Java 编程那些事儿 1——序言

序言

从大学毕业到现在，马上就六年了，这六年中从事过开发，也从事培训工作，相比而言，参加培训工作的时间要长一些。由于工作的特点，遇到了各种各样的学生，在学习编程时遇到了一系列的问题，也有很多迷茫的时候，希望通过编写本书，把相关的问题进行一下总结，把自己的经验和大家进行分享。当然由于一些知识也只是个人见解，也希望大家积极指正，帮助编程的初学者，以及程序开发人员深刻理解基础的概念，更好的学习编程和从事开发工作。

编写一本书，总要有个名字吧，姑且把书名叫做《Java 编程那些事儿》吧，主要是用通俗的语言，解释清楚以下几个内容：

1、 程序设计是什么？
2、 Java 语言相关基础语法及应用
3、 如何建立基础的程序逻辑

以及其它编程和 Java 语言的相关问题，本书的编写打算以在线连载的形式进行，估计编写的周期会比较长，希望能为软件开发行业尽一点自己的微薄之力吧。

陈跃峰

2008-4-2

cqucyf@gmail.com

Java 编程那些事儿 2——程序设计是什么？

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

程序设计，俗称编程，是一个比较专业的概念。初学者，甚至一部分开发人员，都不能很简单的解释这个概念，所以使初学者觉得程序设计是一件很有科技含量，或者是很高深的学科，其实这些都是误解。那么程序设计到底是什么呢？

程序，其实就是把需要做的事情用程序语言描述出来。类似如作家，就是把自己头脑中的创意用文字描述出来。所以学习程序，主要就是三个问题：做什么、怎么做和如何描述，具体如下：

1.做什么

做什么就是程序需要实现的功能。

2. 怎么做

怎么做就是如何实现程序的功能，在编程中，称为逻辑，其实就是实现的步骤。

3. 如何描述

就是把怎么做用程序语言的格式描述出来。

所以，对于有经验的程序设计人员来说，学习新的程序设计语言的速度会比较快，就是因为第1和第2个问题基本解决了，只需要学习第3个问题即可了。

对于“做什么”的问题，可能初学者觉得会比较简单，其实在大型项目开发，例如ERP，企业都不能很详细的说明需要实现具体的功能，这就需要有专门的人员去发掘具体的功能，这个用程序的术语叫做需求分析。举个例子吧，例如某个人要找个女朋友，如果你大概去问他，他会说，找个中等的就可以了，但是这个还不是具体的需求，你可能需要问一下，要求女朋友的年龄是多少，身高是多少等等具体的问题。所以说，搞清楚“做什么”也不是简单的事情，需要专门的人员从事该工作。

对于“怎么做”的问题，是初学者，甚至很有经验的开发人员都头疼的事情，这个称作程序逻辑。因为实际的功能描述和程序设计语言之间不能直接转换，就像作家需要组织自己的思路和语言一样，程序设计人员也需要进行转换，而且现实世界和程序世界之间存在一定的差异，所以对于初学者来说，这是一个非常痛苦的过程，也是开始学习时最大的障碍。由于计算机自身的特点，“怎么做”的问题其实就是数据和操作的问题，某个顶级大师曾经说过：“程序=数据结构+算法”，把这个问题描述的简单准确。那么“怎么做”的问题，就变成了持有那些数据，以及如何操作这些数据的问题。先简单的介绍这么多，大家仔细体会吧。

对于“如何描述”的问题，是学习程序最容易，也是最枯燥的问题。其实就是学“透”一套格式，并且深刻理解语言的特点。学程序语言，就像学汉语差不多，需要学习字怎么写，学习语法结构等，只是不需要像汉语这样学那么多年，但是学好一个语言还是要耐得住寂寞。语法的学习需要细致，只有深刻领悟了语法的格式才能够熟练使用该语言。

前面介绍的是程序的概念，那么为什么叫程序设计，其实这个设计和现实中的设计一样。例如你自己盖个小棚子，只需要简单的规划即可，也就是编程中的小程序，而如果需要建造一栋大楼，肯定需要进行设计吧，程序也是这样。所以把编程叫做程序设计了。

Java 编程那些事儿 3——你适合学习程序设计吗？

作者：陈跃峰

出自：http://blog.csdn.net/mailbomb

程序设计是一个技术活，所以不是适合所有的人。现在很多人听说程序设计人员待遇如何如何的好，都一窝蜂的来学习程序，其实这个现象很不正常，因为程序不一定适合你。其实对于一个人来说，适合你的才是最好的。

就像现在很多小孩子都被家长逼着去学钢琴啊什么，有些小孩根本没有艺术细胞的，所以学习的效果就是差强人意了。

其实程序设计最需要基础扎实了，现在的程序设计学习很偏重程序设计语言的学习，或者直白点说，程序设计课程基本上可以说是在学习程序设计语言，

在上一个内容中已经讲解了程序设计是什么的问题，程序设计语言只是程序设计中最后的环节，也是比较简单的环节，只学会程序语言，离实际工作的距离还很遥远，而更多的程序基础其实是在语言之外的东西。就像会写汉字，熟悉汉语语法的人一定能够成为作家吗？

程序设计的基础不外乎以下几个方面：

1、 一定的英文阅读能力

因为程序设计接触的很多文档都是以英文的形式提供的，一个阅读英文很困难的人，可以学会程序设计，但是不会有很深的造诣。就像一个看不懂字典的人，能学好汉语吗？

2、 较强的数学基础

计算机最核心的功能就是计算，各种程序逻辑都会被转成一定格式的运算，运算需要什么知识呢，肯定是数学了。就像一个数学很差的人能做好会计吗？在程序设计中，需要深刻理解数学，用数学来解决你遇到的各种实际问题，类似于做数学应用题吧。这个基础学要长期的积累。

3、 较强的逻辑思维能力

逻辑思维可能每个程序设计人员都很需要，那么逻辑思维是什么呢？其实就是把一个事情分解成一系列的步骤在有限的时间内做完，这个也是程序设计过程中最灵活的地方。例如你要完成“去罗马”这件事情，那么逻辑有多少种呢？借用一句俗话“条条大道通罗马”来解释这个问题吧，所以程序设计是典型的脑力劳动。可能有些人觉得程序设计就是体力活，这也不错，为什么呢，还是借助一个例子来说明吧，买油翁的故事大家都知道吧，如果你反复做一件相同的事情，可能这个事情对外人来说是脑力劳动，对于不断重复做的人来说，也就只是“唯手熟尔”的体力活罢了。

可能很多初学者对于逻辑思维还不是很清楚，那么举一个比较老套的例子吧，例如实现“把一个大象放到冰箱里”这个事情，逻辑是怎样呢？步骤如下：

a、 打开冰箱

b、 把大象推到冰箱里

c、 关上冰箱的门

当然这只是一个很简单的逻辑。在实际的程序设计中还需要严谨的逻辑思维，保证程序可以正常运行。

那么逻辑严谨又是什么呢？还以上面的例子为例，严谨的逻辑思维应该做如下事情：

a、 冰箱打不开怎么办？

b、 大象不进冰箱怎么办？

c、 关不上冰箱门怎么办？

就像一个运动员来说，良好的体质是基础，同样，对于程序员来说，良好的基础可能帮助你达到更高的高度。当然基础不可能每个人都具备，但是数学基础和逻辑思维能力是必须的。

那你的数学基础如何呢，出个简单的数学题目测试一下你的数学基础吧。

已知一组从 1 开始的数字，第一行从左到右依次是 1 到 10，第二行从左到右依次是 11 到 20，按照每行 10 个的顺序依次类推，则任意整数 n(n>0)位于该组数字的第几行第几列呢？

所以，如何你觉得以上的内容你欠缺很多，可能你就不适合做程序设计这个职业，趁早选择其他的职业吧，这样对你的发展会更有利。如果你觉得以上的内容你大部分都符合，那么你可以尝试学习一下后续的内容——《如何学好程序设计》。

希望大家积极讨论，不足之处请大家积极指正。

Java 编程那些事儿 4——如何学好程序设计？

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

俗话说“兴趣是最好的老师”，但是只靠兴趣是远远不够的，还需要付出艰辛的努力。程序设计是一种技能，需要在较短的时间内学会，就不能像学习汉语一样，通过十几年甚至更长的时间来学好，也不能像英语那样进行业余学习，以至很多大学毕业的人英语水平也不敢恭维，也达不到实用的程度。

那么如何学好程序设计呢？或者更现实一点，如何在短时间内成为一个程序员呢？

在接触的学生中，很多人会问：学习程序设计有捷径吗？一般我都不直接回答，而是这样反问他们：大家都看过武侠小说吧，那么练武有没有捷径呢？可能一部分学生会说没有，而另一些同学会说，练武有捷径的啊，比如什么“辟邪剑谱”、“葵花宝典”之类的，但是走这些捷径需要付出很大的代价，但是的确可以快速炼成绝世武功。可惜的是，学习程序设计连这些付出很大代价的秘籍都没有。

但是在实际的学习中，就像练武一样，如果有位前辈对你进行指点或引导，的确可以提高你学习的速度，但是你还是要付出艰辛的努力。

在介绍如何学好程序设计以前，首先要搞明白，学习程序设计需要学什么，其实不外乎以下内容：

l 程序设计语言
语言是程序最终表达的方式，必须熟练。
l 开发工具
开发工具相当于练武的武器，拿个趁手的武器可以发挥你的潜能
l 开发技术
开发技术就是实现好的功能，可以直接拿来用的结构，类似于武功秘籍，但是一定要熟练到可以灵活使用啊。
l 逻辑思维
如何实现程序的要求功能。
l 设计模式
设计模式就是设计的技巧，类似于写作文时的倒序、插叙什么的。

其实学好程序的方法很简单——“勤学苦练”。多读代码，多写代码，是学好程序的基本方式。需要把各种东西熟练到可以灵活使用的程度，由于学习的时

间一般都比较紧，不能像汉语那样炼成习惯成自然的方式，所以在开始学习的初期伴随着大量的记忆，在记忆的基础上进行使用，仔细记忆并体会每个细节，把合适的内容用在合适的地方。

当然，学习最好的开始是模仿，但是由于程序很灵活，所以一定要在模仿的基础上理解，理解了以后进行创新，把东西理解了以后，这些内容就变成了自己的，即使忘记了，也很容易的捡起来，而不要囫囵吞枣式的学习，那样无法做到灵活的解决问题。

当学会了程序设计以后，就可以成为大师级的人物，像武侠小说里的那些大侠一样，做到“意由心生”、“无招胜有招”了，祝福每个初学者以及从事开发的人员，能够早日进入到该境界。

PS：关于英文阅读能力的锻炼，坚持读 1-2 本英文原版书籍(网上很容易下到很多原版电子书)，就可以获得比较大的提升。

Java 编程那些事儿 5——程序设计介绍小结

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

前面简单的介绍了程序设计的相关知识，程序设计也就是用程序语言表达你自己的思维，所以重要的不是语言，而是你的思维，这个是现在程序设计教学中最大的误区，本书中将以语言和思维并重的方式来介绍 Java 语言，并培养你的逻辑思维。

程序设计的道路不是一帆风顺的，其中布满艰辛，所以如果你打算学习程序设计，那么要做好长期吃苦的准备，俗话说：“板凳要做十年冷”，要有这样的耐心和恒心才能把程序设计学会、学好。

然如果基础不好，那么还想学习程序设计课程的话，将需要付出比一般人更多的努力才可以基本赶上一般人的水平，用句不恰当的话“出来混迟早要还的“，前面欠下的基础知识现在就是还的时候了。

关于工作，也简单的说一下，软件开发行业需要各种层次的人才，其实水平只要达到可以独立工作，也就能找到一份工作，但是要想找到一个不错的工作，而且以后还有良好的发展，需要的就是扎实的基础以及长期的努力。

后续讲解语言部分打算分成以下几大块来进行说明：

1、 计算机基础

计算机软件的基本概念、进制的相关知识、计算机内部的数据表达方式等。

2、 Java 语言的简介。

注：我也不能免俗。

3、 Java 开发环境的安装和配置。

介绍 JDK、Eclipse 的安装、配置以及使用。

4、 Java 相关语法和程序逻辑。

以通俗的语言来介绍语法，深刻理解语法的作用和用途。

可能还会进行一些变更，本人保留最终的解释权，呵呵。

Java 编程那些事儿 6——计算机软件基本概念

作者：陈跃峰
出自：**http://blog.csdn.net/mailbomb**

第一章 计算机基础

本部分主要介绍计算机相关的知识，重点介绍和软件编程相关的基础知识。

其实对于编程来说，计算机的基础是越多越深就越好，但是受时间和精力的限制，每个人了解的其实还是很有限，下面就主要编程中常用的基础知识，遗漏的地方请大家积极指正和补充。

计算机本质的结构就是将所有的内容数据化，其实软件编程也采用的是同样的逻辑，把各种需要保存的状态数字化。

1.1 计算机软件基本概念

l 软件的概念

大家都知道，计算机分为硬件和软件，其实看得见摸得着的算硬件，比如硬盘、主板什么的，摸不着的就算软件了。

按照专业点的说法，软件就是一组指令序列，那么如何理解他呢？举个基本的例子，比如大家到学校报名，学校会给你一个单子，一般上面会写，首先到哪里缴费、然后领证件，体检什么的，反正至少也有那么 10 多条，如果把每条操作都看成指令的话，这个就是软件的本质。

或者按照冯·诺伊曼的计算机体系，计算机就是接受输入，进行处理，反馈结果，其实软件也是这样，提供界面接受用户的输入，根据逻辑进行处理，把结果反馈给用户，无论是普通的软件还是游戏都是这样。

l 计算机为什么是二进制的？

众所周知，现在的计算机都是以二进制存储和运算数据的，那么为什么是二进制而不是常见的十进制呢？

原因很简单，因为现在的计算机是电子计算机，内部只有两个状态，所以就依据这两个状态创建了一种新的进制形式——二进制。这样极大的简化了电子计算机的结构，可以用电流的有无、光线的有无以及磁性的有无等状态来实现数学上的二进制。数学上用 0 和 1 来分别代表这两个状态罢了。

当然，随着科学的发展，以后计算机不再是电子计算机了，那么二进制也可能就消失了。

l 计算机存储单位

既然计算机是电子计算机了，那么存储的最小单位就是一个二进制位，英语是 bit，简写成 b。一位只有 2 个值，0 或者 1。

由于位的单位太小，所以就设计了另外一个概念——字节，英语是 byte，简写成 B。规定 1 个字节是 8 位，即 1B=8b。比如大家接触的 8 位机、16 位机等等，

就是指 CPU 一次处理的最小的数据单位。
再大点的单位就依次是 KB、MB、GB 和 TB 了，他们之间的进制都是 2 的 10 次方，也就是 1024，也就是 1KB=1024B，1MB=1024KB。

这里简单的说一个实际问题，大家买硬盘的时候，比如 160GB，这里厂商使用的进制是 1000，而不是 1024，所以 160 个 GB 格式化以后就大概只有：

(160 X 1000 X1000 X1000) / 1024/1024/1024 = 149GB

Java 编程那些事儿 7——进制的概念

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

1.2 进制的概念

因为不可能为每个数值都创造一个符号，所以需要用基本数字组合出复合的数值，这样就有了进制的概念。

其实所有进制都是人为的创造，都是用来计数方便的。现在最常用的进制是十进制，当然其它的进制也在使用中。例如“半斤八两”这个成语，就反映了古代一斤等于十六两的概念，也就是十六进制计数方式。

计算机编程中常用的进制有二进制、八进制、十进制和十六进制，十进制还是最主要的表达形式。在编程中，大家书写的数值默认为十进制。

对于进制，有两个最基本的概念：基数和运算规则。

l 基数

基数指一种进制中组成的基本数字，也就是不能再拆分的数字。例如十进制是 0-9，二进制是 0 和 1，八进制是 0-7，十六进制是 0-9，A-F(大小写均可)。或者可以简单的这样记忆，假设是 n 进制的话，基数就是[0,n-1]的数字，基数的个数和进制值相同，十进制有十个基数，依次类推。

l 运算规则

运算规则就是进位或借位规则，这个类似于一般计算机书籍中位权的概念，例如对于十进制来说，该规则是“满十进一，借一当十”，也就是低位的数字满十了向高位进一，从高位借到的一，相当于低位上的十。其它的进制也是这样，对于二进制来说，就是“满二进一，借一当二”，八进制和十六进制也是这样。

在数学上表示一个数字是几进制，通常使用如下格式：[数值]进制数，例如[10]2 表示二进制数值 10。

1.2.1 二进制

二进制是计算机内部数据表示的形式，所以学习计算机编程必须熟悉二进制。熟悉二进制有以下几个用途：

l 更容易理解计算机的数据存储方式

计算机内部的很多转换，例如数据类型之间的强转，都可以用二进制解释最终的结果的值。

l 二进制的运算速度高

二进制的运算速度比十进制高的多。例如求 2 的 n 次方，通过移位实现的效率比数学方法高效。

1 使用二进制数值进行数据存储

以二进制的形式存储数值，一个是比较节约资源，可以使用二进制的位来存储信息，例如常见的硬件控制信息，都是二进制的形式进行提供的。

如前所述，二进制包含 0 和 1 两个基数，运算规则是“满二进一，借一当二”，下面简单的介绍一下二进制的计数方式。

例如十进制的 0-9 用二进制进行表达，则依次是：

0，1，10，11，100，101，110，111，1000，1001

说明：数值之间使用逗号进行间隔。

下面是二进制的一些基本运算结果：

1 加法运算

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

1 减法

0 - 0 = 0

0 - 1 = -1

1 - 0 = 1

1 - 1 = 0

1 乘法

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

1 除法

0 / 0 无意义

0 / 1 = 0

1 / 0 无意义

1 / 1 = 1

以下是一些符合的表达式：

110 + 111 = 1101

这些基本的运算结构在实际开发中一般不会直接用到，但是通过这些内容可以加深对于二进制概念的理解。

1.2.2 二进制和十进制之间的转换

由于计算机内部的数据是以二进制进行表达的，而十进制又是日常生

活中最常用的进制,所以它们之间经常需要进行转换。下面介绍一下转换的方式。

1.2.2.1 十进制转换为二进制

十进制整数转换为二进制有三种方法，分别是除二取余、计算器转换和经验法。十进制小数的转换方法最后做简单的介绍。

1. 除二取余法

除二取余法是转换时的最基本方法，也是最通用的方法。规则为：使用十进制和2去除，取每次得到的商和余数，用商继续和2相除，直到商为零为止，第一次得到的余数作为二进制的低位，最后一次得到的余数作为二进制的高位，由余数组成的数字就是转换后二进制的值。例如十进制的13转换为二进制的计算步骤如下：

	商	余数
13 / 2 =	6	1
6 / 2 =	3	0
3 / 2 =	1	1
1 / 2 =	0	1

则计算的最终结果就是1101。

2. 计算器转换

Windows操作系统中的计算器也可以很方便的实现进制之间的转换。在程序菜单中附件子菜单中打开计算器，从打开的计算器的查看菜单中，选择“科学型”，输入你要转换的十进制的数字，例如13，然后界面上数字显示框西侧的“二进制“，则转换后的数值就直接显示在计算器中。

3. 经验法

对于二进制熟悉以后,那么计算十进制对应的数字可以通过一些基本的数学变换来实现，在使用经验法以前，必须熟记2的0-10次方对应的十进制的值，依次是：

1，2，4，8，16，32，64，128，256，512，1024

则转换一些特殊的数字时可以极大的提高转换速度，例如数字65，则可以这样转换：

65 = 64 + 1

64对应的二进制形式为1000000

1对应的二进制形式为1

则65的二进制形式为1000001

这个只适合转换一些特殊的数字，适应性没有除二取余法广泛。

十进制小数的转换采用的一般方法是乘二取整法，规则为：对于小数部分先乘二,然后获得运算结果的整数部分,然后将结果中的小数部分再次乘二，直到小数部分为零为止，则把第一次得到的整数部分作为二进制小数的高位，后续的整数部分作为地位就是转换后得到的二进制小数。需要说明的是，有些十进制小数无法准确的用二进制进行表达，所以转换时符合一定的精度即可，这也是为什么计算机的浮点数运算不准确的原因。

例如0.25转换为二进制小数的步骤如下：

整数部分

0.25 × 2 = 0.5　　0

0.5 × 2 = 1.0　　1

则 0.25 转换为二进制小数为 0.01

如果一个十进制数字既有整数部分，也有小数部分，则分开进行转换即可。

1.2.2.2 二进制转换为十进制

二进制转换为十进制采用的方法是：数字乘位权相加法。下面先以十进制为例来说明该方法，例如十进制数字 345 的值，5 的位权是 1，4 的位权是 10，3 的位权是 100，则有如下表达式成立： 345=5 × 1 + 4 × 10 + 3 × 100，这就是数字乘位权相加法的原理。

其实对于十进制整数的位权很有规则，从右向左第 n 位的位权是十的(n-1)方，例如个位是 10(1-1)，十位是 10(2-1),依次类推。那么二进制整数的位权规律和这个一致，也就是从右向左第 n 位的位权是二的(n-1)方。

例如二进制整数 1011 转换为十进制的表达式为：

[1011]2 = 1 × 20 + 1 × 21 + 0 × 22 + 1 × 23 = 1 + 2 + 0 + 8=11

而对于二进制的小数，也是采用一样的方法，只是二进制小数的位权规则为，小数点后第一位小数的位权是 2 的-1 次方，第二位是 2 的-2 次方，依次类推。

例如二进制小数 0.1101 转换为十进制小数的表达式为

[0.1101]2=1 ×2-1 + 1 ×2-2 + 0 × 2-3 + 1 × 2-4 = 0.5 + 0.25 + 0 + 0.0625=0.8125

同理，如果二进制包含整数和小数部分，则分开进行转换即可。

1.2.3 二进制和八进制、十六进制之间的转换

虽然二进制是计算机内部的数据表达形式，但是由于二进制基数太少，则导致数字比较长，为了简化数字的书写，就创建了八进制和十六进制。八进制和十六进制就是对二进制的简化，所以二进制到八进制和十六进制的转换非常简单。

二进制整数转换为八进制的方法是“三位一并“，也就是从右侧开始，每 3 位二进制数字转换为八进制的一位，依次类推，因为二进制的三位数字可以表达的区间是 000-111，刚好和 0-7 重合。例如：

二进制的 10111 转换为 8 进制为：最后三位 111 转换为 7，前面的数字 10 转换为 2，则转换后得到的八进制数字为 27。

二进制整数转换为十六进制的方法是“四位一并“，例如 10111 转换为十六进制是 0111 转换为 7，1 转换为 1，则转换后得到的十六进制数字是 17。

二进制小数转换为八进制的方法也是“三位一并“，只是转换时从小数的高位开始，也就是小数的左侧开始。例如 0.10111 转换为八进制是 101 转换为 5，110 转换为 6，则转换得到的八进制小数为 0.56。需要特别注意的是，小数最后如果不足三位，一定要在后续补零以后再进行转换。

二进制小数转换为十六进制的方法也是“四位一并”，只是转换时从小数的高位开始。例如二进制小数 0.10111 转换为十六进制小数为，1011 转换为 b，

1000 转换为 8，则转换后得到的十六进制是 0.b8。

如果二进制数包含整数和小数部分，则分开进行转换。

Java 编程那些事儿 8——计算机内部的数据表达

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

1.3 计算机内部的数据表达

计算机内部数据表达的总原则就是：把一切内容数值化、数字化。这个也是编程时处理数据的基本方式，对于编程理解的越深入，则将越认同该原则。

其实计算机也只能这样，因为计算机内部只能存储 0 和 1 两个数字，所以必须把指令、数据、图片、文本等各种各样的内容数字化成 0 和 1 进行存储、传输和显示。

1.3.1 整数的表达

整数有正负之分，但是计算机内部只能存储 0 和 1，则计算机内部将符号数字化，用二进制码的最高位代表符号位，规定该位为 0 代表正，1 代表负。这就是符号数字化的规定。

前面介绍过整数在计算机内部都是以二进制的形式保存的。但是为了计算方便，以及简化 CPU 的结构，所以在存储和运算时都采用补码的形式。

前面介绍的那些直接计算出来的二进制形式，都称作整数的原码。规定正数的原码、反码和补码都是自身。

而对于负数，仔细研究一下其组成格式。以 8 位机为例，也就是一个数字占计算机中的 8 位，也就是一个字节，用最高位存储符号位，其它的位存储数值。例如-8 的原码是 10001000，最高位的 1 代表负数，后续的 7 位代表数值。

负数的反码是指符号位不变，其他的位取反，也就是 0 变 1，1 变 0，则-8 的反码是 11110111。

负数的补码是指在反码的数值位上加 1，运算后得到的结果，只计算数值位，不改变符号位。则-8 的补码是 11111000，该次运算中，低位向高位进行了进位。

规律：补码的补码等于负数的原码。

也就是对负数的补码再求补，则得到的负数的原码。

熟悉整数的表达，对于后续理解数据的区间以及进行强制转换以后得到的数值很有帮助，也是进行位运算的基础。

备注：小数，编程语言中称浮点数，的存储形式和整数不同。

1.3.2 字符的表达

字符指计算机内部单个的符号，如标点符号、英文字母和汉字等等。因为这些字符种类各异，计算机无法直接表达，那么就采用了计算机编程中也常用的方式，对每个字符进行编号，例如规定 a 字符编号为 97，b 字符编号为 98 等等。

由于需要编号的字符很多，就专门规定了一系列字符和编号的对应规则，那么这些对应表就被称作字符集，常见的字符集有 ASCII、GB2312、BIG5 等。

在计算机内部存储、运算和传输时，都只需要使用该编号即可。

字符集比较完美的解决了字符的存储和传输的问题。

所以字符在程序内部可以参与运算，其实参与运算的就是这个字符的编号，字符集规律是很多字符变换逻辑实现的基础。

备注：字符的显示则通过专门的字符显示码实现。

1.3.3 总结

其实计算机内部所有的东西都是以数字的形式存储的，这里只是希望通过这两种简单的结构，使大家了解将数据数字化的思想，这是编程时常用的思想之一。

Java 编程那些事儿 9——网络编程基础

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

1.4 网络编程

对于初学者，或者没有接触过网络编程的程序员，会觉得网络编程涉及的知识很高深，很难，其实这是一种误解，当你的语法熟悉以后，其实基本的网络编程现在已经被实现的异常简单了。

1.4.1 网络编程是什么？

网络编程的本质是两个设备之间的数据交换，当然，在计算机网络中，设备主要指计算机。数据传递本身没有多大的难度，不就是把一个设备中的数据发送给两外一个设备，然后接受另外一个设备反馈的数据。

现在的网络编程基本上都是基于请求/响应方式的，也就是一个设备发

送请求数据给另外一个，然后接收另一个设备的反馈。

在网络编程中，发起连接程序，也就是发送第一次请求的程序，被称作客户端(Client)，等待其他程序连接的程序被称作服务器(Server)。客户端程序可以在需要的时候启动，而服务器为了能够时刻相应连接，则需要一直启动。例如以打电话为例，首先拨号的人类似于客户端，接听电话的人必须保持电话畅通类似于服务器。

连接一旦建立以后，就客户端和服务器端就可以进行数据传递了，而且两者的身份是等价的。

在一些程序中，程序既有客户端功能也有服务器端功能，最常见的软件就是 BT、emule 这类软件了。

下面来谈一下如何建立连接以及如何发送数据。

1.4.2 IP 地址和域名

在现实生活中，如果要打电话则需要知道对应人的电话号码，如果要寄信则需要知道收信人的地址。在网络中也是这样，需要知道一个设备的位置，则需要使用该设备的 IP 地址，具体的连接过程由硬件实现，程序员不需要过多的关心。

IP 地址是一个规定，现在使用的是 IPv4，既由 4 个 0-255 之间的数字组成，在计算机内部存储时只需要 4 个字节即可。在计算机中，IP 地址是分配给网卡的，每个网卡有一个唯一的 IP 地址，如果一个计算机有多个网卡，则该台计算机则拥有多个不同的 IP 地址，在同一个网络内部，IP 地址不能相同。IP 地址的概念类似于电话号码、身份证这样的概念。

由于 IP 地址不方便记忆，所以有专门创造了域名(Domain Name)的概念，其实就是给 IP 取一个字符的名字，例如 163.com、sina.com 等。IP 和域名之间存在一定的对应关系。如果把 IP 地址类比成身份证号的话，那么域名就是你的姓名。

其实在网络中只能使用 IP 地址进行数据传输，所以在传输以前，需要把域名转换为 IP，这个由称作 DNS 的服务器专门来完成。

所以在网络编程中，可以使用 IP 或域名来标识网络上的一台设备。

1.4.3 端口的概念

为了在一台设备上可以运行多个程序，人为的设计了端口(Port)的概念，类似的例子是公司内部的分机号码。

规定一个设备有 216 个，也就是 65536 个端口，每个端口对应一个唯一的程序。每个网络程序，无论是客户端还是服务器端，都对应一个或多个特定的端口号。由于 0-1024 之间多被操作系统占用，所以实际编程时一般采用 1024 以后的端口号。

使用端口号，可以找到一台设备上唯一的一个程序。

所以如果需要和某台计算机建立连接的话，只需要知道 IP 地址或域名即可，但是如果想和该台计算机上的某个程序交换数据的话，还必须知道该程序使用的端口号。

1.4.4 数据传输方式

知道了如何建立连接，下面就是如何传输数据了，先来看一下数据传输的方式。

在网络上，不管是有线传输还是无线传输，数据传输的方式有两种：

<!--[if !supportLists]-->l <!--[endif]-->TCP(Transfer Control Protocol)

