OPTIONS 方法发起一个**预检请求（preflight request），从而获知服务端是否允许该跨域请求**。服务器确认允许之后，才发起实际的 HTTP 请求。在预检请求的返回中，服务器端也可以通知客户端，是否需要携带身份凭证（包括 Cookies 和 HTTP 认证相关数据）。参考文章： <https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access_control_CORS>



需要服务器的配合，CORS 支持以下浏览器，目前来看，浏览器的问题已经越来越不重要了，连淘宝都不支持 IE7 了~~~

Chrome 3+

Firefox 3.5+

Opera 12+

Safari 4+

Internet Explorer 8+

### 前端 jQuery 写法

$.ajax({

    type: "POST",

    url: baseUrl + "/jsonp/post",

**dataType**: 'json',

**crossDomain**: true,

**xhrFields: {**

**withCredentials: true**

**},**

    data: {

        name: "name\_from\_frontend"

    },

    success: function (response) {

        console.log(response)// 返回的 json 数据

        $("#response").val(JSON.stringify(response));

    }

});

dataType: “json”。

crossDomain: true，这里代表使用跨域请求

xhrFields: {withCredentials: true}，这样配置就可以把 cookie 带过去了，不然 session 无法维护（cookie中保存了JSESSIONID），如果没有这个需求，也就不需要配置。

# 实体类（PO DO TO DTO VO BO POJO DAO）

<http://blog.csdn.net/wangxin1982314/article/details/51954264#insertcode>

在项目应用中，VO对应于页面上需要显示的数据（表单），DO对应于数据库中存储的数据（数据表），DTO对应于除二者之外需要进行传递的数据。

## 实体类

实体类有两方面内容，存储数据和执行数据本身相关的操作。这两方面内容对应到实现上，最简单的实体类是POJO类，含有属性及属性对应的set和get方法，实体类常见的方法还有用于输出自身数据的toString方法。

## 领域模型中的实体类

领域模型中的实体类分为四种类型：VO、DTO、DO、PO，各种实体类用于不同业务层次间的交互，并会在层次内实现实体类之间的转化。

业务分层为：视图层（VIEW+ACTION），服务层（SERVICE），持久层（DAO）

## 项目中的实体类

项目中常见的实体类有VO，DO和DTO，命名规则也常是以相应字符串结尾，如\*VO.Java。但是DTO不总是遵循这个规则，而通常与他的用途有关，如写成\*Query.java，表示存储了一个查询条件。项目中实体类出现的业务层次也没有这么严格，例如我们可以在视图层就组装一个DO，也可以将一个VO从持久层传出来，所以与业务分层相关联的划分方法显得有些冗余。从项目代码中抽象出的理解是：VO对应于页面上需要显示的数据，DO对应于数据库中存储的数据，DTO对应于除二者之外需要进行传递的数据。

## PO DO TO DTO VO BO POJO DAO

<http://blog.csdn.net/scholar_man/article/details/48287571>

### PO(persistant object)

持久对象

在 o/r 映射的时候出现的概念，如果没有 o/r 映射，没有这个概念存在了。通常**对应数据模型 ( 数据库 ),** 本身还有部分业务逻辑的处理。可以看成是**与数据库中的表相映射的 java 对象**。最简单的 PO 就是对应数据库中某个表中的一条记录，多个记录可以用 PO 的集合。 PO 中应该不包含任何对数据库的操作。

### DO（Domain Object）

领域对象

就是从现实世界中抽象出来的有形或无形的业务实体。

### TO(Transfer Object)

数据传输对象

在应用程序不同 tie( 关系 ) 之间传输的对象

### DTO（Data Transfer Object）

数据传输对象

这个概念来源于J2EE的设计模式，原来的目的是为了EJB的分布式应用提供粗粒度的数据实体，以减少分布式调用的次数，从而提高分布式调用的性能和降低网络负载，但在这里，我泛指用于**展示层与服务层之间的数据传输对象**。

### VO(value object)

值对象

通常用于业务层之间的数据传递，和 PO 一样也是**仅仅包含数据而已**。但应是抽象出的业务对象 , 可以和表对应 , 也可以不 , 这根据业务的需要 。用 new 关键字创建，由 GC 回收的。

### BO(business object)

业务对象

从业务模型的角度看 , 见 UML 元件领域模型中的领域对象。封装业务逻辑的 java 对象 , 通过调用 DAO 方法 , 结合 PO,VO 进行业务操作。 business object: 业务对象 主要作用是把业务逻辑封装为一个对象。这个对象可以包括一个或多个其它的对象。 比如一个简历，有教育经历、工作经历、社会关系等等。 我们可以把教育经历对应一个 PO ，工作经历对应一个 PO ，社会关系对应一个 PO 。 建立一个对应简历的 BO 对象处理简历，每个 BO 包含这些 PO 。 这样处理业务逻辑时，我们就可以针对 BO 去处理。

