目　录

第一章　Java Web 概述

第一节　Web 应用演化

随着互联网技术与应用的不断发展，普通的静态网页已不能满足网上信息交流的需求，具有交互功能的动态网页得到了广泛的应用。Web 程序设计技术就是用于实现动态交互式功能的网页制作技术，通过 Web 程序语言（CGI、PHP、ASP、JSP、ASP. Net 等）设计的动态网页可以根据用户的即时操作和即时请求，使网页内容发生相应的变化，从而可以实现功能强大的交互式操作。

Web 应用经历了从 C/S 架构到 B/S 架构的演变过程，目前主流网络应用主要采用 B/S 架构。

一、C/S 架构

在 Web 应用普及之前，多数网络应用需要在客户端安装程序，一般称为 C/S 架构，即 Client/Server。这种架构的应用不仅需要编写服务器端的脚本程序，也需要编写客户端程序，用户需要通过安装文件在本机安装，此类应用如 Windows MediaPlayer、QQ、Office 等。

二、B/S 架构

对于 C/S 架构的应用来说，将任务合理分配到 Client 端和 Server 端来实现，因此系统的网络通信开销低，应用服务器运行数据负荷较轻，但是客户端系统升级或功能更新代价高，而且效率低，因此适用于中小型应用程序。另外一种应用架构是 B/S 架构，即 Browser/Server，本机只需安装浏览器软件。B/S 架构的应用程序主要业务逻辑运行在服务器上，只有极少的业务逻辑运行在客户端，主要数据存储在服务器上，极少数据缓存在客户端。这样的客户端简单，系统维护与升级的成本和工作量小。

第二节　Web 服务器脚本技术

设计动态网页的技术有很多，并且这方面的技术在不断地发展变化，它们有各自的特点。

一、CGI

CGI（Common Gateway Interface）是最古老的 Web 编程技术，是一段在服务器端运行的程序，是面向客户端 HTML 页面的接口，就像一座桥，把网页和 Web 服务器中的执行程序连接起来，把 HTML 接收的指令传递给服务器，把服务器执行的结果返回 HTML 页面。

CGI 可以在任何服务器和操作系统上实现，任何程序语言都可以编写 CGI。CGI 属于底层操作，远不及 ASPJSP 和 PHP 容易。因为 ASP 等都提供良好的运作环境，底层操作对于它们而言是不必要的。如果把 CGI 比作机器语言，那么 ASP 等就是汇编语言了。CGI 的底层操作可以为大型系统创作之所用。

二、PHP

PHP（PersonalHomePage）是一种 HTML 内嵌的语言，是一种在服务器端运行的嵌入 HTML 文档的脚本语言，语言的风格类似于 C 语言，现在被很多网站编程人员广泛运用。PHP 独特的语法混合了 C、Java、Perl 以及 PHP 自创新的语法。它可以比 CGI 或者 Perl 更快速地执行动态网页。

PHP 执行效率比较高，而且 PHP 支持几乎所有流行的数据库以及操作系统。但每种数据库的开发语言都完全不同，这就需要开发人员将同样的数据操作用不同的代码写出多种代码库，增加了程序员的工作量。另外，PHP 是完全免费的，所有代码都是开源的，缺少正规的商业支持，无法实现商品化应用的开发。

三、ASP

ASP（ActiveServerPages）是微软推出的用以取代 CGI 的技术。简单来说，ASP 是一套服务器端的脚本运行环境，通过 ASP 可以结合 HTML 网页、ASP 指令和 ActiveX 元素建立动态、交互、高效的 Web 服务器应用程序。

ASP 提供了创建交互网页的简便方法，只要将一些简单的指令嵌入 HTML 文件中，就可以从表单中收集数据。ASP 还可以利用 ADCX Active Data Object，微软开发的一种数据访问模型）方便地访问数据库，使开发基于 WWW 的应用系统成为可能。

四、JSP

JSP（JavaServerPages）也是当前比较热门的 Web 技术，是由 SUN 公司发布的。JSP 为创建高度动态的 Web 应用提供了一个独特的开发环境。

JSP 与 Microsoft 的 ASP 技术非常相似。两者都提供了在 HTML 代码中混合某种程序代码、由语言引擎解释执行程序代码的能力。在 ASP 或 JSP 环境下，HTML 代码主要

负责描述信息的显示样式，而程序代码则用来描述处理逻辑。普通的 HTML 页面只依赖于 Web 服务器，而 ASP 和 JSP 页面需要附加的语言引擎分析和执行程序代码。程序代码的执行结果被重新嵌入 HTML 代码中，然后一起发送给浏览器。ASP 和 JSP 都是面向 Web 服务器的技术，客户端浏览器不需要任何附加的软件支持。

JSP 的特点如下。

（1）将内容的生成和显示进行分离。Web 页面开发人员可以使用 HTML 或者 XML 标志来设计和格式化最终页面，使用 JSP 标识或者小脚本来生成页面上的动态内容。生成的内容逻辑上被封装在标识和 JavaBean 组件中，并且捆绑在小脚本中，所有的脚本在服务器端运行。如果核心逻辑被封装在标志和 JavaBean 中，那么其他人，如 Web 管理人员或页面设计者，能够编辑和使用 JSP 页面，而不影响内容的生成。在服务器端，JSP 引擎解释 JSP 标识和小脚本，生成所请求的内容，并且将结果以 HTML（或者 XML）页面的形式发送回浏览器。这有助于作者保护自己的代码，而又保证了任何基于 HTML 的 Web 浏览器的完全可用性。

（2）生成可重用的组件。绝大多数 JSP 页面信赖于可重用的、跨平台的组件（JavaBean 或者 EnterpriseJavaBean 组件）来执行应用程序所要求的更为复杂的处理。开发人员能够共享和交换执行普通操作的组件，或者使这些组件为更多的使用者或者客户团体所使用。基于组件的方法加速了总体开发过程，并且使各种组织在他们现有的技能和优化结果的开发努力中得到平衡。

（3）采用标识简化页面开发。Web 页面开发人员不都是熟悉脚本语言的编程人员。JSP 技术封装了许多功能，这些功能是在易用的、与 JSP 相关的 XML 标识中，进行动态内容生成所需要的、标准的 JSP 标识能够访问的 JavaBean 组件，设置或者检索组件属性，下载 Applet，以及执行用其他方法更难于编码和耗时的功能。通过开发定制标识库，JSP 技术是可以扩展的。今后，第三方开发人员和其他人员可以为常用功能创建自己的标识库。这使得 Web 页面开发人员能够使用熟悉的工具如同标识一样执行特定功能的构件来工作。

⑷由于 JSP 页面的内置脚本语言是基于 Java 编程语言的，而且所有的 JSP 页面都被编译为 JavaServletJSP 页面就具有 Java 技术的所有好处，包括健壮的存储管理和安全性。

（5）可靠且移植方便。作为 Java 平台的一部分 JSP 拥有 Java 编程语言“一次编写，各处运行”的特点。随着越来越多的供应商将 JSP 支持添加到他们的产品中，可以使用自己所选择的服务器和工具，更改工具或服务器并不影响当前的应用。

第三节 JSP 工作原理

JSP 页面在运行的时候，首先会将整个 JSP 页面编译成一个 Java 文件，而服务器运行时，实际上是在运行 Java 的类文件，这样就达到了一次编写多处运行的目的。这个类文

件就是 JavaServlet。

一、Servlet 技术

（一）Servlet 简介

Servlet 是使用 JavaServletAPI 编写的、适合于 B/S 模式的、运行在 Web 服务器端的 Java 类，具有独立于平台和协议的特性，可以生成动态的 Web 页面。Servlet 和客户端的通信采用“请求/响应”模式，其工作原理如图 1-1 所示。