传输控制协议方式，该传输方式是一种稳定可靠的传送方式，类似于显示中的打电话。只需要建立一次连接，就可以多次传输数据。就像电话只需要拨一次号，就可以实现一直通话一样，如果你说的话不清楚，对方会要求你重复，保证传输的数据可靠。

使用该种方式的优点是稳定可靠，缺点是建立连接和维持连接的代价高，传输速度不快。

<!--[if !supportLists]-->l <!--[endif]-->UDP(User Datagram Protocol)

用户数据报协议方式，该传输方式不建立稳定的连接，类似于发短信息。每次发送数据都直接发送。发送多条短信，就需要多次输入对方的号码。该传输方式不可靠，数据有可能收不到，系统只保证尽力发送。

使用该种方式的优点是开销小，传输速度快，缺点是数据有可能会丢失。

在实际的网络编程中，大家可以根据需要选择任何一种传输方式，或组合使用这两种方式实现数据的传递。

1.4.5 协议的概念

协议(Protocol)是网络编程中一个非常重要的概念，指的是传输数据的格式。因为大家在网络中需要传输各种各样的信息，在程序中获得到的都是一组数值，如何阅读这些数值呢，就需要提前规定好这组数据的格式，在客户端按照该格式生成发送数据，服务器端按照该格式阅读该数据，然后在按照一定的格式生成数据反馈给客户端，客户端再按照该格式阅读数据。现实中类似的例子就是电报编码，每个数字都是用特定的数据表达。

一般程序的协议都分成客户端发送的数据格式，以及服务器端反馈的数据格式，客户端和服务器端都遵循该格式生成或处理数据，实现两者之间的复杂数据交换。

1.4.6 小结

网络编程就是使用 IP 地址，或域名，和端口连接到另一台计算机上对应的程序，按照规定的协议(数据格式)来交换数据，实际编程中建立连接和发送、接收数据在语言级已经实现，做的更多的工作是设计协议，以及编写生成和解析数据的代码罢了，然后把数据转换成逻辑的结构显示或控制逻辑即可。

需要了解更多的网络编程的知识，建议阅读《JAVA2 网络协议内幕》一书。

Java 编程那些事儿 10——Java 语言简介

Java 编程那些事儿 11——JDK 的安装、配置和使用

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

第二章 建立开发环境

“工欲善其事，必先利其器”。

进行程序开发，首先要安装开发相关的软件，并且熟悉这些工具软件的基本使用。本章介绍一下两类开发工具的使用：

l 基础开发工具

基础开发工具是进行程序设计的基础，包含开发中需要的一些基本功能，例如编译、运行等，是其它开发工具的基础。

Java 语言的基本开发工具是 SUN 公司免费提供的 JDK。

实际开发中，为了方便和程序开发的效率，一般不直接使用基础开发工具，所以对于很多基础开发工具，只需要掌握其基本的使用即可。

l 集成开发环境(IDE)

集成开发环境是指将程序设计需要的很多功能，例如代码编辑、代码调试、程序部署等等一系列功能都整合到一个程序内部，方便程序开发，并提高实际的开发效率，简化了程序设计中的很多操作。

Java 语言的集成开发环境很多，常见的有 Eclipse、JBuilder、NetBeans 等等。

由于实际开发中，基本都是使用集成开发环境进行开发，所以在学习中必须熟练掌握该类工具的使用。

一般集成开发环境的使用都很类似，在学习时只要熟练掌握了其中一个的使用，其它的工具学习起来也很简单。

本文以 Eclipse 为例来介绍集成开发环境的基本使用。

2.1 JDK 开发环境

JDK(Java Developer’s Kit)，Java 开发者工具包，也称 J2SDK (Java 2 Software Development Kit)，是 SUN 公司提供的基础 Java 语言开发工具，该工具软件包含 Java 语言的编译工具、运行工具以及执行程序的环境(即 JRE)。

JDK 现在是一个开源、免费的工具。

JDK 是其它 Java 开发工具的基础，也就是说，在安装其它开发工具以前，必须首先安装 JDK。

对于初学者来说，使用该开发工具进行学习，可以在学习的初期把精力放在 Java 语言语法的学习上，体会更多底层的知识，对于以后的程序开发很有帮助。

但是 JDK 未提供 Java 源代码的编写环境，这个是 SUN 提供的很多基础开发工具的通病，所以实际的代码编写还需要在其它的文本编辑器中进行。其实大部分程序设计语言的源代码都是一个文本文件，只是存储成了不同的后缀名罢了。

常见的适合 Java 的文本编辑器有很多，例如 JCreator、Editplus、UltraEdit 等。

下面依次来介绍 JDK 的下载、安装、配置和使用。

2.1.1 JDK 的获得

如果需要获得最新版本的 JDK，可以到 SUN 公司的官方网站上进行下载，下载地址为：

http://java.sun.com/javase/downloads/index.jsp

下载最新版本的“JDK 6 Update 6”，选择对应的操作系统，以及使用的语言即可。

在下载 Windows 版本时，有两个版本可供下载,，分别是：

l Windows Online Installation

在线安装版本，每次安装时都从网络上下载安装程序，在下载完成以后，进行实际的安装。

l Windows Offline Installation

离线安装版本，每次安装时直接进行本地安装。

通常情况下，一般下载离线安装版本。

其实如果不需要安装最新版本的话，也可以在国内主流的下载站点下载 JDK 的安装程序，只是这些程序的版本可能稍微老一些，这些对于初学者来说其实问题不大。

2.1.2 JDK 的安装

Windows 操作系统上的 JDK 安装程序是一个 exe 可执行程序，直接安装即可,在安装过程中可以选择安装路径以及安装的组件等,如果没有特殊要求，选择默认设置即可。程序默认的安装路径在 C:\Program Files\Java 目录下。

2.1.3 JDK 的配置

JDK 安装完成以后，可以不用设置就进行使用，但是为了使用方便，一般需要进行简单的配置。

由于 JDK 提供的编译和运行工具都是基于命令行的，所以需要进行一下 DOS 下面的一个设定，把 JDK 安装目录下 bin 目录中的可执行文件都添加到 DOS 的外部命令中，这样就可以在任意路径下直接使用 bin 目录下的 exe 程序了。

配置的参数为操作系统中的 path 环境变量，该变量的用途是系统查找可执行程序所在的路径。

配置步骤为：

1、“开始”>“设置”>“控制面板”>“系统”

如果控制面板的设置不是经典方式，那么可以在控制面板的“性能和维护”中找到“系统”。

当然，也可以选择桌面上的“我的电脑”，点击鼠标右键，选择“属性”打开。

2、在“系统属性”窗口中，选择“高级”属性页中的“环境变量”按钮。

3、在“环境变量”窗口中，选择“系统变量”中变量名为“Path”的环境变量，双击该变量。

4、把 JDK 安装路径中 bin 目录的绝对路径，添加到 Path 变量的值中，并使用半角的分号和已有的路径进行分隔。

例如 JDK 的安装路径下的 bin 路径是 C:\Program Files\Java\jdk1.6.0_04\bin，则把该路径添加到 Path 值的起始位置，则值为：C:\Program Files\Java\jdk1.6.0_04\bin;C:\Program Files\PC Connectivity Solution\;C:\Program Files\Java\jdk1.6.0_04\bin;C:\j2sdk1.4.2_11\bin;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem

以上路径在不同的计算机中可能不同。

配置完成以后，可以使用如下格式来测试配置是否成功：

1、 打开“开始”>“程序”>“附件”>“命令提示符”

2、 在“命令提示符”窗口中，输入 javac，按回车执行

如果输出的内容是使用说明，则说明配置成功。

如果输出的内容是“’javac’不是内部或外部命令，也不是可执行的程序或批处理文件。”，则说明配置错误，需要重新进行配置。

常见的配置错误为：

1)路径错误，路径应该类似 C:\Program Files\Java\jdk1.6.0_04\bin。

2)分隔的分号错误，例如错误的打成冒号或使用全角的分号。

Java 编程那些事儿 12——第一个 HelloWorld 程序

Java 编程那些事儿 13——Eclipse 基本使用

作者：陈跃峰

出自：http://blog.csdn.net/mailbomb

2.2 Eclipse 使用

Eclipse 是一个开源、免费的集成开发工具。

Eclipse 是实现开发中的主流开发工具之一，熟练使用该工具将在学习，以及以后的实际开发中让你如虎添翼。

如果把程序员类比成军队中的士兵的话，那么集成开发工具就是你手中的枪，你要对它足够的熟悉，并且足够熟练的使用它。

对于开发工具的学习，需要在学习中使用，在使用中学习。

2.2.1 Eclipse 的获得

Eclipse 的安装程序可以从其官方网站上免费下载，地址为：

http://www.eclipse.org

在下载时选择“Eclipse Classic”下载即可，其最新版本为 3.3.2。

需要注意的是，在现在时一定要下载 SDK，而且根据你的操作系统选择对应的版本，例如 Windows 平台上的文件默认是 eclipse-SDK-3.3.2-win32.zip。

以下为 Windows 操作系统为例子来介绍 Eclipse 的安装。

2.2.2 Eclipse 的安装

Eclipse 是一个使用 Java 语言开发的工具软件，所以在安装 Eclipse 以前，一定要安装 JDK，其中 Eclipse3.3.2 要求安装的 JDK 版本在 1.5 及以上。

Eclipse 的安装很简单，只需要解压缩安装文件即可，解压缩的文件没有限制，可以根据实际使用的需要解压缩到任意路径下。

2.2.3 Eclipse 基本使用

Eclipse 安装完成以后，选择 Eclipse 安装目录下的 eclipse.exe 即可启动该软件。

2.2.3.1 工作空间设置

第一次启动 Eclipse 时，会弹出一个标题为“Workspace Launcher”的窗口，该窗口的功能是设置 Eclipse 的 workspace(工作空间)。workspace 是指 Eclipse 新建的内容默认的保存路径，以及 Eclipse 相关的个性设置信息。该窗口中“Workspace”输入框中是需要设置的路径，可以根据个人的需要进行设置，下面的“Use this as default and do not ask again”选择项的意思是：使用这个作为默认设置，以后不要再询问，选中以后的效果是：1、下次启动时不再弹出该窗口，2、把这个设置作为默认设置，不选中该选择项则每次启动时都弹出该窗口。设置完成以后，选择“OK”按钮，就可以启动 Eclipse 了。

2.2.3.2 显示主界面

Eclipse 第一次启动起来以后，会显示一个欢迎界面，选择左上角“Welcome”右侧的“X”关闭欢迎界面，就可以看到 Eclipse 的主界面了。

欢迎界面只显示一次，以后只有在变更了工作空间以后才可能会再次显示。

关于 Eclipse 界面的布局方式，这里暂不介绍，因为介绍时不可避免的要用到一些后续要学到的专业术语，这里先简单介绍一下其使用方式，至于界面的布局方式这个在使用中逐渐去熟悉。

2.2.3.3 Eclipse 基本使用

集成开发环境(IDE)的使用相对来说稍显繁琐，但是对于实际的项目开

发来说却是非常实用的，在初次使用时，需要习惯和适应这种使用方式。

集成开发环境在使用前，需要首先建立Project(项目)，Project是一个管理结构，管理一个项目内部的所有源代码和资源文件，并保存和项目相关的设置信息。

一个项目内部可以有任意多个源文件，以及任意多的资源。

使用Eclipse的基础步骤主要有如下这些：

l 新建项目
l 新建源文件
l 编辑和保存源文件
l 运行程序

2.2.3.3.1 新建项目

新建项目的步骤如下：

1、 选择菜单“File”>“New”>“Java Project”
2、 在“New Java Project”窗口中，进行新建项目的设定，例如输入Test
“Project Name”是必须输入的内容，代表项目名称，在硬盘上会转换成一个文件夹的名称。
“Content”设置项目的内容。
“JRE”部分设置项目使用的JDK版本。
“Project layout”部分设置项目文件内部的目录结构。
3、 选择“Finish”按钮完成设置
项目建立以后，可以到磁盘对应路径下观察一下项目文件夹的结构。

2.2.3.3.2 新建源文件

项目建立以后，或者打开项目以后，就可以新建源文件了。

一个项目中可以包含多个源文件，每个源文件都可以独立执行。

新建源文件的步骤为：

1、 选择菜单“File”>“New”>“Class”
2、 在“New Java Class”向导中，进行新建源文件的设定
“Source folder”代表源代码目录，例如“test/src”，如果该内容和项目保持一致则不需要修改，否则可以选择后续的“Browse…”按钮进行修改。
“Name”代表源文件的名称，例如输入Hello。
“public static void main(String[] args)”选项代表在生成的源代码中包含该代码，选中该设置。
3、 选择“Finish”按钮完成设置，则Eclipse将自动生成符合要求的源代码，并在Eclipse环境中打开。
生成的代码如下：

```
publicclass Hello {

        /**
         *@paramargs
         */
```

```
        publicstaticvoid main(String[] args) {
            // TODO Auto-generated method stub

        }

}
```

2.2.3.3.3 编辑和保存源文件

可以把其中的用于说明的注释内容删除，并添加输出字符串的代码，则代表变为如下内容：

```
publicclass Hello {
    publicstaticvoid main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

选择工具栏的保存按钮，或者按 Ctrl+S 组合键保存源文件，在源文件保存时，Eclipse 会自动编译该代码，如果有语法错误，则以红色波浪线进行提示。

2.2.3.3.4 运行程序

运行程序的方法为：选择源代码空白处，点击右键，选择“Run as”>“1 Java Application”即可运行，当然，你也可以选择 Eclipse 左侧你需要运行的文件名，点击右键，也可以找到一样的菜单进行运行。

这样，程序的运行结果就显示在该界面中了。

Java 编程那些事儿 14——Eclipse 基础使用进阶

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

2.2.4 Eclipse 基础使用进阶

熟悉了 Eclipse 基本的使用以后，下面再补充一下 Eclipse 其它常见的操作，主要包含以下一些技巧：

l 分类项目目录
l 打开项目
l 添加 JDK
l 添加源代码

1 快捷键速查

2.2.4.1 分类项目目录

在默认的Eclipse项目目录下，源代码和class文件都存储在项目根目录下,这样项目目录显得比较凌乱,可以通过以下设置来设置项目目录的结构：

菜单“Window”>“Preferences…”>“Java”>“Build Path”，选择右侧的“Folders”选项，设置“Source folder name”源代码目录名称为src，“Output folder name”输出目录名称为classes，选择“OK”按钮完成设置。

这样在新建eclipse项目时，将在项目目录下自动新建src和classes这两个文件夹，并将新建的源代码文件默认存储在项目目录下的src目录中，将生成的class文件默认存储在项目目录下的classes目录中。

2.2.4.2 打开项目

在日常的使用中，经常需要打开已有的项目，在eclipse中打开已有的eclipse项目的操作步骤为：

菜单“File”>“Import”>“General”>“Existing Projects into Workspace”，选择“Next”按钮打开导入窗口，选择“Select root directory”后面的“Browse…”按钮选择项目的根目录，此时项目名称会现在导入窗口下面的空白区域内，选择“Finish”按钮，即可完成打开项目的操作。

2.2.4.3 添加JDK

默认的Eclipse中只集成一个JDK，如果在新建项目时需要不同版本的JDK，则需要首先在Eclipse中集成对应的JDK，然后才可以在新建项目时进行选择。

添加新的JDK到Eclipse中的步骤如下：

菜单“Window”>“Preferences…”>“Java”>“Installed JREs”，选择右侧的“Add”按钮，然后选择弹出窗口中的“JRE home directory”后的“Browse…”按钮选择需要添加的JDK安装目录的根目录，例如C:\j2sdk1.4.2_11，选择“OK”完成添加即可。

2.2.4.4 添加源代码

将已有的java源代码添加到已有的项目中，这样才可以在eclipse中运行该程序，添加的步骤为：

1、 复制该文件，而非该文件的内容

2、 选择Eclipse界面左侧的项目列表中，对应项目的源代码目录，例如Hello项目下的src目录

3、 选择ctrl+v粘贴即可

2.2.4.5 快捷键速查

为了方便对于eclipse的操作，Eclipse提供了常见快捷键的功能和列表，可以选择菜单“Help”>“ Key Assist”查看。

2.3 总结

开发环境配置和使用部分就简单的介绍这么，其实只掌握这些还满足不了实际开发的要求，还需要在使用中大量的进行学习，并熟练这些工具软件的使用。

Java 编程那些事儿 15——如何学好 Java 语法

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

第三章 Java 基础语法

学习一个程序语言，首先需要学习该语言的格式，这个格式就是语言的语法。语法，对于初学者来说，是学习程序最枯燥的地方，因为语法部分需要记忆的内容太多，而且需要理解很多的知识。

而对于曾经接触过其他程序设计语言的人来说，学习语法的速度特别快，主要是因为语法部分涉及的很多概念已经理解，而且大部分语言的语法格式比较类似。

本章就来详细介绍一下 Java 语言的基础语法，希望能够通过本章的学习掌握 Java 语言的语法格式。

3.1 如何学好 Java 语法

对于初学者来说，学习 Java 语法是一个长期、艰苦的过程，所以要做好吃苦的准备，而且语法的学习会随着对于语言理解的加深，而体会到更多设计的巧妙。

语法格式只是学习程序时最基础的知识，在实际的开发中，需要根据程序的需要，使用恰当的格式去实现程序逻辑，所以语法一定要熟练。

学习语法主要有三种学习方式：

1 在理解的基础上记忆

这个是最理想的学习语法的方式，通过这样学习语法会觉得很轻松，而且对于语法的使用也把握的很准确。

对于这种方式，需要在学习的初期深刻理解语法的功能，体会语法适用的场合，记忆语法的实现格式。

但是在实际的学习中，由于初学者未接触过开发，很多的概念无法深刻理解，所以很多人还无法实现使用该方式来进行学习。

1 在记忆的基础上理解

使用这种方式，首先需要记忆住该语法格式，然后在记忆以及后续的练习中逐步

体会语法的用途，这也是大部分初学者学习语法的方式。
通过这种方式学习语法，也可以在一定的锻炼以后成为合格的程序员。
1 在未理解的基础上记忆
这种方式是学习语法是最笨拙的方式，也是很多很努力学习，但是无法理解语法格式的学生。
使用这种方式其实没有真正理解语法的格式，而只是生硬的去进行记忆，很多时候还无法灵活的去运用这些格式，所以学习的效果也打了一定的折扣。

学习语法时，主要需要学习和理解以下这些内容：
1 语法的格式
这个必须进行记忆，熟记以后可以提高写代码的速度。
1 语法概念
理解相关的语法概念，例如变量、方法、数组等等
1 语法的用途
语法的适用领域。
1 大量练习
通过大量的练习深刻体会语法的使用。

关于语法的学习就介绍这么多，下面就进入设计巧妙的 Java 语法世界开始正式的学习了。

Java 编程那些事儿 16——代码框架、关键字和标识符

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

3.2 代码框架

“罗马不是一天建成的”，所以想只学习几天的语法或者一两周的语法就能很熟练的编程，是不实际的想法。说个极端的例子，你的英语学了多少年了，能很流利的与人交流和书写文章吗？当然，就程序语法来说，比英语简单多了。在开始学习 Java 时，不可能把所有的语法都一下子介绍清楚，但是如果需要把程序正确的运行起来，那么还必须不少的语法知识，为了在学习的初期可以让自己编写的代码编译通过，并且能够执行，所以特提供一个简单的代码框架，方便大家初期的练习。

代码框架的结构如下：

```
public class 文件名{
        public static void main(String[] args){
                你的代码
        }
}
```

使用该代码框架时，只需要把“文件名”的位置换成自己的文件名，

并且在“你的代码”的位置写自己的代码即可，使用示例：

```
public class Hello{
        public static void main(String[] args){
                System.out.println(“Hello world!”);
        }
}
```

则在该示例代码中，“文件名”被替换成了 Hello，“你的代码”被替换成了 System.out.println(“Hello world!”);,在后续的示例中，给出的代码片段，除非特别说明，都是应该写在“你的代码”位置的代码，后面就不专门声明了。

3.3 关键字

关键字(keyword)，也称保留字(reserved word)，是指程序代码中规定用途的单词。也就是说，只要在程序代码内部出现该单词，编译程序就认为是某种固定的用途。

关键字列表及中文解释如下，格式为：关键字(中文解释)：

abstract(抽象的) continue(继续) for(当…的时候) new(新建) switch(转换)
assert(断言) default(默认) if(如果) package(打包) synchronized(同步)
boolean(布尔) do(做) goto(跳转到) private(私有的) this(这个)
break(中断) double(双精度) implements(实现) protected(受保护的) throw(抛出，动词)
byte(字节) else(否则) import(引入) public(公共的) throws(抛出，介词)
case(情形) enum(枚举) instanceof(是…的实例) return(返回) transient(瞬时的)
catch(捕获) extends(继承) int(整数) short(短整数) try(尝试)
char(字符) final(最终的) interface(接口) static(静态的) void(空的)
class(类) finally(最终地) long(长整数) strictfp(精确浮点) volatile(易变的)
const(常量) float(单精度浮点) native(本地的) super(超级的) while(当…的时候)

说明：其中 goto 和 const 的用途被保留，在语法中未使用到这两个关键字。
在实际学习时，必须牢记关键字的意义以及拼写。
后续学习的语法知识，大部分都是使用关键字和一些符号组成的，关键字的意义基本上就代表了该种语法格式的用途。

3.4 标识符

标识符，也就是标识的符号，指程序中一切自己指定的名称，例如后续语法中涉及到的变量名称、常量名称、数组名称、方法名称、参数名称、类名、接口名、对象名等等。

其实程序中除了一些分隔符号，如空格、括号和标点符号以外，只有三类名称：

l 关键字

l 系统功能名称

如 System.out.println 中的 System、out 和 println。

l 标识符

对于英语不好的学习者来说，第一类和第二类都是需要熟悉和记忆的，而第三类名称，也就是标识符，可以由程序开发者自己进行设定。

通常情况下，为了提高程序的可读性，一般标识符的名称和该标识符的作用保持一致。

标识符的命名规则主要有如下几个要求：

1、 不能是关键字

2、 只能以字母、下划线(_)和美元符号($)开头

需要特别注意的是，标识符不能以数字字符开头。

3、 不能包含特殊字符，例如空格、括号和标点符号等等。

通常情况下，标识符一般全部是字母，或者使用字母和数字的组合。

Java 编程那些事儿 17——基本数据类型

作者：陈跃峰

出自：http://blog.csdn.net/mailbomb

3.5 基本数据类型

程序中最核心的就是一系列的数据，或者叫程序状态，计算机为了方便的管理数据，就为数据设定了一组类型，这样在为数据分配内存以及操作数据时都比较方便，这就是数据类型的由来。其实现实生活中也存在各种数据类型，例如数字型，字符型等，数字型又可以划分为整数型和小数型，只是没有很可以的划分罢了。

在数据类型中，最常用也是最基础的数据类型，被称作基本数据类型。可以使用这些类型的值来代表一些简单的状态。

3.5.1 概述

学习数据类型的目的就是在需要代表一个数值时，能够选择合适的类型。当然，有些时候好几种类型都适合，那就要看个人习惯了。

学习数据类型需要掌握每种数据类型的特征，以及对应的细节知识，这样会有助于对类型的选择。所以在初次学习时，需要记忆很多的内容

Java 语言的基本数据类型总共有以下 8 种，下面是按照用途划分出的 4 个类别：

l 整数型：byte(字节型)、short(短整型)、int(整型)、long(长整型)
l 小数型：float(单精度浮点型)、double(双精度浮点型)
l 字符型
l 布尔型

3.5.2 整数型

整数型是一类代表整数值的类型。当需要代表一个整数的值时，可以根据需要从 4 种类型中挑选合适的，如果没有特殊要求的话，一般选择 int 类型。4 种整数型区别主要在每个数据在内存中占用的空间大小和代表的数值的范围。具体说明参看下表：

整数型参数表

类型名称	关键字	占用空间(字节)	取值范围	默认值
字节型	byte	1	-27—27-1	0
短整型	short	2	-215—215-1	0
整型	int	4	-231—231-1	0
长整型	long	8	-263—263-1	0

说明：1、Java 中的整数都是有符号数，也就是有正有负。

2、默认值指在特定的情况下才自动初始化，具体的情况后续将有叙述。

3、程序中的整数数值默认是 int 以及 int 以下类型，如果需要书写 long 型的值，则需要在数值后面添加字母 L，大小写均可。

4、程序中默认整数是十进制数字，八进制数字以数字字符 0 开头，例如 016、034 等，十六进制数字以数字字符 0 和字母 x(不区分大小写)开头，例如 0xaf、0X12 等。

3.5.3 小数型

小数型是一类代表小数值的类型。当需要代表一个小数的值时，可以根据需要从以下 2 种类型中挑选合适的。如果没有特殊要求，一般选择 double 类型。

由于小数的存储方式和整数不同，所以小数都有一定的精度，所以在计算机中运算时不够精确。根据精度和存储区间的不同，设计了两种小数类型，具体见下表：

小数型参数表

类型名称	关键字	占用空间(字节)	取值范围	默认值
单精度浮点型	float	4	-3.4E+38—3.4E+38	0.0f
双精度浮点型	double	8	-1.7E+308—1.7E+308	0.0

说明：1、取值范围以科学计数法形式进行描述。

2、在程序中，小数的运算速度要低于整数运算。

3、float 类型的小数，需要在小数后加字母 f，不区分大小写，例如 1.01f。

3.5.4 字符型

字符型代表特定的某个字符，按照前面介绍的知识，计算机中都是以字符集的形式来保存字符的，所以字符型的值实际只是字符集中的编号，而不是实际代表的字符，由计算机完成从编号转换成对应字符的工作。

Java 语言中为了更加方便国际化，使用 Unicode 字符集作为默认的字符集，该字符集包含各种语言中常见的字符。

在程序代码中，字符使用一对单引号加上需要表达的字符来标识，例如’A’、’a’等，当然也可以直接使用字符编码，也就是一个非负整数进行表示。

字符型参数表

类型名称	关键字	占用空间(字节)	取值范围	默认值
字符型	char	2	0-216-1	0

说明：1、字符型的编号中不包含负数。

2、字符型由于存储的是编号的数值，所以可以参与数学运算。

3、字符型可以作为 Java 语言中的无符号整数使用。

4、字符型的默认值是编号为 0 的字符，而不是字符 0

3.5.5 布尔型

布尔型代表逻辑中的成立和不成立。Java 语言中使用关键字 true 代表成立，false 代表不成立。布尔型是存储逻辑值的类型，其实很多程序中都有逻辑值的概念，Java 把逻辑的值用布尔型来进行表达。

布尔型参数表

类型名称	关键字	占用空间(字节)	取值范围	默认值
布尔型	boolean		true 或 false	false

说明： 1、布尔型占用的空间取决于 Java 虚拟机(JVM)的实现，可能是 1 位也可能是 1 个字节。

3.5.6 小结

这里简单的介绍了 8 种基本数据类型的基本特征，在实际的程序设计中，可以根据需要选择对应的类型。

由于 Java 语言是一种强类型的语言，所以在使用数据类型是需要小心。

Java 编程那些事儿 18——变量和常量

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

3.6 变量和常量

在程序中存在大量的数据来代表程序的状态，其中有些数据在程序的运行过程中值会发生改变，有些数据在程序运行过程中值不能发生改变，这些数据在程序中分别被叫做变量和常量。

在实际的程序中，可以根据数据在程序运行中是否发生改变，来选择应该是使用变量代表还是常量代表。

3.6.1 变量

变量代表程序的状态。程序通过改变变量的值来改变整个程序的状态，或者说得更大一些，也就是实现程序的功能逻辑。

为了方便的引用变量的值，在程序中需要为变量设定一个名称，这就是变量名。例如在 2D 游戏程序中，需要代表人物的位置，则需要 2 个变量，一个是 x 坐标，一个是 y 坐标，在程序运行过程中，这两个变量的值会发生改变。

由于 Java 语言是一种强类型的语言，所以变量在使用以前必须首先声明，在程序中声明变量的语法格式如下：

```
数据类型 变量名称;
例如：int x;
```

在该语法格式中，数据类型可以是 Java 语言中任意的类型，包括前面介绍到的基本数据类型以及后续将要介绍的复合数据类型。变量名称是该变量的标识符，需要符合标识符的命名规则，在实际使用中，该名称一般和变量的用途对应，这样便于程序的阅读。数据类型和变量名称之间使用空格进行间隔，空格的个数不限，但是至少需要 1 个。语句使用“;”作为结束。

也可以在声明变量的同时，设定该变量的值，语法格式如下：

数据类型 变量名称 = 值;

例如：int x = 10;

在该语法格式中，前面的语法和上面介绍的内容一致，后续的“=”代表赋值，其中的“值”代表具体的数据。在该语法格式中，要求值的类型需要和声明变量的数据类型一致。

也可以一次声明多个相同类型的变量，语法格式如下：

数据类型 变量名称 1,变量名称 2，…变量名称 n;

例如：int x,y,z;

在该语法格式中，变量名之间使用“,”分隔，这里的变量名称可以有任意多个。

也可以在声明多个变量时对变量进行赋值，语法格式如下：

数据类型 变量名称 1=值 1，变量名称 2=值 2，…变量名称 n=值 n;

例如：int x = 10,y=20,z=40;

也可以在声明变量时，有选择的进行赋值，例如：int x,y=10,z;

以上语法格式中，如果同时声明多个变量，则要求这些变量的类型必须相同，如果声明的变量类型不同，则只需要分开声明即可，例如：