### POJO(plain ordinary java object)

简单无规则 java 对象

纯的传统意义的 java 对象。就是说在一些 Object/Relation Mapping 工具中，能够做到维护数据库表记录的 persisent object 完全是一个符合 Java Bean 规范的纯 Java 对象，没有增加别的属性和方法。我的理解就是最基本的 Java Bean ，只有属性字段及 setter 和 getter 方法！。

### DAO(data access object)

数据访问对象

是一个 sun 的一个标准 j2ee 设计模式， 这个模式中有个接口就是 DAO ，它负持久层的操作。为业务层提供接口。此对象用于访问数据库。通常和 PO 结合使用， DAO 中包含了各种数据库的操作方法。通过它的方法 , 结合 PO 对数据库进行相关的操作。夹在业务逻辑与数据库资源中间。配合 VO, 提供数据库的 CRUD 操作.

# Apache

<https://zh.wikipedia.org/wiki/Apache%E8%BD%AF%E4%BB%B6%E5%9F%BA%E9%87%91%E4%BC%9A>

Apache**软件基金会**（Apache Software Foundation，简称为ASF），是**专门为支持开源软件项目而办的一个非营利性组织**。在它所支持的Apache项目与子项目中，所发行的软件产品都遵循Apache许可证（Apache License）。

Apache软件基金会正式创建于1999年，它的创建者是一个自称为“Apache组织”的群体。

## 项目

1. **Tomcat**：一个运行Java Servlet与JavaServer Pages（JSP）的容器。
2. HTTP Server：可以在UNIX，MS-Windows，Macintosh和Netware操作系统下运行的HTTP服务器的项目。
3. **Ant**：基于Java语言的构建工具，类似于C语言的Make工具。
4. **Hadoop**：并发运算编程工具和分布式文件系统。
5. **ibatis**：一个基于JAVA语言的数据持久化框架。
6. **Cordova**：一个基于Javascrip html css 的混生App开发计划。
7. **Maven**：项目集成构建工具。
8. **Shiro**：权限管理.
9. **Struts**：一个基于J2EE平台的MVC 设计模式的Web应用框架。
10. **Subversion**：一个软件版本管理系统。
11. **Lucene**: 全文检索

……

# JAVA事务,JTA,JDBC,JDO,DAO,JNDI概念

<http://dyldragon.iteye.com/blog/789374>

通常认为事务仅与数据库相关，事务必须服从ISO/IEC指定的ACDI原则：

ACDI：**原子性，一致性，隔离性，持久性**。

1. 什么是java事务
2. 为什么需要
3. java事务的类型，有三种：JDBC事务，JTA（java Transcation API）事务，容器事务

**JDBC事务**：由java.sql.Connection对象控制，可通过如下方法控制事务：setAutoCommit，

getAutoCommit，commit，rollback。JDBC事务不能跨越多个数据库。

**JTA事务**：高层的，与实现无关，与协议无关的。JTA允许应用程序执行分布式事务处理。

**容器事务**：容器事务主要是J2EE**应用服务器**提供的，容器事务大多是基于JTA完成，这是一个基于JNDI的，

相当复杂的API实现。这项功能由J2EE应用服务器提供。这使得我们可以简单的指定将哪个方法加入事务，

一旦指定，容器将负责事务管理任务。

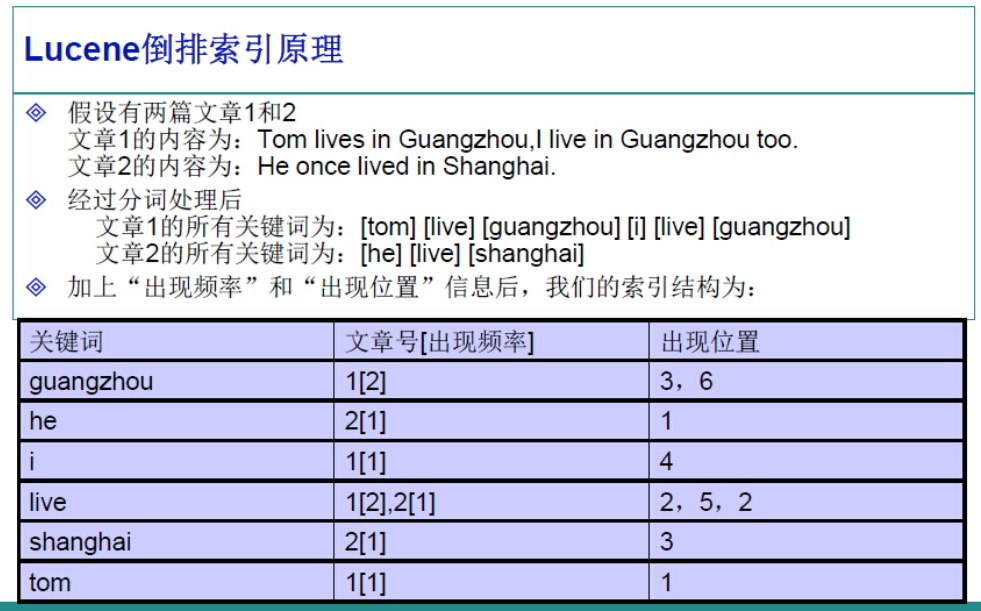
1. 几种事务的差异，JDBC事务控制的局限性在一个数据库连接内，但是其使用简单。 JTA事务的功能强大，事务可以跨越多个数据库或多个DAO，使用也比较复杂。 容器事务，主要指的是J2EE应用服务器提供的事务管理，局限于EJB应用使用。