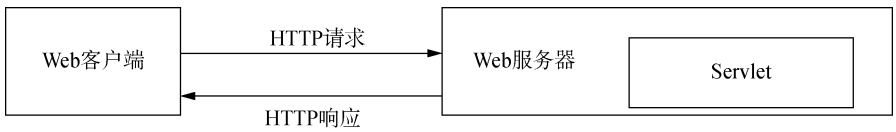

图 1-1　Servlet 的工作原理

（二）Servlet 的优点

（1）可移植性：Servlet 利用 Java 语言开发，具有 Java 的跨平台性，Servlet 程序可以在任何操作系统上运行。

（2）功能强大：包括网络和 URL 访问、通过 JDBC 访问远程数据库、通过对象序列化使用 JavaBean、通过 JNDI 使用 EJB、通过 JXPA 访问 Web 服务等。

（3）性能优良：Servlet 程序在加载执行之后，它的实例在一段时间内会一直驻留在服务器的内存中，若有请求，服务器会直接调用 Servlet 实例来服务。并且当多个客户请求一个 Servlet 时，服务器会为每个请求者启动一个线程来处理，所以效率高。

（4）可靠性：Servlet 有强类型检查功能，并且利用 Java 的垃圾回收机制避免内存管理上的问题。另外，Servlet 能够安全地处理各种错误，不会因为发生程序上的逻辑错误而导致整体服务器系统的崩溃。

（三）JSP 与 Servlet 的关系

JSP 与 Servlet 的关系如图 1-2 所示，可以概括为两点。

（1）JSP 的实现是基于 Servlet 的，JSP 页面在运行之前要被解释成 JavaServlet。

（2）当 JSP 容器接到对一个 JSP 页面的请求后，首先判断与 JSP 文件对应的 Servlet 类的名字，如果该类不存在或比 JSP 文件旧，容器就会重新创建一个等价的 Servlet 类并编译它。

二、JSP 生命周期

如果客户端请求的是一个 JSP 页面，这时服务器会将该页面编译为一个 Servlet 类，

并自动加载形成 Servlet 实例，把执行结果返回客户端。因此，JSP 网页在执行时会经历以下几个阶段。

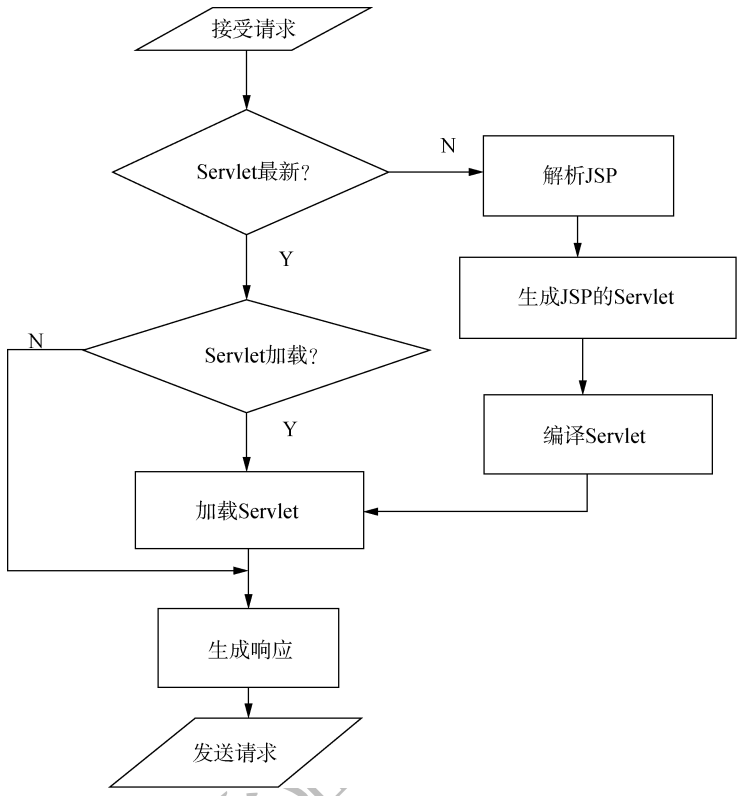

图 1-2 JSP 与 Servlet 的关系示意图

（一）编译阶段

在客户端发出 request 请求后，JSP 容器将 JSP 转译成 Servlet 的源代码，再将产生的 Servlet 的源代码编译成为 Servlet 类。

（二）初始化阶段

Servlet 容器加载 Servlet 类，创建 Servlet 实例并调用 Servlet 的 init 方法进行初始化。

（三）响应客户请求阶段

对于到达 Servlet 容器的客户请求，利用容器所创建的 Servlet 实例的对象调用 service 方法处理请求信息，响应至客户端。

（四）终止阶段

当 Web 应用被终止，或 Servlet 容器终止运行，或 Servlet 容器重新装载 Servlet 的新

实例时，Servlet 容器会先调用 Servlet 的 destroy 方法。在 destroy 方法中，可以释放 Servlet 所占用的资源。

第四节　超文本传输协议

超文本传输协议（Hyper Text Transfer Protocol，HTTP）是万维网（World Wide Web，WWW，也简称为 Web）上应用最为广泛的一种网络协议，是一个属于应用层的面向对象的协议，适用于分布式超媒体信息系统。HTTP 主要用在客户端（浏览器）和 Web 服务器之间进行通信。所有的 WWW 文件都必须遵守这个标准。

HTTP 于 1990 年提出，经过十几年的使用与发展，得到了极大的扩展和完善。HTTP 有 3 个版本，依次是 0.9、1.0、1.1。目前在 WWW 中普遍使用的版本是 1.1。HTTPng 是发展中的下一代协议，在效率和性能上会有更进一步的提高。

客户端（浏览器）和 Web 服务器之间要进行通信，首先使用可靠的 TCP 连接（默认端口为 80），然后浏览器要先向服务器发送请求信息，服务器在接收到请求信息后，做出响应，返回相应的信息，浏览器接收到来自服务器的响应信息后，对这些数据进行解释执行。所以 HTTP 可以分成两部分：HTTP 请求和 HTTP 响应。

一、HTTP 请求

HTTP 请求格式如下：

```
<request-line>
<headers>
<blank line>
[<request-body>]
```

在 HTTP 请求中，第一行必须是一个请求行（requestline），用来说明请求类型、要访问的资源以及使用的 HTTP 版本。紧接着是一个头部（header）小节，用来说明服务器要使用的附加信息，如声明浏览器所用语言，请求正文的长度等。在头部之后是一个空行，指示头部结束。在此之后可以添加任意的其他数据，称为主体（body），其中可以包含客户提交的查询字符串信息。

下面是一个 HTTP 请求的例子：

```
GET /sample. jsp HTTP/1.1
Accept: image/gif, imge/ jpeg, */*
Accept-Language: zh-cn
Connection: Keep-Alive
```

```
Host: localhost
User-Agent: Mozilla/4. 0 (compatible; MSIE 5.01; Windows NT 5.0)
Accept-Encoding: gzip, deflate
username = users&password = 1234
```

在 HTTP 中，定义了大量的请求类型，其中 GET 请求和 POST 请求是最主要的。只要在 Web 浏览器上输入一个 URL，浏览器就将基于该 URL 向服务器发送一个 GET 请求，以告诉服务器获取并返回资源（也就是对网页的访问）。而 POST 请求在请求主体中为服务器提供了一些附加的信息。通常，当填写一个在线表单并提交它时，这些填入的数据将以 POST 请求的方式发送给服务器。

二、HTTP 响应

HTTP 服务器接到请求后，经过处理，会给予相应的响应信息，其格式与 HTTP 请求相似，如下所示：

```
<status-line>
<headers>
<blank line>
[<response-body>]
```

在 HTTP 响应中与请求唯一真正的区别在于第一行中用状态信息代替了请求信息，状态行（statusline）通过提供一个状态码来说明所请求的资源情况。在响应信息的头部也包含很多信息，如服务器类型、日期时间、内容类型和长度等。在头部之后同样需要用一个空行指示头部结束。再接下来的就是服务器返回的内容，如一个 HTML 页面。