```
int n = 3;
boolean b = true;
char c;
```

在程序中，变量的值代表程序的状态，在程序中可以通过变量名称来引用变量中存储的值，也可以为变量重新赋值。例如：

```
int n = 5;
n = 10;
```

在实际开发过程中，需要声明什么类型的变量，需要声明多少个变量，需要为变量赋什么数值，都根据程序逻辑决定，这里列举的只是表达的格式而已。

3.6.2 常量

常量代表程序运行过程中不能改变的值。

常量在程序运行过程中主要有 2 个作用：

l 代表常数，便于程序的修改

l 增强程序的可读性

常量的语法格式和变量类型，只需要在变量的语法格式前面添加关键字 final 即可。在 Java 编码规范中，要求常量名必须大写。

则常量的语法格式如下：

final 数据类型 常量名称 = 值;

final 数据类型常量名称 1 = 值 1, 常量名称 2 = 值 2,……常量名称 n = 值 n;

例如：

```
final double PI = 3.14;
```

```
    final char MALE=’M’,FEMALE=’F’;
```
在 Java 语法中，常量也可以首先声明，然后再进行赋值，但是只能赋值一次，示例代码如下：
```
        final int UP;
        UP = 1;
```
常量的两种用途对应的示例代码分别如下：
1 代表常数
```
final double PI = 3.14;
int r =5;
double l = 2 * PI * r;
double s = PI * r * r;
```
在该示例代码中，常量 PI 代表数学上的∏值，也就是圆周率，这个是数学上的常数，后续的变量 r 代表半径，l 代表圆的周长，s 代表圆的面积。
则如果需要增加程序计算时的精度，则只需要修改 PI 的值 3.14 为 3.1415926，重新编译程序，则后续的数值自动发生改变，这样使代码容易修改，便于维护。
1 增强程序的可读性
```
int direction;
final int UP = 1;
final int DOWN = 2;
final int LEFT = 3;
final int RIGHT = 4;
```
在该示例代码中，变量 direction 代表方向的值，后续的四个常量 UP、DOWN、LEFT 和 RIGHT 分辨代表上下左右，其数值分别是 1、2、3 和 4，这样在程序阅读时，可以提高程序的可读性。

3.6.3 语句块

在程序中，使用一对大括号{}包含的内容叫做语句块，语句块之间可以互相嵌套，嵌套的层次没有限制，例如：
```
        {
                int a;
        }
```
语句块的嵌套：
```
        {
                int b;
                {
                        char c;
                }
        }
```
以上代码只是演示语法，没有什么逻辑意义。在后续的语法介绍中，还会有语句块的概念，就不再重复介绍了。

3.6.4 变量的作用范围

每个变量都有特定的作用范围，也叫做有效范围或作用域，只能在该范围内使用该变量，否则将提示语法错误。通常情况下，在一个作用范围内部，不能声明名称相同的变量。

变量的作用范围指从变量声明的位置开始，一直到变量声明所在的语句块结束的大括号为止。例如以下代码：

```
{
        {
                int a = 10;
                a = 2;
        }
        char c;
}
```

在该代码中，变量 a 的作用范围即从第三行到第五行，变量 c 的作用范围即从第六行到第七行。

3.6.5 常量的作用范围

常量的作用范围和变量的作用范围规则完全一致。

3.6.6 总结

对于变量和常量的内容就介绍这么，下面是一个完整的代码，可以在 JDK 或 Eclipse 中进行编译和运行。代码文件名为 VariableAndConst.java，示例代码如下：

```
public class VariableAndConst{
        public static void main(String[] args){
                int n = 0;
                char c = ‘A’;
                System.out.println(n);
                n = 10;
                System.out.println(n);
                System.out.println(c);
        }
}
```

说明：在该代码中，System.out.println 的功能是输出变量中存储的值。

Java 编程那些事儿 19—数据类型转换

作者：陈跃峰
出自：**http://blog.csdn.net/mailbomb**

3.7 数据类型转换

Java 语言是一种强类型的语言。强类型的语言有以下几个要求：
l 变量或常量必须有类型
要求声明变量或常量时必须声明类型，而且只能在声明以后才能使用。
l 赋值时类型必须一致
值的类型必须和变量或常量的类型完全一致。
l 运算时类型必须一致
参与运算的数据类型必须一致才能运算。

但是在实际的使用中，经常需要在不同类型的值之间进行操作，这就需要一种新的语法来适应这种需要，这个语法就是数据类型转换。

在数值处理这部分，计算机和现实的逻辑不太一样，对于现实来说，1 和 1.0 没有什么区别，但是对于计算机来说，1 是整数类型，而 1.0 是小数类型，其在内存中的存储方式以及占用的空间都不一样，所以类型转换在计算机内部是必须的。Java 语言中的数据类型转换有两种：
l 自动类型转换
编译器自动完成类型转换，不需要在程序中编写代码。
l 强制类型转换
强制编译器进行类型转换，必须在程序中编写代码。

由于基本数据类型中 boolean 类型不是数字型，所以基本数据类型的转换是出了 boolean 类型以外的其它 7 种类型之间的转换。下面来具体介绍两种类型转换的规则、适用场合以及使用时需要注意的问题。

3.7.1 自动类型转换

自动类型转换，也称隐式类型转换，是指不需要书写代码，由系统自动完成的类型转换。由于实际开发中这样的类型转换很多，所以 Java 语言在设计时，没有为该操作设计语法，而是由 JVM 自动完成。
l 转换规则
从存储范围小的类型到存储范围大的类型。
具体规则为：

byte→short(char)→int→long→float→double

也就是说 byte 类型的变量可以自动转换为 short 类型，示例代码：

```
byte b = 10;
short sh = b;
```

这里在赋值时，JVM 首先将 b 的值转换为 short 类型，然后再赋值给 sh。
在类型转换时可以跳跃。示例代码：

```
byte b1 = 100;
int n = b1;
```

l 注意问题
在整数之间进行类型转换时，数值不发生改变，而将整数类型，特别是比较大的整数类型转换成小数类型时，由于存储方式不同，有可能存在数据精度的损失。

3.7.2 强制类型转换

强制类型转换，也称显式类型转换，是指必须书写代码才能完成的类型转换。该类类型转换很可能存在精度的损失，所以必须书写相应的代码，并且能够忍受该种损失时才进行该类型的转换。
l 转换规则
从存储范围大的类型到存储范围小的类型。
具体规则为：

```
double→float→long→int→short(char)→byte
```

语法格式为：

```
(转换到的类型)需要转换的值
```

示例代码：

```
double d = 3.10;
int n = (int)d;
```

这里将 double 类型的变量 d 强制转换成 int 类型，然后赋值给变量 n。需要说明的是小数强制转换为整数，采用的是“去 1 法”，也就是无条件的舍弃小数点的所有数字，则以上转换出的结果是 3。整数强制转换为整数时取数字的低位，例如 int 类型的变量转换为 byte 类型时，则只去 int 类型的低 8 位(也就是最后一个字节)的值。
示例代码：

```
int n = 123;
byte b = (byte)n;
int m = 1234;
byte b1 = (byte)m;
```

则 b 的值还是 123，而 b1 的值为-46。b1 的计算方法如下：m 的值转换为二进制是10011010010,取该数字低8位的值作为b1的值,则b1的二进制值是11010010，按照机器数的规定，最高位是符号位，1 代表负数，在计算机中负数存储的是补码，则该负数的原码是 10101110，该值就是十进制的-46。
l 注意问题
强制类型转换通常都会存储精度的损失，所以使用时需要谨慎。

3.7.3 其它

后续的复合数据类型，如类和接口等，也存在类似的转换。

Java 编程那些事儿 20—空白、语句结束和注释

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

3.8 空白

在前面的内容中，已经介绍了在编写代码中，单词和单词之间需要使用空格进行间隔，至于空格的数量则不限制。

而实际的编码中，为了使代码的结构清晰，一般需要在代码的前面加入一定数量的空格，例如如下格式：

```
public class Blank{
        public static void main(String[] args){
                int n;
                {
                        n = 10;
                }
                System.out.println(n);
        }
}
```

在该代码中，除了第一行和最后一行外，每行都包含一定数量的空格，这种编码的格式称为代码缩进。

在实际代码中，只要存在包含关系，则代码就应该缩进，语句块是最典型的包含关系之一。说明：在编译时，每行开始的空格都会被忽略掉。

3.9 语句结束

Java 语法中，语句以“;”作为行结束符，例如：

```
int n = 2;
```

通常情况下，代码中每行只写一句代码，但是也可以写多句，例如：

```
int n = 2; byte b = 10;
```

但是一般为了代码结构清楚，只在一行中书写一句代码。

有些时候代码本身比较长，则也可以把一句代码写在多行，而代码语句结束的地方书写一个“;”即可。

在实际代码中，一般大括号开始和结束的地方，以及大部分小括号结束的地方都不需要书写“;”来进行结束。

3.10 注释

注释(comment)是对代码功能和用途的说明。在程序中编写适当的注释，将使程序代码更容易阅读，增强代码的可维护性。

注释在程序编译时都会被忽略，所以注释不会增加class文件的大小。

Java语言中注释的语法有三种：单行注释、多行注释和文档注释。

3.10.1 单行注释

单行注释指只能书写一行的注释。单行注释属于注释中最简单的语法格式，用于对于代码进行简单的说明，例如变量功能等。

单行注释的语法格式为：

```
//注释内容
```

注释以两个斜线开始，该行后续的内容都是注释的内容，注释内容不能换行。

单行注释一般书写在需要说明的代码的上一行，或者该行代码的结束处，示例结构如下：

```
//循环变量
int i = 0;
char sex; //性别
```

3.10.2 多行注释

多行注释指可以书写任意多行的注释。多行注释一般用于说明比较复杂的内容，例如程序逻辑或算法实现原理等。

多行注释的语法格式为：

```
/*
        注释内容
*/
```

注释以一个斜线和一个星号开始，后续是注释内容，注释内容可以书写任意多行，最后注释以一个星号和一个斜线结尾。

很多多行注释在每行以星号开始，这个不是语法必需的。

3.10.3 文档注释

文档注释指可以被提取出来形成程序文档的注释格式，这是Java语言

有特色的注释格式。一般对于程序程序的结构进行说明，例如类、属性、方法和构造方法进行说明，这些概念在后续将详细介绍。

文档注释的语法格式为：

```
/**
        注释内容
 */
```

注释以一个斜线和两个星号开始，后续是注释内容，注释内容可以书写任意多行，最后注释以一个星号和一个斜线结尾。

在后续内容中还将完善该注释的语法格式。

3.10.4 其它

在规范的代码中，一般有 10%-20%的注释，也就是每 100 行代码中包含 10-20 行注释的内容，但是国内很多开发公司都远远达不到这个要求。

另外需要特别注意的是，在实际的项目开发中，在修改代码后，一定要对应的修改注释的内容，保持代码和注释内容的同步。

Java 编程那些事儿 21——算术运算符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

第四章 运算符

计算机，顾名思义，就是计算的机器，所以在编程中，也要进行大量的计算(运算)，运算的符号简称为运算符。

由于计算机可以进行各种运算，所以提供了很多的运算符号，这些运算符号一部分是现实里经常使用的，也有不少是计算机中新增的。

学习运算符，首先要掌握每种运算的运算规则，然后在适当的时候使用对应的运算符。这需要对于运算符最够的熟悉，并具备一定的计算机基础知识。

运算符的种类很多，为了方便学习，以下按照类别来进行介绍。

4.1 算术运算符

算术运算符，也称数学运算符，是指进行算术运算的符号，语法中对应的符号、功能以及说明参看下表

表 4-1 算术运算符

符号	名称	功能说明
+	加	加法运算
-	减	减法运算

* 乘 乘法运算
/ 除 除法运算
% 取余 求两个数字相除的余数

在算术运算符中，+、-、*和/的运算规则和数学基本相同，在四则运算中，乘除优先于加减，计算时按照从左向右的顺序计算，不同的地方在于：
1 程序中乘号不能省略，在数学上可以写 y = 2x，但是程序中必须写成 y=2 * x。
1 运算结果的类型和参与运算的类型中最高的类型一致，例如整数加整数还是整数。影响最大的是除法，整数除整数结果还是整数，例如 10/3 的结果是 3，而不是 3.333。
接着来说说取余运算符，%的功能是取两个数字相除的余数，例如 10%3 表示计算 10 除以 3 的余数，则结果应该是 1。取余运算在编程中的用途也比较大，比较常见的用途有：控制规则变化，控制随机数字的区间等。
算术运算符基本使用的示例代码如下：

```
int n = 3 + 5;
int a = 10;
int b = 20;
int c = a * b;
double d = 100.2;
double d1 = d + a;
```

在算术运算符部分，需要特别注意的一个语法现象是“晋升”。晋升指低于 int 的 3 种数字类型(byte、short 和 char)进行算术运算后，结果会自动提升成 int 类型。示例代码如下：

```
byte b1 = 10;
byte b2 = 20;
byte b3 = b1 + b2; //语法错误，类型不匹配
 int n = b1 + b2;     //或者 byte b3 = (byte)(b1 + b2);
```

在程序中使用算术运算符实现程序中的数学运算，在运算时也可以加入小括号，和数学一样，在程序中也是先计算小括号内部的，然后再计算小括号外部的内容，示例代码如下：

```
int a = 1;
int b = 2;
int c = 3;
int d = c * (a + b) + c;
```

另外一个需要注意的就是，变量在计算时必须被赋值，否则直接报语法错误，例如：

```
int n;
int m = 2 * n;
```

Java 编程那些事儿 22—比较运算符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.2 比较运算符

比较运算符实现数据之间大小或相等的比较。

比较运算符运算的结果是一个 boolean 类型的值，如果比较结果成立则为 true，否则为 false。

Java 语言中比较运算符的表示和功能见下表 4-2。

表 4-2 比较运算符

符号	名称	功能说明
>	大于	比较左侧数字是否大于右侧数字
<	小于	比较左侧数字是否小于右侧数字
>=	大于等于	比较左侧数字是否大于或等于右侧数字
<=	小于等于	比较左侧数字是否小于或等于右侧数字
==	等于	比较左侧数字是否等于右侧数字
!=	不等于	比较左侧数字是否不等于右侧数字

比较运算符的运算规则和现实中的规则一样。需要注意的问题主要有以下几个：

l boolean 类型只能比较相等和不相等，不能比较大小。

l >=的意思是大于或等于，两者成立一个即可，所以 5>=5 成立。

l 在数学上表示的区间[1,10)，也就是数字大于等于 1 同时小于 10，在程序中不能写成如下格式：1<=n<10，这种书写在语法上是错误的，如果需要表达这种区间，则参看 4.3 逻辑运算符实现部分。

l 判断相等的符号是两个等号，而不是一个等号，这个需要特别小心。

比较运算使用的示例代码如下：

```
int a = 10;
boolean b = (a > 3); //该条件成立，则将值 true 赋值给变量 b
boolean c = (b == true); //条件成立，结果为 true
```

在实际代码中，数值、变量以及运算结果都可以直接参与比较，只是程序中为了增强可读性，有些时候需要将比较分开进行书写。

比较运算符是程序设计中实现数据比较的基础，也是很多逻辑实现的基础，在程序逻辑中，经常通过比较一定的条件，来判断后续的程序该如何执行。

Java 编程那些事儿 23—逻辑运算符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.3 逻辑运算符

逻辑运算符是指进行逻辑运算的符号。逻辑运算主要包括与(and)、或(or)和非(not)三种，在程序中主要用来连接多个条件，从而形成更加复杂的条件。

逻辑运算符的运算结果是 boolean 类型。

参与逻辑运算的数据也必须是 boolean 类型。

关于逻辑运算符的种类和说明参看表 4-3。

表 4-3 逻辑运算符

符号	名称	功能说明
&&	逻辑与	两个条件同时为 true 才为 true，否则为 false
||	逻辑或	两个条件有一个为 true 则为 true，否则为 false
!	逻辑非	只操作一个数据，对数据取反

逻辑运算符使用示例代码：

```
boolean b1 = true;
boolean b2 = false;
boolean b3 = b1 && b2; //则 b3 的值是 false
b3 = b1 || b2;         //则 b3 的值是 true
b3 = !b1;              //则 b3 的值是 false
```

在实际程序中，可以根据逻辑的需要使用对应的逻辑运算符号。实际使用示例：

1 表示变量 n 是否属于[0,10)区间

```
int n = 4;
                    boolean b = (n >=0) && (n < 10);
```

对于变量 n 来说，只要 n 同时满足大于等于零，且小于 10，则位于[0,10)区间，由于程序中无法书写 0<=n<10 这样的条件，则必须通过逻辑运算符进行连接。

1 表示变量 n 不属于[0,10)区间

一种写法是：

```
int n = 4;
boolean b = !((n >= 0) && (n < 10));
```

这里，对于属于该区间的条件取反，则可以获得不属于该区间的条件。

另一种写法是：

```
int n = 4;
boolean b = (n < 0) || (n >= 10);
```

这里做了一个简单的变通，如果变量 n 不属于该区间，则在数学上只需要满足 n

小于 0 或者 n 大于等于 10 的任何一个条件即可，这样的或者关系在程序中实现时使用逻辑或实现。

在程序设计中，根据逻辑需要，使用对应的逻辑运算符，可以实现相对比较复杂的组合条件，从而实现对应程序的功能。

最后说一下&&和&的区别，其实在进行逻辑与运算时，既可以使用&&也可以使用&，在功能上本身没有区别。两者区别的位置在，对于&来说，如果左侧条件为false，也会计算右侧条件的值，而对于&&来说，如果左侧的条件为false，则不计算右侧的条件，这种现象被称作短路现象。

示例代码：

```
int n = -1;
boolean b1 = (n >= 0) && (n < 10);
boolean b2 = (n >= 0) & (n < 10);
```

则对于第二行代码来说，两个条件都将被计算，而对于第三行代码来说，因为 n >= 0 这个条件不成立，则 n < 10 根本不会被执行。当然，两者得到的最终结果是一样的。

对于现在的代码来说，区别不大，但是如果后续的条件是一个方法(方法的概念后续将介绍到)，则将影响程序逻辑。

验证&和&&功能的示例代码如下：

```
public class Test{
        public static void main(String[] args){
                int n = 10;
                  boolean b = (n < 8) && ((n =
1) != 0);
                int m = 20;
                  boolean b1 = (m < 8) & ((m =
1) != 0);

                System.out.println(n);
                System.out.println(m);
        }
}
```

最后编辑：2008-5-26 修正&和&&的错误

Java 编程那些事儿 24—赋值运算符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.4 赋值运算符

赋值运算符是指为变量或常量指定数值的符号。最基本的赋值运算符是“=”。

由于Java语言是强类型的语言，所以赋值时要求类型必须匹配，如果类型不匹配时需要能自动转换为对应的类型，否则将报语法错误。示例代码：

```
byte b = 12;        //类型匹配，直接赋值
int n = 10;         //类型匹配，直接赋值
double d = 100; //类型不匹配，系统首先自动将100转换成100.0，然后赋值
char c = -100;    //类型不匹配，无法自动转换，语法错误
```

需要强调的是，只能为变量和常量赋值，不能为运算式赋值，例如：

```
int a = 10;
int b = 2;
a + b = 100; //不能为运算式a + b赋值，语法错误
```

常量只能赋值一次，否则也将出现语法错误，例如：

```
final int N = 10;
N = 20; //常量只能赋值一次，语法错误
```

在基本的赋值运算符基础上，可以组合算术运算符，以及后续将学习的位运算符，从而组成复合赋值运算符。赋值运算符和算术运算符组成的复合赋值运算符如下表4-4所示。

表4-4 复合赋值运算符

符号	名称	功能说明
+=	加等	把变量加上右侧的值然后再赋值给自身
-=	减等	把变量减去右侧的值然后再赋值给自身
*=	乘等	把变量乘以右侧的值然后再赋值给自身
/=	除等	把变量除以右侧的值然后再赋值给自身
%=	取余等	把变量和右侧的值取余然后再赋值给自身

实际使用示例：

```
int n = 2;
n += 3;
```

说明：计算以前n的值是2，也就是把n + 3的值，也就是5再赋值给n，经过运算以后n的值为5，因为该代码只执行一次，所以不会导致循环。

依次类推，其它的复合赋值运算符也是这样：

```
int n = 10;
n -= 2; //则n的值是8
n *= 3; //因为n的初值是8，则n运算后的结果是24
n /= 5; //因为n的初值是24，则n运算后的值是4
```

注意：复合赋值运算不会改变结果的类型，所以在有些时候运算在逻

辑上会出现错误，但是符合计算中数值的表述。例如：

```
byte b = 127;
b += 1;
System.out.println(b);
```

根据前面的介绍，byte 类型的取值区间是-128~127，由于复合赋值运算符不改变结果的类型，则导致结果是-128，而不是 128。原因如下：

l byte 类型值 127 的机器数是 01111111，0 表示正数，后续的数值表示 127

l 该数值加 1 后，得到的数值是 10000000，二进制加法

l 而 10000000 在 byte 类型中恰好是-128 的机器数表示形式

其它类型的符合运算符也可能存在类似的情况，使用时需要注意。

Java 编程那些事儿 25—位运算符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.5 二进制运算符

由于计算机内部的数据都以二进制的形式存在，所以在 Java 语言中提供了直接操作二进制的运算符，这就是下面要讲解的位运算符和移位运算符。

使用二进制的运算符，可以直接在二进制的基础上对数字进行操作，执行的效率比一般的数学运算符高的多，该类运算符大量适用于网络编程、硬件编程等领域。

二进制运算符在数学上的意义比较有限。

在 Java 代码中，直接书写和输出的数值默认是十进制，Java 代码中无法直接书写二进制数值，但是可以书写八进制和十六进制数字，八进制以数字 0 开头，例如 016，十六进制以数字 0 和 x 开头，例如 0x12,0xaf 等等。

在计算二进制运算时，Java 语言的执行环境(JRE)首先将十进制的数字转换为二进制，然后进行运算。如果输出结果的值，则数字会被转换成十六进制进行输出。

需要注意的是：

1、正数的机器数是原码，负数的机器数是补码，计算时需要小心。关于二进制和补码的计算可以参看《Java 编程那些事儿 7——进制的概念》和《Java 编程那些事儿 8——计算机内部的数据表达》。

2、整数型的计算结果都是 int 型，而不管是对 byte 还是 short 进行二进制运算。

4.5.1 位运算符

Java 语言中的位运算符主要有 4 种：&(位与)、|(位或)、^(异或)和~(按位取反)，下面依次介绍运算规则和使用示例。

1 &(AND)

运算规则：参与运算的数字，低位对齐，高位不足的补零，对应的二进制位都为 1，则运算结果为 1，否则为 0。

适用场合：屏蔽数字中某一位或某些位。因为任何数和 0 与都是 0。

示例代码：

```
int a = 4;
int b = 10;
int c = a & b;
```

计算过程：

4 的二进制形式为 0000 0000 0000 0000 0000 0000 0000 0100

10 的二进制形式为 0000 0000 0000 0000 0000 0000 0000 1010

按照计算规则,结果为 0000 0000 0000 0000 0000 0000 0000 0000

这个数字转换为十进制就是数字 0

1 | (OR)

运算规则：参与运算的数字，低位对齐，高位不足的补零，对应的二进制位有一个为 1 则为 1，否则为 0。

适用场合：将数字中某一位或某些位修改成 1。因为 1 和任何数或都是 1。

示例代码：

```
int a = 4;
int b = -10;
int c = a | b;
```

计算过程：

4 的二进制形式为 0000 0000 0000 0000 0000 0000 0000 0100

10 的二进制形式为 1111 1111 1111 1111 1111 1111 1111 0110

按照计算规则,结果为 1111 1111 1111 1111 1111 1111 1111 0110

这个二进制数转换为十进制就是数字-10。

1 ^(XOR)

运算规则：参与运算的数字，低位对齐，高位不足的补零，对应的二进制位相同为零，不相同为 1。

适用场合：判断数字对应的位是否相同。

示例代码：

```
int a = 4;
int b = 10;
int c = a ^ b;
```

计算过程：

4 的二进制形式为 0000 0000 0000 0000 0000 0000 0000 0100

10 的二进制形式为 0000 0000 0000 0000 0000 0000 0000 1010

按照计算规则,结果为 0000 0000 0000 0000 0000 0000 0000 1110
这个数字转换为十进制就是数字 14
1 ~(NOT)
运算规则：只操作一个数字，将该数字中为 1 的位变成 0，为 0 的位变成 1。
适用场合：反转数字的内容
示例代码：

```
int a = 4;
int c = ~a;
```

计算过程：
4 的二进制形式为 0000 0000 0000 0000 0000 0000 0000 0100
按照计算规则,结果为 1111 1111 1111 1111 1111 1111 1111 1011
这个数字转换为十进制就是数字-5。

其实位运算和实际的应该实现保持一致，也就是提供的电路级运算符号，每种运算符都有对应的电路实现。

实际使用简单示例：
1 把任意数字转换为正数
假设 n 是一个任意的整数，则把 n 转换为正数的代码为：

```
int m = n & 0x7fffffff;
```

1 判断任意数字倒数第三位的值是否为 1
假设 n 是一个任意的整数，则判断的代码为：

```
int m = n & 0x4;
boolean b = (m != 0);
```

1 将任意数字倒数第四位置为 1
假设 n 是一个任意的整数，则代码为：

```
int m = n | 0x8;
```

Java 编程那些事儿 26—移位运算符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.5.2 移位运算符

移位运算符就是在二进制的基础上对数字进行平移。按照平移的方向和填充数字的规则分为三种：<<(左移)、>>(带符号右移)和>>>(无符号右移)。

在移位运算时，byte、short 和 char 类型移位后的结果会变成 int 类型，对于 byte、short、char 和 int 进行移位时，规定实际移动的次数是移动次数和 32 的余数，也就是移位 33 次和移位 1 次得到的结果相同。移动 long 型的数值时，规定实际移动的次数是移动次数和 64 的余数，也就是移动 66 次和移动 2 次得到的结果相同。

三种移位运算符的移动规则和使用如下所示：

1 <<
运算规则：
按二进制形式把所有的数字向左移动对应的位数，高位移出(舍弃)，低位的空位补零。
语法格式：
需要移位的数字 << 移位的次数
例如： 3 << 2，则是将数字 3 左移 2 位
计算过程：
3 << 2
首先把 3 转换为二进制数字 0000 0000 0000 0000 0000 0000 0000 0011，然后把该数字高位(左侧)的两个零移出，其他的数字都朝左平移 2 位，最后在低位(右侧)的两个空位补零。则得到的最终结果是 0000 0000 0000 0000 0000 0000 0000 1100，则转换为十进制是 12。
数学意义：
在数字没有溢出的前提下，对于正数和负数，左移一位都相当于乘以 2 的 1 次方，左移 n 位就相当于乘以 2 的 n 次方。
1 >>
运算规则：
按二进制形式把所有的数字向右移动对应巍峨位数，低位移出(舍弃)，高位的空位补符号位，即正数补零，负数补 1。
语法格式：
需要移位的数字 >> 移位的次数
例如 11 >> 2，则是将数字 11 右移 2 位
计算过程：
11 的二进制形式为：0000 0000 0000 0000 0000 0000 0000 1011，然后把低位的最后两个数字移出，因为该数字是正数，所以在高位补零。则得到的最终结果是 0000 0000 0000 0000 0000 0000 0000 0010。转换为十进制是 3。
数学意义：
右移一位相当于除 2，右移 n 位相当于除以 2 的 n 次方。
1 >>>
运算规则：
按二进制形式把所有的数字向右移动对应巍峨位数，低位移出(舍弃)，高位的空位补零。对于正数来说和带符号右移相同，对于负数来说不同。
其他结构和>>相似。

4.5.3 小结

二进制运算符，包括位运算符和移位运算符，使程序员可以在二进制基础上操作数字，可以更有效的进行运算，并且可以以二进制的形式存储和转换数据，是实现网络协议解析以及加密等算法的基础。

但是，在实际使用中，为了使代码可读性强，还是大量使用一般的算术运算符来进行数字运算。

Java 编程那些事儿 27—其它运算符

郑州游戏学院 陈跃峰
出自：**http://blog.csdn.net/mailbomb**

4.6 其它运算符

对于无法归类，或者单独占一类的运算符，将在下面进行介绍。

l ++、--

这两个运算符是程序中的递增和递减运算符。其意义参照以下示例代码：

```
int n = 0;
n++; // n = n + 1
System.out.println(n);
```

n++的意义就是对原来变量 n 的值加 1 以后再赋值给自身，因为原来变量 n 的值是 0，加 1 以后则变成 1。

同理，递减运算符的意义也是这样，示例代码：

```
int m = 0;
m--;
System.out.println(m);
```

m—的意义就是对原来变量 m 的值减 1 以后再赋值给自身，则 m 的值变成-1。

需要注意的是++和—只能操作变量，而不能操作其他的内容，以下使用都是错误的：

```
int a = 0;
int b = 0;
(a + b)++;   //错误
final int M = 1;
M++;      //错误
5++;      //错误
```

在实际书写时，++和—既可以写在变量的前面，也可以写在变量的后面，例如：

```
int k = 0;
k++;
++k;
```

同理，--也可以这样，那么这些在实际使用中有什么区别呢？其实对于变量的值来说，没有区别，也就是++无论写后面还是写前面，变量的值肯定增加 1，--无论写在后面还是前面，变量的值都减 1。其最大的区别在于整个式子的值，如 n++，规则如下：

1）++或—写在变量前面，则该式子的值等于变量变化以前的值。

2）++或—写在变量后面，则该式子的值等于变量变化以后的值。

示例代码如下：