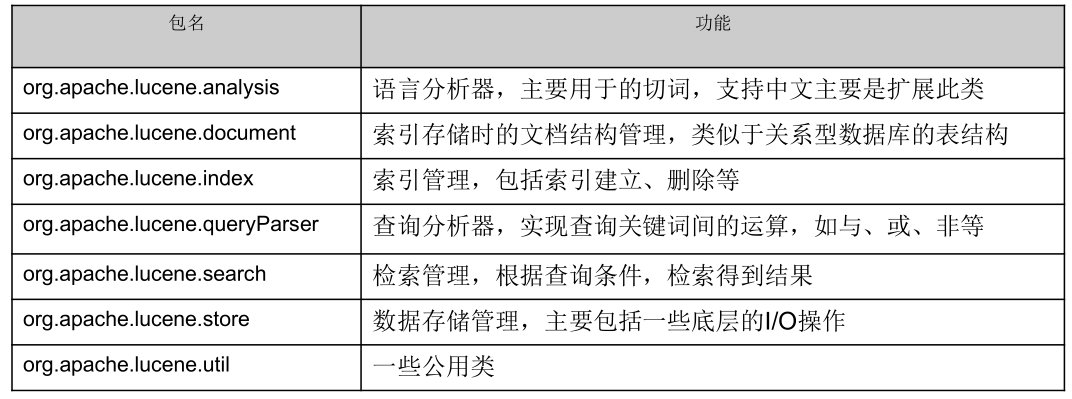
# 全文检索Apache Lucene

<http://www.cnblogs.com/xing901022/p/3933675.html>

<https://pan.baidu.com/s/1jGxGi6E>

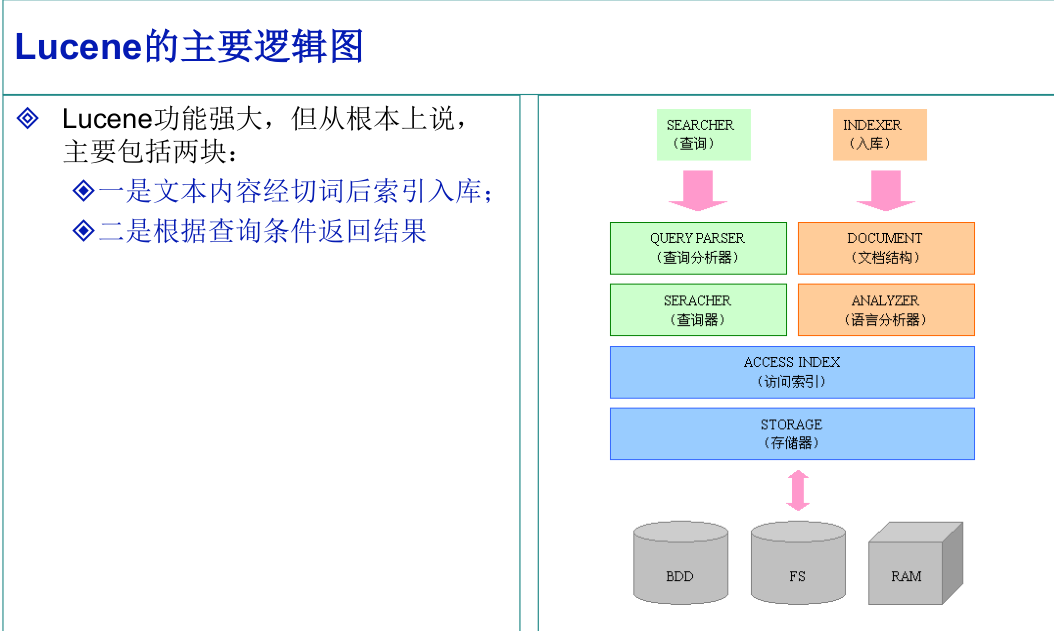
我们一个文件夹中，或者一个磁盘中有很多的文件，记事本、world、Excel、pdf，我们想根据其中的关键词搜索包含的文件。例如，我们输入Lucene，所有内容含有Lucene的文件就会被检查出来。这就是所谓的全文检索。