下面是一个 HTTP 响应的例子：

```
HTTP/1.1 200 OK
Server: ApacheTomcat/6. 0.14
Date: Fri, 20 Nov 2009 10: 30: 15 GMT
Content-Type: text/ html
Last-Modified: Fri, 20 Nov 2009 10: 40: 25 GMT
Content-Length: 112
<html>
<head>
<title>HTTP 响应示例</title>
</head>
<body>
Hello HTTP!
```

```
</body>
</html>
```

浏览器的每次请求都要求建立一次单独的连接，在处理完每一次的请求后，就自动释放连接。

HTTP 的主要特点可概括如下。

（1）支持客户/服务器模式。

（2）简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。HTTP 简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

（3）灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。

（4）无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，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。

（5）无状态：HTTP 是无状态协议。无状态是指协议对于事务处理没有记忆能力。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，这样可能导致每次连接传送的数据量增大。另外，在服务器不需要先前信息时它的应答就较快。

第五节　安装与配置 JDK、Tomcat

一、实验内容

练习 JDK、Tomcat 的安装与环境变量配置。

在 JDK 环境下编写执行简单的 Java 程序。

运行与测试 Tomcat 服务器。

二、实验步骤

（一）安装 JDK

JDK（Java Development Kit）软件开发工具包是整个 Java 的核心，包括 Java 运行环境 JRE（Java Runtime Environment）、Java 工具及 Java 基础的类库等。JDK 可在网站免费下载：http://www.oracle，com/technetwork/java/javase/downloads/index.html。进入下载页面，接受协议，选择正确的操作系统后即可下载 JDK 的安装程序。

运行 JDK 安装程序启动安装向导，单击“下一步”按钮进入定制安装界面，选择“开发工具”，设置安装位置，单击“下一步”按钮安装所选组件。

开发工具安装完毕后，开始安装 JRE。根据需要更改目标文件夹（建议与 JDK 安装在同一位置），单击“下一步”按钮开始安装 JRE。完成 JRE 的安装，最终完成整个 JDK 的安装。安装成功后会显示成功信息。

（二）配置 JDK 环境变量

JDK 安装完成以后，为了在命令行状态中任何路径下都能访问编译器和虚拟机，还需要设置一下环境变量。打开“控制面板”，单击“系统与安全”进入“系统”界面，单击左侧的“高级系统设置”，弹出“系统属性”窗口，打开“高级”选项卡，单击最下面的“环境变量”按钮，弹出“环境变量”编辑对话框。对话框中包括用户变量和系统变量两个部分，如果设置用户的环境变量，则只允许当前用户使用，其他用户不能使用；如果设置系统的环境变量，则此计算机的每个用户均可使用。

单击选中系统变量中的 Path 变量，然后单击“编辑”按钮，在变量值最前面加入“C:\ProgramFiles\Java\jdkl.8.0_131\bin;”，即把 JDK 安装后的 bin 目录加入 Path 中，注意用分号隔开。单击“确定”按钮完成系统环境变量的配置。

（三）测试 JDK

为测试 JDK 是否安装成功，在任务栏的搜索框中输入 cmd，单击打开“命令提示符”窗口，在命令行中输入 javac，如果 JDK 安装且配置正确。

（四）编写简单的 Java 程序

编写运行一个 Java 程序分为 3 步：编写 Java 源程序、编译源程序、运行 .class 字节码文件。

下面编写命令行程序，在命令行窗口输出“HelloWorld”。

1. 编写 Java 源程序

用“记事本”或其他文本编辑器建立一个文档，命名为 HelloWorld.java（注意在“文件资源管理器”窗口的“查看”选项卡的“显示/隐藏”选项区中勾选“文件扩展名”复选框）。然后打开文件，编辑内容如下。

```
public class HelloWorld {
    public static void main (String [ ] args) {
        System. out. println ( " HelloWorld! " ) ;
    }
}
```

文件建立以后，保存到任意目录（以 C:\为例），这就是源程序文件（扩展名为 .java）。

2. 编译 Java 源程序

进入命令提示符窗口，利用 cd 命令，切换到源程序所在目录（在这里运行“cd \”切换到 C 盘根目录），然后输入下面的编译命令：

```
javac HelloWorld. java↙
```

该语句会调用 JDK 路径中 bin 文件夹下面的 javac. exe，对 HelloWorld. java 进行编译。

如果编译命令执行后，没有任何提示，则说明程序被顺利编译。在当前目录下会生成一个可供虚拟机解释运行的类文件 HelloWorld. class（扩展名为 . class）。

3. 执行 Java 程序

继续输入以下命令可以执行程序：

```
java HelloWorld↙
```

程序运行后会在屏幕上显示运行结果：

```
HelloWorld!
```

虚拟机会在当前路径下寻找 HelloWorld. class，解释执行该类文件。

（五）安装与配置 Tomcat

Tomcat 是一款比较优秀的 JavaWeb 服务器，是在 SUN 公司的 JSWDKXJavaServer-WebDevelopmentKit）的基础上发展起来的一个优秀的 Servlet/JSP 容器，它是 Apache-Jakam 软件组织的一个子项目。它不但支持 JSP 和 Servlet 的开发，而且还具备了商业 Web 应用容器的特征，如 Tomcat 管理和控制平台、安全域管理关键以及 Tomcat 阀等。Tomcat 已成为目前开发企业 JavaWeb 应用程序的最佳服务器选择之一。

Tomcat 安装过程非常简单，完毕后在安装目录下找到 bin 文件夹，通过运行其中的 startup，bat 就可以运行 Tomcat。Tomcat 在启动时会读取环境变量信息，因此需要配置一个“JAVA_HOME”系统环境变量，JAVA_HOME 变量值为 JDK 的安装目录。

在系统变量面板中单击“新建”按钮，变量名输入 JAVA_HOME（不区分大小写），变量值输入 JDK 的安装目录，单击“确定”按钮，此时再运行 startup，bat。正常启动后可以进行测试，打开 IE 浏览器，输入 http://localhost：8080。

三、实验小结

JDK 和 Tomcat 的安装过程较为简单，但需要注意系统环境变量（Path）的配置，这一步容易出现错误。安装配置完毕后，要学会简单的测试方法，环境配置正确后才可以开始编写 JSP 程序。

四、补充练习

(1) 使用 JDK 编写一个简单的 Java 程序，在命令提示符窗口输出当前的系统时间。提示：创建 java.util.Date 类的实例得到当前的系统时间。

(2) JBoss 是另外一款免费的 JavaWeb 服务器应用服务框架，它一般捆绑 Tomcat 作为内核，但其作用比 Tomcat 强大。尝试自学下载、安装与配置 JDK+JBoss 服务器环境。

第六节 部署运行第一个 JSP 程序

一、实验内容

在 Tomcat 服务器上部署站点。

使用记事本或 UltraEdit 等代码编辑器编写一个简单的 JSP 程序。

运行 JSP 程序，在浏览器中查看显示信息。

理解 JSP 工作原理与生命周期。

二、实验步骤

(一) 部署站点

安装 Tomcat 后，其安装目录中的 webapps 目录用于部署发布各个站点。所谓在 Tomcat 中发布一个站点，其实就是在 webapps 下构建一个目录，属于该站点的资源全部部署在该目录下即可。

(二) 编写 JSP 代码

在 test 站点文件夹中新建一个 JSP 文档，命名为 firstJSP.jsp。使用记事本或 UltmEdit 打开文档，编写代码如下。

```
<% @ page import = " java.util.* " contentType = " text/html; charset = gb2312" %>
  <HTML>
    <HEAD>
      <TITLE>firstJSP</TITLE>
    </HEAD>
    <BODY>
      <% = new Date ( ) %>
```

```
    </BODY>
</HTML>
```

