### Java Web复习指南

题目类型

单项选择（10道）、填空（5道）、简答（5道）、编程题

##### 考试范围

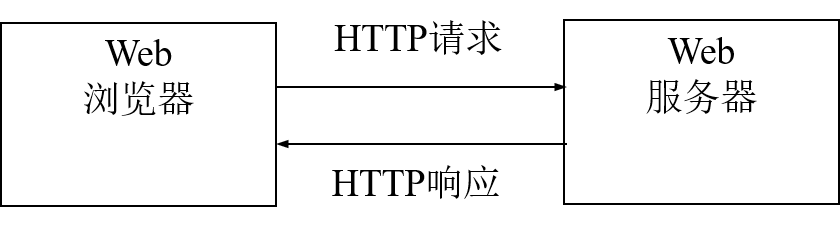
###### Web基础知识

* HTML基本概念

超文本标记语言。作用是定义超文本文档的结构和格式。HTML描述了一个网站的结构语义随之线索的呈现，使之成为一种标记语言而非编程语言。

* HTTP协议

超文本传送协议。HTTP协议是浏览器和web服务器通信的基础。它是一个基于请求-响应的无状态的协议。



**三点注意事项：**

HTTP是无连接的：限制每次连接只处理一个请求。服务器处理完各户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

HTTP是媒体独立的：这意味着，只要客户端和服务器知道如何处理的数据内容，任何类型的数据都可以通过HTTP发送。

HTTP是无状态的：无状态是指协议对于事物处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传。

**八种请求类型：**

OPTIONS：返回服务器针对特定资源所支持的HTTP请求方法。

HEAD：向服务器所要与GET请求相一致的响应，只不过响应体将不会被返回。

GET：向特定的资源发出请求。

POST：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。

PUT:向指定资源位置上传其最新内容

DELETE：请求服务器删除Request-URI所标识的资源。

TRACE：回显服务器收到的请求，主要用于测试或诊断。

CONNECT：HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。

**常见HTTP状态码：**

200-请求成功

301-资源或网页被永久转移到其他URL

404-请求的资源或网页不存在

500-内部服务器错误

* Tomcat的基本概念，配置，部署

Tomcat是Apache软件基金会的开源产品，是Servlet和JSP技术的实现。Tomcat服务器本身具有作为Web服务器运行的能力，因此不需要一个单独的Web服务器。

另：URI统一资源标识符、URL统一资源定位符。

URL组成-协议名称、主机地址、端口号和资源的名称。

###### Servlet

* Servlet基本概念

Servlet是运行在Web服务器端的Java程序，此程序是基于HTTP协议的，在服务器端运行（如Tomcat），是Jsp出现前Java中用于构建Web应用的一项重要技术。

实际上，JSP页面运行前会由JSP容器将其翻译成Servlet，真正在服务器端运行的是Servlet。

Servlet规范提供了一个标准的，平台独立的框架实现再Servlet和容器之间的通信。该框架是由一组Java接口和类组成，它们称为Servlet API。

**主要功能：**

读取客户提交的数据。

读取由浏览器发送的隐式请求数据。

生成结果。

向客户端发送显式数据。

发送隐式的HTTP响应数据。

**Servlet的优势（与CGI对比）：**

性能明显更好。

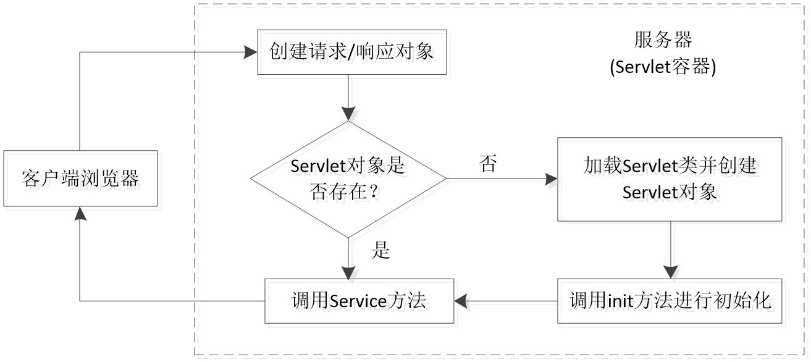
Servlet在Web服务器的地址空间内执行。这样它就没必要再创建一个单独的进程来处理每个客户端请求。