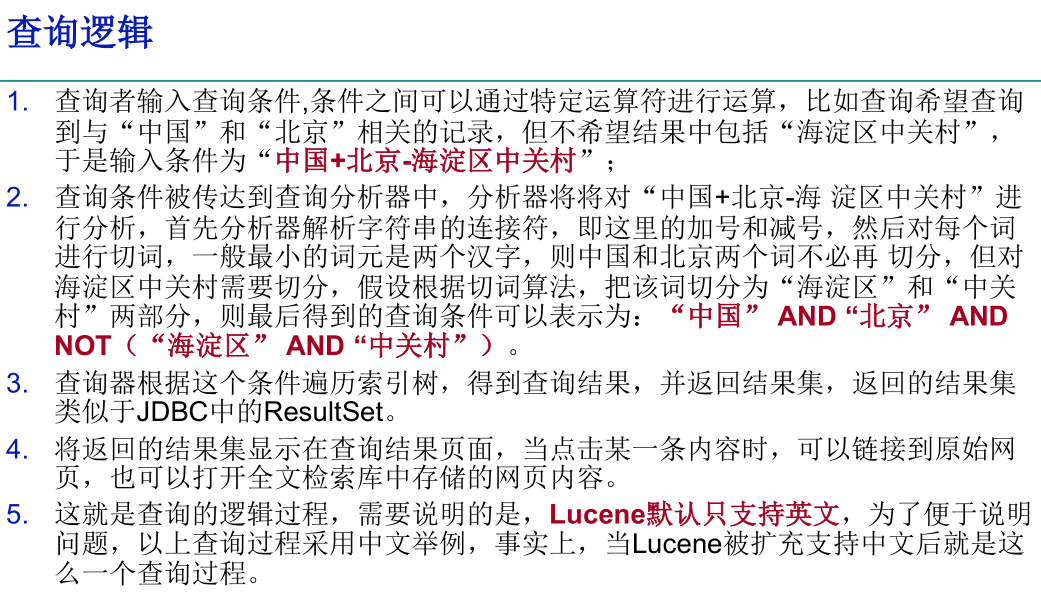


我们可以看到，Lucene的使用主要体现在两个步骤：

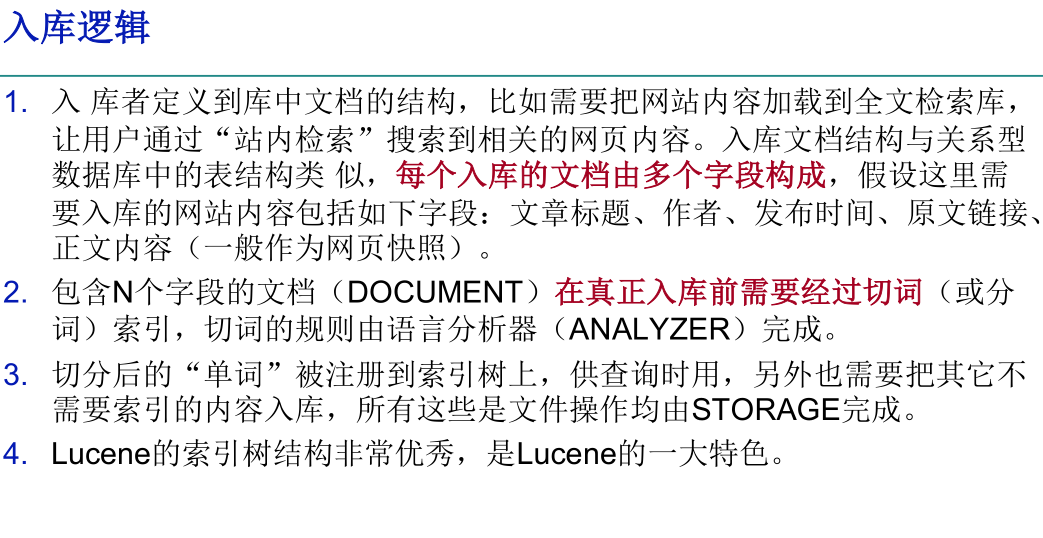
1. 创建索引，通过IndexWriter对不同的文件进行索引的创建，并将其保存在索引相关文件存储的位置中。
2. 通过索引查寻关键字相关文档。



## 查询逻辑



## 入库逻辑



## 核心入库类

### IndexWriter

IndexWriter是在索引过程中的中心组件。IndexWriter这个类创建一个新的索引并且添加文档到一个已有的索引中。 你可以把IndexWriter 想象成让你可以**对索引进行写操作的对象**，但是不能让你读取或搜索。

### Directory

Directory类**代表一个Lucene索引的位置**。它是一个抽象类. 其中的两个实现:

 第一个是 FSDirectory，它表示一个存储在文件系统中的索引的位置。

 第二个是 RAMDirectory，它表示一个存储在内存当中的索引的位置。

### Analyzer

在一个文档**被索引之前，首先需要对文档内容进行分词处理**，从而剔除一些冗余的词句（例如：a，the,they等),这部分工作就是由 Analyzer 来做的。

### Document

Document文档**类似数据库中的一条记录**，可以由好几个字段（Field）组成，并且字段可以套用不同的类型。一个Field代表与这个文档相关的元数据。元数据如作者、标题、主题、修改日期等等，分别做为文档的字段索引和存储。

Document的方法：

void add(Fieldable field)添加一个字段（Field）到Document中;

String get(String name)从文档中获得一个字段对应的文本.