```
int n = 1;
int m= 1;
```

```
n++;     //n 的值变为 2
++m;    //m 的值变为 2
int k = n++; //n 的初始值是 2，则 n++的值是 2，结果 n 的值为 3，k 的值为 2
int j = ++m; //m 的初始值是 2，则++m 的值是 3，结果 m 的值是 3，j 的值为 3
```

同理，--也是这样。

下面是一个稍微综合点的示例：

```
int a = 0;
int b = 0;
a = b++; //a 为 0，b 为 1
a = ++b; //a 为 2，b 为 2
b = a++; //a 为 3，b 为 2
a = ++b; //a 为 3，b 为 3
```

说明：注释部分为对应行代码运行以后，a 和 b 的值。

在程序开发中，可以使用该区别简化代码的书写，但是不推荐这样做，因为这样将增加阅读代码的难度。

l +、-

前面介绍过加减运算符，其实+、-还有另外一个意义，也就是代表正负，通常情况下正号可以省略，而负号可以和数值、变量以及运算式进行结合，示例代码如下：

```
int a = 0;
int b = 1;
int c = -5;
c = -a;
c = -(a + b);
```

l ？：

这个运算符称为条件运算符，其作用是根据判断的结果获得对应的值，语法格式如下：

```
条件式 ？值 1 ： 值 2
```

语法要求条件式部分必须是 boolean 类型，可以是 boolean 值，也可以是 boolean 变量，或者是关系运算符或逻辑运算符形成的式子，值 1 和值 2 必须能够转换成相同的类型。

功能说明：如果条件式的结果是 true，则整个式子的值取值 1 的值，否则取值 2 的值。示例代码如下：

```
int x = 10;
int y = 20;
int max = x > y ? x : y; //因为 x 大于 y，则取变量 x 的值，然后赋值给 max
int a = -10;
int abs = a > 0 ? a : -a; //实现求绝对值得功能
```

l ()

括号，也是运算符的一种，作用是可以让括号内部的计算首先进行，这个和数学上一致，只是程序代码中可以使用这个组合任意的合法运算式。示例代码为：

```
int a = 1 + 2 * 3;
int a = (1 + 2) * 3; //和以上代码的运行结果不一致
```

其实每个运算符都有自己的优先级，使用括号可以提升对应式子的优先级。关于运算符优先级的概念，后续将进行介绍。

Java 编程那些事儿 28—运算符优先级

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.7 运算符优先级

在实际的开发中，可能在一个运算符中出现多个运算符，那么计算时，就按照优先级级别的高低进行计算，级别高的运算符先运算，级别低的运算符后计算，具体运算符的优先级见下表：

运算符优先级表

优先级	运算符	结合性
1	() [] .	从左到右
2	! +(正) -(负) ~ ++ --	从右向左
3	* / %	从左向右
4	+(加) -(减)	从左向右
5	<< >> >>>	从左向右
6	< <= > >= instanceof	从左向右
7	== !=	从左向右
8	&(按位与)	从左向右
9	^	从左向右
10	|	从左向右
11	&&	从左向右
12	||	从左向右
13	?:	从右向左
14	= += -= *= /= %= &= |= ^= ~= <<= >>= >>>=	从右向左

说明：

1、 该表中优先级按照从高到低的顺序书写，也就是优先级为 1 的优先级最高，优先级 14 的优先级最低。

2、 结合性是指运算符结合的顺序，通常都是从左到右。从右向左的运算符最典型的就是负号，例如 3+-4，则意义为 3 加-4，符号首先和运算符右侧的内容结合。

3、 instanceof 作用是判断对象是否为某个类或接口类型，后续有详细介绍。
4、 注意区分正负号和加减号，以及按位与和逻辑与的区别
其实在实际的开发中，不需要去记忆运算符的优先级别，也不要刻意的使用运算符的优先级别，对于不清楚优先级的地方使用小括号去进行替代，示例代码：

```
int m = 12;
int n = m << 1 + 2;
int n = m << (1 + 2); //这样更直观
```

这样书写代码，更方便编写代码，也便于代码的阅读和维护。

Java 编程那些事儿 29—表达式

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

4.8 表达式

由运算符和变量、常数或常量组成的式子称为表达式。例如 2+3,a*b 等。表达式是组成程序的基本单位，也是程序运算时的基本单位。

在程序代码内部，每个表达式都有自己对应的数据类型，具体参看下表：

表达式结果类型

序号	运算符	结果类型
1	算术运算符	数字型
	位运算符	
	移位运算符	
	递增、递减运算符	
2	比较运算符	布尔型
	逻辑运算符	
3	赋值运算符	和变量类型一致
4	条件运算符	和两个值中类型高的一致

对于序号 1 和 2 的运算符组成的表达式，由于比较直观，就不再单独举例了，对于 3 和 4 说明如下：

```
int n = 10;
int m = 10;
 n =( m = 10);    //则表达式 m=10 的类型是变量 m 的类型，也是 m 的值
boolean b = false;
boolean b1 = true;
 b = (b1 = true); //则表达式 b1 = true 的类型是布尔型，值是 true
```

```
double d;
```
d = 10 > 0 ? 1.0 : 2; //由于 1.0 是 double 型，2 是整数型，则表达式的类型是 double

对于由多个运算符组成的表达式，其最终的类型由最后一个运算符决定。

在实际的程序代码中，大部分的表达式不能单独成为代码中的一行，否则程序会提示语法错误，例如：

```
int a = 10;
int b = 20;
a + b;     //不能单独成行
```

在表达式中，能够单独成行的运算符包括赋值运算符和递增、递减运算符。

4.9 总结

本部分系统的讲解了 Java 语言中运算符的知识，并且介绍了实际使用过程中需要注意的问题，在学习时需要熟记每种运算符的作用，然后在实际项目中根据需要使用对应的运算符来实现程序的功能。

Java 编程那些事儿 30—流程控制基础

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

第五章 流程控制

流程就是指程序执行的顺序，流程控制就是指通过控制程序执行的顺序实现要求的功能。流程控制部分是程序中语法和逻辑的结合，也是程序中最灵活的部分，是判断一个程序员能力的主要方面。

众所周知，算法是程序逻辑的核心，而算法的绝大部分代码都是流程控制实现的。

流程控制就是将程序员解决问题的思路，也就是解决问题的步骤，使用程序设计语言的语法格式描述出来的过程。

5.1 流程控制基础

流程控制在程序语言实现时，通过三类控制语句进行实现：

1 顺序语句

顺序语句是一种自然的语句，没有特定的语法格式，总体的执行流程就是先写的

代码先执行，后写的代码后执行。
使用顺序语句时，需要特别注意代码的执行顺序。
1 条件语句
条件语句，又称判断语句或分支语句，在程序中有对应的语法格式，执行流程是根据条件是否成立，决定代码是否执行。如果条件成立，也就是 true，则执行对应的代码，否则不执行。
使用条件语句时，需要特别注意条件是否准确以及功能部分的书写。
1 循环语句
循环语句是一种计算机内部特有的语句，指重复执行的代码，在程序中有对应的语法格式，执行的流程是每次判断条件是否成立，然后决定是否重复执行。循环语句是流程控制部分最灵活、最复杂，也是功能最强大的一类语句。
使用循环语句时，需要注意循环条件以及循环功能部分的书写。

在程序中，任意复杂的流程，都只通过以上三类语句的组合、嵌套来进行实现，所以在学习流程控制时，首先需要对于三类语句有个基础的认识，然后熟悉相关的语法，进行针对的练习，最后灵活使用这三类语句解决实际的问题。、

另外，需要强调的是，根据逻辑的需要，各种语句可以任意进行嵌套，也就是在一个语句的内部书写其它的语句，这样可以实现更加复杂的逻辑。

后续的讲解也按照该顺序进行，本部分会附带部分流程控制的练习。

5.2 顺序语句

顺序语句是流程控制语句中最简单的一类语句，在代码中没有语法格式，只需要根据逻辑的先后顺序依次书写即可，所以在实际书写以前，首先要思考清楚对应的逻辑顺序，然后再开始对应的书写。

需要注意的是，在实际的代码中，有些时候代码书写的先后会影响程序的逻辑，例如如下输出的代码。

示例代码 1：

```
int n = 10;
n += 2;
System.out.println(n);
```

示例代码 2：

```
int n = 10;
System.out.println(n);
n += 2;
```

则由于代码书写的顺序不同，即使相同的代码，示例代码 1 中输出的值是 12，而示例代码 2 中输出的值是 10。类似的逻辑在实际的项目中也有很多。

Java 编程那些事儿 31——if 语句语法(1)

郑州游戏学院 陈跃峰

出自：**http://blog.csdn.net/mailbomb**

5.3 条件语句

条件语句，是程序中根据条件是否成立进行选择执行的一类语句，这类语句在实际使用中,难点在于如何准确的抽象条件。例如实现程序登录功能时，如果用户名和密码正确，则进入系统，否则弹出“密码错误”这样的提示框等。

本部分对于条件语句的介绍，重点在于语法讲解和基本的使用，更详细的使用参看后续的综合示例部分。

在 Java 语言中，条件语句主要有两类语法：if 语句和 switch 语句。

5.3.1 if 语句

if 关键字中文意思是如果，其细致的语法归纳来说总共有三种：if 语句、if-else 语句和 if-else if-else 语句，下面分别进行介绍。

5.3.1.1 if 语句

该类语句的语法格式为：

```
if(条件表达式)
    功能代码;
```

语法说明：if 是该语句中的关键字，后续紧跟一对小括号，该对小括号任何时候不能省略，小括号的内部是具体的条件，语法上要求该表达式结果为 boolean 类型。后续为功能的代码，也就是当条件成立时执行的代码，在程序书写时，一般为了直观的表达包含关系，功能代码一般需要缩进。

需要特别注意的是：

1、 这里的功能代码只能是一行，关于多行结构的功能代码，后续将说明。

2、 if(条件表达式)后续一般不书写分号

if 语句的执行流程为：如果条件表达式成立，则执行功能代码，如果条件表达式不成立，则不执行后续的功能代码。

示例代码：

```
int a = 10;
if(a >= 0)
        System.out.println(“a 是正数”);
if( a % 2 == 0)
        System.out.println(“a 是偶数”);
```

在该示例代码中，第一个条件是判断变量 a 的值是否大于等于零，如果该条件成立则执行输出，第二个条件是判断变量 a 是否为偶数，如果成立也输出。

注意以下代码的执行流程：

```
int m = 20;
if( m > 20)
```

```
        m += 20;
        System.out.println(m);
```

按照前面的语法格式说明，只有 m+=20;这行代码属于功能代码，而后续的输出语句和前面的条件形成顺序结构，所以该程序执行以后输出的结果为 20。

如果当条件成立时，需要执行的语句有多句，则可以使用语句块来进行表述，语法格式如下：

```
if(条件表达式){
        功能代码块;
}
```

使用这种语法格式，使用一个代码块来代替前面的功能代码，这样可以在代码块内部书写任意多行的代码，而且也使整个程序的逻辑比较清楚，所以在实际的代码编写中推荐使用该种逻辑。

Java 编程那些事儿 32—if 语句语法(2)

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.3.1.2 if-else 语句

if-else 语句实现了封闭的条件，在程序中使用的更加常见。其中 else 关键字的作用是“否则”，即条件不成立的情况。

if-else 语句的语法格式如下：

```
if(条件表达式)
        功能代码 1;
else
        功能代码 2;
```

语法说明：其中前面的部分和 if 语句一样，else 部分后面是功能的代码，按照该语法格式，功能代码只能有一句。

执行顺序：如果条件成立，则执行 if 语句中的功能代码 1，否则执行 else 中的功能代码 2。

示例代码为：

```
int n = 12;
if(n % 2 != 0)
        System.out.println(“n 是奇数”);
else
        System.out.println(“n 不是奇数”);
```

则因为 n%2 的值是 0，条件不成立，则执行 else 语句的代码，程序输出“n 不是奇数”。

在实际使用时，为了结构清楚，以及可以在功能代码部分书写多行代码，一般把功能代码部分使用代码块，则语法格式为：

```
if(条件表达式){
```

```
        功能代码块
    }else{
        功能代码块
    }
```

当程序中有多个 if 时，else 语句和最近的 if 匹配。示例代码：

```
    if(条件 1)
        功能代码 1;
    if(条件 2)
        功能代码 2;
    else
        功能代码 3;
```

则这里的 else 语句和条件 2 对应的 if 语句匹配，前面的条件 1 是一个独立的语句。在实际代码中，可以使用大括号使整个程序的结构更加清楚。

对于 if-else 语句来说，因为 if 的条件和 else 的条件是互斥的，所以在实际执行中，只有一个语句中的功能代码会得到执行。

在实际开发中，有些公司在书写条件时，即使 else 语句中不书写代码，也要求必须书写 else，这样可以让条件封闭。这个不是语法上必须的。

Java 编程那些事儿 33——if 语句语法(3)

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.3.1.3 if-else if-else 语句

在现实中，有些时候的条件不是一个，而是一组相关的条件，例如将阿拉伯数字转换为中文大写，根据分数转换为对应的等级等，都是多条件的结构，在程序中为了避免写多个 if 语句的结构，提供了一类专门的多分支语句，这就是 if-else if-else 语句。

if-else if-else 语句的语法格式为：

```
    if(条件 1)
        功能代码 1;
    else if(条件 2)
        功能代码 2;
    else if(条件 3)
        功能代码 3;
    ……
    else
        功能代码;
```

语法说明：

1、 else if 是 else 和 if 两个关键字，中间使用空格进行间隔。
2、 条件 1 到条件 n 都是 boolean 类型
3、 else if 语句可以有任意多句

4、 最后的 else 语句为可选

5、 如果功能代码部分不是语句块，也就是不用大括号，就只能写一句。

执行流程：当条件 1 成立时，则执行功能代码 1；当条件 1 不成立且条件 2 成立时，则执行功能代码 2；如果条件 1、条件 2 都不成立且条件 3 成立，则执行功能代码 3，依次类推，如果所有条件都不成立，则执行 else 语句的功能代码。其执行流程的流程图如上所示。

下面是一个实现根据月份的值，输出该月份包含的日期数，2 月份全部输出 28，不考虑闰年的示例代码：

```
int month = 3;
int days = 0;   //日期数
if(month == 1){
        days = 31;
}else if(month == 2){
        days = 28;
} else if(month == 3){
        days = 31;
} else if(month == 4){
        days = 30;
} else if(month == 5){
        days = 31;
} else if(month == 6){
        days = 30;
} else if(month == 7){
        days = 31;
} else if(month == 8){
        days = 31;
} else if(month == 9){
        days = 30;
} else if(month == 10){
        days = 31;
} else if(month == 11){
        days = 30;
} else if(month == 12){
        days = 31;
}
System.out.println(days);
```

再来看一个示例代码，该代码的功能是实现将百分制的成绩转换为 A、B、C、D 和 E，代码如下：

```
int score = 87;
if(score >= 90){
        System.out.println( ‘A’ );
} else if(score >= 80){
```

```
            System.out.println( ‘B’ );
        } else if(score >= 70){
            System.out.println( ‘C’ );
        } else if(score >= 60){
            System.out.println( ‘D’ );
        } else{
            System.out.println( ‘E’ );
        }
```

从该代码中可知，每个 else if 语句在书写时是有顺序的，在实际书写时，必须按照逻辑上的顺序进行书写，否则将出现逻辑错误。

if-else if-else 语句是 Java 语言中提供的一个多分支条件语句，但是在判断某些问题时，会书写的比较麻烦，所以在语法中提供了另外一个语句——switch 语句来更好的实现多分支语句的判别。

Java 编程那些事儿 34——switch 语句语法

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.3.2 switch 语句

switch 关键字的中文意思是开关、转换的意思，switch 语句在条件语句中特别适合做一组变量相等的判断，在结构上比 if 语句要清晰很多。

switch 语句的语法格式为：

```
switch(表达式){
        case 值 1:
                功能代码 1;
                [break;]
        case 值 2:
                功能代码 2;
                [break;]
        ……
        default:
                功能代码 1;
                [break;]
}
```

语法说明：

1、 表达式的类型只能为 byte、short、char 和 int 这 4 种之一。
2、 值 1、值 2…值 n 只能为常数或常量，不能为变量。
3、 功能代码部分可以写任意多句。
4、 break 关键字的意思是中断，指结束 switch 语句，break 语句为可选。

5、 case 语句可以有任意多句，是标号语句。
6、 default 语句可以写在 switch 语句中的任意位置，功能类似于 if 语句中的 else。
执行流程：当表达式的值和对应 case 语句后的值相同时，既从该位置开始向下执行，一直执行到 switch 语句的结束，在执行中，如果遇到 break 语句，则结束 switch 语句的执行。
则在 if-else if-else 语句中，根据月份获得每个月的天数，不考虑闰年，的示例代码如下：

```
        int month = 10;
        int days = 0;
        switch(month){
                case 1:
                        days = 31;
                        break;
case 2:
                        days = 28;
                        break;
case 3:
                        days = 31;
                        break;
case 4:
                        days = 30;
                        break;
case 5:
                        days = 31;
                        break;
case 6:
                        days = 30;
                        break;
case 7:
                        days = 31;
                        break;
case 8:
                        days = 31;
                        break;
case 9:
                        days = 30;
                        break;
case 10:
                        days = 31;
                        break;
case 11:
```

```
                        days = 30;
                        break;
                case 12:
                        days = 31;
                        break;
        }
        System.out.println(days);
```

根据 switch 语句的语法，该代码也可以简化为如下格式：

```
        int month = 10;
        int days = 0;
        switch(month){
                case 2:
                        days = 28;
                        break;
                case 4:
                case 6:
                case 9:
                case 11:
                        days = 30;
                        break;
                        default:
                                days = 31;
        }
        System.out.println(days);
```

代码说明：因为 switch 语句每次比较的是相等关系，所以可以把功能相同的 case 语句合并起来，而且可以把其他的条件合并到 default 语句中，这样可以简化 case 语句的书写。该代码的结构比最初的代码简洁很多了。

虽然在语法上 switch 只能比较相等的结构，其实某些区间的判别也可以通过一定的变换使用 switch 语句进行实现。例如 if-else if-else 语句示例中的分数转换的示例，则分数的区间位于 0-100 之间，如果一个一个的去比较，case 语句的数量会比较多，所以可以做一个简单的数字变换，只比较分数的十位及以上数字，这样数字的区间就缩小到了 0-10，则实现的代码如下：

```
        int score = 87;
        switch(score / 10){
                case 10:
                case 9:
                        System.out.println( 'A' );
                        break;
                case 8:
                        System.out.println( 'B' );
                        break;
                case 7:
```

```
                System.out.println( 'C' );
                break;
            case 6:
                System.out.println( 'D' );
                break;
            default:
                System.out.println( 'E' );
        }
```

当然，switch 语句不是很适合进行区间的判别，更多的区间判别一般还是使用 if-else if-else 语句进行实现。

5.3.3 小结

if 语句可以实现程序中所有的条件，switch 语句特别适合一系列点相等的判别，结构显得比较清晰，而且执行速度比 if 语句要稍微快一些，在实际的代码中，可以根据需要来使用对应的语句实现程序要求的逻辑功能。

Java 编程那些事儿 35——while 语句语法

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.4 循环语句

循环语句在程序设计中用来描述有规则重复的流程。在实际的程序中，存在很多需要重复执行的流程，为了简化这些重复的执行流程，在程序设计语言中新增了该类语句。

在学习循环语句时，最重要的就是发现流程的规律，然后再用程序设计语言将该规律描述出来，从来实现程序要求的流程。

循环语句是流程控制中最复杂，也是最有用、最难掌握的语句，在最初接触时，首先要熟悉基本的语法，然后需要能够快速观察出流程的规律，这个观察能力需要依靠大量的阅读和编写程序进行培养，这就是基本的逻辑思维，然后将该规律描述出来即可。所以在学习循环语句时，学习语法只是基本的内容，更多的是培养自己观察规律的能力，这个才是真正学习循环语句时的难点，也是重点。

本节主要讲述循环语句的三种语法格式：while 语句、do-while 语句和 for 语句。

5.4.1 while 语句

while 关键字的中文意思是“当……的时候”，也就是当条件成立时循环执行对应的代码。while 语句是循环语句中基本的结构，语法格式比较简单。

while 语句语法格式：

```
while(循环条件)
        循环体;
```

为了结构清楚，并且使循环体部分可以书写多行代码，一般把循环体处理成代码块，则语法格式变为：

```
while(循环条件){
        循环体;
}
```

语法说明：和 if 语句类似，如果不是用代码块的结构，则只有 while 后面的第一个语句是循环体语句。在该语法中，要求循环条件的类型为 boolean 类型，指循环成立的条件，循环体部分则是需要重复执行的代码。

执行流程：在执行 while 语句时，首先判断循环条件，如果循环条件为 false，则直接执行 while 语句后续的代码，如果循环条件为 true，则执行循环体代码，然后再判断循环条件，一直到循环条件不成立为止。

下面结合具体的示例来演示一下 while 语句的基本使用。首先我们来实现一个无限循环，也称死循环，具体代码如下：

```
while(true){
        System.out.println( ‘a’ );
}
```

下面讲解一下该 while 语句的执行顺序，首先判断 while 语句的循环条件，条件成立，则执行循环体的代码，输出字符 a，然后再判别循环条件，条件成立，继续执行循环体代码，输出 a，再判断循环条件……，依次类推，因为循环条件一直成立，所以该程序的功能是一直输出 a，永不停止。

说明：在控制台下执行死循环的程序，可以按 Ctrl+C 结束循环，在 Eclipse 中运行死循环的程序，可以选择执行窗口中的红色按钮“Terminate”结束程序。

下面是使用 while 语句输出 0-9 这 10 个数字，程序实现的原理是使用一个变量代表 0-9 之间的数字，每次输出该变量的值，每次对该变量的值加 1。变量的值从 0 开始，只要小于数字 10 就执行该循环。具体的示例代码如下：

```
int i = 0;
while(i < 10){
        System.out.println(i); //输出变量的值
        i++; //变量的值增加 1
}
```

其执行流程为：

1、 执行 int I = 0;

2、 判断 i<10 是否成立，如果不成立则结束，否则执行下一步

3、 输出变量 i 的值
4、 i 的值增加 1
5、 跳转到步骤 2 继续执行
需要注意的是，首先 while 语句中的条件是循环成立的条件，也就是该条件成立则继续循环，所以在书写时注意。另外，内部代码的书写有顺序，同样是上面的代码，如果调整内部代码的顺序，如下所示：

```
int i = 0;
 while(i < 10){
        i++; //变量的值增加 1
        System.out.println(i); //输出变量的值
 }
```

则程序的执行结果将变为输出数字 1-10。所以在循环语句中，代码之间的顺序也影响整个程序的逻辑。
下面是用循环实现一个简单的数学逻辑，求 1-10 这 10 个数字的和。程序的原理是这样：声明一个变量 i，从 1 变化到 10，在声明一个变量 sum，每次和 i 的值相加以后赋值给自身，下次再使用 sum 是变化以后的 i 相加，循环结束以后，得到的结果就是数字 1-10 之间所有数字的和。
示例代码如下：

```
int i = 1;      //循环变量
int sum = 0;  //数字和
while(i <= 10){
        sum += i;    //和当前的 i 值相加
        i++;         //变量 i 增加 1
}
```

这样，第一次循环是把和 1 相加的结果赋值给 sum，然后再使用 sum 的值和 2 相加再赋值给 sum，依次类推，则得到 1-10 之间所有变量的和。
在使用循环语句时，发现规律需要的时间比编写和调试代码需要的时间多得多，所以要善于发现规律，善于思考。

Java 编程那些事儿 36—do-while 语句语法

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.4.2 do-while 语句

do-while 语句由关键字 do 和 while 组成，是循环语句中最典型的“先循环再判断”的流程控制结构，这个和其它 2 个循环语句都不相同。
do-while 语句的语法格式为：

```
do{
        循环体;
```

```
}while(循环条件);
```

语法说明：在 do-while 语句中，循环体部分是重复执行的代码部分，循环条件指循环成立的条件，要求循环条件是 boolean 类型，值为 true 时循环执行，否则循环结束，最后整个语句以分号结束。

执行流程：当执行到 do-while 语句时，首先执行循环体，然后再判断循环条件，如果循环条件不成立，则循环结束，如果循环条件成立，则继续执行循环体，循环体执行完成以后再判断循环条件，依次类推。

实现和 while 语句实现的类似的示例，则死循环的示例代码为：

```
do{
        System.out.println( ‘a’ );
}while(true);
```

实现输出 0-9 这 10 个数字的循环为：

```
int i = 0;
do{
        System.out.println(i);    //输出变量的值
        i++;                      //变量增加 1
}while(i < 10);
```

实现求 1-10 这 10 个数字的和的代码为：

```
int i = 1;
int sum = 0;
do{
        sum += i; //求和
        i++;      //变量增加 1
}while(i < 10);
System.out.println(sum);    //输出和
```

同理，实现求 5 的阶乘的代码类似，在数学上 5 的阶乘指 1×2×3×4×5，数学上规定 0 的阶乘等于 1，在实际计算时，阶乘的值增加的非常快，所以需要注意结果不能溢出。其具体代码为：

```
int i = 1;
int result = 1;
do{
        result *= i;
        i++;
}while(i <= 5);
System.out.println(result);
```

在实际的程序中，do-while 的优势在于实现那些先循环再判断的逻辑，这个可以在一定程度上减少代码的重复，但是总体来说，do-while 语句使用的频率没有其他的循环语句高。

Java 编程那些事儿 37—for 语句语法

郑州游戏学院 陈跃峰

出自：**http://blog.csdn.net/mailbomb**

5.4.3 for 语句

for 关键字的意思是“当…的时候”，是实际开发中比较常用的循环语句，其语法格式相对于前面的循环语句来说稍显复杂，但是在熟悉以后，将会发现其语法安排的比较条理，把循环控制和循环体很清晰的分开。

for 语句的语法格式为：

```
for(初始化语句;循环条件;迭代语句){
        循环体;
}
```

语法说明：

1、 和其它流程控制语句一样，语句中的大括号不是语法必须的，但是为了结构清楚以及在循环体部分可以书写多行代码，一般使用大括号。

2、 初始化语句作用是在循环开始以前执行，一般书写变量初始化的代码，例如循环变量的声明、赋值等。该语句可以为空。

3、 循环条件是循环成立的条件，要求必须为 boolean 类型，如果该条件为空，则默认为 true，即条件成立。

4、 迭代语句是指循环变量变化的语句，一般书写 i++、i—这样的结构，当然，该语句也可以为空

5、 循环体指循环重复执行的功能代码。

执行流程：

1、 执行初始化语句

2、 判断循环条件，如果循环条件为 false，则结束循环，否则执行下一步

3、 执行循环体

4、 执行迭代语句

5、 跳转到步骤 2 重复执行

需要注意的是：for 语句中的各个语句都可以为空，初始化语句在 for 语句执行时执行且只执行一次。

依据 for 语句的语法格式，则最简单的 for 语句是如下格式：

```
for(;;);
```

由于循环条件为空时，默认为 true，则循环条件恒成立，该循环的循环体即最后的一个分号，这样的语句称作空语句，则该循环是一个死循环，循环体是空语句。

在实际书写代码时，一般把循环控制部分都写在 for 语句的小括号内部，而循环体只书写和逻辑相关的代码，这种结构使逻辑显得很清晰。

使用 for 语句输出的 0-9 之间数字的代码如下：

```
for(int i = 0;i < 10;i++){
        System.out.println(i);
}
```

则该语句的执行流程为：

1、 执行 int i = 0;

2、 判断 i<10，如果条件不成立则结束，否则继续执行下一步
3、 执行 System.out.println(i);
4、 执行 i++
5、 跳转到步骤 2 继续执行
类似的示例代码，实现求 1-100 之间数字的和，代码如下：

```
int sum = 0;
for(int i = 1;i <= 100;i++){
        sum +=i;
}
System.out.println(sum);
```

这些是一些基本的 for 语句的使用，在通常情况下，for 语句和 while 语句之间可以实现很简单的转换，例如下面是一个使用 for 语句书写的 while 格式的代码：

```
int i = 0;
for(;i < 10;){
        System.out.println(i);
        i++;
}
```

关于 for 语句的深入使用请参看后续的综合示例部分的讲解。

5.4.4.小结

这里介绍了基本的循环控制语句的语法格式，在程序设计时，必须理解每种语句的语法格式和对应的特点，才能在实际使用时依据自己的逻辑进行灵活运用。

和前面的条件语句一样，在实际使用时，循环控制语句之间也可以进行相互的嵌套来解决复杂的逻辑，在语法上对于嵌套的层次没有限制。

while 语句和 for 语句在循环语句中属于“先判断再循环”的结构，而 do-while 语句属于“先循环再判断”的结构，所以从语法角度来看，do-while 语句的循环体至少会执行一次，在实际使用时 while 语句和 for 语句之间可以进行很方便的替换。

Java 编程那些事儿 38—break 和 continue 语句

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.5 break 和 continue 语句

break 和 continue 语句是和循环语句紧密相关的两种语句。其中 break

关键字的意思是中断、打断，continue 关键字的意思是继续。使用这两个关键字可以调节循环的执行。

5.5.1 break 语句

break 语句在前面的 switch 语句中已经介绍过，功能的话是中断 switch 语句的执行，在循环语句中，break 语句的作用也是中断循环语句，也就是结束循环语句的执行。

break 语句可以用在三种循环语句的内部，功能完全相同。下面以 while 语句为例来说明 break 语句的基本使用及其功能。

示例代码：