Servlet是独立于平台的。

服务器上的Java安全管理器执行了一系列限制，以保护服务器计算机上的资源。因此Servlet是可信的。

Java类库的全部功能对Servlet来说都是可用的。

**工作原理：**



Web服务器（Servlet容器）接受到客户端请求，容器创建“请求和响应”对象，并判断请求的Servlet对象是否存在。

如果存在，则直接调用此Servlet对象的Service方法，并将“请求和响应”对象作为参数传递。

如果不存在，容器负责加载Servlet类，创建Servlet对象并实例化，然后调用Servlet的init方法进行初始化，之后调用Service方法。

在Service方法中，通过请求对象获取客户端提交的数据并处理，然后通过响应对象将处理结果返回客户端。

* HttpServlet类

HttpServlet抽象类用来实现针对HTTP协议的Servlet。

* ServletConfig接口和ServletContext接口

**ServletConfig接口：**

ServletConfig接口为用户提供了有关Servlet配置信息。

Servlet配置包括Servlet名称、Servlet上下文对象、Servlet初始化参数。

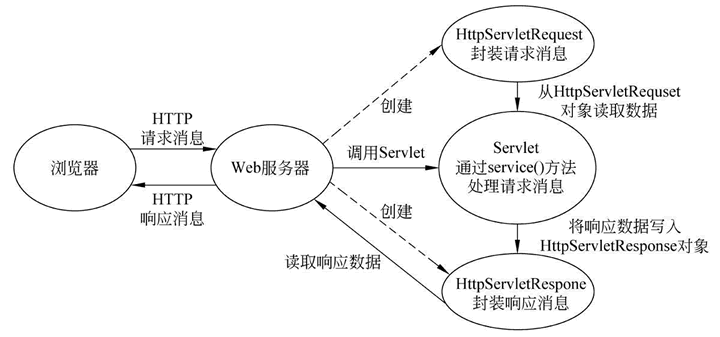
在一个web应用可以存在多个ServletConfig对象。

在Servlet的配置文件web.xml中，可以使用一个或多个<init-param>标签为servlet配置一些初始化参数。

**ServletContext接口：**

当Servlet容器启动时，会为每个Web应用创建一个唯一的ServletContext对象代表当前的Web应用，该对象封装了当前Web应用的所有信息。

* HttpServletResponse对象和HttpServletRequest对象



**HttpServletRequest**接口扩展了ServletRequest接口并提供了针对HTTP请求操作方法。

定义列获取请求行、请求头和请求消息体的相关方法。通过这个对象提供的方法，可以获得客户端请求的所有信息，包括请求的地址，请求的参数，请求的数据，上传的文件，客户端的ip甚至客户端操作系统都包含在期内。

**HttpServletResponse**接口扩展了ServletResponse接口并提供了针对HTTP的发送响应方法。

这个对象中封装了向客户端发送数据、发送响应头、发送响应状态码的方法。

**发送响应消息体相关的方法：**

GetOutputSream方法：输出对象是GetOutputSream，二进制数据。

getWriter方法：输出对象为PrintWriter，字符文本格式。

两个方法互相排斥，调用了其中的任何一个方法后，就不能再调用另一方法。

* doGet和doPost

**doGet**：

GET请求来自于一个URL的正常请求，或者来自于一个未指定的METHOD的HTML表单，它由doGet方法处理。

**doPost：**

POST请求来自于一个特别指定了METHOD为POST的HTML表单，它由doPost方法处理。

* RequestDispatch对象

**请求转发（RequestDispatcher）：**

客户首先发送一个请求到服务器端，服务器端发现匹配的servlet，并指定它去执行，当这个servlet执行完之后，它要调用getRequestDispacther()方法，把请求转发给指定的URL,整个流程都是在服务器端完成的，而且是在同一个请求里面完成的。