### Field

Field 对象是用来描述一个文档的某个**属性**的，比如一封电子邮件的标题和内容个 可以用两个 Field 对象分别描述。

## 核心搜索类

### IndexSearcher

IndexSearcher是用来在建立好的索引上进行搜索的， 它只能以只读的方式打开一个索引，所以可以有多个IndexSearcher的实例在一个索引上进行操作。

Search方法

1. 返回值为Hits：

search(Query query) 、search(Query query, Filter filter) 、search(Query query, Sort sort) 、search(Query query, Filter filter, Sort sort)

1. 返回索引中**得分较高的文档集合**,这些方法中,都带有一个int形式参数,表示取出置于TopDocs集合中的文档数量：

search(Query query, Filter filter, int n)、search(Weight weight, Filter filter, int n)、search(Query query, Filter filter, int n, Sort sort)、search(Weight weight, Filter filter,int n, Sort sort)

1. 无返回值

search(Query query, Filter filter, HitCollector results)、search(Query query, HitCollector results) 、search(Weight weight, Filter filter,HitCollector results)

### Term

Term是**搜索的基本单元**。一个Term对象有两个String类型的域组成：字段的名称和字段的值。在搜索时，你可能创建Term对象并和TermQuery同时使用。其中第一个参数代表了要在文档的哪一个Field上进行查找，第二个参数代表了要查询的关键词。

Query q = new TermQuery(new Term(“fieldName”, “queryWord ”));

这段代码使Lucene找出在fieldName字段中含有单词queryWord的所有文档。因为TermQuery对象继承自它的抽象父类Query，你可以在等式的左边用Query类型。

### Query

Query是一个抽象类，这个类的目的是把用户输入的查询字符串封装成Lucene能够识别的Query。

#### TermQuery

TermQuery是抽象类Query的一个子类，它同时也是Lucene支持的最为基本的一个查询类。

### Hits

Hits是用来保存搜索的结果的。基于性能考虑，Hits的实例并不从索引中加载所有匹配查询的所有文档，而是每次一小部分。

public final int length()

public final Document doc(int n)

public final float score(int n)

public final int id(int n)

public Iterator iterator()

### BooleanQuery

布尔搜索：BooleanQuery是实际开发过程中经常使用的一种Query。它其实是一个组合的Query，在使用时可以把各种Query对象添加进去并标明它们之间的**逻辑关系**。

### 其他

BooleanClause布尔搜索、RangeQuery范围搜索、PrefixQuery 前缀搜索、PhraseQuery短语搜索、FuzzyQuery模糊搜索、WildcardQuery通配符搜索、SpanQuery 跨度搜索、RegexQuery正则表达式搜索、MultiFieldQueryParser 多域搜索、MultiSearcher多索引搜索、ParallelMultiSearcher多线程搜索。

## 使用

Lucene使用前的准备主要包括三个步骤：1. **将数据转化为文本，2. 分析，3. 将它保存至索引**

**在索引中清除Document**：Document的删除是由IndexReader来完成的。这个类并不立即从索引中删除Document。它只做个删除的标志，等待IndexReader的close()方法调用时真正的Document删除。除了我们通过指定Document编号来删除单个Document之外，可以用IndexReader的**delete**(Term)方法删除多个Document。使用这个删除方法，允许你删除所有包含指定Term的Document

**恢复Document**：因为Document的删除延迟到IndexReader实例关闭时才执行，Lucene允许程序改变想法并恢复已做删除标记的Document。对IndexReader的undeleteAll()方法的调用通过清除索引目录中的.del文件来恢复所有删除的Document。所以在关闭IndexReader实例关闭之后Document就保留在索引中了。只能使用与删除Document时同一个IndexReader实例，才能调用**undeleteAll**()来恢复Document。

**更新索引中的Document**：Lucene并没有提供更新方法；Document必须首先从索引中删除然后再重新添加它。（新版本中封装了更新方法）

1. 打开IndexReader。2. 删除所有你要删除的Document。3. 关闭IndexReader。 4. 打开IndexWriter。5. 添加你要添加的所有Document。6. 关闭IndexWriter。

# Redis

<http://www.yiibai.com/redis/redis_quick_guide.html>

Redis是一个使用ANSI **C**编写的**开源**、**支持网络、基于内存**、可选**持久性**的**键值对**存储**数据库**。Redis是最流行的键值对存储数据库。