```
int i = 0;
while(i < 10){
        i++;
        if(i == 5){
                break;
        }
}
```

则该循环在变量 i 的值等于 5 时，满足条件，然后执行 break 语句，结束整个循环，接着执行循环后续的代码。

在循环语句中，可以使用 break 语句中断正在执行的循环。

在实际的代码中，结构往往会因为逻辑比较复杂，而存在循环语句的嵌套，如果 break 语句出现在循环嵌套的内部时，则只结束 break 语句所在的循环，对于其它的循环没有影响，示例代码如下：

```
for(int i = 0;i < 10;i++){
        for(int j = 0;j < 5;j++){
                System.out.println(j);
                if(j == 3){
                        break;
                }
        }
}
```

则该 break 语句因为出现在循环变量为 j 的循环内部，则执行到 break 语句时，只中断循环变量为 j 的循环，而对循环变量为 i 的循环没有影响。

在上面的示例代码中，如果需要中断外部的循环，则可以使用语法提供的标签语句来标识循环的位置，然后跳出标签对应的循环。示例代码如下：

```
lable1:
        for(int i = 0;i < 10;i++){
                for(int j = 0;j < 5;j++){
                        System.out.println(j);
                        if(j == 3){
                                break label1;
```

```
                    }
                }
            }
```

说明：这里的 label1 是标签的名称，可以为 Java 语言中任意合法的标识符，标签语句必须和循环匹配使用，使用时书写在对应的循环语句的上面，标签语句以冒号结束。如果需要中断标签语句对应的循环时，采用 break 后面跟标签名的方式中断对应的循环。则在该示例代码中 break 语句中断的即循环变量为 i 的循环。

同样的功能也可以使用如下的逻辑实现：

```
boolean b = false;
for(int i = 0;i < 10;i++){
    for(int j = 0;j < 5;j++){
        System.out.println(j);
        if(j == 3){
            b = true;
            break;
        }
    }
    if(b){
        break;
    }
}
```

该示例代码中，通过组合使用 **2** 个 **break** 以及一个标识变量，实现跳出外部的循环结构。

5.5.2 continue 语句

continue 语句只能使用在循环语句内部，功能是跳过该次循环，继续执行下一次循环结构。在 **while** 和 **do-while** 语句中 **continue** 语句跳转到循环条件处开始继续执行，而在 **for** 语句中 **continue** 语句跳转到迭代语句处开始继续执行。

下面以 **while** 语句为例，来说明 **continue** 语句的功能，示例代码如下：

```
int i = 0;
while(i < 4){
    i++;
    if(i == 2){
        continue;
    }
    System.out.println(i);
}
```

则该代码的执行结果是：

```
1
3
```

4

在变量 **i** 的值等于 **2** 时，执行 **continue** 语句，则后续未执行完成的循环体将被跳过，而直接进入下一次循环。

在实际的代码中，可以使用 **continue** 语句跳过循环中的某些内容。

和前面介绍的 **break** 语句类似，**continue** 语句使用在循环嵌套的内部时，也只是跳过所在循环的结构，如果需要跳过外部的循环，则需要使用标签语句标识对应的循环结构。示例代码如下：

```
lable1:
    for(int i = 0;i < 10;i++){
        for(int j = 0;j < 5;j++){
            System.out.println(j);
            if(j == 3){
                continue label1;
            }
        }
    }
```

这样在执行 **continue** 语句时，就不再是跳转到 **j++**语句，而是直接跳转到 **i++** 语句。

5.5.3 小结

在实际的代码中，可以根据需要使用 **break** 和 **continue** 语句调整循环语句的执行，**break** 语句的功能是结束所在的循环，而 **continue** 语句的功能是跳过当次循环未执行的代码，直接执行下一次循环

Java 编程那些事儿 39—流程控制综合示例 1

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.6 综合示例

在一般的学校学习流程控制时，重点是放在流程控制的相关语法，其实为了能成为一个合格的程序员，仅仅学好语法是远远不够的，还需要通过大量的练习来适应程序设计语言的思维方式，并且熟练地把自己的解决问题的步骤形成代码，这些都需要通过大量的阅读代码和编写代码来实现。

所以在学习流程控制时，重点是解决实际的问题，而不是仅仅停留在语法层面上，这个是很多在校学生学习程序时最突出的一个问题。

在遇到一个实际问题时，首先要能够思考出解决这个问题的数学步骤或逻辑步骤，然后才能编写对应的代码，所以遇到实际问题是，一定要积极思考，

并且善于思考，对于一个相同的问题，不同的逻辑就可以写出不同的代码，所以在思考解决问题的方式时，需要进行发散性的思维，而这些理性的思维很多都是建立在数学基础以及对语法的熟悉基础之上。

下面，通过一系列的实际问题，来说明解决实际问题的步骤以及书写的对应的代码。

5.6.1 示例讲解

5.6.1.1 最大公约数

问题：求两个自然数的最大公约数。

这两个都是基础的数学问题，最大公约数指两个数字公共的约数中最大的，例如数字 6 的约数有 1、2、3、6，数字 9 的约数有 1、3、9，则数字 6 和数字 9 的公共约数有 1 和 3，其中 3 是最大的公约数。

第一种思路：从 1 开始循环，每次把符合要求(即同时是两个数字的约数)的值都存储起来，那么最后一个存储起来的就是最大的约数。

则实现的代码如下：

```
int n = 6;
int m = 9;
int result = 1;
for(int i = 1;i <= n;i++){
        if((n % i == 0) && (m % i == 0)){
                result = i;
        }
}
System.out.println(result);
```

使用该思路，每次都存储得到的公共约数，那么最后一个存储的就是两个数字的最大公约数。

第二种思路：从两个数字中最小的数字开始循环，每次减 1，那么第一次得到的公共约数就是所求的最大公约数。

则实现的代码如下：

```
int n = 6;
int m = 9;
int result = n > m ?m : n;
for(int i = result;i >= 1;i--){
        if((n % i == 0) && (m % i == 0)){
                result = i;
                break; //结束循环
        }
}
System.out.println(result);
```

当然，解决这个问题，还有很多其它的方法，这里演示的这两种实现

只是最自然的实现而已，采用类似的原理也可以求两个数字的最小公倍数的结构。

5.6.1.2 百元百鸡问题

问题描述：每只母鸡 3 元，每只公鸡 4 元，每只小鸡 0.5 元，如果花 100 元钱买 100 只鸡，请问有哪些可能？说明：每种鸡的数量都可以为零。

其实这个问题是数学上的组合问题，只需要把所有的情况列举出来，然后来判断是否符合要求即可。这样的重复列举的问题，在程序上可以使用循环进行解决。

第一种思路：当母鸡的数量为 0 时，公鸡的数量从 0-100，当公鸡的数量每变化一次，小鸡的数量就从 0 变化到 100，使用如下数值组合来描述这个思路：

母鸡数量 公鸡数量 小鸡数量

0 0 从 0 变化到 100

1 0 从 0 变化到 100

2 0 从 0 变化到 100

……

0 1 从 0 变化到 100

1 1 从 0 变化到 100

……

100 100 100

上面列举出了所有公鸡、母鸡和小鸡的数量都是0-100时的所有组合，总计是101的三次方种，这样的穷举结构直接存在嵌套，在程序实际实现时，通过循环之间的嵌套就可以实现，则实现的代码如下：

```
for(int i = 0;i <= 100;i++){ //母鸡数量
    for(int j = 0;j <= 100;j++){ //公鸡数量
        for(int k = 0;k <= 100;k++){ //小鸡数量
            //判断数量是否为 100，以及金额是否为 100
            if((i +j + k == 100) && (i * 3 + j * 4 + k * 0.5 == 100)){
System.out.println(“母鸡数量:” + i + “ 公鸡数量:” + j + “ 小鸡数量” + k);
```

```
                                        }
                                    }
                                }
                            }
```

按照 for 语句的执行流程，循环变量变化 1，则内部的循环执行一次，而在循环嵌套时，循环体又是一个新的循环，则该循环执行完成的一组循环。这里通过循环的嵌套实现了所有数值的穷举。在循环的内部，只需要按照题目要求判断一下数量和金额是否符合要求即可。

但是这样的代码效率比较差，可以通过简单的优化来提高程序的执行效率。

第二种思路：由于母鸡每只的金额是 3 元，所以 100 元最多购买的母鸡数量是 100/3=33 只，同理 100 元最多购买的公鸡数量是 25 只，而按照 100 元 100 只的要求，小鸡的数量应该为 100 减去公鸡和母鸡的数量，这样代码就可以简化为如下的结构：

```
                for(int i = 0;i <= 33;i++){ //母鸡数量
                        for(int j = 0;j <= 25;j++){ //公鸡数量
                                int k = 100 - i - j; //小鸡数量
                                        //判断金额是否为 100
                                         if (i * 3 + j * 4 + k
* 0.5 == 100){

System.out.println(“母鸡数量:” + i + “ 公鸡数量:” + j + “ 小鸡数量” + k);
                                        }
                                }
                        }
                }
```

这样，就可以大幅提高程序的执行效率，从而提高程序的执行速度。当然该代码还可以继续进行优化，那样可以再次提供程序的执行效率。

Java 编程那些事儿 40—流程控制综合示例 2

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.6.1.3 喝汽水问题

问题：共有 1000 瓶汽水，每喝完后一瓶得到的一个空瓶子，每 3 个空瓶子又能换 1 瓶汽水，喝掉以后又得到一个空瓶子，问总共能喝多少瓶汽水，最后还剩余多少个空瓶子？

这个问题其实是个比较典型的递推问题，每 3 个空瓶都可以再换 1 瓶新的汽水，这样一直递推下去，直到最后不能换到汽水为止。

第一种思路：每次喝一瓶，每有三个空瓶子就去换一瓶新的汽水，直

到最后没有汽水可以喝为止。在程序中记忆汽水的数量和空瓶子的数量即可。

则实现的代码如下：

```
int num = 1000;        //汽水数量
int drinkNum = 0;      //喝掉的汽水数量
int emptyNum = 0;    //空瓶子的数量
while(num > 0){       //有汽水可以喝
        num--;          //喝掉一瓶
        emptyNum++; //空瓶子数量增加 1
        drinkNum++;    //喝掉的汽水数量增加 1
        if(emptyNum == 3){ //有 3 个空瓶子，则去换
                num++;    //汽水数量增加 1
                emptyNum = 0;    //空瓶子数量清零
        }
}
System.out.println(“总共喝掉瓶数:” + drinkNum);
System.out.println(“剩余空瓶子数:” + emptyNum);
```

执行该程序，输出结果如下：

```
总共喝掉瓶数：1499
剩余空瓶子数：2
```

在该代码中，每次循环喝掉一瓶汽水，则汽水数量减少 1，空瓶子数增加 1，喝掉的总汽水瓶数增加 1，每次判断空瓶子的数量是否达到 3，如果达到 3 则换 1 瓶汽水，同时空瓶子的数量变为零。这种思路比较直观，但是循环的次数比较多，所以就有了下面的逻辑实现。

第二种思路：一次把所有的汽水喝完，获得所有的空瓶子，再全部换成汽水，然后再一次全部喝完，再获得所有的空瓶子，依次类推，直到没有汽水可喝为止。

则实现的代码如下：

```
int num = 1000;        //汽水数量
int drinkNum = 0;      //喝掉的汽水数量
int emptyNum = 0;     //空瓶子的数量
while(num > 0){       //有汽水可以喝
        drinkNum += num; //喝掉所有的汽水
        emptyNum += num; //空瓶子数量等于上次剩余的加上这次喝掉的数量
        num = emptyNum / 3; //兑换的汽水数量
        emptyNum -= num * 3; //本次兑换剩余的空瓶子数量
}
System.out.println(“总共喝掉瓶数:” + drinkNum);
System.out.println(“剩余空瓶子数:” + emptyNum);
```

在该代码中，每次喝掉所有的汽水，也就是 num 瓶，则喝掉的总瓶数每次增加 num，因为每次都可能剩余空瓶子(不足 3 个的)，则总的空瓶子数量是

上次空瓶子数量加上本次喝掉的num瓶。接着是对话汽水，则每次可以兑换的汽水数量是空瓶子的数量的1/3，注意这里是整数除法，而本次兑换剩余的空瓶子数量是原来的空瓶子数量减去兑换得到汽水数量的3倍，这就是一次循环所完成的功能，依次类推即可解决该问题。

5.6.1.4 水仙花数

问题：水仙花数指三位数中，每个数字的立方和和自身相等的数字，例如370，3 × 3 × 3 + 7 × 7 × 7 + 0 × 0 × 0 =370，请输出所有的水仙花数。

该问题中体现了一个基本的算法——数字拆分，需要把一个数中每位的数字拆分出来，然后才可以实现该逻辑。

实现思路：循环所有的三位数，拆分出三位数字的个位、十位和百位数字，判断3个数字的立方和是否等于自身。

则实现的代码如下所示：

```
for(int i = 100;i < 1000;i++){ //循环所有三位数
        int a = i % 10;          //个位数字
        int b = (i / 10) % 10; //十位数字
        int c = i / 100;        //百位数字
        //判断立方和等于自身
         if(a * a * a + b * b * b + c * c * c == i){
                System.out.println(i);
        }
}
```

在该代码中，拆分个位数字使用i和10取余即可，拆分十位数字时首先用i除以十，去掉个位数字，并使原来的十位数字变成个位，然后和10取余即可，因为i是一个三位数，所以i除以100即可得百位数字，因为这里都是整数除法，不存在小数的问题。然后只需要判断立方和是否等于自身即可。

注意：因为i是循环变量，这里不能改变i的值，不然可能造成死循环。

Java 编程那些事儿 41—流程控制综合示例 3

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.6.1.5 99 乘法表

问题：在控制台打印数学上的99乘法表

该类问题是发现数字的规律，然后将数值的规律用程序描述出来。实际实现时，可能需要耐心的进行调试。在这里，需要实现数字的多行输出，前面

使用的System.out.println是输出内容并换行，后续再输出的内容就再下一行显示，如果需要在输出时不换行，则可以使用System.out.print进行输出。

99乘法表的规则是总计9行，每行单独输出，第一行有1个数字，第二行有2个数字,依次类推，数字的值为行号和列号的乘积。

实现思路：使用一个循环控制打印9行，在该循环的循环体中输出该行的内容，一行中输出的数字个数等于行号，数字的值等于行号和列号的成绩。

实现代码如下：

```
for(int row = 1;row <= 9;row++){        //循环行
    for(int col = 1;col <= row;col++){   //循环列
        System.out.print(row * col);  //输出数值
        System.out.print(' ');        //输出数字之间的间隔空格
    }
    System.out.println();             //一行输出结束，换行
}
```

该程序的输出为：

```
1

2 4

3 6 9

4 8 12 16

5 10 15 20 25

6 12 18 24 30 36
```

7 14 21 28 35 42 49

8 16 24 32 40 48 56 64

9 18 27 36 45 54 63 72 81

在该输出中，数字之间的对齐有些问题，第四行和第五行的对齐就很明显。那么如果在输出时想让数字对齐，那么就要首先思考数字为什么不能对齐？则问题直观的出现在有些数字是一位数有些是两位数，发现了原因就可以着手解决了，如果想实现数字的左对齐，则在一位数字的后续多输出一个空格，如果想实现数字的右对齐，则只需要在一位数字的前面输出一个空格即可。

以下代码实现了数字的右对齐：

```
for(int row = 1;row <= 9;row++){        //循环行
    for(int col = 1;col <= row;col++){   //循环列
        if(row * col < 10){      //一位数
            System.out.print(' ');
        }
        System.out.print(row * col);  //输出数值
        System.out.print(' ');        //输出数字之间的间隔空格
    }
    System.out.println();             //一行输出结束，换行
}
```

所以在实际书写代码时，代码的位置对于程序逻辑的影响很大，在编写代码时，需要认真考虑代码书写的位置。

5.6.1.6 打印图形

问题：在控制台中打印如下格式的图形

```
    *
   ***
  *****
 *******
*********
```

由于受到控制台输出的限制，只能按照行的上下，依次进行输出，所以解决打印图形的问题时，只能按照从上到下依次输出每行的内容，关键是仔细观察，发现图形的规律。

第一种思路：外部循环循环 5 次打印 5 行，每行的内容分为两部分：空格和星号，每行空格的数量是 5 减去行号个，每行星号的数量是行号的 2 倍减 1 个，在外部循环内部先打印空格，再打印星号，每个都只打印 1 个，使用数量控制对应的打印次数，打印完星号以后换行。

则实现的代码如下：

```
for(int row = 1;row <= 5;row++){       //循环行
    //打印空格
    for(int c1 = 0;c1 < 5 - row;c1++){
        System.out.print(' ');
    }
    //打印星号
    for(int c2 = 0;c2 < 2 * row - 1;c2++){
        System.out.print('*');
    }
```

```
            //换行
            System.out.println();
        }
```

该代码中 row 的循环用于控制打印的行数，row 变量的值代表行号，内部的循环体分为三部分：打印空格，打印星号和换行，打印的数量参看图形的规律部分。

第二种思路：外部循环循环 5 次打印 5 行，内部每行打印的总字符数量是 4 加行号个，其中前 5-行号个字符是空格，后续的字符是星号，所有字符打印完成以后换行。

则实现的代码如下：

```
        for(int row = 1;row <= 5;row++){      //循环行
            //循环总的字符数
            for(int col = 0; col < 4 + row;col++){
                if(col < 5 - row){  //打印空格
                    System.out.print(' ');
                }else{ //打印星号
                    System.out.print('*');
                }
            }
            //换行
            System.out.println();
        }
```

该代码的总体思路和第一种思路一样，都是按行打印，只是在考虑问题时首先考虑字符总的数量，把这个数量作为循环次数，内部控制那些该输出字

符那些该输出星号即可。

5.6.1.7 质数判断

问题：判断一个自然数是否是质数。

质数指只能被 1 和自身整除自然数，也称素数，最小的质数是 2。对于自然数来说，任何一个数字都可以被 1 和自身整除。

实现思路：利用数学上的反证法进行判断。则问题转换为只需要判断不能被 1 和自身以外的任何一个数字整除即可。则假设判断的数字是 n 的话，则这些数字的区间是[2,n-1]和大于 n 的所有数字。在数学上 n 不可能被大于 n 的数字整除，所以程序只需要判断[2,n-1]之间的数字即可，如果被该区间的任何一个数字整除了，则说明不是质数。

则实现的代码如下：

```
int n = 23;

boolean b = true;  //存储是否是质数，假设是质数

for(int i = 2;i < n;i++){

        //如果整除，则不是质数

        if(n % i == 0){

                b = false;

                break;      //后续比较没有意义，结束循环

        }

}

//输出是否是质数

if(b){

        System.out.println("是质数");
```

```
}else{
        System.out.println("不是质数");
}
```

该代码是最容易思考出来的一种实现，其实在数学上只需要判断 n 是否可以被 2 到 n 的二次方根之间的数字即可。则实现的代码变为如下：

```
int n = 23;
boolean b = true;  //存储是否是质数，假设是质数
for(int i = 2;i <= Math.sqrt(n);i++){
        //如果整除，则不是质数
        if(n % i == 0){
                b = false;
                break;      //后续比较没有意义,结束循环
        }
}
//输出是否是质数
if(b){
        System.out.println("是质数");
}else{
        System.out.println("不是质数");
}
```

通过缩小判断数字的区间，可以显著提高程序的执行效率。说明：这里的 Math.sqrt 的功能是计算 n 的二次方根。

关于流程控制的综合示例部分就介绍这么多，下面将整理一下流程控制的综合练习。

Java 编程那些事儿 42—流程控制综合练习

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

5.6.2 综合练习

本部分是一些整理的关于流程控制部分的综合练习，可以通过这些练习熟悉 Java 语言的基本语法，以及锻炼逻辑思维能力。

练习题：

1、 计算数字 12 和 18 的最小公倍数。

2、 如果苹果 1 元/个， 桔子 2 元/个， 芒果 4 元/个，若是用 10 元去买，有几种组合呢？

3、 一只猴子第一天摘下若干个桃子，当即吃了一半，还不瘾，又多吃了一个，第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个，第 10 天早上想再吃时，发现只剩下一个桃子了。请问猴子第一天一共摘了多少个桃子？

4、 计算 30 的阶乘。

5、 一个农场有头母牛，现在母牛才一岁，要到四岁才能生小牛，四岁之后，每年生一头小牛。假设每次生的都是母牛，并且也遵守 4 年才生育并生母牛的原则，并且无死亡，请问 n 年后共有多少头牛？

6、 角谷猜想问题：日本一位中学生发现一个奇妙的“定理”，请角谷教授证明，而教授无能为力，于是产生角谷猜想。猜想的内容是：任给一个自然数，若为偶数除以 2，若为奇数则乘 3 加 1，得到一个新的自然数后按照上面的法则继续演算，若干次后得到的结果必然为 1。试编写代码验证该猜想是否正确。

7、 输出 20 个如下规律的数列： 1 1 2 3 5 8 13……

8、 输出 30 个如下规律的数列： 1 3 6 10 15 21 ……

9、 输出任意一个三位数中的个位数字和百位数字对调的数值，例如如果三位数是 235，则输出 532。

10、求 100 以内所有质数的和。

这里只列举了部分流程控制的练习，希望大家可以积极补充一些经典的流程控制练习习题，大家一起提高。

5.6.3 综合练习部分答案

```
int n = 12;
int m = 18;
int result = 1;
for(int i = m*n;i >= 18;i--){
if((i % n == 0) && (i % m == 0)){
result = i;
}
}
System.out.println(result);
```

jinyi81 发表于 2008 年 8 月 27 日 18:50:16 举报

```
int n = 12;
int m = 18;
int result = 1;
for(int i = m*n;i >= 18;i--){
if((i % n == 0) && (i % m == 0)){
result = i;
}
}
System.out.println(result);
```

jinyi81 发表于 2008 年 8 月 27 日 18:59:43 ;
举报

```
2,
for(int i = 0;i <= 10;i++){
for(int j = 0;j <= 5;j++){
for(int k = 0;k <= 2;k++){
```

```
if (i + j * 2 + k * 4 == 10){
System.out.println(i+" "+j+" "+k);

}}}}}}
```

yangjie83102 发表于 2008 年 9 月 9 日 11:14:20 举报

第三题：

```
class Eat
{
public static void main(String[] args)
{
int yushu =1 ;//每天剩余的桃子
int toal = 0 ;//桃子的总数
for (int day=5/*day 是天数*/; day>1;day-- )
{
yushu = 2*yushu+2;
}
toal = yushu;
System.out.println(toal);
}
}
```

star 发表于 2008 年 11 月 24 日 20:09:20 aboolean=false;

}

} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/06/20/2570811.aspx 发表时间:2008 年 11 月 24 日 20:09:20" ;j="+j+" i="+i+">举报

```
//1、 计算数字 12 和 18 的最小公倍数。
/**
```

```
* 最小公倍数（Least Common Multiple，缩写 L.C.M.），
* 对于两个整数来说，指该两数共有倍数中最小的一个。计算最小公倍数时，
* 通常会借助最大公因数（gcd/hcf）来辅助计算。
*/
int a=12,b=18;
boolean aboolean=true;
for(int i=1;aboolean&&i<a;i++)
for(int j=1;aboolean&&j<b;j++)
{
if(a*i==b*j)
{
int ab=b*j;
System.out.println("i="+i+";j="+j+";ab="+ab);
aboolean=false;
}
}
```

star 发表于 2008 年 11 月 24 日 20:24:20 ;

}

}

} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/06/20/2570811.aspx 发表时间:2008 年 11 月 24 日 20:24:20">举报

```
//2、 如果苹果 1 元/个， 桔子 2 元/个， 芒果 4 元/个，若是用 10 元去买，有几种组合呢？
int a=1,b=2,c=4,n=10;
for(int i = 0;i <= n/a;i++){ //苹果数量
for(int j = 0;j <= n/b;j++){ //桔子数量
for(int k = 0;k <= n/c;k++)//芒果数量
```

```
//判断金额是否为 n
if (i * a + j * b + k * c == n){
System.out.println("苹果数量：" + i + " ;桔子数量：" + j +
" ;芒果数量 " + k);
}
}
}
```

---

star 发表于 2008 年 11 月 24 日 23:01:44 ；

```
}
}
}
/**3、 一只猴子第一天摘下若干个桃子，当即吃了一半，
还不瘾，又多吃了一个,第二天早上又将剩下的桃子吃掉一半，
又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个，
第 10 天早上想再吃时，发现只剩下一个桃子了。请问猴子第一天一共摘了多少
个桃子？
*/
int n=1,t=10;
for(int i=t-1;i>0;i--)
{
n=(n+1)*2;
System.out.println("猴子第"+i+"天一共有 " + n + " 个桃子");
}
System.out.println("猴子第一天一共摘了 " + n + " 个桃子");
//4、 计算 30 的阶乘。
int n=30;
double nj=1;
for(int i=1;i<=n;i++)
```

```
{
nj*=i;
System.out.println(i+" 的阶乘是 " + nj );
}
System.out.println(n+" 的阶乘是 " + nj );
/**5、 一个农场有头母牛(a)，现在母牛才一岁(b)，要到四岁(c)才能生小牛，
* 四岁之后，每年生一头小牛(d)。假设每次生的都是母牛，
* 并且也遵守 4 年才生育并生母牛的原则，并且无死亡，请问 n 年后共有多少(x)
头牛？
*/
int a=1,b=1,c=4,d=1,n=50,x;
int[] m=new int[c+2];
//m[i]为 i 岁的母牛
for(int i=0;i 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/0
6/20/2570811.aspx 发表时间:2008 年 11 月 24 日 23:01:44">举报
```

```
//2、 如果苹果 1 元/个， 桔子 2 元/个， 芒果 4 元/个，若是
用 10 元去买，有几种组合呢？
int a=1,b=2,c=4,n=10;
for(int i = 0;i <= n/a;i++){ //苹果数量
for(int j = 0;j <= n/b;j++){ //桔子数量
for(int k = 0;k <= n/c;k++)//芒果数量
//判断金额是否为 n
if (i * a + j * b + k * c == n){
System.out.println("苹果数量：" + i + " ;桔子数量：" + j +
" ;芒果数量 " + k);
}
}
}
```

```
/**3、 一只猴子第一天摘下若干个桃子，当即吃了一半，
还不瘾，又多吃了一个,第二天早上又将剩下的桃子吃掉一半，
又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个，
第 10 天早上想再吃时，发现只剩下一个桃子了。请问猴子第一天
一共摘了多少个桃子？
*/
int n=1,t=10;
for(int i=t-1;i>0;i--)
{
n=(n+1)*2;
System.out.println("猴子第"+i+"天一共有 " + n + " 个桃子
");
}
System.out.println("猴子第一天一共摘了 " + n + " 个桃子
");
//4、 计算 30 的阶乘。
int n=30;
double nj=1;
for(int i=1;i<=n;i++)
{
nj*=i;
System.out.println(i+" 的阶乘是 " + nj );
}
System.out.println(n+" 的阶乘是 " + nj );
/**5、 一个农场有头母牛(a)，现在母牛才一岁(b)，要到四岁(c)
才能生小牛，
* 四岁之后，每年生一头小牛(d)。假设每次生的都是母牛，
* 并且也遵守 4 年才生育并生母牛的原则，并且无死亡，请问 n 年
```

后共有多少(x)头牛？

*/

int a=1,b=1,c=4,d=1,n=50,x;

int[] m=new int[c+2];

//m[i]为 i 岁的母牛

for(int i=0;i

---

star 发表于 2008 年 11 月 24 日 23:03:01 * 四岁之后，每年生一头小牛(d)。假设每次生的都是母牛，

* 并且也遵守 4 年才生育并生母牛的原则，并且无死亡，请问 n 年后共有多少(x)头牛？

*/

int a=1,b=1,c=4,d=1,n=50,x;

int[] m=new int[c+2];

//m[i]为 i 岁的母牛

for(int i=0;i<=c+1;i++)

{

m[i]=0;

}

m[b]=a;

for(int i=1;i<=n;i++)

{

m[c+1]+=m[c];

for(int j=c;j>=1;j--)

{

m[j]=m[j-1];

}

m[0]=m[c+1]*d;

x=0;

```
 for(int j=0;j<=c+1;j++)
 {
 x+=m[j];
 System.out.print(" ;?+j+?-?+m[j]);
 }
System.out.println("---第"+i+"年:"+x);
}
```

文章链接:http://blog.csdn.net/Mailbomb/archive/2008/06/20/2570811.aspx 发表时间:2008 年 11 月 24 日 23:03:01">举报

```
/**5、 一个农场有头母牛(a)，现在母牛才一岁(b)，要到四岁(c)才能生小牛，
* 四岁之后，每年生一头小牛(d)。假设每次生的都是母牛，
* 并且也遵守 4 年才生育并生母牛的原则，并且无死亡，请问 n 年后共有多少(x)头牛？
*/
int a=1,b=1,c=4,d=1,n=50,x;
int[] m=new int[c+2];
//m[i]为 i 岁的母牛
for(int i=0;i<=c+1;i++)
{
m[i]=0;
}
m[b]=a;
for(int i=1;i<=n;i++)
{
m[c+1]+=m[c];
for(int j=c;j>=1;j--)
{
```

```
m[j]=m[j-1];
}
m[0]=m[c+1]*d;
x=0;
for(int j=0;j<=c+1;j++)
{
x+=m[j];
System.out.print(";"+j+"-"+m[j]);
}
System.out.println("---第"+i+"年:"+x);
}
```

---

star 发表于 2008 年 11 月 24 日 23:03:54 ；