**重定向（sendRedirect）:**

客户发送一个请求到服务器，服务器匹配servlet，这都和请求转发一样，servlet处理完之后调用了sendRedirect()这个方法，这个方法是response的方法，所以，当这个servlet处理完之后，看到response.senRedirect()方法，立即向客户端返回这个响应，响应行告诉客户端你必须要再发送一个请求，去访问URL，紧接着客户端受到这个请求后，立刻发出一个新的请求，去请求URL,这里两个请求互不干扰，相互独立，在前面request里面setAttribute()的任何东西，在后面的request里面都获得不了。可见，在sendRedirect()里面是两个请求，两个响应。

* Cookie对象和Session对象

**会话跟踪：**

HTTP本身是无状态协议，它不保存连接交互信息，一次响应完成之后即连接断开，下一次请求需要重新建立连接，服务器不记录上次连接的内容。因此如果判断两次连接是否是同一用户，就需要使用会话跟踪技术来解决。

**常见会话跟踪技术：**

URL重写：在URL结尾附加会话ID标识，服务器通过会话ID识别不同用户。

隐藏表单域：将会话ID埋入HTML表单隐藏域提交到服务端。

Cookie：第一次请求时服务器主动发送一段信息给浏览器（即Cookie），下次请求时浏览器自动附带该段信息发送给服务器，服务器读取Cookie识别用户。

Session：服务器为每个用户创建一个Session对象保存到内存，并生成一个sessionID放入Cookie发送给浏览器，下次访问时sessionID会随Cookie传回来，服务器再根据sessionID找到对应Session对象。

Cookie技术将会话数据保存在浏览器客户端；

Session技术将会话数据保存在服务器端。

Session依赖于cookie，如果cookie被禁用session将失效。

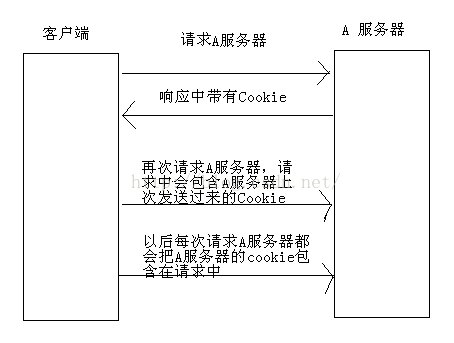
**Cookie技术的特点：**

Cookie技术将会话数据保存在浏览器客户端。

Cookie数据只能是非中文的字符串类型的数据。

浏览器可以保存多个Cookie，但是浏览器一般只允许存放300geCookie，每个站点最多存放20个Cookie。

每个Cookie的大小限制为4KB。



**Cookie注意细节：**

如果创建了一个cookie，并将他发送到浏览器，默认情况下它是一个会话级别的cookie（即存储在浏览器的内存中），用户退出浏览器之后即被删除。若希望浏览器将该cookie存储在磁盘上，则需要使用maxAge，并给出一个以秒为单位的事件。将最大时效设为0则是命令浏览器删除该cookie。