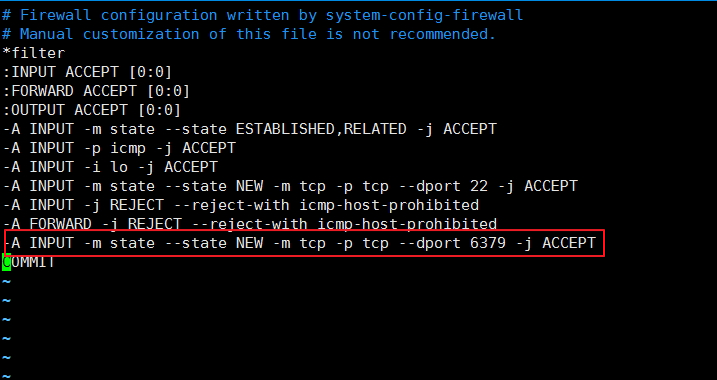
## 数据模型

Redis的外围由一个键、值映射的字典构成。与其他非关系型数据库主要不同在于：Redis中值的类型不仅限于字符串，还支持如下抽象数据类型：

1. 字符串列表
2. 无序不重复的字符串集合
3. 有序不重复的字符串集合
4. 键、值都为字符串的哈希表

值的类型决定了值本身支持的操作。Redis支持不同无序、有序的列表，无序、有序的集合间的交集、并集等高级服务器端原子操作。

<http://www.jb51.net/article/122154.htm>



# Tomcat的三种高级运行模式

<http://blog.csdn.net/ITresource/article/details/54927166>

1. **Tomcat Connector（连接器）有两种：HTTP/1.0和AJP/1.3**，AJP(Apache JServ Protocol):AJP是面向数据包的基于TCP/IP的协议，它在Apache和Tomcat的实例之间提供了一个专用的通信信道。
2. **Tomcat Connector(连接器)有三种运行模式：bio nio apr**

## bio(blocking I/O)

阻塞式I/O操作，表示Tomcat使用的是传统的Java I/O操作(即java.io包及其子包)。是基于JAVA的HTTP/1.1连接器，Tomcat7 或以下，在 Linux 系统中默认使用这种方式。一般而言，bio模式是三种运行模式中性能最低的一种。

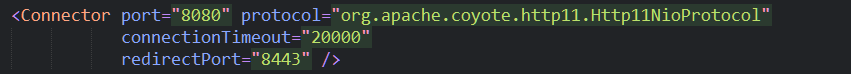
**一个线程处理一个请求，缺点：并发量高时，线程数较多，浪费资源**

## nio(new/non-blocking I/O)

是Java SE 1.4及后续版本提供的一种新的I/O操作方式(即java.nio包及其子包)。Java nio是一个基于缓冲区、并能提供**非阻塞I/O**操作的Java API，因此nio也被看成是**non-blocking I/O**的缩写。它拥有比传统I/O操作(bio)更好的并发运行性能。要让Tomcat以nio模式来运行只需要在Tomcat安装目录/conf/server.xml 中将对应的中protocol的属性值改为 org.apache.coyote.http11.**Http11NioProtocol**即可。

利用 Java 的异步请求 IO 处理，可以**通过少量的线程处理大量的请求**

Tomcat8 以上版本在 Linux 系统中，默认使用的就是NIO模式，不需要额外修改 ，Tomcat7必须修改Connector配置来启动



## apr(Apache Portable Runtime/Apache可移植运行时)

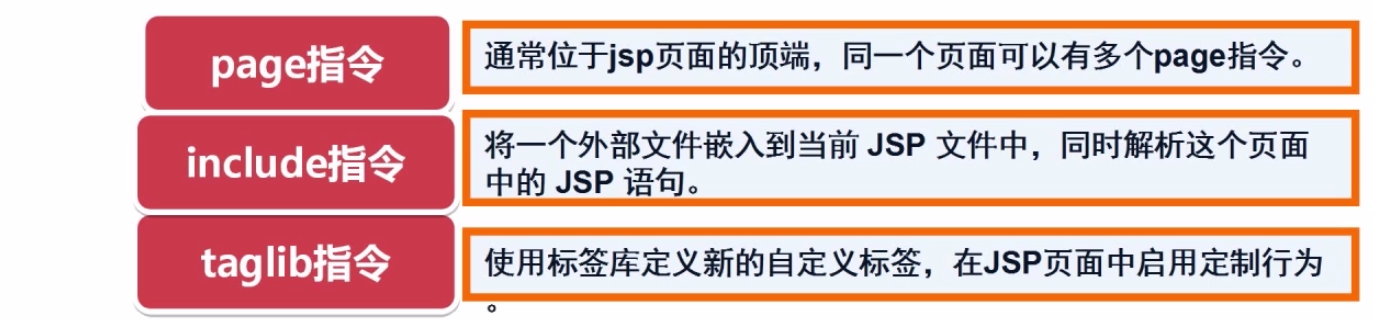
Tomcat将以JNI的形式调用Apache HTTP服务器的核心动态链接库来处理文件读取或网络传输操作，从而大大地提高Tomcat对静态文件的处理性能。Tomcat apr也**是在Tomcat上运行高并发应用的首选模式**。从操作系统级别来解决异步的IO问题。