（三）运行 JSP 程序

启动 Tomcat 服务器，在浏览器地址栏中输入地址：http://localhost：8080/test/firstJSP.jsp。

三、实验小结

本次实验使用 JDK＋Tomcat 服务器环境，在 webapps 目录下部署了一个站点，并创建一个 JSP 页面，这个页面的作用是在页面上输出当前的系统时间。

初学者在实践时容易犯以下错误。

（一）路径错误

如果在运行 JSP 程序时出现“404”错误，说明访问资源不存在，可以检查地址书写是否正确。地址是区分大小写的。

（二）语法错误

如果在运行 JSP 程序时出现“500”错误，说明脚本代码语法错误，可以根据提示信息查找代码错误。。

此外，还容易出现端口占用等异常现象，需要初学者耐心地解决各种问题。

四、补充练习

（1）部署一个“web1”站点，编写 JSP 程序，能够在浏览器上输出“你好！欢迎来到这里！”。

提示：在页面上输出语句可使用脚本代码＜%out.println（“你好！欢迎来到这里");%＞。

（2）部署一个“web2”站点，编写 JSP 程序，能够显示“3＋5”的计算结果，并标记为红色。

提示：在页面上计算并输出结果，可使用表达式＜%＝3＋5%＞；红色样式显示可以使用＜fontcolor＝“red”＞…＜/font＞标签。

第二章　Java 程序设计基础

第一节　类和对象

JavaWeb 应用开发需要用到 Java 基础知识，本章将讲解 Web 应用开发中的 Java 基础必备知识，包括封装、继承与多态机制。Java 的封装机制是在使用抽象思维对事物的特征和行为进行提取的基础上，实现细节隐藏和权限访问的控制。JavaBean 是一种 Java 语言写成的可重用组件，用于封装数据和封装业务逻辑。

在现实世界中，万物皆对象。而要定义这些对象的本质，就要使用类，换句话说，就是要用类来定义这些对象的特征。所以，类是 Java 语言的基础。在类中，定义了数据和操作数据的代码，然后使用类来定义一个一个的对象。类是 Java 中一种重要的复合数据类型，创建一个新类，就相当于创建了一种新的数据类型。类是对象的模板，对象是类的一个实例。因此，所有的 Java 程序都是基于类的。

一、类的成员

在 Java 类中，使用数据来描述客观事物的特征，称为属性，使用数据的操作来描述客观事物的行为，称为方法。属性和方法统称为类的成员。在一个类中，既可以只包含对象的属性，也可以只包含对象的方法，但一个完整的类要同时包含对象的属性和方法，这也是比较常见的类的定义形式。

为了更好地理解 Java 类，下面举个简单的例子来进行说明。例如，要描述“学生”这一类客观事物，学生具有“学号”“姓名”“年龄”这样的属性，还有“上课”这样的行为，因此学生类的定义如下所示。

创建一个学生类，封装学生的“学号”“姓名”“年龄”属性以及“上课”方法。

```
public class Student {
    String num;
    String name;
    int age;
    void attendClass ( ) {
        System. out. println (" 学生去上课。");
```

```
    }
}
```

在这个例子中，使用 num、name 和 age 三个变量来定义学生的三个属性，使用 attend-Class（）方法实现学生上课的行为。

二、对象初始化与构造方法

（一）对象初始化

定义好一个类以后，还不能完全使用它，除非只使用其中的 static 成员。要使用一个类时，大多数情况下需要创建一个该类的对象，这个创建过程叫作对象的实例化过程或初始化过程。

继续上一节的例子，定义好了 Student 类，现在可以使用 Student 来实例化对象，比如定义小明、小丽两个学生实体，就可以使用如下代码：

```
Student xiaoming = new Student ();
Student xiaoli = new Student ();
```

实例化语句中 new 关键字的作用就是分配一段内存空间，利用后面指定的构造方法 Student（）建立一个对象。所谓构造方法是指在类中定义的，并在对象被创建时所调用的方法，该方法的名称与类名相同，不需要指定返回值，也不需要使用 return 返回。引入构造方法的主要目的是完成对象被创建时的初始化工作。

使用 new 关键字实例化对象并给对象进行初始化。

```
public class Student {
    String num;
    String name;
    int age;
    void attendClass () {
        System. out. println (" 学生去上课。");
    }
    public static void main (String [] args) {
        Student xiaoming = new Student () ;
        xiaoming. name = ” 小明”;
        xiaoming. attendClass ( ) ;
    }
}
```

程序运行结果：

学生去上课。

该程序在 main（）方法中使用 new 关键字创建一个对象 xiaoming 后，对 xiaoming 的 name 属性进行了初始化，并调用该对象的 attendClass（）方法进行输出。

（二）构造方法

在类中，除了上面例子中用到的自定义方法和 main（）方法外，类还有一个特殊的方法，就是构造方法。当使用一个类创建对象时 Java 会自动调用该类的构造方法。即使没有显式的定义，每个类也会有一个隐藏的构造方法，这也正是为什么即使没有定义构造方法，仍然可以使用它去定义对象了。构造方法有以下几个重要特征。

（1）构造方法的名字与类名相同。

（2）构造方法没有返回类型。

（3）使用类创建对象时，构造方法将立即被调用。

（4）构造方法主要用于对象属性的初始化。

Java 会自动为每个类提供一个默认的构造方法，但是用户一旦定义了自己的构造方法 Java 就不会再使用默认的构造函数了。

使用构造方法对属性进行初始化。

```
public class Student {
    Str ing num;
    Str ing name;
    int age;
    public Student ( String stuNum, String stuName, int stuAge) {
        num = stuNum;
        name = stuName;
        age = stuAge;
    }
    void attendClass ( String name ) {
        System. out. println (" 学生" + “name” + " 去上课");
    }
    public static void main ( String [ ] args) {
        Student xiaoming = new Student ( " 2009001", " 小明", 23);
        xiaoming, attendClass ( ) ;
    }
}
```

程序运行结果：

学生小明去上课。

该示例使用构造方法 Student（StringstuNum，StringstuName，intstuAge）实现对象 xiaoming 的初始化，即给该对象的三个属性进行赋值。可以看出，有了自定义的构造方法后 Java 将不再调用默认的构造方法。

三、this 关键字

不难发现，在上面的例子中，构造方法的三个局部变量是与类的三个实例变量（属性）相对应的，只有这样才能在定义对象时完成对各个属性的初始化过程。但有些情况下，局部变量与实例变量的名字是完全一致的，这种情况之所以被允许，正是因为 Java 定义了 this 这个关键字。this 可以在引用当前对象的所有方法内使用。也就是，this 总是调用该方法对象的一个引用。这样就可以直接引用对象，能用它来解决可能在实例变量和局部变量之间发生的任何同名的冲突。

使用 this 关键字来存取同名的实例变量。

```
public class Student {
    String num;
    String name;
    int age;
    public Student ( String num, String name, int age) {
        this. num = num;
        this. name = name;
        this. age = age;
    }
}
```

本例中使用的是另外一种 Student 类构造方法的写法，它使用了与实例变量相同的名字作为构造方法的局部变量，然后使用 this 关键字来进行赋值存取，防止实例变量被隐藏。

四、包

Java 语言使用包来对类和接口进行更好的管理。一个包就相当于操作系统中的文件夹，包中的类和接口就相当于文件。

包的名字有层次关系，各层之间以点分隔。定义一个包的语法格式为：

```
package pkg1 [. pkg2 [ . pkg3 …] ];
```

包的定义要放在程序的第一行，它的前面只能有注释或空行。一个文件中最多只能有一条 package 语句。

创建一个包，用于存储类 A。

```
package com. test;  ····· //定义一个包
public class A {                                    //定义一个类
    public void add (int i, int j) {                //两个数求和并输出
    System. out. println (i + j);
    }
}
```

引用上面中创建的包，调用类 A 的 add 方法求两个数的和。

```
import com. test. * ;                               //引入所需的类
public class B {
      public static void main (String [ ] args) {
        A a = new A ( );
        a. add (2, 3);                              //调用 A 类的 add ( ) 方法
              }
}
```