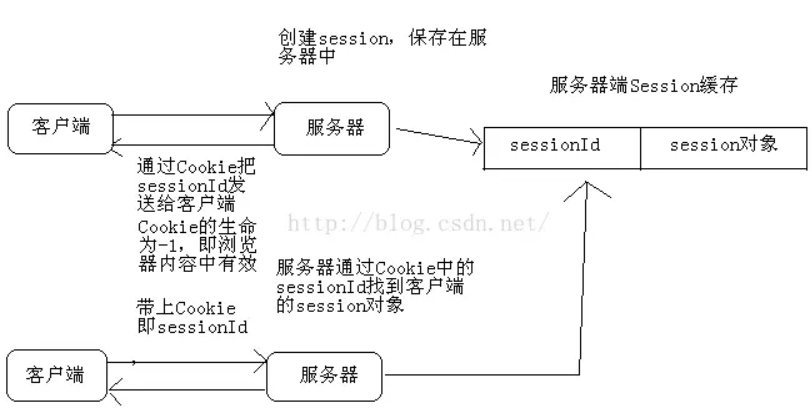
Session技术的特点：

Session技术将会话保存在服务器端。

Session可以保存对象。

Session可以存储中文字符串。

Session可以保存超过4KB的会话数据。



* Servlet的生命周期

Servlet不是独立的应用程序，它不能由用户或程序员直接调用，它的产生与销毁完全由容器管理。

Servlet有良好的生存期的定义，包括如何加载、实例化、初始化、处理客户端请求以及如何被移除。这个生存期由javax.servlet.servlet接口的init、service和destroy方法表达。

**Servlet生命周期包括以下几个阶段：**

加载和实例化Servlet类。

调用init方法初始化servlet实例。

一旦初始化完成，容器从客户收到请求时就将调用它的service方法。

最后容器在servlet实例上调用destroy方法使它进入销毁状态。

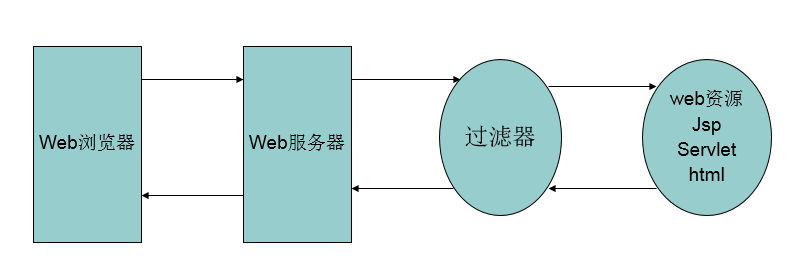
Init和destroy在Servlet周期只能被调用一次。

* Filter过滤器

Servlet Filter可以动态地拦截请求和响应，以变换或使用包含在请求或响应中的信息。

在客户端的请求访问后端资源之前，拦截这些请求。

在服务器的响应发送回客户端之前，处理这些响应。



**Filter的特点：**

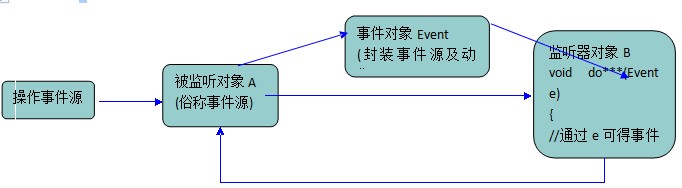
Servlet过滤器可能检查和修改ServletRequest和ServletResponse对象。

可以指定Servlet过滤器和特定的URL关联，只有当客户请求访问此URL时，才会触发该过滤器工作。

多个Servlet过滤器可以被串联起来，形成管道效应，协同修改请求和响应对象。

* Listener监听器

将挺起就是一个实现特定接口的java程序，这个程序专门用于监听另一个java对象的方法调用或属性改变，当被监听对象发生如web应用的启动和停止，用户session的开始和结束等事件后，监听器某个方法将被执行。