```
else if(n==2)
System.out.println("输出"+n+"个如下规律的数列：1 1");
else
{
System.out.print("输出"+n+"个如下规律的数列：1 1");
for(int i=3;i<=n;i++)
{
c=b;
b=a+b;
a=c;
System.out.print(" "+b);
}
}
```

文章链接:http://blog.csdn.net/Mailbomb/archive/2008/06/20/2570811.aspx 发表时间:2008 年 11 月 24 日 23:03:54">举报

/**6、 角谷猜想 问题：日本一位中学生发现一个奇妙的“定理”，

* 请角谷教授证明，而教授无能为力，于是产生角谷猜想。猜想的内容是：
* 任给一个自然数，若为偶数除以 2，若为 奇数则乘 3 加 1，得到一个新的自然
* 数后按照上面的法则继续演算，若干次后得到的结果必然为 1。
* 试编写代码验证该猜想是否正确。
*/

```
long a=1112;
while(!(a==1))
{
if(a%2==0)
{
a=a/2;
}
else
{
a=a*3+1;
}
System.out.println("a="+a);
}
//7、 输出 20 个如下规律的数列： 1 1 2 3 5 8 13……
int a=1,b=1,c=0,n=20;
if(n==1)
System.out.println("输出"+n+"个如下规律的数列：1");
else if(n==2)
System.out.println("输出"+n+"个如下规律的数列：1 1");
else
{
```

```
System.out.print("输出"+n+"个如下规律的数列：1 1");
for(int i=3;i<=n;i++)
{
c=b;
b=a+b;
a=c;
System.out.print(" "+b);
}
}
```

star 发表于 2008 年 11 月 24 日 23:05:14 ；

```
for(int i=1;i<=n;i++)
{
System.out.print((i*i+i)/2+" ");
}
/**9、 输出任意一个三位数中的个位数字和百位数字对调的数值，
* 例如如果三位数是 235，则输出 532。
*/
int i=235,a=i % 10,b = (i / 10) % 10,c = i / 100;
System.out.println(a*100+b*10+c);
//10、 求 100 以内所有质数的和。
int n = 0;
for(int i=2;i<=100;i++)
{
boolean b = true; //存储是否是质数，假设是质数
for(int j = 2;j <= Math.sqrt(i);j++){
//如果整除，则不是质数
if(i % j == 0){
b = false;
```

```
break; //后续比较没有意义，结束循环
}
}
if(b)
{
n+=i;
System.out.println(";"+i+" : "+n);
}

}
```

文章链接:http://blog.csdn.net/Mailbomb/archive/2008/06/20/2570811.aspx 发表时间:2008 年 11 月 24 日 23:05:14">举报

```
//8、 输出 30 个如下规律的数列： 1 3 6 10 15 21 ……
/**
* 1 3 6 10 15 21 …的通项公式?
* 解：（n^2+n)/2
*/
int n=30;
System.out.print("输出 30 个如下规律的数列：");
for(int i=1;i<=n;i++)
{
System.out.print((i*i+i)/2+" ");
}
/**9、输出任意一个三位数中的个位数字和百位数字对调的数值，
* 例如如果三位数是 235，则输出 532。
*/
int i=235,a=i % 10,b = (i / 10) % 10,c = i / 100;
System.out.println(a*100+b*10+c);
```

```
//10、 求 100 以内所有质数的和。
int n = 0;
for(int i=2;i<=100;i++)
{
boolean b = true; //存储是否是质数，假设是质数
for(int j = 2;j <= Math.sqrt(i);j++){
//如果整除，则不是质数
if(i % j == 0){
b = false;
break; //后续比较没有意义，结束循环
}
}
if(b)
{
n+=i;
System.out.println(";"+i+" : "+n);
}

}
```

---

star 发表于 2008 年 11 月 24 日 23:07:18 ；

```
else if(n==2)
System.out.println("输出"+n+"个如下规律的数列：1 1");
else
{
System.out.print("输出"+n+"个如下规律的数列：1 1");
for(int i=3;i<=n;i++)
{
c=b;
```

```
b=a+b;
a=c;
System.out.print(" "+b);
}
}
```

文章链接:http://blog.csdn.net/Mailbomb/archive/2008/06/20/2570811.aspx 发表时间:2008 年 11 月 24 日 23:07:18">举报

```
/**6、 角谷猜想 问题：日本一位中学生发现一个奇妙的“定理”，
* 请角谷教授证明，而教授无能为力，于是产生角谷猜想。猜想的内容是：
* 任给一个自然数，若为偶数除以 2，若为 奇数则乘 3 加 1，得到一个新的自然
* 数后按照上面的法则继续演算，若干次后得到的结果必然为 1。
* 试编写代码验证该猜想是否正确。
*/
long a=1112;
while(!(a==1))
{
if(a%2==0)
{
a=a/2;
}
else
{
a=a*3+1;
}
System.out.println("a="+a);
}
```

```
//7、 输出 20 个如下规律的数列： 1 1 2 3 5 8 13……
int a=1,b=1,c=0,n=20;
if(n==1)
System.out.println("输出"+n+"个如下规律的数列：1");
else if(n==2)
System.out.println("输出"+n+"个如下规律的数列：1 1");
else
{
System.out.print("输出"+n+"个如下规律的数列：1 1");
for(int i=3;i<=n;i++)
{
c=b;
b=a+b;
a=c;
System.out.print(" "+b);
}
}
```

star 发表于 2008 年 12 月 1 日 17:49:30 举报

```
function monkey($day)
{
if ($day==1)
{
$number = 1;
}else
{
$number = (monkey($day-1) + 1) * 2;
}
return $number;
```

```
}
echo $total = monkey(9);
```

猴子问题 PHP 解决

Java 编程那些事儿 43—数组概述

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

第六章 数组

数组(Array)是 Java 语言中内置的一种基本数据存储结构，通俗的理解，就是一组数的集合，目的是用来一次存储多个数据。

数组是程序中实现很多算法的基础，可以在一定程度上简化代码的书写。

Java 语言中的数组可以分为：一维数组和多维数组，本部分将首先以一维数组为基础进行讲解，最后讲解多维数组的概念和相关的理解、使用。

6.1 数组概述

在程序中，如果需要存储一个数值的话，则可以在代码中声明一个变量来进行存储，但是有些时候，为了程序操作的方便，需要将一组相关的数值存储在一起，这就是数组出现的最初原因。

在实际使用时，数组的目的就是存储一组相关的数据，例如棋牌游戏中的一副牌等，从存储数值的角度考虑，其作用是和变量等价的。

则实际使用时，数组名称是一个整体，类似学校里的班级名称，为了能够方便的访问数组中的某个具体的值，对数组中的值进行强制的编号，这个编号称作数组的下标，类似班级中每个学员的序号。在实际引用数组中的值时，使用数组名称和下标一起进行指定，类似于 XX 班级序号为 n 的学员。

为了数组管理的方便，所以在语法上要求数组中存放的每个元素类型必须相同。数组中的每个具体的数值也称作数组元素。

在内存中，数组存储在连续的区域内部，因为数组中每个元素的类型相同，则占用的内存大小也一致，所以在访问数组中的元素时可以直接根据数组在内存中的起始位置以及下标来计算元素的位置，因此数组的访问速度很高。

实际使用时，每个数组都有长度，为了方便管理，在 Java 语言中，在初始化数组时，必须指定数组的长度，而且一旦指定，长度就不能改变，除非在重新初始化该数组。

了解了数组以上相关的概念以后，在实际使用数组时，数组的类型、

数组的长度以及数组中每个元素的功能，都由程序员根据需要进行指定，这些都需要一定的设计技巧，也是初学者最不熟悉的问题。这个问题可以简单的这么理解，就像有了一块泥巴，只要你按照规则来做，可以根据你的需要做成任意的结构，制作过程和制作方法都由制作者进行设计。

虽然数组从结构上来看，只是把以前语法中的多个变量存储在一起，通过数组名称组合上下标的方式进行使用，这个简单的变化，将极大的简化程序算法的实现，所以说数组是数据存储方式的很大革新。或者套用一句时髦的话——“数据存储的一小步，确实算法实现的一大步”。数组的另外一个变革就是下标可以使用变量进行代表，这样在访问数组的值时会更加灵活，这个也是理解数组的关键。

总结来说，主要有以下几点：

1、 数组中的元素类型必须相同。
2、 数组的长度一旦指定即不能改变。
3、 数组中的值通过数组名和下标组合起来进行访问。

Java 编程那些事儿 44—数组基础语法

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.2 数组基本语法

了解了数组的概念以后，下面来看一下数组的语法格式。数组的语法格式主要有四种：数组声明、数组初始化、引用数组元素和获得数组长度。

6.2.1 数组声明

和变量类似，数组在使用以前也必须声明，数组的声明语法格式为：

```
数据类型 数组名称[]
```

或：

```
数据类型[] 数组名称
```

例如：

```
int m[];
char c[];
double d[];
```

这里的数据类型可以是Java语言的任意数据类型，也就是说既可以是基本数据类型也可以是复合数据类型。在声明数组时使用一对中括号，该对中括号既可以放在数据类型的后面，也可以放在数组名称的后面。数组名称是一个标识符，可以根据需要设置其名称，在程序中使用该名称代表该数组。

这两种声明的语法格式在实际使用时完全等价，可以根据习惯进行使

用。

数组声明以后在内存中不占用空间，没有地址，由于数组是复合数据类型，所以声明完成以后其默认值是 null。

数组声明以后不能直接使用，必须对其初始化以后才可以进行使用。

6.2.2 数组初始化

数组初始化就是对数组进行赋值。数组的初始化语法分为两种：静态初始化和动态初始化。静态初始化适用于已知数组所有元素的值，一次初始化所有元素，动态初始化只申请空间，每个元素的值是数组声明时数据类型对应的初始值。

6.2.2.1 静态初始化

静态初始化，也称数组的整体赋值，是一次为数组中所有元素依次进行赋值的语法，通过可以语法可以指定数组中每个元素的值，同时也指定了数组的长度。

语法格式为：

```
数据类型[] 数组名称 = {值 1,值 2,……,值 n};
```

例如：

```
int[] m = {1,2,3,4};
char c[] = { ‘a’,’ f’,’ d’ };
```

静态初始化必须和数组的声明位于同一行，换句话说，只能在声明数组的同时进行静态初始化。数组中的所有元素书写一对大括号的内部，系统按照值的书写顺序依次为数组运算进行赋值，例如数组 m，则将 1 赋值给 m 数组的第一个元素，2 赋值给 m 数组的第二个元素，依次类推，数组的总长度等于静态初始化时数值的个数。在实际书写时，需要注意，值的类型必须和数组声明时的类型匹配，或者可以自动进行转换。

在实际程序中，静态初始化一般书写一组已知的无规律数值，这样书写起来比较简单，格式比较统一。

6.2.2.2 动态初始化

动态初始化，也就是只为数组指定长度，并且在内存中申请空间。动态初始化可以不必和数组的声明放在一起，也可以重新初始化一个初始化的数组。

动态初始化的语法格式：

```
数据类型[] 数组名称 = new 数据类型[长度];
```

例如：

```
int[] m = new int[10];
char[] c;
n = new char[3];
```

动态初始化使用 new 关键字进行初始化，new 关键字后续的数据类型要求和数组声明时的数据类型一样，中括号内部是需要初始化的数组长度，该长

度值可以是数字也可以是整型变量，如果是整型变量则不能为 long 型。在实际使用时，也可以先声明再进行动态初始化。

动态初始化指定了数组的长度，在内存中申请了对应长度的空间，而每个元素的值取数组数据类型对应的默认值。默认值的规定如下：

a、 boolean 类型的默认值是 false。

b、 其它 7 种基本数据类型是 0。说明：char 的默认值是编码为 0 的字符，而不是字符 0。

c、 复合数据类型的初始值是 null。

动态初始化只专注于为数组申请对应长度的空间，具体存储的元素的值可以根据需要依次进行指定。

6.2.3 引用数组元素

数组是一组数的集合，在实际使用时还需要引用数组中的每个元素。则引用数组中元素的语法格式为：

数组名称[下标]

其中下标指数组中每个元素的索引值，Java 语法规定数组中的第一个元素索引值是 0，第二个是 1，依次类推。在程序书写时，下标位置既可以书写常数也可以书写变量。而整个引用元素的表达式可以看作是一个变量，该变量的类型和数组的类型一致。

示例代码如下：

```
int[] m = {3,2,4,6};
m[1] = 4;
m[2] = m[3] + m[0];
```

在代码中，可以使用变量作为下标，示例代码如下：

```
char[] ch = new char[10];
int i = 2;
ch[i] = ‘a’;
```

使用变量作为数组的下标，极大的增强了数组元素使用的灵活性，也是灵活使用数组必须深刻理解的内容。

因为数组的下标都从 0 开始，所以有效的数组下标区间是 0 到数组的长度减 1，其它的下标都是非法的。在代码中出现非法的下标不会出现语法错误，但是会导致运行时出现异常。

6.2.4 获得数组长度

为了方便的操作数组，Java 语法中提供了获得数组长度的语法格式。对于一个已经初始化完成的数组，获得该数组长度的语法格式为：

数组名称.length

示例代码如下：

```
int[] n = {1,2,3,4,6};
```

```
int len = n.length;
```

则在该代码中 n.length 代表数组 n 的长度，由数组的初始化可以看出数组 n 的长度是 5，则变量 len 的值将是 5。使用该语法，可以只需要知道数组的名称就可以获得数组的长度，便于灵活操作数组。

综合前面下标的语法和长度的语法，则输出数组 n 中所有元素的代码为：

```
for(int i = 0;i < len;i++){
        System.out.println(n[i]);
}
```

这种使用数组的方式称作数组的遍历，遍历数组是使用数组的基础，也是很多和数组相关逻辑实现的基础。

关于数组的语法就介绍这么，下面通过一些示例来演示数组的实际使用。

Java 编程那些事儿 45—数组使用示例 1

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.3 数组使用示例

本部分通过一系列的示例，熟悉数组的基本语法，并演示一些使用数组可以解决的基本问题。在实际使用数组时，数组的长度以及数组中每个元素存储的数据，都根据逻辑需要进行设计。

6.3.1 循环初始化数组元素

要求：初始化一个长度为 100 的 int 数组，数组中的元素依次为 1-100。

这是一个基本的使用，主要是熟悉数组的语法，在实际编写时，需要发现数组下标和数组元素值之间的规律即可。

第一种思路：循环数组的下标 0-99，则和下标对应的元素的值比数组下标多 1。

则实现代码如下：

```
int[] m = new int[100];
for(int i = 0;i < m.length;i++){
        m[i] = i + 1;
}
```

代码说明：声明并初始化一个长度是 100 的数组，使用循环循环数组的下标，下标的区间是[0,m.length-1]，其中 m.length 表示数组的长度。在实际赋值时，把数组的下标做成变量，则当 i 的值为 0 时，则 m[i]就是 m[0]，依

次类推，按照题目的要求，则数值的规律是 i+1，这样循环结束以后，数组 m 中的值就依次是 1-100 了。

第二种思路：循环数组的值 1-100，则下标比数组的值下 1。

则实现的代码如下：

```
int[] m = new int[100];
for(int i = 1;i <= 100;i++){
        m[i - 1] = i;
}
```

该代码中循环变量 i 的值从 1 循环到 100，则数组的下标是 i - 1，这样也可以实现题目要求的功能。

6.3.2 输出数列

要求：输出 1 1 2 3 5 8 13……这样的数列，输出该数列的前 20 个数字。

该题是一个基本的数字逻辑，在实际解决该问题时，首先要发现该数字的规律，然后按照该规律来设计数组即可。

实现思路：数字的规律是除了数列里的前两个数字以外，其它的数字都满足该数字等于前两个数字的和，由于题目要求输出前 20 个数字，所以需要一个长度为 20 的数组，第一个和第二个数字直接赋值，后续的数字通过前两个数字元素得到。

则实现的代码如下：

```
int[] num = new int[20];
num[0] = 1;
num[1] = 1;
//循环初始化
for(int i = 2;i < num.length;i++){
        num[i] = num[i - 1] + num[i - 2];
}
//循环输出
for(int i = 0;i < num.length;i++){
        System.out.print(num[i]);
        System.out.print( ‘ ‘);
}
System.out.println(); //换行
```

在该代码中，初始化一个长度为 20 的数组，首先将数组中的前两个元素赋值成 1，然后循环对后续的元素的赋值，如果当前元素的下标是 i，则它前一个元素的下标是 i-1，再前面一个元素的下标是 i-2，只需要将这 2 个元素的值相加，然后赋值给当前元素即可。后面使用一个循环，输出数组中所有的元素，元素和元素之间有一个间隔的空格，在输出所有的元素以后换行。

6.3.3 歌手打分

要求：在歌唱比赛中，共有10位评委进行打分，在计算歌手得分时，去掉一个最高分，去掉一个最低分，然后剩余的8位评委的分数进行平均，就是该选手的最终得分。如果已知每个评委的评分，求该选手的得分。

该题实际上涉及到求数组的最大值、最小值，以及求数组中所有元素的和，也是数组方便统计的用途体现。

实现思路：求出数组元素的最大值、最小值以及和，然后使用和减去最大值和最小值，然后除以8获得得分。

则实现的代码如下：

```
int[] score = {90,78,90,96,67,86,78,92,79,85}; //评委打分
int sum = 0;        //存储和
int max =score[0];  //存储最大值
int min = score[0]; //存储最小值
//求和
for(int i = 0;i < score.length;i++){
        sum += score[i];
}
//获得最大值
for(int i = 1;i < score.length;i++){
        //比较
                        if(max < score[i]){
max = score[i];
        }
}
//获得最小值
for(int i = 1;i < score.length;i++){
        //比较
                        if(min > score[i]){
min = score[i];
        }
}
//计算平均分
double avg = (sum - max - min)/8.0;
System.out.println(avg);
```

在该代码中，实现数组求和的思路和以前的一样，就是每次加一个元素，然后用得到的结果再和后续的元素依次相加。求最大值的思路是首先假设第一个元素最大，把score[0]赋值给max，然后使用max的值和后续依次比较，如果后续的元素比max大，则把该值赋值给max，然后再和后续的元素比较，求最小值的思路和最大值的思路一样。然后计算平均分就完成了题目的要求。

该代码虽然结构比较清晰，但是效率不高，为了追求效率，可以把以上三个循环合并起来，代码如下：

```
for(int i = 0;i < score.length;i++){
        sum += score[i]; //求和
        //获得最大值
                if(max < score[i]){
max = score[i];
        }
        //获得最小值
                if(min > score[i]){
min = score[i];
        }
}
```

这样虽然在结构上稍微复杂了一些，但是效率得到了改善。在实际编写程序时，一般会在功能书写完成以后，对代码进行优化，提高程序的执行效率。

Java 编程那些事儿 46—数组使用示例 2

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.3.4 判断数组元素是否重复

要求：判断一个数组中是否存在相同的元素，如果存在相同的元素则输出“重复”，否则输出“不重复”。

该题中如果需要判断数组中元素是否重复，则需要对数组中的元素进行两两比较，如果有任意一组元素相等，则该数组中的元素存在重复，如果任意一组元素都不想等，则表示数组中的元素不重复。

实现思路：假设数组中的元素不重复，两两比较数组中的元素，使用数组中的第一个元素和后续所有元素比较，接着使用数组中的第二个元素和后续元素比较，依次类推实现两两比较，如果有一组元素相同，则数组中存储重复，结束循环。把比较的结果存储在一个标志变量里，最后判断标志变量的值即可。

则实现的代码如下：

```
int[] n = {1,2,3,1,0};
boolean flag = true;   //假设不重复
 for(int i = 0;i < n.length - 1;i++){  //循环开始元素
        for(int j = i + 1;j < n.length;j++){ //循环后续所有元素
                //如果相等，则重复
                if(n[i] == n[j]){
```

```
                                        flag = false; //设置标
志变量为重复
                                        break;      //结束循环
                                }
                        }
                }
                //判断标志变量
                if(flag){
                        System.out.println(“不重复”);
                }else{
                        System.out.println(“重复”);
                }
```

在该代码中，flag 变量存储是否重复，true 代表不重复，false 代表重复。外部循环中循环变量 i 代表第一个元素的下标，内部循环中循环变量 j 代表后续元素的下标，当 i 为零时和后续所有元素比较，然后当 i 为 1 时也和后续所有元素比较，依次类推，这样实现所有元素之间的两两比较。然后如果元素相同，则代表有重复，把 flag 变量的值置成 flase，结束循环。最后根据 flag 变量的值就可以判断是否重复了。

6.3.5 判断数组是否对称

要求：判断数组元素是否对称。例如{1}、{1,2,0,2,1}，{1,2,3,3,2,1}这样的都是对称数组。

该题中用于判断数组中的元素关于中心对称，也就是说数组中的第一个元素和最后一个元素相同，数组中的第二个元素和倒数第二个元素相同，依次类推，如果比较到中间，所有的元素都相同，则数组对称。

实现思路：把数组长度的一半作为循环的次数，假设变量 i 从 0 循环到数组的中心，则对应元素的下标就是数组长度-i-1，如果对应的元素有一组不相等则数组不对称，如果所有对应元素都相同，则对称。

则实现的代码如下：

```
        int[] n = {1,2,0,2,1};
         boolean flag = true; //假设对称
          for(int i = 0;i < n.length/2;i++){ //循环数组长度
的一半次
                //比较元素
                if(n[i] != n[n.length - i - 1]){
                        flag = false;   //不对称
                        break;       //结束循环
                }
         }
        if(flag){
                System.out.println(“对称”);
```

```
}else{
        System.out.println(“不对称”);
}
```

在该代码中，flag 作为标志变量，值为 true 代表对称，false 代表不对称，因为是两两比较，只需要比较数组的长度一半次即可，如果对应的元素不相同则数组不对称，结束循环。最后判断标志变量的值，就可以获得数组是否对称了。

6.3.6 数制转换

要求：将十进制数字转换为二进制数字。

在前面介绍过，十进制数字转换为二进制数字时一般使用除二取余法，该方法很规则，在程序中可以通过循环实现，在程序中只需要把得到的数字存储起来即可。

实现思路：将除二取余得到的第一个数字存储在数组中第一个元素，第二次得到的余数存储在数组中第二个元素，依次类推，最后反向输出获得的数字即可。

实现代码如下：

```
int n = 35;
int[] m = new int[32];
//拆分数字
int num = 0;
while(n != 0){
        m[num] = n % 2; //存储余数
        num++;          //拆分数字增加 1
        n /= 2;         //去掉余数
}
//输出拆分后的数字
for(int i = num - 1;i >= 0;i--){
        System.out.print(m[i]);
}
System.out.println();
```

在该代码中，因为 int 是 32 位的，所以最多需要长度是 32 的数组即可。在存储时把拆分出的第一个数字，也就是二进制的低位，存储在数组的第一个元素，num 代表拆分出的数字的个数以及数组下标，一直拆分到 n 的值为零时结束。循环结束后，因为拆分出来的数字个数是 num，所以只需要反向输出数组中 0 到 num-1 下标的元素即可。

Java 编程那些事儿 47—数组使用示例 3

郑州游戏学院 陈跃峰

出自：**http://blog.csdn.net/mailbomb**

6.3.7 数字统计

要求：统计一个整数中出现最多的数字。如果数字个数一样，则以最大的数字为准，例如 1 输出 1，121 输出 1，23231 输出 3。

该题是一个综合的题目，在实际分析时可以分解成三个问题：1、把整数中的每个数字拆分出来，2、统计拆分出的数字中 0-9 每个的个数，3、获得数字个数的最大值。

实现思路：

1、 拆分数字：整数和 10 取余可以获得该整数的个位值，然后用该整数除以 10 可以去掉个位(整数除法)，按照这种结构实现循环，并把拆分出的数字(也就是余数)存储到数组中。

2、 统计数字：声明一个长度是 10 的整型数组，使用这个数组中的第一个元素保存数字 0 出现的次数，第二个元素保存数字 1 出现的次数，依次类推。使用循环实现数字个数的统计。

3、 获得最大值对应的数字：获得个数数组中最大值的下标，就是需要的数字。

则实现的代码如下：

```
int m = 1232312;
 int[] n = new int[10]; //存储拆分后的数字
 int num = 0; //存储拆分出的数字个数
 while(m != 0){ //未拆分完
          n[num] = m % 10;    //获得个位数字
          num++;              //拆分出的数字个数加 1
          m /= 10;            //去掉拆分出的数字
 }
  int[] count = new int[10]; //存储 0-9 数字出现的次数
 //统计数字出现的次数
 for(int i = 0;i < num;i++){
          count[n[i]]++;
 }
 //获得最大值的下标
 int index = 0;
 for(int i = 0;i < count.length;i++){
          if(count[index] <= count[i]){
                   index = i;
          }
 }
 //输出
 System.out.println(index);
```

在该代码中，拆分的十进制的数字，首先拆分出个位，并存储到 n 数组中，然后

通过除 10 去掉拆分出的数字，继续执行循环，一直运算到 m 为 0 时为止，变量 num 保存拆分出的数字的个数。使用数组 count 记忆 0-9 每个数字出现的次数，count[0]存储 0 出现的次数，count[1]存储 1 出现的次数,依次类推，所以当 n[i]的值为几时，只需要 count[n[i]]增加 1 即可。最后使用循环获得最大数字的下标，适用<=进行比较，可以保证当个数相同时取后续的数字，这样就可以通过循环获得最大数值的下标，按照数组 count 的结构，数组的下标和就是数字的值。

6.3.8 数组编码

要求：设有一数组 A，长度是 N，内部的数据是 0 到 N-1 之间的所有数字，例如当 N 等于 5 时，数组为：A={0,3,2,1,4}。针对 A 数组，有一个对应的编码数组 B，B 的长度和 A 的长度相等，规定数组 B 中元素的值规定如下：

a、 B[0]的值为 0

b、 B[i]的值是 A 数组中 A[i]以前的值中比 A[i]小的元素的个数。

c、 例如示例中 A 数组{0,3,2,1,4}对应的编码数组 B 的值为{0,1,1,1,4}。

现在已知 A 数组，编码代码计算对应的编码数组 B。

该题是一个基本的数组变换题目，只要熟悉了题目的要求以后，按照题目的要求求解对应的数组 B 即可。

实现思路：初始化一个长度和 A 数组一样的 B 数组，初始化第一个元素的值为 0，循环统计比 A[i]元素小的数字个数，把个数值赋值给对应的 B[i]即可。

则实现的代码如下：

```
int[] A = {0,3,2,1,4};
int[] B = new int[A.length];
B[0] = 0; //初始化第一个元素，可选
for(int i = 1;i < A.length;i++){
        int count = 0; //计数变量
        //统计小于 A[i]元素的数量
        for(int j = i - 1;j >= 0;j--){
                if(A[j] < A[i]){
                        count++;
                }
        }
        B[i] = count;   //赋值
}
```

该代码中，按照数组 B 中值的规定，统计 A[i]以前比 A[i]小的元素个数，然后把得到的结果赋值给 B[i]即完成题目的要求。

6.3.9 数组排序

要求：将数组中的元素按照从小到大的顺序(升序)进行排列。

数组的排序是实现很多数组操作的基础，在实际使用时也有很多的排序方法，这里以冒泡排序为例来说明数组的排序算法。

实现思路：每次排序一个元素，总的排序次数是数组的长度减 1 次。第一次时，首先比较第一个和第二个元素，如果第一个元素比第二个元素大，则交换这两个元素的值，然后比较第二个和第三个元素，如果第二个比第三个大则交换，依次类推，这样当第一次交换完成以后，数组中的最后一个元素一定是数组中最大的元素。第二次时，只比较数组的前长度减一个元素，比较步骤和第一次相同，依次类推，每次都少比较一个元素，最终获得的就是排序完成的数组。

则实现的代码如下：

```
int[] m = {2,10,3,4,2};
for(int i = 0;i < m.length - 1;i++){ //排序次数
        //两两比较，实现排序
        for(int j = 0;j < m.length - 1 - i;j++){
                if(m[j] > m[j + 1]){
                        //交换
                        int temp = m[j];
                        m[j] = m[j + 1];
                        m[j + 1] = temp;
                }
        }
}
//输出排序后的元素
for(int i = 0;i < m.length;i++){
        System.out.println(m[i]);
}
```

冒泡排序通过数组中元素的两两比较和交换，实现数组中元素的排序。其中循环变量为 i 的循环代表排序的次数，总的排序次数是数组的长度减 1 次。内部的循环变量为 j 的循环实现未排序元素的两两比较，其中循环条件可以保证 i 增加 1，内部比较的元素减少 1，这个在功能上就是不比较排过序的元素。

Java 编程那些事儿 48—多维数组基础

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.4 多维数组基础

在学校里，由于一个班的人数不多，所以按照顺序编号即可，当人数增多时，例如对于学校里的人，在编号时就要增加层次，例如 XX 班 XX 号。在部队中也是这样，XX 师 XX 团 XX 营 XX 连 XX 排 XX 班，这里的层次就比较深了。为了管理数据的方便，一般要加深管理的层次，这就是多维数组的由来。

多维数组，指二维以及二维以上的数组。二维数组有两个层次，三维数组有三个层次，依次类推。每个层次对应一个下标。

在实际使用中，为了使结构清晰，一般对于复杂的数据都是用多维数组。

关于多维数组的理解，最终的是理解数组的数组这个概念，因为数组本身就是一种复合数据类型，所以数组也可以作为数组元素存在。这样二维数组就可以理解成内部每个元素都是一维数组类型的一个一维数组。三维数组可以理解成一个一维数组，内部的每个元素都是二维数组。无论在逻辑上还是语法上都支持“数组的数组”这种理解方式。

通常情况下，一般用二维数组的第一维代表行，第二维代表列，这种逻辑结构和现实中的结构一致。

和一维数组类似，因为多维数组有多个下标，那么引用数组中的元素时，需要指定多个下标。

6.5 多维数组语法

下面以二维数组为例，来介绍多维数组的语法。

6.5.1 多维数组声明

多维数组的声明：