程序运行结果：

```
5
```

调用 A 类的 add（）方法实现两个整数的加法，所以程序开头使用 import 关键字引用 com. test 这个包中的所有类（* 代表该包中定义的所有类，也可以只引用该包中的某个类），这样才可以访问 A 类。

需要指出的是，java，lang 是 Java 语言使用最广泛的包，它所包括的类是其他包的基础。这个包无须显式引用，它总是被编译器自动调入。

第二节　类的继承

继承是面向对象程序设计中的一个重要特性，通过继承可以实现源代码的复用，提高程序的可维护性。所谓继承，即在现有类的基础上建立新类的处理过程。利用继承，可以先创

建一个拥有共同属性的一般类，根据该一般类再创建具有特殊属性的新类。由继承而得到的类称为子类（或称为派生类，subclass），被继承的类称为父类（或称为超类，superclass）。直接或间接被继承的类都是父类。

在继承关系中，子类从父类继承了所有非私有的属性和方法。通过继承，可以让子类复用父类的代码，同时也允许子类增加自己特有的属性和方法。这样使程序结构更加清晰，既减少了程序的编码量，又降低了程序维护的工作量。

一、父类与子类

Java 中继承是通过 extends 关键字实现的。定义类时在 extends 关键字后指明新定义类的父类，使得在两个类之间建立继承关系。新定义的子类就可以继承父类中非私有的属性和方法作为自己的成员属性和成员方法。但父类的构造方法不能被继承。

一个类的子类声明格式如下：

```
class SubClass [extends SuperClass] {
    ……
}
```

定义父类人类与其子类学生类、教师类的继承关系。

```
//People. java 定义父类人类 People
    class People {
    String name;
    int age;
    public void eat () {
        System. out. println (" 人都是要吃饭的。");
    }
}
//Student. java 学生类 Student 继承 People 类
class Student extends People {
    public void attendClass () {
        System. out. println (" 学生要听课。");
    }
}
/ /Teacher. java 教师类 Teacher 继承 People 类
class Teacher extends Peoplel
    public void teaching () {
        System. out. println (" 教师要讲课。");
    }
}
//TestExtend. java 测试类
public class TestExtend {
    public static void main (String [ ] args) {
        Student xiaoming = new Student ();
```

```
        xiaoming. name = " 张小明";
        xiaoming. age = 20;
        System. out. println ( " ----------学生小明的描述----------");
        System. out. println (" 小明的名字叫" + xiaoming. name);
        System. out. println (" 小明的年龄是" + xiaoming. age);
        xiaoming. attendClass ();
        Teacher li = new Teacher () ;
        li. name = " 李华";
        li. age = 30;
        System. out. println (----------李老师的描述----------") ;
        System. out. println (" 李老师的名字叫" + li. name) ;
        System. out. println (" 李老师的年龄是" + li. age);
        li. teaching ();
        li. eat ();
    }
}
```

程序运行结果：

```
----------学生小明的描述----------
小明的名字叫张小明
小明的年龄是 20
学生要听课。
----------李老师的描述----------
李老师的名字叫李华
李老师的年龄是 30
教师要讲课。
人都是要吃饭的。
```

该事例中，定义了三个类，分别是 People、Student 和 Teacher。其中 People 是父类，定义了人类具有的属性，包括姓名和年龄，以及 cat（）方法。通过 extends 关键字，Student 继承 People 的所有属性和方法，同时还添加自己所特有的方法 attendClass（）。而 Teacher 也继承 People 的所有属性和方法，同时添加自己特有的方法 teaching（）。

二、方法重写

方法重写即是在子类中对父类中的方法保持名字不变，重写父类的方法体，以便完成不同的工作。比如，现实世界中，哺乳动物都会发声，但是不同的哺乳动物的发声方式不

同。比如，猫是“啼喵”声，狗是“汪汪”声。因此在描述“叫”这个方法时就可以在不同的子类中各自重写哺乳类的 call（）方法。

方法重写的应用。

```
//Mammaul. java 父类 Manmaul 类
class Mammaul {
    public void call () {
        System. out. println (" 哺乳动物是会叫的。");
    }
}
//Cat. java Cat 类继承 Mammaul 类
class Cat extends Mammaul {
    public void call () { ····· //重写父类 Mammaul 类的 call () 方法
        System. out. println (" 猫咪喵喵叫。");
    }
}
//Dog. java Dog 类继承 Manmaul 类
class Dog extends Mammaul {
    public void call () {              //重写父类 Mammaul 类的 call () 方法
        System. out. println (" 狗狗汪汪叫。");
    }
}
//Testoverride. java 测试类
public class TestOverride {
    public static void main (String [ ] args) {
        Cat cat = new Cat ();
        System. out. println (" -------------------猫猫的叫声描述-------------------")
        System. out. println (" 猫猫怎么叫呢?");
        cat. call ();
        Dog dog = new Dog ();
        System. out. println (" -------------------狗狗的叫声描述-------------------")
        System. out. println (" 狗狗又怎么叫呢?");
        dog, call ();
    }
}
```

程序运行结果：

```
----------猫猫的叫声描述----------
猫猫怎么叫呢?
猫咪喵喵叫。
----------狗狗的叫声描述----------
狗狗又怎么叫呢?
狗狗汪汪叫。
```

该示例中，父类 Mammaul 中定义一个方法 call（）。子类 Cat、Dog 均对父类中的 call（）方法进行了重写。可以看到，call（）方法在不同的类中名字、返回类型以及参数列表都一样，只是方法的具体实现有所区别。当子类的实例分别调用 call（）方法时，调用的都是自己对应的 call（）方法，因此得到的结果也是不一样的。

三、super 关键字

子类在重写了父类的方法后，常常还需要使用父类中被重写的方法，这时就要调用父类中的方法 Java 中通过 super 关键字来实现对父类中被重写的方法的调用。

super 关键字调用父类中的方法可以被分成两种情况：调用父类的构造方法和调用父类中被重写的方法。

调用父类构造方法的 super 关键字用法。

```
class Mammaul {
    public Mammaul () ····· //默认的构造方法
        System. out. println (" 我是哺乳动物。");
    }
    public Mammaul (int legs) {                    //带参的构造方法
        System. out. println (" 我是一只" + legs + " 条腿的哺乳动物。");
    }
}
class Dog extends Marmaul {
    public Dog () {
        super (4);                                 //调用父类的带参构造方法
        System. out. println (" 我是一只狗。");
    }
}
//测试类
public class TestSuper {
    public static void main (String [] args) {
        Dogdog = new Dog ();
```

```
    }
}
```

程序运行结果：

```
我是一只 4 条腿的哺乳动物。
我是一只狗。
```

使用 super 关键字调用父类中被重写的方法。

```
class Mammaul {
    public void call () {
        System. out. println (" 哺乳动物会叫。");
    }
}
class Dog extends Mammaul {
    public void call () {
        super. call ();                          //调用父类的 call () 方法
        System. out. println (" 狗狗会汪汪叫。");
}
}
//测试类
public class TestSuper2 {
    public static void main (String [ ] args) {
        Dog dog = new Dog ();
        dog. call ();
    }
}
```

程序运行结果：

```
哺乳动物会叫。
狗狗会汪汪叫。
```

该示例中用 super 关键字实现子类对父类中被重写的方法的调用。尽管父类 Mammaul 中的 call () 方法被重写，不能直接调用，但是 super 允许调用定义在父类中的被重写的方法。

同样的，可以利用 super 关键字访问父类中被子类隐藏的属性。

四、访问修饰符

访问修饰符的作用是说明被声明的内容（类、属性、方法）的访问权限。合理地使用

访问修饰符，可以降低类和类之间的耦合性（关联性），从而降低整个项目的复杂度，便于整个项目的开发和维护。在Java语言中访问控制权限有4种：public、private、protected及无修饰符。