**常用监听器：**

监听session、request、context的创建与销毁。

监听对象属性变化。

###### JSP

* JSP基本概念

全称Java Server Pages，是一种动态网页开发技术。

JSP是一种Java Servlet，主要用于实现Java Web应用程序的用户界面部分。

* JSP的语法

JSP页面内容可以分成 元素 和 模板数据 两部分。

元素是在JSP基本语法中定义的内容：

JSP容器在转换阶段将元素翻译成响应的java代码

JSP定义的元素有四种类型：指令、脚本、动作和表达式语言。

JSP页面中其他的所有内容都是模板数据：

JSP容器对模板数据不做处理，例如HTML内容，会直接送到客户端执行。

脚本元素：

声明：

**<%!** declaration; [ declaration; ] ... **%>**

可以声明变量、方法、类、类变量、数组等。

表达式：

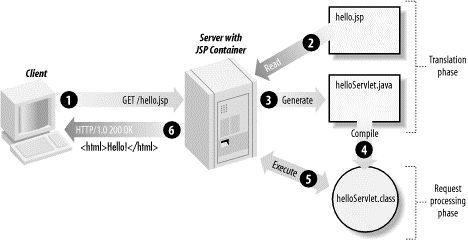
**<%=** expression **%>**

“\<%=”三个字符之间不能有空格，表达式不能有“；”。

JSP注释

**<%-**- This is JSP comment --**%>**

* JSP的执行过程



* 与Servlet的区别

JSP可以很方便的编写或者修改HTML网页而不用面对大量的println语句

* JSP指令

JSP指令是在JSP程序中嵌入一段java代码。

基本语法：

**<%** Java代码 **%>**

JSP指令用于设置整个页面属性，并告诉引擎如何处理该页面，它并不向客户端产生任何输出。通过JSP指令可以设置页面的引入类、内容类型和编码、错误处理和会话信息。

指令元素的语法格式：

**<%@** 指令名 属性名1="属性值" 属性名2="属性值" … **%>**

| **指令** | **描述** |
| --- | --- |
| <%@ page ... %> | 定义页面的依赖属性，比如脚本语言、error页面、缓存需求等等 |
| <%@ include ... %> | 包含其他文件 |
| <%@ taglib ... %> | 引入标签库的定义，可以是自定义标签 |

按照惯例，page指令被编码在JSP页面的顶部。

* JSP的内置对象
* JSP动作

| **语法** | **描述** |
| --- | --- |
| jsp:include | 用于在当前页面中包含静态或动态资源 |
| jsp:useBean | 寻找和初始化一个JavaBean组件 |
| jsp:setProperty | 设置 JavaBean组件的值 |
| jsp:getProperty | 将 JavaBean组件的值插入到 output中 |
| jsp:forward | 从一个JSP文件向另一个文件传递一个包含用户请求的request对象 |
| jsp:plugin | 用于在生成的HTML页面中包含Applet和JavaBean对象 |
| jsp:element | 动态创建一个XML元素 |
| jsp:attribute | 定义动态创建的XML元素的属性 |
| jsp:body | 定义动态创建的XML元素的主体 |
| jsp:text | 用于封装模板数据 |

* JSP隐含对象

JSP隐含对象是JSP容器为每个页面提供的Java对象，开发者可以直接使用它们而不用显式声明。JSP隐含对象也被称为预定义变量。

| **对象** | **描述** |
| --- | --- |
| request | HttpServletRequest类的实例 |
| response | HttpServletResponse类的实例 |
| out | PrintWriter类的实例，用于把结果输出至网页上 |
| session | HttpSession类的实例 |
| application | ServletContext类的实例，与应用上下文有关 |
| config | ServletConfig类的实例 |
| pageContext | PageContext类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | Exception类的对象，代表发生错误的JSP页面中对应的异常对象 |

* Include与forward

另：JSP生命周期

###### EL和JSTL

* JavaBean的基本概念

在软件开发时，一些数据和功能需要在很多地方使用，为了方便将它们移植，可以利用JavaBean技术对这些数据和功能进行封装，做到一次编写到处运行。

JavaBean是一种规范，而不是一种技术或工具。

* EL的基本概念、标识符、保留字、变量、常量、运算符

Expression Language又称JSP表达式语言，它主要用于替换JSP页面中的脚本表达式<%=%>,从各种类型的web域中检索java对象，获取数据。Web服务器对于request域中的属性是以object类型来存储的，在得到该属性时使用的java语言脚本就应该是(objiect)request.getAttribute(xxx)，它对于实际应用还必须进行强制类型了转换。