APR是**使用原生C语言编写的非堵塞I/O，利用了操作系统的网络连接功能，速度很快**。 但是需先安装apr和native，若直接启动就支持apr，能大幅度提升性能，

与配置nio运行模式一样，也需要将对应Connector的protocol属性值改为 org.apache.coyote.http11.Http11AprProtocol。

配置之前需要安装APR以及与 Tomcat-native 整合（之前先安装依赖）

# JSP三大指令



# ASM

<https://mp.weixin.qq.com/s?__biz=MzAxOTc0NzExNg==&mid=2665513528&idx=1&sn=da8b99016aeb4ede2e3c078682be0b46&chksm=80d67a7bb7a1f36dbbc3fc9b3a08ca4b9fae63dbcbd298562b9372da739d5fa4b049dec7ed33&scene=21#wechat_redirect>

<https://www.ibm.com/developerworks/cn/java/j-lo-asm30/>

**可以动态的修改已经编译过的class , 还可以动态的生成新的java class, 注意动态指可以完全在运行时，在内存中完成。**

ASM 是一个 **Java 字节码操控框架**。它能被用来动态生成类或者增强既有类的功能。ASM 可以直接产生二进制 class 文件，也可以在类被加载入 Java 虚拟机之前动态改变类行为。Java class 被存储在严格格式定义的 .class 文件里，这些类文件拥有足够的元数据来解析类中的所有元素：类名称、方法、属性以及 Java 字节码（指令）。ASM 从类文件中读入信息后，能够改变类行为，分析类信息，甚至能够根据用户要求生成新类。

核心主要是三个类， **ClassReader**, **ClassWriter** , **ClassVisitor**, 用这三类就可以去解析一个class 字节码， 获得字段，方法等信息， 当然最重要的可以对这些信息进行修改， 最终形成一个新的class。之后交由JVM加载。

# JMS

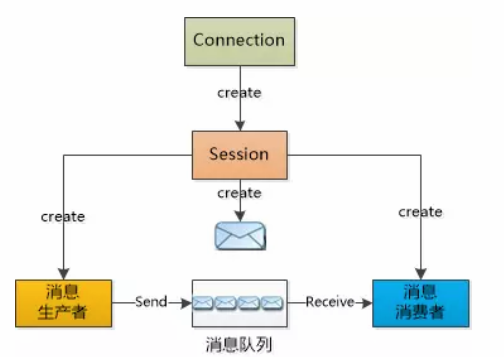
<http://blog.csdn.net/jiuqiyuliang/article/details/46701559>

<https://mp.weixin.qq.com/s?__biz=MzAxOTc0NzExNg==&mid=2665513515&idx=1&sn=380bb1cb56d4151fd3acc5aa86f1da9a&chksm=80d67a68b7a1f37e3d98fe4495eab4db097eedd695c99fbd8704cc0464595842c4da598b99e3&scene=21#wechat_redirect>

JMS是java的消息服务，**JMS的客户端之间可以通过JMS服务进行异步的消息传输**。

有两种模型：**点对点**模型和**发布/订阅**模型

## 点对点P2P（Point to Point）



**消息队列（Queue）、发送者(Sender)、接收者(Receiver)**

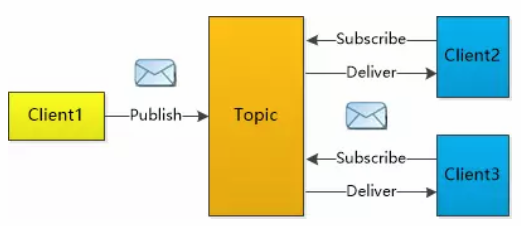
每个消息都被发送到一个特定的队列，接收者从队列中获取消息。队列保留着消息，直到他们被消费或超时。

**特点**：

1. **每个消息只有一个消费者**（Consumer）(即一旦被消费，消息就从消息队列中删除)
2. 发送者和接收者之间**在时间上没有依赖性**，也就是说当发送者发送了消息之后，不管接收者有没有正在运行，它不会影响到消息被发送到队列
3. **接收者在成功接收消息之后需向队列应答成功**

如果你希望发送的每个消息都应该被成功处理的话，那么你需要P2P模式。

## 发布/订阅模式Pub/Sub



**主题（Topic）、发布者（Publisher）、订阅者（Subscriber）**

客户端将消息发送到主题。多个发布者将消息发送到Topic,系统将这些消息传递给**多个订阅者**。

**Pub/Sub的特点**

1. 每个消息**可以有多个消费者**
2. 发布者和订阅者之间**有时间上的依赖性**。针对某个主题（Topic）的订阅者，它**必须创建一个订阅者之后，才能消费发布者的消息，而且为了消费消息，订阅者必须保持运行的状态**。
3. 为了缓和这样严格的时间相关性，JMS允许订阅者创建一个可持久化的订阅。这样，即使订阅者没有被激活（运行），它也能接收到发布者的消息。