（1）public：用public修饰的成分是公有的，也就是说它可以被其他任何对象访问。

（2）private：类中限定为private的成员只能被这个类本身在类内访问，在类外不可见。

（3）protected：用protected修饰的成分是受保护的，可以被同一包中的其他类或异包中的子类访问。

（4）无修饰符（默认）：public、private、protected这三个限定符不是必须写的。如果不写，表示可以被所在的包中其他类访问。

访问修饰符的使用。

```
public class Shape {
    private double per imeter ;
    private void setPer imeter () {
        System. out. println (" 计算该形状的周长。");
    }
    private void getPer imeter () {
        System. out. println (" 输出该形状的周长。");
    }
}
public class Test {
    public static void main (String [ ] args) (
        Shape shape = new Shape ();
        shape. setPerimeter () ;
        shape. getPerimeter ();
    }
}
```

程序在进行编译时，报错信息如下：

```
Test. java：4：setPer imeter () 可以在 Shape 中访问 private
            shape. setPerimeter ();
Test. java：5 ：getPerimeter () 可以在 Shape 中访问 private
            shape. getPerimeter ();
```

原因是setPerimeter（）和getPerimeter（）方法在Shape类中声明时用private修饰符修饰，和perimeter属性一样完全封闭在类内，不允许外界访问。可将这两个方法的修饰符去掉，或改为protected、public即可实现访问。

第三节　抽象类和接口

有时我们创建一个类是为了让其他类来继承它，这样在定义该类时可以只定义一个所有子类共享的一般形式，至于细节交给各个子类去实现，这样的类称为抽象类。另外一些时候，我们需要定义一个类只知道要做什么但不知如何去做，即所有的实现都交给实现它的类来完成，这样的类我们把它定义成接口。

一、抽象类与抽象方法

抽象类使用 abstract 关键字来创建，里面所定义的抽象方法也使用 abstract 来定义。抽象类是抽象的，因此没有具体的实例。抽象方法没有内容，因此不需要被执行。如果一个类继承一个抽象类，那么子类就必须要重写父类中所有的抽象方法，否则它本身仍然是一个抽象类。

利用抽象类实现多态。

```
abstract class Animal {
    abstract void breathe ();
    abstract void move ();
    void breed () {
        System. out. println (" 动物会繁衍。");
    }
}
class Dog extends Animal {
    void breathe () { ·····  //重写父类 Animal 中的抽象方法 breathe ()
        System. out. println (" 狗狗用肺呼吸。");
    }
void move () {                          //重写父类 Animal 中的抽象方法 move ()
        System. out. println (" 狗狗用四条腿跑。");
    }
}
class Fish extends Animal {
    void breathe () {                  //重写父类 Animal 中的抽象方法 breathe ()
        System. out. println (" 鱼儿用鳃呼吸。");
    }
void move () {                          //重写父类 Animal 中的抽象方法 move ()
```

```
        System. out. println (" 鱼儿会游泳。");
    }
}
//测试类
public class TestAbstract {
    public static void main (String [ ] args) {
        Animal animal = new Dog ();
        animal. breed ();          //调用从父类继承的成员方法
        animal. breathe ();        //调用子类 Dog 中重写的抽象方法
        animal. move ();
        Fish fish = new Fish ();
        fish. breed ();
        fish. breathe ();         //调用子类 Fish 中重写的抽象方法
        fish. move ();
    }
}
```

程序运行结果：

```
动物会繁衍。
狗狗用肺呼吸。
狗狗用四条腿跑。
动物会繁衍。
鱼儿用鳃呼吸。
鱼儿会游泳。
```


该示例中，首先定义了一个抽象类 Animal，其中声明了两个抽象方法 breathe（）和 move（）以及一个实现了的成员方法 breed（）。接着定义两个子类 Dog 和 Fish，并分别实现方法 breathe（）和 move（）。最后，在测试类中生成类 Dog 的对象并把它的引用返回到 Animal 型的变量 animal 中，生成类 Fish 的对象将引用返回 Fish 型变量 fish 中。由于对象的多态性产生了上面的运行结果。

二、定义与实现接口

与 C++不同，Java 不允许多重继承，也就是说一个子类至多只能有一个父类。为了弥补这一不足，Java 中的接口可以实现多重继承，即一个类可以实现多个接口，这个机制使接口能够发挥出更加灵活、强大的功能。

interface 是声明接口的关键字。在 Java 中，接口中的属性只允许是静态常量，也就是

static 或 final 类型的。接口中的方法都是一些没有实现具体操作的抽象方法，并且要相关，即保持较高的内聚性。也就是说，接口定义的仅仅是某些特定功能的对外规范，而并没有真正实现这些功能，需要其实现类去实现。接口中无论是属性或方法，默认都是 public 类型。

Java 用关键字 implements 表示实现一个接口，在实现类中必须实现接口中定义的所有方法，且可以使用接口中定义的常量。

这个程序分为三部分：第一部分是 Action 接口的定义；第二部分是 Person 类利用 implements 关键字实现了 Action 接口内的所有抽象方法，并定义了自己的私有属性；第三部分是 Test 测试类。Action 接口中定义的 name 字符串，虽然没有任何修饰符，但默认是 publicstaticfinal 类型，且可以被 Person 类使用。

接口的定义与实现方法看似很简单，但是在实际应用中比较难以理解，首先必须明确接口不同于类的一些特性。

接口中的方法可以有参数列表和返回类型，但不能有任何方法体实现。

接口中可以包含属性，但是会被隐式地声明为 static 和 final，存储在该接口的静态存储区域内，而不属于该接口。

接口中的方法可以被声明为 public 或不声明，但结果都会按照 public 类型处理。

如果没有实现接口中所有方法，那么创建的仍然是一个接口，即接口可以继承接口(使用关键字 extends)。在继承时，父接口传递给子接口的只是方法说明，而不是具体实现，这在一定程度上消除了完全的多继承所带来的复杂性。一个接口可以有一个以上的父接口，一个类可以在继承某一父类的同时实现多个接口，即允许多重继承。

第四节　JavaBean 技术

JavaBean 是基于 Java 的组件技术，它提供了创建和使用以组件形式出现的 JavaBean 类的方法。JavaBean 通过封装属性和方法成为具有某种功能或者处理某个业务的对象，通过 JavaBean 的方法调用可以处理几乎所有通过 Java 编程可以完成的工作，这本身也是 JSP 的极大优势所在。这种方式大大拓展了编程人员的可操纵范围，不用再像以往其他类似语言一样，因为找不到实现所需功能的组件而一筹莫展。可以说，JavaBean 组件是目前具有功能强大和开发简单双重特点的最好的组件方式。

在 Java 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能，通过 JavaBean 的组合可以快速生成新的应用程序。JavaBean 具有以下特性。

可以实现代码的重复利用。

易维护性、易使用性、易编写性。

可以在支持 Java 在任何平台上工作，而不需要重新编译。

可以在内部、网内或者网络之间进行传输。

可以以其他部件的模式进行工作。

一、封装数据的 JavaBean

经过分析不难发现，用户使用浏览器对网页进行浏览等操作，本质是对数据的操作或者执行某种业务逻辑。因此，在编写 JSP 程序之前，应首先进行实体的抽象。抽象是进行面向对象程序设计的第一步，简单来说，需要解决以下 3 个问题。

参与活动的实体。

实体具有的属性。

实体具有的行为（即方法）。

比如，就“用户登录”这一功能来讲，参与登录活动的实体是一个个“用户”，于是可以抽象出 User 类；每一个用户都具有 ID、用户名和密码等属性，于是在 User 类中封装这些成员属性；每一个用户都要在登录活动中进行判断其账号是否合法这一行为，于是可以在 JavaBean（业务 Bean）中封装该业务逻辑的方法。