基础操作符：

| **操作符** | **描述** |
| --- | --- |
| . | 访问一个Bean属性或者一个映射条目 |
| [] | 访问一个数组或者链表的元素 |
| ( ) | 组织一个子表达式以改变优先级 |
| + | 加 |
| - | 减或负 |
| \* | 乘 |
| / or div | 除 |
| % or mod | 取模 |
| == or eq | 测试是否相等 |
| != or ne | 测试是否不等 |
| < or lt | 测试是否小于 |
| > or gt | 测试是否大于 |
| <= or le | 测试是否小于等于 |
| >= or gt | 测试是否大于等于 |
| && or and | 测试逻辑与 |
|  |  |
| ! or not | 测试取反 |
| empty | 测试是否空值 |

隐含对象：

| **隐含对象** | **描述** |
| --- | --- |
| pageScope | page 作用域 |
| requestScope | request 作用域 |
| sessionScope | session 作用域 |
| applicationScope | application 作用域 |
| param Request | 对象的参数，字符串 |
| paramValues | Request对象的参数，字符串集合 |
| header | HTTP 信息头，字符串 |
| headerValues | HTTP 信息头，字符串集合 |
| initParam | 上下文初始化参数 |
| cookie | Cookie值 |
| pageContext | 当前页面的pageContext |

可以在表达式中直接使用这些对象，就像使用变量一样。

* JSTL的基本概念、标签

JSP标准标签库是一个JSP标签集合，它封装了JSP应用的通用核心代码。

核心标签：

引用语法：

1 **<%@** taglib prefix="c"

2 uri="http://java.sun.com/jsp/jstl/core" %\>

| **标签** | **描述** |
| --- | --- |
| <c:out> | 用于在JSP中显示数据，就像<%= ... > |
| <c:set> | 用于保存数据 |
| <c:remove> | 用于删除数据 |
| <c:catch> | 用来处理产生错误的异常状况，并且将错误信息储存起来 |
| <c:if> | 与我们在一般程序中用的if一样 |
| <c:choose> | 本身只当做和的父标签 |
| <c:when> | 的子标签，用来判断条件是否成立 |
| <c:otherwise> | 的子标签，接在标签后，当标签判断为false时被执行 |
| <c:import> | 检索一个绝对或相对 URL，然后将其内容暴露给页面 |
| <c:forEach> | 基础迭代标签，接受多种集合类型 |
| <c:forTokens> | 根据指定的分隔符来分隔内容并迭代输出 |
| <c:param> | 用来给包含或重定向的页面传递参数 |
| <c:redirect> | 重定向至一个新的URL. |
| <c:url> | 使用可选的查询参数来创造一个URL |

格式化标签：

引用语法：

1 **<%@** taglib prefix="fmt"

2 uri="http://java.sun.com/jsp/jstl/fmt" **%>**

| **标签** | **描述** |
| --- | --- |
| <fmt:formatNumber> | 使用指定的格式或精度格式化数字 |
| <fmt:parseNumber> | 解析一个代表着数字，货币或百分比的字符串 |
| <fmt:formatDate> | 使用指定的风格或模式格式化日期和时间 |
| <fmt:parseDate> | 解析一个代表着日期或时间的字符串 |
| <fmt:bundle> | 绑定资源 |
| <fmt:setLocale> | 指定地区 |
| <fmt:setBundle> | 绑定资源 |
| <fmt:timeZone> | 指定时区 |
| <fmt:setTimeZone> | 指定时区 |
| <fmt:message> | 显示资源配置文件信息 |
| <fmt:requestEncoding> | 设置request的字符编码 |

SQL标签：

引用语法：

1 **<%@** taglib prefix="sql"

2 uri="http://java.sun.com/jsp/jstl/sql" **%>**

标签 描述

<sql:setDataSource>指定数据源

<sql:query>运行SQL查询语句

<sql:update>运行SQL更新语句

<sql:param>将SQL语句中的参数设为指定值

<sql:dateParam>将SQL语句中的日期参数设为指定的java.util.Date 对象值<sql:transaction>在共享数据库连接中提供嵌套的数据库行为元素，将所有语句以一个事务的形式来运行