```
数据类型[][] 数组名称;
数据类型[] 数组名称[];
数据类型   数组名称[][];
```

以上三种语法在声明二维数组时的功能是等价的。同理，声明三维数组时需要三对中括号，中括号的位置可以在数据类型的后面，也可以在数组名称的后面，其它的依次类推。

例如:

```
int[][] map;
char c[][];
```

和一维数组一样，数组声明以后在内存中没有分配具体的存储空间，也没有设定数组的长度。

6.5.2 多维数组初始化

和一维数组一样，多维数组的初始化也可以分为静态初始化(整体赋值)和动态初始化两种，其语法格式如下。

6.5.2.1 静态初始化

以二维数组的静态初始化为例，来说明多维数组静态初始化的语法格

式。示例代码如下：

```
int[][] m = {
        {1,2,3},
        {2,3,4}
};
```

在二维数组静态初始化时，也必须和数组的声明写在一起。数值书写时，使用两个大括号嵌套实现，在最里层的大括号内部书写数字的值。数值和数值之间使用逗号分隔，内部的大括号之间也使用逗号分隔。

由该语法可以看出，内部的大括号其实就是一个一维数组的静态初始化，二维数组只是把多个一维数组的静态初始化组合起来。

同理，三维数组的静态初始化语法格式如下：

```
int[][][] b = {
        {
                {1,2,3},
                {1,2,3}
        },
        {
                {3,4,1},
                {2,3,4}
        }
};
```

说明：这里只是演示语法格式，数值本身没有意义。

6.5.2.2 动态初始化

二维数组动态初始化的语法格式：

```
数据类型[][] 数组名称 = new 数据类型[第一维的长度][第二维的长度];
数据类型[][] 数组名称;
 数组名称 = new 数据类型[第一维的长度][第二维的长度];
```

示例代码：

```
byte[][] b = new byte[2][3];
int m[][];
m = new int[4][4];
```

和一维数组一样，动态初始化可以和数组的声明分开，动态初始化只指定数组的长度，数组中每个元素的初始化是数组声明时数据类型的默认值。例如上面初始化了长度为 2X3 的数组 b，和 4X4 的数组 m。

使用这种方法，初始化出的第二维的长度都是相同的，如果需要初始化第二维长度不一样的二维数组，则可以使用如下的格式：

```
int n[][];
n = new int[2][]; //只初始化第一维的长度
//分别初始化后续的元素
```

```
n[0] = new int[4];
n[1] = new int[3];
```

这里的语法就体现了数组的数组概念，在初始化第一维的长度时，其实就是把数组 n 看成了一个一维数组，初始化其长度为 2，则数组 n 中包含的 2 个元素分别是 n[0]和 n[1]，而这两个元素分别是一个一维数组。后面使用一维数组动态初始化的语法分别初始化 n[0]和 n[1]。

6.5.3 引用数组元素

对于二维数组来说，由于其有两个下标，所以引用数组元素值的格式为：

```
数组名称[第一维下标][第二维下标]
```

该表达式的类型和声明数组时的数据类型相同。例如引用二维数组 m 中的元素时，使用 m[0][0]引用数组中第一维下标是 0，第二维下标也是 0 的元素。这里第一维下标的区间是 0 到第一维的长度减 1，第二维下标的区间是 0 到第二维的长度减 1。

6.5.4 获得数组长度

对于多维数组来说，也可以获得数组的长度。但是使用数组名.length 获得的是数组第一维的长度。如果需要获得二维数组中总的元素个数，可以使用如下代码：

```
int[][] m = {
          {1,2,3,1},
          {1,3},
          {3,4,2}
};
int sum = 0;
for(int i = 0;i < m.length;i++){ //循环第一维下标
          sum += m[i].length;      //第二维的长度相加
}
```

在该代码中，m.length 代表 m 数组第一维的长度，内部的 m[i]指每个一维数组元素，m[i].length 是 m[i]数组的长度，把这些长度相加就是数组 m 中总的元素个数。

Java 编程那些事儿 49—多维数组使用示例 1

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.6 多维数组使用示例

多维数组在实际使用时，更多的在于数组的设计，在实际使用中，一般对于多维数组的统计相对来说比一维数组要少一些，更多的设计数组的大小，并规定数组中存储值的含义，在代码中按照值的规定使用数组。

所以在实际使用多维数组以前，需要考虑清楚：

1 需要几维数组
1 每一维的长度是多少
1 按照怎样的规则存储值
1 数组值的意义是什么

6.6.1 拉丁方阵

要求：实现任意阶拉丁矩阵的存储和输出

拉丁矩阵是一种规则的数值序列，例如 4 阶的拉丁矩阵如下所示：

1 2 3 4
2 3 4 1
3 4 1 2
4 1 2 3

该矩阵中的数字很规则，在实际解决该问题时，只需要把数值的规律描述出来即可。

实现思路：声明一个变量 n，代表矩阵的阶，声明和初始化一个 nXn 的数组，根据数据的规律，则对应的数值为(行号 + 列号 + 1)，当数值比 n 大时，取和 n 的余数。

实现的代码如下：

```
                int n = 6;
                 int[][] arr = new int[n][n];
                 int data; //数值
                 //循环赋值
                 for(int row = 0;row < arr.length;row++){
                                          for(int  col  =  0;col  <
arr[row].length;col++){
                                    data = row + col + 1;
                                    if(data <= n){
                                              arr[row][col] = data;
                                    }else{
                                              arr[row][col] = data %
n;
                                    }
                          }
                 }
```

```
                //输出数组的值
                for(int row = 0;row < arr.length;row++){
                                        for(int col = 0;col < arr[row].length;col++){
                                System.out.print(arr[row][col]);
                                System.out.print(' ');
                        }
                        System.out.println();
                }
```

该代码中变量 data 存储行号+列号+1 的值，每次在赋值时判别 data 的值是否小于等于 n，根据判断的结果赋值对应数组元素的值。

在解决实际问题时，观察数字规律，并且把该规律使用程序进行表达，也是每个程序员需要的基本技能。

6.6.2 杨辉三角

要求：实现 10 行杨辉三角元素的存储以及输出。

杨辉三角是数学上的一个数字序列，该数字序列如下：

```
                1
1          1
1          2 1
1          3 3 1
1          4 6 4 1
```

该数字序列的规律为，数组中第一列的数字值都是 1，后续每个元素的值等于该行上一行对应元素和上一行对应前一个元素的值之和。例如第五行第二列的数字 4 的值，等于上一行对应元素 3 和 3 前面元素 1 的和。

实现思路：杨辉三角第几行有几个数字，使用行号控制循环次数，内部的数值第一行赋值为 1，其它的数值依据规则计算。假设需要计算的数组元素下标为(row,col)，则上一个元素的下标为(row - 1,col)，前一个元素的下标是(row - 1,col - 1)。

实现代码如下：

```
        int[][] arr = new int[10][10];
        //循环赋值
        for(int row = 0;row < arr.length;row++){
                for(int col = 0;col <= row;col++){
                        if(col == 0){ //第一列
                                arr[row][col] = 1;
                        }else{
                                        arr[row][col] = arr[row - 1][col] + arr[row - 1][col - 1];
                        }
                }
```

```
}
//输出数组的值
for(int row = 0;row < arr.length;row++){
        for(int col = 0;col <= row;col++){
                System.out.print(arr[row][col]);
                System.out.print(' ');
        }
        System.out.println();
}
```

该题目中数字之间的规律比较简单，主要是理解数组的下标基本的处理，加深对于数组下标的认识，控制好数组元素的值。

Java 编程那些事儿 50—多维数组使用示例 2

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.6.3 存储图形结构

要求：根据数组中的值，在对应位置绘制指定的字符。规定 0 绘制空格，1 绘制星号(*)。数组的值如下所示：

```
{
        {0,0,0,1,0,0,0},
        {0,0,1,0,1,0,0},
        {0,1,0,0,0,1,0},
        {1,0,0,0,0,0,1},
        {0,1,0,0,0,1,0},
        {0,0,1,0,1,0,0},
        {0,0,0,1,0,0,0}
}
```

该题目是一个基本的数组应用，数组中的值存储的是控制信息，程序根据数组中的值实现规定的功能。

实现思路：循环数组中的元素，判断数组中的值，根据值绘制对应的字符即可。

实现的代码如下所示：

```
int[][] map = {
        {0,0,0,1,0,0,0},
        {0,0,1,0,1,0,0},
        {0,1,0,0,0,1,0},
        {1,0,0,0,0,0,1},
        {0,1,0,0,0,1,0},
```

```
                                {0,0,1,0,1,0,0},
                                {0,0,0,1,0,0,0}
                        };
                        //输出数组的值
                        for(int row = 0;row < map.length;row++){
                                                for(int col = 0;col <
map[row].length;col++){
                                switch(map[row][col]){
                                case 0:
                                        System.out.print(' ');
                                        break;
                                case 1:
                                        System.out.print('*');
                                        break;
                                }
                        }
                        System.out.println();
                }
```

类似的代码在游戏开发中，可以用来代表游戏中的地图数据，或者俄罗斯方块等益智游戏中地图块的值。

6.6.4 螺旋数组

要求：存储和输出 nXm 的螺旋数组，其中 n 和 m 为大于 0 的整数。

以下是一些螺旋数组的示例：

```
1          2  3  4                    1  2  3  4  5
12 13 14 5                   14  15  16  17 6
11  16 15 6                    13  20  19  18  7
10 9  8  7                   12  11  10   9  8
```

4X4 螺旋数组　　4X5 螺旋数组

对于螺旋数组来说，其中的数值很有规则，就是按照旋转的结构数值每次加 1，实现该功能需要对数组和流程控制有角深刻的认识。

实现思路：声明一个变量来代表需要为数组元素赋的值，对于其中的数字来说，每个数字都有一个移动方向，这个方向指向下一个元素，根据该方向改变数组的下标，如果到达边界或指向的元素已经赋值，则改变方向。

实现代码如下：

```
                int n = 4;
                 int m = 5;
                 int[][] data = new int[n][m];
                 int dire;   //当前数字的移动方向
                 final int UP = 0;   //上
                 final int DOWN = 1; //下
```

```
                final int LEFT = 2; //左
                final int RIGHT = 3;//右
                dire = RIGHT;
                int value = 1;    //数组元素的值
                int row = 0;     //第一维下标
                int col = 0;     //第二维下标
                data[0][0] = 1; //初始化第一个元素
                while(value < n * m){
                        switch(dire){
                                case UP:
                                        row--; //移动到上一行
                                         if(row < 0){ //超过边
界
                                                row++; //后退
                                                dire = RIGHT;
                                                 continue; //
跳过该次循环
                                                        }else
if(data[row][col] != 0){//已赋值
                                                row++; //后退
                                                dire = RIGHT;
                                                 continue; //
跳过该次循环
                                        }
                                        break;
                                case DOWN:
                                        row++; //移动到下一行
                                        if(row >= n){ //超过边
界
                                                row--; //后退
                                                dire = LEFT;
                                                 continue; //
跳过该次循环
                                                        }else
if(data[row][col] != 0){//已赋值
                                                row--; //后退
                                                dire = LEFT;
                                                 continue; //
跳过该次循环
                                        }
                                        break;
                                case LEFT:
```

```
                                        col--; //移动到前一列
                                         if(col < 0){ //超过边
界
                                                col++; //后退
                                                dire = UP;
                                                 continue; //
跳过该次循环
                                                        }else
if(data[row][col] != 0){//已赋值
                                                col++; //后退
                                                dire = UP;
                                                 continue; //
跳过该次循环
                                        }
                                        break;
                                case RIGHT:
                                        col++; //移动到后一行
                                        if(col >= m){ //超过边
界
                                                col--; //后退
                                                dire = DOWN;
                                                 continue; //
跳过该次循环
                                                        }else
if(data[row][col] != 0){//已赋值
                                                col--; //后退
                                                dire = DOWN;
                                                 continue; //
跳过该次循环
                                        }
                                        break;
                        }
                        value++; //数值增加1
                        data[row][col] = value;//赋值
                }
                //输出数组中的元素
                for(int i = 0;i < data.length;i++){
                        for(int j = 0;j < data[i].length;j++){
                                if(data[i][j] < 10){//右对齐
                                        System.out.print(' ');
                                }
                                System.out.print(data[i][j]);
```

```
                System.out.print(' ');
            }
            System.out.println();
        }
```

在该代码中 dire 代表当前元素的移动方向，每个根据该变量的值实现移动，如果移动时超出边界或移动到的位置已赋值，则改变方向，并跳过本次循环，如果移动成功，则数值增加 1，对数组元素进行赋值。

对于多维数组来说，更多的是设计数组的结构，并根据逻辑的需要变换数组的下标，实现对于多维数组元素的操作。

Java 编程那些事儿 51—多维数组练习

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

6.7 数组综合练习

1、计算两个矩阵 A、B 的乘积矩阵 C。

矩阵 A＝{1,2,3,4,5,6}；

矩阵 B＝{7,8,9,10,11,12}。

两个矩阵的乘积仍然是矩阵。若 A 矩阵有 m 行 p 列，B 矩阵有 p 行 n 列,则它们的乘积 C 矩阵有 m 行 n 列。C=A*B 的算法：

Cij= (i=0，1，……，m-1；j=0，1，……，n-1)

设 A、B、C 矩阵用 3 个 2 维数组表示：a 数组有 3 行 2 列，b 数组有 2 行 3 列，则 c 数组有 3 行 3 列。

如：

```
c[0][0]= a[0][0]*b[0][0]+a[0][1]*b[1][0];
c[1][0]= a[1][0]*b[0][1]+a[1][1]*b[1][1];
```

2、计算并输出 nXn 的蛇形矩阵。(n>0)

例如 4X4 的蛇形矩阵如下：

```
1  3  4 10
2  5  9 11
6  8  12 15
7 13 14 16
```

3、使用 1-9 这 9 个数字填充一个 3X3 的数组，要求输出所有可能的情况。

备注：希望大家积极补充。

6.8 数组综合练习部分答案

star 发表于 2008 年 11 月 26 日 14:57:03 * 矩阵 A＝{1,2,3,4,5,6}；

* 矩阵 B＝{7,8,9,10,11,12}。

* 两个矩阵的乘积仍然是矩阵。若 A 矩阵有 m 行 p 列，

* B 矩阵有 p 行 n 列,则它们的乘积 C 矩阵有 m 行 n 列。C=A*B 的算法：

* Cij= (i=0，1，……，m-1；j=0，1，……，n-1)

* 设 A、B、C 矩阵用 3 个 2 维数组表示：a 数组有 3 行 2 列，b 数组有 2 行 3 列，则 c 数组有 3 行 3 列。

* 如：

* c[0][0]= a[0][0]*b[0][0]+a[0][1]*b[1][0]；

* c[1][0]= a[1][0]*b[0][1]+a[1][1]*b[1][1]；

*/

int m=6,p=1,n=6;

int[][] A=new int[m][p],B=new int[p][n],C=new int[m][n];

for(int i=0;i for(int j=0;j

A[i][j]=i+1;

for(int i=0;i

for(int j=0;j B[i][j]=j+7;

for(int i=0;i for(int j=0;j for(int k=0;k

C[i][j]+=A[i][k]*B[k][j];

for(int i=0;i {

for(int j=0;j {

if(C[i][j]<10)

System.out.print(" ?);

举报

/**1、计算两个矩阵 A、B 的乘积矩阵 C。

```
*      矩阵 A＝{1,2,3,4,5,6}；
*      矩阵 B＝{7,8,9,10,11,12}。
* 两个矩阵的乘积仍然是矩阵。若 A 矩阵有 m 行 p 列，
* B 矩阵有 p 行 n 列,则它们的乘积 C 矩阵有 m 行 n 列。C=A*B
的算法：
*    Cij= (i=0，1，……，m-1；j=0，1，……，n-1)
* 设 A、B、C 矩阵用 3 个 2 维数组表示：a 数组有 3 行 2 列，b
数组有 2 行 3 列，则 c 数组有 3 行 3 列。
*      如：
*      c[0][0]= a[0][0]*b[0][0]+a[0][1]*b[1][0]；
*      c[1][0]= a[1][0]*b[0][1]+a[1][1]*b[1][1]；
*/
int m=6,p=1,n=6;
int[][] A=new int[m][p],B=new int[p][n],C=new int[m]
[n];
for(int i=0;i<m;i++)
for(int j=0;j<p;j++)
A[i][j]=i+1;
for(int i=0;i<p;i++)
for(int j=0;j<n;j++)
B[i][j]=j+7;
for(int i=0;i<m;i++)
for(int j=0;j<n;j++)
for(int k=0;k<p;k++)
C[i][j]+=A[i][k]*B[k][j];
for(int i=0;i<m;i++)
{
for(int j=0;j<n;j++)
```

```
{
if(C[i][j]<10)
System.out.print(" ");
System.out.print(C[i][j]);
System.out.print(" ");
}
System.out.println(" ");
}
```

star 发表于 2008 年 11 月 26 日 15:11:40 0)

```
* 例如 4X4 的蛇形矩阵如下：
* 1 3 4 10 1 3 4 10 11
* 2 5 9 11 2 5 9 12 19
* 6 8 12 15 6 8 13 18 20
* 7 13 14 16 7 14 17 21 24
* 15 16 22 23 25
*/
//DOWN UPRIGHT RIGHT LEFTDOWN
int n=5;
int[][] data = new int[n][n];
int dire; //当前数字的移动方向
final int UPRIGHT = 0; //上右
final int DOWN = 1; // 下
final int LEFTDOWN= 2; //左下
final int RIGHT = 3;// 右
dire = DOWN;
int value = 1; //数组元素的值
int row = 0; //第一维下标
int col = 0; //第二维下标
```

```
 data[0][0] = 1; //初始化第一个元素
 data[n-1][n-1] = n*n;
 while(value < n * n-1){
 System.out.print(" (?+dire+?)?);
 switch(dire){
case DOWN://DOWN -UPRIGHT +RIGHT -LEFTDOWN
row++; //移动到下一行
if(row>=n){ //超过边界
row--; //后退
dire = RIGHT;
continue; //跳过该次循环
}
else
{
value++; //数值增加 1
文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/05/26137
36.aspx 发表时间:2008 年 11 月 26 日 15:11:40">举报
```

```
/**2、计算并输出 nXn 的蛇形矩阵。(n>0)
* 例如 4X4 的蛇形矩阵如下:
* 1 3 4 10 1 3 4 10 11
* 2 5 9 11 2 5 9 12 19
* 6 8 12 15 6 8 13 18 20
* 7 13 14 16 7 14 17 21 24
* 15 16 22 23 25
*/
//DOWN UPRIGHT RIGHT LEFTDOWN
int n=5;
int[][] data = new int[n][n];
```

```
int dire; //当前数字的移动方向
final int UPRIGHT = 0; //上右
final int DOWN = 1; // 下
final int LEFTDOWN= 2; //左下
final int RIGHT = 3;// 右
dire = DOWN;
int value = 1; //数组元素的值
int row = 0; //第一维下标
int col = 0; //第二维下标
data[0][0] = 1; //初始化第一个元素
data[n-1][n-1] = n*n;
while(value < n * n-1){
System.out.print("("+dire+")");
switch(dire){
case DOWN://DOWN -UPRIGHT +RIGHT -LEFTDOWN
row++; //移动到下一行
if(row>=n){ //超过边界
row--; //后退
dire = RIGHT;
continue; //跳过该次循环
}
else
{
value++; //数值增加 1
```

star 续 2 发表于 2008 年 11 月 26 日 15:15:52

```
if(col==0)
{
dire = UPRIGHT;
}
```

```
else if(col==n-1)
{
dire = LEFTDOWN;
}
}
break;
case UPRIGHT://+DOWN -UPRIGHT +RIGHT LEFTDOWN
row--;col++; //移动到上一行，右一列
if(col>=n)//超过边界
{
row++;col--; //后退
dire = DOWN;
continue; //跳过该次循环
}
else if(row<0)
{ //超过边界
row++;col--; //后退
dire = RIGHT;
continue; //跳过该次循环
}
else
{
value++; //数值增加 1
data[row][col] = value;//赋值
dire = UPRIGHT;
}
break;
case RIGHT://+DOWN -UPRIGHT RIGHT -LEFTDOWN
```

```
col++; //移动到右一列
if(col>=n){ / 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/05/2613736.aspx 发表时间:2008 年 11 月 26 日 15:15:52">举报
        data[row][col] = value;//赋值
        if(col==0)
        {
        dire = UPRIGHT;
        }
        else if(col==n-1)
        {
        dire = LEFTDOWN;
        }
        }
        break;
        case UPRIGHT://+DOWN -UPRIGHT +RIGHT LEFTDOWN
        row--;col++; //移动到上一行，右一列
        if(col>=n)//超过边界
        {
        row++;col--; //后退
        dire = DOWN;
        continue; //跳过该次循环
        }
        else if(row<0)
        { //超过边界
        row++;col--; //后退
        dire = RIGHT;
        continue; //跳过该次循环
        }
```

```
        else
        {
        value++; //数值增加 1
        data[row][col] = value;//赋值
        dire = UPRIGHT;
        }
        break;
        case RIGHT://+DOWN -UPRIGHT RIGHT -LEFTDOWN
        col++; //移动到右一列
        if(col>=n){ /
```

---

star 续 2 发表于 2008 年 11 月 26 日 15:18:19

```
 dire = DOWN;
 continue; //跳过该次循环
 }
 else
 {
 value++; //数值增加 1
 data[row][col] = value;//赋值
 if(row==0)
 {
 dire = LEFTDOWN;
 }
 else if(row==n-1)
 {
 dire = UPRIGHT;
 }
 }
 break;
 case LEFTDOWN://+DOWN UPRIGHT +RIGHT -LEFTDOWN
```

```
row++;col--; //移动到下一行，左一列
if(row>=n){ //超过边界
row--;col++; //后退
dire = RIGHT;
continue; //跳过该次循环
}else if(col<0)//超过边界
{
row--;col++; //后退
dire = DOWN;
continue; //跳过该次循环
} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/05/2613736.aspx 发表时间:2008 年 11 月 26 日 15:18:19">举报
            col--; //后退
            dire = DOWN;
            continue; //跳过该次循环
            }
            else
            {
            value++; //数值增加 1
            data[row][col] = value;//赋值
            if(row==0)
            {
            dire = LEFTDOWN;
            }
            else if(row==n-1)
            {
            dire = UPRIGHT;
            }
```

```
}
break;
case LEFTDOWN://+DOWN UPRIGHT +RIGHT -LEFTDOW
N
row++;col--; //移动到下一行，左一列
if(row>=n){ //超过边界
row--;col++; //后退
dire = RIGHT;
continue; //跳过该次循环
}else if(col<0)//超过边界
{
row--;col++; //后退
dire = DOWN;
continue; //跳过该次循环
}
```

---

star 续 2 发表于 2008 年 11 月 26 日 15:19:33 {

```
value++; //数值增加 1
data[row][col] = value;//赋值
dire = LEFTDOWN;
}
break;
}
}
System.out.println();
//输出数组中的元素
for(int i = 0;i < data.length;i++){
for(int j = 0;j < data[i].length;j++){
if(data[i][j] < 10){//右对齐
```

```
System.out.print(' ');
}
System.out.print(data[i][j]);
System.out.print(' ');
}
System.out.println();
} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/05/2613736.aspx 发表时间:2008 年 11 月 26 日 15:19:33">举报
        else
        {
        value++; //数值增加 1
        data[row][col] = value;//赋值
        dire = LEFTDOWN;
        }
        break;
        }
        }
        System.out.println();
        //输出数组中的元素
        for(int i = 0;i < data.length;i++){
        for(int j = 0;j < data[i].length;j++){
        if(data[i][j] < 10){//右对齐
        System.out.print(' ');
        }
        System.out.print(data[i][j]);
        System.out.print(' ');
        }
        System.out.println();
```

}

star 发表于 2008 年 11 月 26 日 15:22:48 // 3、使用 1-9 这 9(v)个数字填充一个 3X3(mxn)的数组，要求输出所有可能的情况。

```
 //这里将其矩阵拉直以简化
 int v[]=new int[]{1,2,3,4};//,5,6,7,8,9};
 int a[][]=new int[factorial(v.length)][v.length];
 a=fill_in(v.length,v);
// 输出数组中的元素
 for(int i = 0;i < factorial(v.length);i++){
 for(int j = 0;j < v.length;j++){
 if(a[i][j] < 10){//右对齐
 System.out.print(' ');
 }
 System.out.print(a[i][j]);
 System.out.print(' ');
 }
 System.out.println();
 }
 System.out.println(" ='==="+a.length);
```

举报

```
public static void main(String[] args){
// 3、使用 1-9 这 9(v)个数字填充一个 3X3(mxn)的数组，要求输出所有可能的情况。
//这里将其矩阵拉直以简化
int v[]=new int[]{1,2,3,4};//,5,6,7,8,9};
int a[][]=new int[factorial(v.length)][v.length];
a=fill_in(v.length,v);
// 输出数组中的元素
```

```
for(int i = 0;i < factorial(v.length);i++){
for(int j = 0;j < v.length;j++){
if(a[i][j] < 10){//右对齐
System.out.print(' ');
}
System.out.print(a[i][j]);
System.out.print(' ');
}
System.out.println();
}
System.out.println("===="+a.length);
}
static int[][] fill_in(int f,int[] v)
{
if(f==1)
{
int[][] temp=new int[1][1];
temp[0][0]=v[0];
//// 输出数组中的元素
// for(int i = 0;i < temp.length;i++){
// for(int j = 0;j < temp[i].length;j++){
// if(temp[i][j] < 10){//右对齐
// System.out.print(' ');
// }
// System.out.print(temp[i][j]);
// System.out.print(' ');
// }
//
```

star 续 3 发表于 2008 年 11 月 26 日 15:25:49 ¿

举报

```
// System.out.println("===="+temp.length);
return temp;
}
else
{
int[][] temp=fill_in(f-1, v );
int[][] b=new int[factorial(f)][f];
for(int i = 0;i <factorial(f-1);i++)//
for(int k = 0;k < f;k++)//插入位置
{
for(int j = 0;j < k;j++)
{
b[factorial(f-1)*k+i][j]=temp[i][j];
}
b[factorial(f-1)*k+i][k]=v[f-1];
for(int j = k+1;j < f;j++)
{
b[factorial(f-1)*k+i][j]=temp[i][j-1];
}
}
//// 输出数组中的元素
// for(int i = 0;i < b.length;i++){
// for(int j = 0;j < b[i].length;j++){
// if(b[i][j] < 10){//右对齐
// System.out.print(' ');
// }
```

```
// System.out.print(b[i][j]);
// System.out.print(' ');
// }
// System.out.println();
// }
// System.out.println("===="+b.length);
```

star 续 3 发表于 2008 年 11 月 26 日 15:28:06 举报

```
return b;
}
}
// 计算 n 的阶乘 factorial。
static int factorial(int n)
{
int nj=1;
for(int i=1;i<=n;i++)
nj*=i;
return nj;
}
```

Java 编程那些事儿 52—方法声明

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

第七章 方法

方法(method)，在面向过程的语言中称作函数(function)，在汇编语言中称作子程序，是一个代码功能块，实现某个特定的功能。在实际的程序开发中，方法是一种基础的组织代码的方式。本部分就介绍方法相关的概念、相关语法以及实际使用时需要注意的问题。

7.1 方法概述

方法的概念来源于数学上的函数，在数学中，当数据具有一定的规律时，就是用一个函数来代码该数字的规律，例如 f(n)=n 则代表 1、2、3、……这样的一个数列。在数学上 n 是参数，对于确定的 n 值只有一个 f(n)的值和它对应。

方法是一组为了实现特定功能的代码块的集合。方法在语法上的功能主要有以下两个：

l 结构化代码

将代码按照功能进行组织，使代码的结构比较清晰，容易阅读和修改，也就是程序的可维护性强。

l 减少代码重复

一个固定的功能，可能会在程序中多次使用，在使用时只需要调用写好的方法，而不用重复书写对应的功能代码。

方法在书写时需要注意以下两点：

l 逻辑严谨

方法实现的一个完整的功能，所以在书写时要考虑到各种可能的情况，并对每种情况做出恰当的处理。

l 通用性强

方法实现的是一种功能，在实际实现时，可以根据需要，使方法具备一定的通用性，除非必要，否则不要写专用的方法。

在 Java 语言中，恰当的使用方法，将使程序更加优雅，便于阅读和使用。下面就来介绍方法声明的语法格式。

7.2 方法声明

方法声明写在代码中类声明的内部，方法声明的外部，伪代码如下：

```
public class Hello{
        方法声明 1
        方法声明 2
        ……
}
```

在 Java 语言中，方法声明之间没有顺序。

方法声明，就是声明一种新的功能，或者说创造一种新的功能。例如以下是一个求 int 数据绝对值的方法声明代码：