实际上 JavaBean 本质上只是一个 Java 类而已，但必须符合一些标准化的规范。一个标准的封装数据的 JavaBean 通常具有几项特征。

是一个公共（public）类。

具有 public 的无参构造方法。

有一组 get 类型的公共方法，可以供外部对象得到内部的属性值。

可以通过一组 set 类型的公共方法，来改变内部的属性值。

JavaBean 的属性是内部核心的重要信息，一般设置为私有（private）权限。当 JavaBean 被实例化为一个对象时，可以通过使用 set 方法改变它的属性值，也就等于改变了这个 JavaBean 对象的状态。而这种状态的改变，常常也伴随着一连串的数据处理动作，使得其他相关的属性值也跟着发生变化。

二、封装业务的 JavaBean

狭义上的 JavaBean 用于封装数据，广义上的 JavaBean 还用于封装业务，即处理过程。

通常情况下，业务逻辑会首先以接口形式进行抽象，然后进行业务逻辑的实现。

第五节　使用集合类存储对象

编写程序，一般都要采用一定的数据结构来描述解决的问题。在 java.util 包中的集合架构是一个统一的体系结构，该体系结构用于创建和操作一些重要的被广泛使用的数据结构。集合就是把多个元素组合成单一实体的对象，Java 集合架构支持两种类型的集合：Collections 和 Maps。它们分别在接口 Collection 和 Map 中定义。Collection 是最基本的集

合接口，一个 Collection 代表一组 Object，即 Collection 的元素。一些 Collection 允许相同的元素而另一些不允许，一些能排序而另一些不能排序。

一、List 集合

次序是 List 最重要的特点，它确保维护元素特定的顺序。List 是有序的 Collection，使用此接口能够精确地控制每个元素插入的位置。用户能够使用索引（元素在 List 中的位置，类似于数组下标）来访问 List 中的元素，这类似于 Java 的数组。和下面要提到的 Set 不同，List 允许有相同的元素。

除了具有 Collection 接口必备的 iterator（）方法外，List 还提供一个 listIterator（）方法，返回一个 Listlterator 接口，和标准的 Iterator 接口相比，Listlterator 多了一些 add（）之类的方法，使用它可以从两个方向遍历 List，也可以从 List 中间插入和删除元素（只推荐 LinkedLLst 使用）。List 接口常用方法有以下五种。

list. add（）：添加数据。

list. remove（）：删除数据。

listjemoveAll（）：删除所有数据。

list. retainAlK）：保留交集。

list. subList（sizel，size2）：返回 sizel 到 size2 之间的数据。

实现 List 接口的常用类有 LinkedList、ArrayList、Vector 和 Stack。

ArrayLLst 实现了可变大小的数组。它允许对元素进行快速随机访问，但是向它中间插入与移除元素的速度很慢。每个 ArrayList 实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。当需要插入大量元素时，在插入前可以调用 ensureCapacity 方法来增加 ArrayList 的容量以提高插入效率。Listlterator 只应该用来由后向前遍历 ArrayList，而不是用来插入和删除元素，因为这比 LinkedList 开销要大很多。

二、Set 集合

Set 是一种不包含重复的元素的 Collection，而且 Set 最多有一个 null 元素。Set 与 Collection 有完全一样的接口。Set 接口不保证维护元素的次序。实现 Set 接口的常用类有 HashSet 和 TreeSet。在这里重点介绍 HashSet。

HashSet 是为快速查找而设计的 Set。

三、Map 集合

Map 没有继承 Collection 接口，Map 提供 key 到 value 的映射。一个 Map 中不能包含相同的 key，每个 key 只能映射一个 value。Map 中的元素是键值成对的对象，像个小型数据库，最典型的应用就是数据字典。另外，Map 可以返回其所有键组成的 Set 和其所有值

组成的 Collection，或其键值对组成的 Set，并且还可以像数组一样扩展多维 Map。Map 有两种比较常用的实现：HashMap 和 TreeMap。在这里重点介绍 HashMap。

在各种 Map 中，HashMap 用于快速查找。集合中的每一个元素对象包含一对键和值，集合中没有重复的键，但值对象可以重复。例如，如下程序语句：

```
Map map = new HashMap ();
map. put (" 1" , "Mon");
map. put (" 1", " Monday" );
map. put (" 2", " Monday");
```

由于第一次和第二次加入 Map 中的键都是 1，所以第一次加入的值将被覆盖，而第二个和第三个的值虽然相同，但是键不一样，所以分配不同的地址空间，不会发生覆盖，也就是说一共有两个元素在 map 这个 Map 类型集合中。

第六节　JDBC 技术

JDBC 是 Java 数据库连接（JavaDatabaseConnectivity）的简称，是一种用于执行 SQL 语句的 JavaAPI，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。通过 JDBC 提供的 API，可以向各种关系数据库发送 SQL 语句，进行数据库操作，完成数据库应用程序的开发。同时，由于 JDBC 是由纯 Java 语言编写的，所以用 Java 和 JDBC 开发的数据库应用程序可以在任何支持 Java 的平台上运行。JDBCAPI 是 Java 平台（包括 J2SE、J2EE）的一部分，有两个包：java，sql 和 javax. sql。现在这两个包已成为 Java 核心框架的组成部分。

一、java，sql 包

JDBCAPI 包括一个框架（来自 java，sql 包），凭借此框架可以动态地安装不同驱动程序用来访问不同数据源、发送 SQL 语句、处理返回的结果集或更新数据记录等。表 2.2 所示为执行 JDBC 数据库操作的常用 API。

一般情况下，使用 JDBC 访问数据库步骤如下。

导入 JDBCAPI：首先利用 import 语句导入 java，sql 包。

装载驱动程序：针对不同 DBMS，使用 Class 类的 forName 方法加载驱动程序类的支持。

建立数据库连接：使用 DriverManager 类的 getConnection 方法，指明数据库或数据源的 URL，以及登录 DBMS 的用户名及口令，创建数据库连接对象（Connection 接口对象）。

创建 JDBCStatements 对象：使用已有的 Connection 数据库连接对象创建一个 Statement 对象，该对象把 SQL 语句通过适当的方法发送到 DBMS。

执行语句：对 SELECT 语句来说，使用 executeQuery 方法执行，返回结果是 ResultSet 类型的结果集；对 INSERT、UPDATE、DELETE 语句来说，使用 executeUpdate 方法执行，返回结果是影响的行数。

处理结果：对返回的结果集或影响行数进行处理，可以进行显示、判断等操作。

关闭资源：与各种对象创建的顺序相反，依次关闭 ResultSet、Statement、Connection 对象。

二、创建数据库连接

在对数据库中的数据进行查询或更新操作之前，首先要创建与数据库之间的连接，也就是利用加载的驱动程序类创建 Connection 对象。

三、关闭数据库连接

关闭对象很简单。需要说明的是这里的“对象”是一个复数，指代了 3 个实际的对象，它们是 ResultSet 对象、Statement 对象和 Connection 对象。关闭对象一般形式如下：

```
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;
rs. close ();
stmt. close ();
conn close ();
```

注意：

(1) 它们总是按这样的顺序被创建、被关闭，这由系统保证。

(2) 不一定要显式关闭它们。在程序结束时，它们会被系统自动关闭。

(3) 显式关闭某个对象，那么在该对象上建立的其他对象都将被关闭。其后使用该对象和在其上建立的对象都是非法的。

例如，显式关闭 Statement 对象 stmt，那么建立在其上的 ResultSet 对象 rs 也将被关闭。如果没有建立新的 Statement 对象并赋值给 stmt，那么使用 stmt 是非法的，使用 rs 也是非法的。

四、Statement 类和 PreparedStatement 类

与数据源建立连接后，就可以建立语句对象执行某些查询。SQL 操作中 SELECT 查询是最基本、最常用的操作，将数据库中的数据以某种视图模式显示给用户。JDBC 以 Statement 类和 PreparedStatement 类的对象作为 SQL 语句发送和执行的容器，以 ResultSet 类的对象存储 SELECT 语句执行后返回的结果。