另外还有XML标签，例如

|  |  |
| --- | --- |
| <x:out> | 与<%= ... >,类似，不过只用于XPath表达式 |

JSTL函数，例如

|  |  |
| --- | --- |
| fn:contains() | 测试输入的字符串是否包含指定的子串 |

自定义标签，例如

详情[ch6.html](file:///C:\Users\Wonye\Documents\WeChat%20Files\wxid_ps58iwsyr1s822\FileStorage\File\2020-11\ch6.html)

###### JDBC

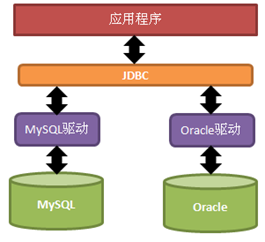
* JDBC的基本概念

JDBC是一种用于执行SQL语句的Java API。

JDBC架构原理：  
JDBC API：提供应用程序到JDBC管理器连接。

JDBC驱动程序API：支持JDBC管理器到驱动程序连接。

JDBC是接口，驱动是接口的实现，没有驱动将无法完成数据库连接，从而不能操作数据库！每个数据库厂商都需要提供自己的驱动，用来连接自己公司的数据库，也就是说驱动一般都有数据库生成厂商提供。



常见的JDBC组件：

* **DriverManager**：此类管理数据库驱动程序列表。 使用通信子协议将来自java应用程序的连接请求与适当的数据库驱动程序进行匹配。在JDBC下识别某个子协议的第一个驱动程序将用于建立数据库连接。
* **Driver**：此接口处理与数据库服务器的通信。我们很少会直接与Driver对象进行交互。 但会使用DriverManager对象来管理这种类型的对象。 它还提取与使用Driver对象相关的信息。
* **Connection**：此接口具有用于联系数据库的所有方法。 连接(Connection)对象表示通信上下文，即，与数据库的所有通信仅通过连接对象。
* **Statement**：使用从此接口创建的对象将SQL语句提交到数据库。 除了执行存储过程之外，一些派生接口还接受参数。
* **ResultSet**：在使用Statement对象执行SQL查询后，这些对象保存从数据库检索的数据。 它作为一个迭代器并可移动ResultSet对象查询的数据。
* **SQLException**：此类处理数据库应用程序中发生的任何错误。
* JDBC的常用API
* JDBC的程序实现
* **import** **java.sql.\*** ;
* 2 **import** **java.math.\*** ;
* 3
* 4 @Test
* 5 **public** void demo()**throws** Exception{
* 6 *// 查询所有的分类信息*
* 7 *// 注意：使用JDBC规范，采用都是java.sql包下的内容*
* 8 *//1 注册驱动*
* 9 Class.forName("com.mysql.jdbc.Driver");
* 10 *//2 获得连接*
* 11 String url = "jdbc:mysql://localhost:3306/mydb";
* 12 Connection conn = DriverManager.getConnection(url,"root", "root");
* 13 *//3获得语句执行者*
* 14 Statement st = conn.createStatement();
* 15 *//4执行SQL语句*
* 16 ResultSet rs = st.executeQuery("select \* from category");
* 17 *//5处理结果集*
* 18 **while**(rs.next()){
* 19 *// 获得一行数据*
* 20 Integer cid = rs.getInt("cid");
* 21 String cname = rs.getString("cname");
* 22 System.out.println(cid + " , " + cname);
* 23 }
* 24 *//6释放资源*
* 25 rs.close();
* 26 st.close();
* 27 conn.close();

###### MVC设计模式

* MVC设计模式的基本概念

**M 代表 模型（Model）**： 模型就是数据，就是 dao,bean

**V 代表 视图（View）** ： 就是网页, JSP，用来展示模型中的数据

**C 代表 控制器（controller)**：控制器的作用就是把不同的数据(Model)，显示在不同的视图(View)上，Servlet 扮演的就是这样的角色。

* 两种常见的设计模式

###### Spring

* Spring的基本概念

是最受欢迎的企业级java应用程序开发框架。

* 依赖注入
* 面向切面的编程

又臭又长。