如果你希望发送的消息可以不被做任何处理、或者被一个消息者处理、或者可以被多个消费者处理的话，那么可以采用Pub/Sub模型

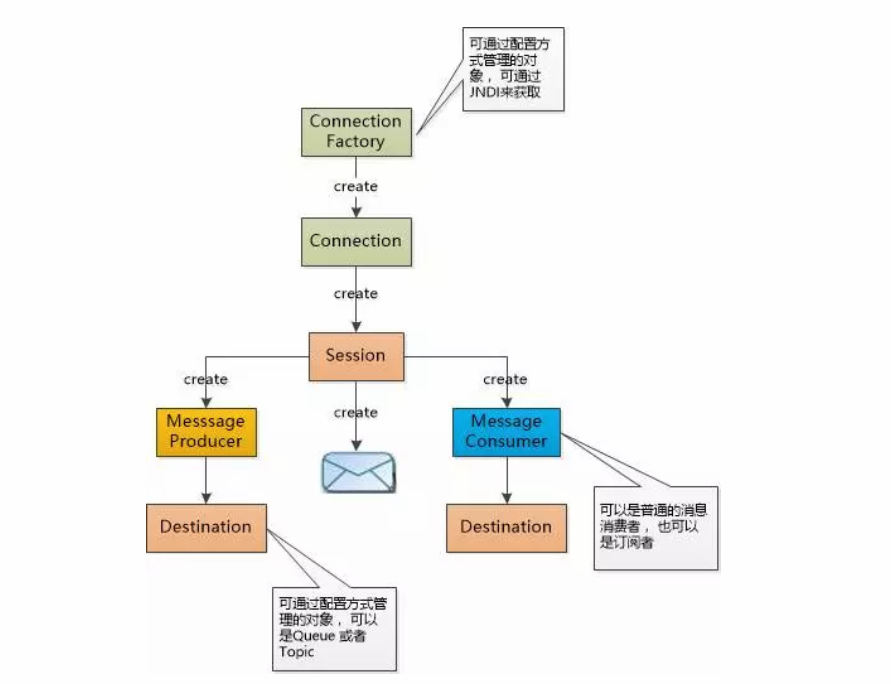
## 消息的消费

在JMS中，消息的产生和消费是异步的。对于消费来说，JMS的消费者可以通过两种方式来消费消息。

**同步** ：订阅者或接收者调用receive方法来接收消息，receive方法在能够接收到消息之前（或超时之前）将一直**阻塞**

**异步** ：订阅者或接收者可以注册为一个消息监听器。当消息到达之后，系统自动调用监听器的onMessage方法。

## JMS编程模型



(1) **ConnectionFactory**

创建Connection对象的工厂，针对两种不同的jms消息模型，分别有QueueConnectionFactory和TopicConnectionFactory两种。可以通过JNDI来查找ConnectionFactory对象。

(2) **Destination**

Destination的意思是消息生产者的**消息发送目标**或者说消息消费者的**消息来源**。对于消息生产者来说，它的Destination是某个队列（Queue）或某个主题（Topic）;对于消息消费者来说，它的Destination也是某个队列或主题（即消息来源）。

所以，Destination实际上就是两种类型的对象：Queue、Topic可以通过JNDI来查找Destination。

(3) **Connection**

Connection表示在客户端和JMS系统之间建立的链接（对TCP/IP socket的包装）。Connection可以产生一个或多个Session。跟ConnectionFactory一样，Connection也有两种类型：QueueConnection和TopicConnection。

(4) **Session**

Session是我们**操作消息的接口**。可以通过session创建生产者、消费者、消息等。Session提供了事务的功能。当我们需要使用session发送/接收多个消息时，可以将这些发送/接收动作放到一个事务中。同样，也分QueueSession和TopicSession。

(5) **消息的生产者**

消息生产者由Session创建，并用于将消息发送到Destination。同样，消息生产者分两种类型：QueueSender和TopicPublisher。可以调用消息生产者的方法（send或publish方法）发送消息。

(6) **消息消费者**

消息消费者由Session创建，用于接收被发送到Destination的消息。两种类型：QueueReceiver和TopicSubscriber。可分别通过session的createReceiver(Queue)或createSubscriber(Topic)来创建。当然，也可以session的creatDurableSubscriber方法来创建持久化的订阅者。

(7) **MessageListener**

消息监听器。如果注册了消息监听器，一旦消息到达，将自动调用监听器的onMessage方法。EJB中的MDB（Message-Driven Bean）就是一种MessageListener。