PreparedStatement类扩展自Statement类，增加了对包含在语句中的参数记录器设值的能力。其不同之处有几个方面：首先，Statement类对象用于执行简单的不含有参数的SQL语句，而PreparedStatement类对象用于执行带有IN参数的SQL语句。IN参数是指在SQL语句被创建时尚未确定值的参数。其次，PreparedStatement类对象执行的是已准备好的或预编译的SQL语句，因而一旦执行一次，就可多次重用。SQL语句中的参数标识符用“?”表示，可使用指定输入值去改变它。而Statement类对象执行的是未经编译的SQL语句。

如果使用Statement类，则先把“select * fromauthorswhereau_id= '172-32-1176'”存在SQL缓存池中，当查询的au_id不同时，需要重新编译存储到SQL缓存池，这样执行就慢。

如果使用PreparedStatement类执行，SQLi语句是“select * fromauthorswhereau_id=?”，则该SQL语句存储在SQL缓存池，后面执行相同查询，就不需要再次编译SQL语句，只需将对应的au_id值带入执行就可以。

（一）创建执行Statement对象

创建Statement对象是通过Connection类的createStatemento方法实现的，代码如下：

```
try {
    Class. forName ( " com. microsoft. sqlserver. jdbc. SQLServerDriver");
                                                        //加载驱动程序
    //getConnection 方法指明连接数据库的数据源、用户名及口令
    Connection conn = DriverManager. getConnection (
……" jabc: sqlserver: //localhost: 1433; databaseName = pubs", " sa", " sa");
    Statement stmt = conn. createStatement ();
} catch (SQLExceptione) {
    e. printStackTrace ();
}
```

在数据库连接conn基础上建立Statement对象stmt，就可以通过该对象执行SQL语句了。Statement提供了3种常用的执行SQL语句的方法。

1. Result Setexecute Query（Stringsql）throwsSQLException

该方法执行给定的SELECT语句，返回一个ResultSet类型的对象存储结果。例如：

```
try {
    Statement stmt = conn. createStatement () ;
    ResultSet rs = stmt. executeQuery ( " select * from authors" );
} catch (SQLException e) {
```

```
        e. printStackTrace () ;
    }
```

这段代码通过 Statement 对象的 executeQuery 方法，执行了一条 SELECT 语句，返回的结果集是一个 ResultSet 对象 rs。

2. intexecute Update（Stringsql）throws SQL Exception

执行给定的 SQL 语句，一般为 INSERT、UPDATE 或 DELETE 等对数据库引起更新的语句，或者为不返回任何内容的 SQL 语句（DDL 语句）。当执行的是 INSERT、UPDATE 或 DELETE 语句时，返回值为表示影响的记录行数；返回值为 0，表示执行的是不返回任何内容的 SQL 语句。例如：

```
    try {
        Statement stmt = conn. createStatement () ;
        int i = stmt. executeUpdate (" insert into authors (au_id, au_lname, au_
fname, contract) values
         ('656-12-2222', 'Lee', 'Grace')");
    } catch (SQLException e) {
        e. printStackTrace ();
    }
```

该代码通过 Statement 对象的 executeUpdate 方法向 pubs 数据库中的 authors 表中添加了一条作者记录，因此返回 i 的值为 1。

3. booleanexecute（Stringsql）throws SQL Exception

该方法可以执行任意的 SQL 语句，当返回结果为 ResultSet 对象时，返回值为 true，否则为 false。例如：

```
    try {
        Statement stmt = conn. createStatement () ;
        Boolean flag = stmt. execute ( " insert into authors (au_id, au_lname, au_
fname, contract)
        values ('656-12-2222', 'Lee', 'Grace', 1)");
    } catch (SQLExceptione) {
        e. printStackTrace ();
    }
```

该代码通过 Statement 对象的 execute 方法向 pubs 数据库中的 authors 表中添加了一条作者记录，由于 INSERT 语句返回的不是 ResultSet 对象，因此 flag 的值为 false。

（二）创建执行 Prepared Statement 对象

执行预编译语句需要创建一个 PreparedStatement 对象，可以使用 Connection 对象的 prepareStatement（）方法创建。PreparedStatement 对象执行的是含有一个或多个 IN 参数的 SQL 语句。创建代码如下：

```
try {
    Class. forName ( " com. microsoft. sqlserver. jdbc. SQLServerDriver");
                                                        //加载驱动程序
    //getConnection 方法指明连接数据库的数据源、用户名及口令
    Connection conn = Dr iverManager. getConnection (
…… …… …… …… …" jadbc: qlserver: //localhost: 1433; databaseName = pubs", " sa", " sa");
    PreparedStatement pstmt = conn. preparedStatement ( " update authors set au_fname = ? where au_id = ?");
    pstmt. setString (1, " Peter");
    pstmt. setString (2, " 656-12-2222");
    pstmt. executeUpdate ();
} catch (SQLException e) {
    e. pr intStackTrace ();
}
```

该代码实现了 au_id 为“656-12-2222”作者的 au_fname 修改为“Peter”。需要注意的是 IN 参数的赋值操作。PreparedStatement 类提供了一系列的 setXXX（id，value）方法来为 IN 参数赋值，其中“XXX”表示与要赋值的参数相对应的数据类型名。如常用的字符型参数赋值用 setString（），为整数型参数赋值用 setlnt（）。id 指 SQL 语句中 IN 参数即“?”出现的顺序，从 1 开始。value 表示为参数赋的值。代码中 pstmt. setString（l，" Peter"）表示 SQL

语句中第一个“?”用字符型数据“Peter”替换，pstmt. setString 表示 SQL 语句中第二个“?”用字符型数据“656-12-2222”替换。所以执行的完整 SQL 语句为“updateauthorssetau—fname=‘Peter1whereau—id=’656-12-2222’”。

由于 PreparedStatement 继承自 Statement，所以 Statement 类所提供的三种常用的执行 SQL 语句的方法 execute（）、executeQuery（）和 executeUpdate（）也被继承，只是被重写为方法调用不用参数，即 PreparedStatement 对象调用的执行 SQL 语句的方法是无参方法。

五、ResultSet 结果集

JDBC 使用 ResultSet 对象存储 SQL 语句执行返回的结果集。

实质上，ResultSet 对象不仅具有数据存储的功能，它同时具有操纵数据、对数据进行更新等功能。通过对数据库的查询操作，可以将 SQL 语句中 SELECT 语句的查询结果返回并存储在结果集中。例如：

```
try {
    Statement stmt = conn. createStatement ();
    ResultSet rs = stmt. executeQuery ( " select * from authors" );
} catch (SQLExceptione) {
    e. printStackTrace ();
}
```

（一）结果集中的行操作

为了能访问结果集中的数据行，ResultSet 类提供了行指针方法 next（）来进行操作。声明如下：

```
Booleannext () throws SQL Exception
```

当 next（）方法执行后，当前行指针向下移动一行。但是要注意，初始状态下，行指针并不是指向结果集的第一行，而是指向结果集第一行记录的前面。所以结果集返回后，第一次调用 next（）方法后，行指针才指向结果集的第一行了。

通过 next（）方法，程序就可以对整个结果集的所有记录进行遍历。

（二）结果集中的列操作

ResultSet 类提供了一系列 getXXX（）方法来从结果集中获取某一列的值。其中，“XXX”表示与要获取的字段类型相对应的 Java 数据类型名。此类型的方法允许用户通过字段名或字段索引来确定具体要获取的列。如要获取 authors 表中字符型的 au_fname 列的值，可以通过以下方法获取：

```
String fname = rs. getString (" au_fname") ;
```

在读取某一条记录的字段数据时，除了可如上述方法指明字段名称外，还可以按照读取字段顺序指明索引。注意，列索引是从 1 开始。例如，rs. getStringrau _ idn）、rS. getString（Mau_fnamen）与 rs. getString（1）、rs. getString（3）效果相同，但建议采用指明字段名称的方法，这样程序可读性较好。

进行列操作和行操作返回了 ResultSet 对象结果集后，可以将某些特定字段的数据读取出来。