```
public int abs(int n){
        if(n > 0){
                return n;
        }else{
                return  -n;
        }
```

}

这里就实现了求 int 值绝对值的功能，为了使该功能通用，使用一个参数 n 代表需要求绝对值的数值，在方法内部使用求绝对值的逻辑：正数的绝对值是自身，负数的绝对值是相反数，使用 return 语句将方法运算的结果返回。

具体方法声明的语法格式如下：

访问控制符［修饰符］返回值类型 方法名称(参数列表){
方法体
}

在实际声明一个方法时，需要依次确定以上内容。下面是具体的说明：

1、 访问控制符

访问控制符限定方法的可见范围，或者说是方法被调用的范围。方法的访问控制符有四种，按可见范围从大到小依次是：public、protected，无访问控制符，private。其中无访问控制符不书写关键字即可。具体的范围在后续有详细介绍。

2、 修饰符

修饰符是可选的，也就是在方法声明时可以不书写。

修饰符是为方法增加特定的语法功能，对于方法实现的逻辑功能无影响。方法的访问控制符有五个,依次是：

u static——静态的
u final——最终的
u abstract——抽象的
u synchronized——同步的
u native——本地的

具体修饰符的作用在后续内容中将详细介绍。

3、 返回值类型

返回值类型是指方法功能实现以后需要得到的结果类型，该类型可以是 Java 语言中的任意数据类型，包括基本数据类型和复合数据类型。如果方法功能实现以后不需要反馈结果，则返回值类型书写为 void。

在实际书写方法时，需要首先考虑一下方法是否需要反馈结果，如果反馈结果，则结果的类型是什么？这个根据方法的需要进行确定，例如上面求绝对值的方法，int 类型的绝对值还是 int 类型，所以把返回值类型做成 int 型。

在方法声明里声明返回值类型，便于方法调用时获得返回值，并对返回值进行赋值以及运算等操作。

4、 方法名称

方法名称是一个标识符，用来代表该功能块，在方法调用时，需要方法名称来确定调用的内容。

为了增强代码的可读性，一般方法名称标识符和该方法的功能一直，例如实现数组排序的方法，可以将方法名称设定为 sort。

在 Java 编码规范中，要求方法的首字母小写，而方法名称中单词和单词间隔的第一个字母大写，例如 bubbleSort。

5、 参数列表

参数列表是声明方法需要从外部传入的数据类型以及个数，例如上面求 int 类型绝对值的方法，每次需要从外部传入一个 int 类型的值，这就需要在参数列表部

分进行声明，语法格式为：

数据类型 参数名称

多个参数时的格式为：

数据类型 参数名称 1,数据类型 参数名称 2,……

声明参数时，类型在前，名称在后，如果有多个参数时，参数和参数之间使用逗号进行分割。

参数的值在方法调用时进行指定，而在方法内部，可以把参数看作是已经初始化完成的变量，直接进行使用。

参数列表部分是方法通用性的最主要实现部分，理论上来说，参数越多，方法的通用性越强，在声明方法时，可以根据需要确定参数的个数，以及参数的类型。参数在参数列表中的排列顺序只和方法调用时有关。

6、 方法体

方法体是方法的功能实现代码。方法体部分在逻辑上实现了方法的功能，该部分都是具体的实现代码，不同的逻辑实现代码区别会比较大。

在方法体部分，如果需要返回结果的值，则可以使用 return 语句，其语法格式为：

```
return 结果的值;
```

或无结果返回时：

```
return;
```

如果方法的返回值类型不是 void，则可以使用 return 返回结果的值，要求结果值的类型和方法声明时返回值类型必须一致。如果返回值类型是 void 时，可以使用 return 语句实现方法返回，而不需要返回值。当代码执行到 return 语句时，方法结束，所以 return 语句的后续书写顺序的代码，例如：

```
return 0;
int n = 0; //语法错误，永远无法执行到该语句
```

另外，如果返回值类型不是 void 时，需要保证有值返回，例如下面的方法就有语法错误：

```
public int test(int a){
        if(a < 0){
                return 0;
        }
}
```

则该方法的声明代码中，当 a 的值大于等于零时，则没有返回值，这语法上称作返回值丢失，这个也是在书写 return 语句时需要特别注意的问题。

Java 编程那些事儿 53—方法声明示例

郑州游戏学院 陈跃峰

出自：http://blog.csdn.net/mailbomb

7.3 方法声明示例

方法实现的是功能，在实际声明方法时，不仅要根据需要确定访问控制符、修饰符、返回值类型、方法和参数列表这些信息，还要按照功能要求的逻辑实现方法体的代码。在实际设定时每个内容都需要根据功能的结构选择最恰当的内容。

下面通过一系列的示例来演示如何进行选择和设定。

7.3.1 判断某个整数是否是偶数

功能要求：判断一个整数是否是偶数

简单分析：为了实现判断任意的整数，需要在判断时从外部传入一个整数，在方法声明时，需要将该整数声明为参数。判断的结果是偶数或不是偶数，只有 2 个状态，可以使用能够代表 2 个状态的数据类型进行代表，最直观的就是 boolean 类型了。

该方法实现的代码如下：

```
public boolean isEven(int n){
        return n % 2 == 0;
}
```

根据逻辑的需要，访问控制符选择 public，修饰符为空，返回值类型做成 boolean，参数列表部分传入一个整型的参数， 这样方法声明的结构就做好了。

偶数的判断，只需要判断一下余数是否为零即可，如果余数为零则成立，否则不成立，直接把比较表达式的值作为方法的返回值返回。

7.3.2 数组排序

功能要求：实现整数数组数据从小到大(升序)的排序

简单分析：为了实现通用性，需要每次传递需要排序的数组进入方法内部，所以方法声明中需要有一个整型数组参数。为了直观，可以把排序以后的数组返回。说明：随着后续的学习，这个返回值不是必须的。

该方法实现的代码如下：

```
public int[] bubbleSort(int[] m){
    for(int i = 0;i < m.length - 1;i++){
        for(int j = 0;j < m.length - 1 - i;j++){
            if(m[j] > m[j + 1]){
                int temp = m[j];
                m[j] = m[j + 1];
                m[j + 1] = temp;
            }
```

```
            }
        }
        return m;
    }
```

在该方法内部，使用冒泡法实现数组的排序，最后将排序完成的数组作为返回值反馈回来。在实际使用时，可以将数组看成是一种普通的数据类型，也可以作为方法的返回值以及参数列表中的类型进行使用。

Java 编程那些事儿 54—方法调用

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

7.4 方法调用

方法声明是创建一个新的功能，声明出来的方法在需要的时候可以通过调用执行该方法的功能，方法只有被调用才能被执行。

在 Java 程序中有一个基本的 main 方法，其方法声明如下：

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

该方法包含在开始的代码框架中，这是一个特殊的方法。Java 语法规定，J2SE 的代码都从该方法开始执行，如果一个代码中没有 main 方法，则该代码不能被直接运行。所以 main 方法也称作 J2SE 程序的入口方法，在运行程序时，自动调用对应代码中的 main 方法开始程序的执行。

由于上面讲述的原因，所以一个方法如果需要得到执行，则需要直接或间接在 main 方法中进行调用。

在调用方法时，程序的执行流程会进入方法的内部，当执行到方法内部的 return 语句或执行完方法内部的代码以后，则返回到调用该方法的位置继续向下执行。

方法调用的语法分为以下两种：

1 一个类内部的方法调用

指调用以及被调用的方法都在一个类的内部。

1 不同类之间的方法调用

指调用以及被调用的方法位于不同的类内部。

由于类的概念现在还没有涉及到，所以这里指讲一个类内部的方法调用语法，关于不同类之间的方法调用则在后续的章节中进行介绍。

7.4.1 一个类内部方法调用语法

在前面的代码框架中，以下代码就是声明类的结构：

```
public class 文件名{
```

在该声明后续大括号内部的代码，称作一个类的内部。

在进行方法调用时，调用的语法格式和 static 修饰符有关，所以按照一个方法在声明时是否有 static 修饰分为两类：

l 有 static 修饰的称作静态方法

l 没有 static 修饰的称作非静态方法

这样一个类内部的方法调用就存在四种情况：

l 在非静态方法内部调用非静态方法

l 在非静态方法内部调用静态方法

l 在静态方法内部调用静态方法

l 在静态方法内部调用非静态方法

其中前三种情况的调用都是直接调用，直接调用的语法格式为：

方法名(参数 1 值,……);

这里方法名为被调用的方法名称，后续紧跟一对小括号，括号内部依次书写调用该方法时传入参数的值，语法上要求传入参数的个数、每个参数的类型都必须和方法声明时保持一致。而这里调用的表达式就代表方法的返回值，可以根据需要使用返回值进行赋值。

示例代码如下：

```
public class CallMethod{
        public static void main(String[] args){
                int a = 10;
                int b = 2;
                int c = 3;
                int d = 32;
                max(a,b); //只比较，比较以后返回值丢失
                int n = max(5,a); //比较，并把返回值赋值给变量 n
                int m = max(c,d); //比较，并把返回值赋值给变量 m
                //比较 a、b、c、d 四个数字的最大值
                int maxNumber = max(max(a,b),max(c,d));
        }

        public static int max(int a,int b){
                if(a > b){
                        return a;
                }else{
                        return b;
                }
        }
}
```

该示例在静态的 main 方法内部调用静态的 max 方法，因为 max 方法在声明时参数列表为 2 个 int 的参数，则调用的时候必须传入 2 个 int 值，可以是

int 的变量也可以是 int 数值。则调用的基本格式为：

max(参数值 1,参数值 2)

调用的表达式可以在代码中单独成行，当方法的返回值类型不是 void 时，可以接收方法的返回值，也可以不接收。方法的返回值是一个确定类型的值，所以在以上比较时，方法调用之间可以进行嵌套。

```
int maxNumber = max(max(a,b),max(c,d));
```

其中 max(a,b)是获得 a 和 b 的最大值，max(c,d)是获得 c 和 d 的最大值，然后比较两个获得的最大值，该代码的功能和以下代码的功能相同：

```
int maxNumber = max(a,max(b,max(c,d)));
```

总得来说，前三种情况的调用都是使用该种格式。

对于最后一种情况，也就是在静态方法内部调用非静态的结构在语法上则比较复杂，以下是一个简单的示例：

```
public class CallMethod2{
        public static void main(String[] args){
                CallMethod2 cm = new CallMethod2();
                int n = cm.max(1,2);
        }

        public int max(int a,int b){
                if(a > b){
                        return a;
                }else{
                        return b;
                }
        }
}
```

在该代码涉及的语法格式在后续的代码中将进行讲解，这里只做简单的说明。其中：

```
CallMethod2 cm = new CallMethod2();
```

该行代码声明并创建了一个CallMethod2类型的对象cm。接着的代码：

```
int n = cm.max(1,2);
```

在调用 max 方法时，则使用对象名.方法名调用对应的方法，这里参数的规则和上面介绍的相同。

7.4.2 方法调用后的执行流程

在方法调用时，程序的执行流程和以前的结构就有所区别。简单说，就是当遇到方法调用时，程序的执行流程将跳转到被调用的方法内部，直到被调用的方法返回时，才从调用的位置继续向下执行。

以下是一个演示调用过程的示例代码：

```
public class CallMethod3{
```

```
        public static void main(String[] args){
                System.out.println(1);
                printTest();
                System.out.println(2);
                max(10,20);
                System.out.println(3);
        }

        public static int max(int a,int b){
                System.out.println("进入 max 方法内部！");
                if(a > b){
                        return a;
                }else{
                        return b;
                }
        }

        public static void printTest(){
                 System.out.println("进入 printTest 方法内部！");
                int a = 10;
                 System.out.println("printTest 方法执行完毕!");
        }
}
```

则该代码执行时的输出如下：

1.进入 printTest 方法内部！printTest 方法执行完毕!

2. 进入 max 方法内部！

3. 从程序执行的输出就可以清晰的看出方法调用时代码的执行顺序。

Java 编程那些事儿 55—方法重载和参数传递

郑州游戏学院 陈跃峰

出自：http://blog.csdn.net/mailbomb

7.5 方法相同

在 Java 语言中，方法相同的概念和其它程序设计语言不尽相同，Java 语言中的方法相同指方法名称和参数列表都相同，其中参数列表相同指参数个数、参数类型和参数排列顺序等相同，参数名称可以不相同。相同的方法访问控

制符、返回值类型可以不相同。

以下是一下相同的方法：

```
public void test(int a,double[] d)

private int test(int i,double[] d1)
```

在同一个类内部，不能声明相同的方法，否则将出现语法错误。

7.6 方法重载

方法重载(overload)是一种语法现象，指在一个类内部出现了多个方法名相同，但是参数列表不同的方法。

方法重载的作用是将功能相同，但是参数列表不同的方法采用相同的方法名称，便于程序员使用。根据方法相同的概念，重载的方法都是不相同的方法。

在 Java 提供的 API 中，大量应用重载的概念，方便程序员对于系统功能方法的实际使用。

恰当的使用重载，可以增强代码的可维护性。

以下是方法重载的示例：

```
public void a(int a){}

public int  a(){}

public void a(int a,String s){}
```

在以上示例方法中，方法的名称都是 a，而参数列表却各不相同，这些方法实现了重载的概念。但是仔细观察可以发现，这些重载的方法的返回值不尽相同，因为返回值类型和方法的重载无关。

通常情况下，重载的方法在访问控制符、修饰符和返回值类型上都保持相同，这个不是语法的要求，只是将这些制作成一致以后，便于实际使用。

7.7 参数传递

在方法调用时，需要根据方法声明传入适当的参数，通过每次调用方法时传参，极大的增强了方法的统一性，避免了方法内部功能代码的重复。但是在实际传递参数时，如果在方法内部修改了参数的值，则调用时使用的变量是否发生改变呢？

例如如下代码：

```
/**
 * 参数传递代码示例
 */
public class TransferValueDemo {
        public static void main(String[] args) {
                int m = 10;
                int[] a = {1,2,34};
                test(m,a);
                System.out.println(m);
                System.out.println(a[0]);
        }
        public static void test(int n,int[] t){
                n = 0;
                t[0] = 123;
        }
}
```

则执行该程序以后，程序的输出结果是：10　123。则在调用 test 方法时，同样都是传入参数，为什么变量 m 的值未改变，而 a[0]的值发生了改变呢？下面就来说明该问题。

在参数传递时，一般存在两种参数传递的规则，在 Java 语言中也是这样，这两种方式依次是：

l　按值传递(by value)

按值传递指每次传递参数时，把参数的原始数值拷贝一份新的，把新拷贝出来的数值传递到方法内部，在方法内部修改时，则修改的时拷贝出来的值，而原始的值不发生改变。

说明：使用该方式传递的参数，参数原始的值不发生改变。

l　按址传递(by address)

按址传递指每次传递参数时，把参数在内存中的存储地址传递到方法内部，在方法内部通过存储地址改变对应存储区域的内容。由于在内存中固定地址的值只有一个，所以当方法内部修改了参数的值以后，参数原始的值发生改变。

说明：使用该方式传递的参数，在方法内部修改参数的值时，参数原始的值也发生改变。

在 Java 语言中，对于那些数据类型是按值传递，那些数据类型是按址传递都作出了硬性规定，如下所示：

l　按值传递的数据类型：八种基本数据类型和 String

l　按址传递的数据类型：除 String 以外的所有复合数据类型，包括数组、类和接口

按照这里的语法规则，则上面的代码中变量 m 的类型是 int，属于按值传递，所以在方法内部修改参数的值时 m 的值不发生改变，而 a 的类型是数组，属于按址传递，所以在方法内部修改参数的值时，原始的值发生了改变。

按值传递和按址传递在实际使用时，需要小心，特别是在方法内部需要修改参数的值时。有些时候，对于按值传递的参数需要修改参数的值，或者按址传递时，不想修改参数的值，下面是实现这两种方式时的示例代码。

按值传递时通过返回值修改参数的值：

```
/**
 * 按值传递的类型通过返回值修改参数的值
 */

public class TransferValueDemo1 {

                public static void main(String[] args) {

                        int m = 10;

                        m = test1(m);  //手动赋值

                        System.out.println(m);

                }

                public static int test1(int n){

                        n = 15;

                        return n;

                }

}
```

在该示例代码中，通过把修改以后的参数 n 的值返回，来为变量 m 赋值，强制修改按值传递参数的值，从而达到修正参数值的目的。

按址传递时通过重新生成变量避免修改参数的值：

```
/**
 * 按址传递时通过重新生成变量避免修改参数的值
 */

public class TransferValueDemo2 {

                        public static void main(String[] args) {

                                int[] a = {1,2,3};

                                test2(a);
```

```
                System.out.println(a[0]);
        }
        public static void test2(int[] m){
                int[] n = new int[m.length];
                        for(int i = 0;i < m.length;i++){
                                n[i] = m[i];
                }
                n[0] = 10;
        }
}
```

在该示例代码中，通过在方法内部创新创建一个数组，并且把传入数组中每个参数的值都赋值给新创建的数组，从而实现复制数组内容，然后再修改复制后数组中的值时，原来的参数内容就不发生改变了。

这里系统介绍了 Java 语言中参数传递的规则，深刻理解这些规则将可以更加灵活的进行程序设计。例如使用复合数据类型按址传递的特性可以很方便的实现多参数的返回，代码示例如下：

```
public int test3(int[] m,int[] n){……}
```

则该方法中，实际上返回了三种值，一个 int 的返回值，数组 m 的值，数组 n 的值，这只是参数传递的一种基本使用，在 JDK 提供的 API 文档中也大量的存在该方法。

7.8 小结

本部分系统的介绍了方法的概念，方法的语法，方法的调用以及和方法有关的一些其它语法知识，深刻理解这些语法是进行程序设计的基础。

Java 编程那些事儿 56—方法练习

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

7.9 方法练习

要求：编写对应功能的方法，并通过调用测试方法的功能是否实现。

1、 求byte类型绝对值的方法。分别使用以下数据测试方法功能实现是否完善、合理：-10、0、100、-128。

2、 将小写字符转换为大写字符，如果对应的字符不是小写字符，则返回自身。测试数据：’a’、’F’、’2’。

3、 求整型数组中所有元素的和。测试数据：{1}、{0,3,2,12}。

4、 将小数四舍五入成整数的方法。测试数据：0.1、1.7

5、 将数组按照升序(从小到大)进行排列的方法。测试数据：{3,10,9,2,3}

6、 打印1-100之间所有的偶数，每行显示10个数字，数字右对齐。

7、 将对应的数字字符(如’1’、’4’等)转换为对应数字(如1、4)的方法，如果不能转换，则返回-1。测试数据：’5’、’a’、’-‘。

7.10 方法练习部分答案

---

star 发表于 2008 年 11 月 27 日 22:06:27 +

"合理：-10、0、100、-128。");

System.out.println(absoluteValue(a1)+" "+absoluteValue(a2)+" "+

absoluteValue(a3)+" "+absoluteValue(a4));

}

static int absoluteValue(byte a)

{

```
int b=0;
if (a<0)
b=-a;
if(a<0)
b=-(a+1)+1;
return b;
} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/21/2682383.aspx 发表时间:2008 年 11 月 27 日 22:06:27">举报
```

```
/** 1、 求 byte 类型绝对值的方法。
* 分别使用以下数据测试方法功能实现是否完善、合理：-10、0、100、-128。
*/
byte a1=-10,a2=0,a3=100,a4=-128;
System.out.println("分别使用以下数据测试方法功能实现是否完善、" +
"合理：-10、0、100、-128。");
System.out.println(absoluteValue(a1)+" "+absoluteValue(a2)+" "+
absoluteValue(a3)+" "+absoluteValue(a4) );
}
static int absoluteValue(byte a)
{
int b=0;
if (a<0)
b=-a;
if(a<0)
b=-(a+1)+1;
return b;
```

}

star 发表于 2008 年 11 月 27 日 22:07:53 +

"则返回自身。测试数据：'a'、'F'、'2'。");

System.out.println(ansiUpperCase(a1)+" "+ansiUpperCase(a2)+" "

+

ansiUpperCase(a3));

}

static char ansiUpperCase(char a)

{

char c='!';

if(a>='a'&&a<='z')

c=(char) (a-32);

else if(a>='A'&&a<='Z')

c=a;

return c;

}

//3、 求整型数组中所有元素的和。测试数据：{1}、{0,3,2,12}。

int[] a1={1},a2={0,3,2,12};

System.out.println("求整型数组中所有元素的和。测试数据：{1}、{0,3,2,1

2}。");

System.out.println(intArraySum(a1)+" "+intArraySum(a2));

}

static int intArraySum(int[] a)

{

int sum=0;

for(int i=0;i<a.length;i++)

sum+=a[i];

return sum;

} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/21/2682383.aspx 发表时间:2008 年 11 月 27 日 22:07:53">举报

```
/**2、 将小写字符转换为大写字符，如果对应的字符不是小写字符，
* 则返回自身。测试数据：’a’、’F’、’2’。
*/
char a1='a',a2='F',a3='2';
System.out.println("将小写字符转换为大写字符，如果对应的字符不是小写字符，" +
"则返回自身。测试数据：’a’、’F’、’2’。");
System.out.println(ansiUpperCase(a1)+" "+ansiUpperCase(a2)+" "+
ansiUpperCase(a3));
}
static char ansiUpperCase(char a)
{
char c='!';
if(a>='a'&&a<='z')
c=(char) (a-32);
else if(a>='A'&&a<='Z')
c=a;
return c;
}
//3、 求整型数组中所有元素的和。测试数据：{1}、{0,3,2,12}。
int[] a1={1},a2={0,3,2,12};
System.out.println("求整型数组中所有元素的和。测试数据：{1}、{0,3,2,12}。");
System.out.println(intArraySum(a1)+" "+intArraySum(a
```

```
2));
}
static int intArraySum(int[] a)
{
int sum=0;
for(int i=0;i<a.length;i++)
sum+=a[i];
return sum;
}
```

star 发表于 2008 年 11 月 27 日 22:08:30 ；

```
System.out.println(roundTo(a1)+" "+roundTo(a2));
}
static long roundTo(double a)
{
long sum=0;
if((a*10)%10>=5)
sum=(long)a/1+1;
else

sum=(long)a/1;
return sum;
}
/**5、 将数组按照升序(从小到大)进行排列的方法。
* 测试数据：{3,10,9,2,3}
*/
int[] a={3,10,9,2,3};
System.out.println("将数组按照升序(从小到大)进行排列的方法。" +
"测试数据：{3,10,9,2,3}");
```

```
int[] b=ascendingSort(a);
for(int i=0;i<b.length;i++)
System.out.print(b[i]+" ");
}
static int[] ascendingSort(int[] a)
{
for(int i=0;i<a.length;i++)
for(int j=1;j<a.length-i;j++)
if(a[j-1]>a[j])
{
int temp=a[j];
a[j]=a[j-1];
a[j-1]=temp;
}
return a;
} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/21/2682383.aspx 发表时间:2008 年 11 月 27 日 22:08:30">举报
```

```
//4、 将小数四舍五入成整数的方法。测试数据：0.1、1.7
double a1=0.1,a2=1.7;
System.out.println("将小数四舍五入成整数的方法。测试数据：0.1、1.7");
System.out.println(roundTo(a1)+" "+roundTo(a2));
}
static long roundTo(double a)
{
long sum=0;
if((a*10)%10>=5)
sum=(long)a/1+1;
```

```
else

sum=(long)a/1;
return sum;
}
/**5、 将数组按照升序(从小到大)进行排列的方法。
* 测试数据：{3,10,9,2,3}
*/
int[] a={3,10,9,2,3};
System.out.println("将数组按照升序(从小到大)进行排列的方法。" +
"测试数据：{3,10,9,2,3}");
int[] b=ascendingSort(a);
for(int i=0;i<b.length;i++)
System.out.print(b[i]+" ");
}
static int[] ascendingSort(int[] a)
{
for(int i=0;i<a.length;i++)
for(int j=1;j<a.length-i;j++)
if(a[j-1]>a[j])
{
int temp=a[j];
a[j]=a[j-1];
a[j-1]=temp;
}
return a;
}
```

star 发表于 2008 年 11 月 27 日 22:09:10 ;

allEvenNumberPrint(a);

}

static void allEvenNumberPrint(int a)

{

int k=0;

for(int i=1;i<=a;i++)

if(i%2==0)

{

if(i<10)

System.out.print(" ");

System.out.print(" "+i);

k++;

if(k%10==0)

System.out.println();

}

return ;

} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/21/2682383.aspx 发表时间:2008 年 11 月 27 日 22:09:10">举报

//6、 打印 1-100 之间所有的偶数，每行显示 10 个数字，数字右对齐。

int a=100;

System.out.println("打印 1-100 之间所有的偶数，每行显示 10 个数字，数字右对齐。");

allEvenNumberPrint(a);

}

static void allEvenNumberPrint(int a)

{

```
int k=0;
for(int i=1;i<=a;i++)
if(i%2==0)
{
if(i<10)
System.out.print(" ");
System.out.print(" "+i);
k++;
if(k%10==0)
System.out.println();
}
return ;
}
```

---

star 发表于 2008 年 11 月 27 日 22:11:02 +

"如果不能转换，则返回-1。测试数据:'5'、'a'、'-'。");

System.out.print(numericCharacterToNumeric(a1)+" "+

numericCharacterToNumeric(a2)+" "+

numericCharacterToNumeric(a3));

}

static int numericCharacterToNumeric(char a)

{

int b=-1;

if(a>='0'&&a<='9')

b=a-48;

return b;

} 文章链接:http://blog.csdn.net/Mailbomb/archive/2008/07/21/2682383.aspx 发表时间:2008 年 11 月 27 日 22:11:02">举报

/**7、将对应的数字字符(如'1'、'4'等)转换为对应数字(如 1、4)的

```
方法，
* 如果不能转换，则返回-1。测试数据：’5’、’a’、’-‘。
*/
char a1='5',a2='a',a3='-';
System.out.println("将对应的数字字符(如’1’、’4’等)转换为对应
数字(如 1、4)的方法，" +
"如果不能转换，则返回-1。测试数据：’5’、’a’、’-‘。");
System.out.print(numericCharacterToNumeric(a1)+" "+
numericCharacterToNumeric(a2)+" "+
numericCharacterToNumeric(a3));
}
static int numericCharacterToNumeric(char a)
{
int b=-1;
if(a>='0'&&a<='9')
b=a-48;
return b;
}
```

Java 编程那些事儿 57—面向对象基础

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

第八章 面向对象

在程序中，最核心的是数据结构和算法，不同的程序需要根据需要设计不同的数据结构，然后依赖程序的功能以及数据结构设计对应的算法，这种设计方式是程序的底层设计，也就是解决具体的功能。

当程序项目复杂到一定程度时，就不仅要专注于底层的设计，更要对

程序的结构进行设计，面向对象(Object-Oriented，简称 OO)就是一种常见的程序结构设计方法。

面向对象思想的基础是将相关的数据和方法放在一起，组合成一种新的复合数据类型，然后使用新创建的复合数据类型作为项目的基础。

8.1 面向对象简介

前面介绍的有关 Java 语言的基础知识，只是程序的基础知识，而本章介绍的面向对象的相关知识，则是和设计有关的知识。

面向对象的设计方式采用的是从外到内的设计方式，先设计整个项目的结构，然后再根据关联关注内部的每个细节。再分解整个项目时，也是按照模块化进行分解的。就像要制造一辆汽车，面向对象的设计思路是这样的：首先汽车要生产发动机、变速箱等模块，然后再去考虑每个模块的具体实现。使用这种设计思路，把每个部分都模块化，便于将功能进行分解，可以开发更复杂的项目。

再将模块划分出来以后，然后就来设计每个具体的模块，再设计模块时，如果模块还很复杂，则可以继续进行分解。如果模块已经划分的足够细致了，那么就可以进行具体的设计了。

设计具体模块的方式是确定模块需要的核心数据的结构，以及该模块需要具备的功能，也就是本章一开始提到的数据结构和算法，使每个模块都成为一个独立的完整结构，可以向其它的模块提供对应的服务(功能)。

整个系统(项目)则通过模块之间的互相关联运转起来，而每个模块只需要开放一个接口给其它的模块即可。

上面提到的就是面向对象的设计方式，总结起来是两大部分：

l 模块划分

l 模块实现

在具体的面向对象编程(Object-Oriented Programm，简称 OOP)中，划分出来的每个模块一般称为类(class)，而模块内部的数据称为 field，一般称为属性，模块内部的功能一般称为方法(method)。

按照面向对象的设计方式，在实际的项目开发过程中，面向对象技术一般分为 3 个部分：

l 面向对象分析(Object-Oriented Analysis，简称 OOA)

该步骤按照面向对象的思考方式提取项目的需求信息，一般由系统分析员负责，本部分形成文档为《项目需求分析说明书》。

l 面向对象设计(Object-Oriented Design，简称 OOD)

该步骤按照《项目需求分析说明书》进行模块划分，以及进行模块的概要设计，一般由高级程序员负责，本部分形成文档为《项目概要设计说明书》。

l 面向对象编程(Object-Oriented Programm，简称 OOP)

该步骤按照《项目概要设计说明书》细化每个模块的结构，一般由程序员负责，本部分形成文档为《项目详细设计说明书》。

最后由编码员(Coder)按照《项目详细设计说明书》进行具体的编码。这个就是面向对象开发的标准过程的简单描述。

而实际的程序开发过程中，则更关注于 OOP 部分，也就是实际实现时的具体设计以及编码的问题。

面向对象技术除了这些设计方式以外，还有很多的概念和语法知识需要在编程时进行学习，下面以 Java 语言的语法为基础来介绍面向对象编程的内容。

Java 编程那些事儿 58—类(一)

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

8.2 类

类(class)是 Java 语言的最小编程单位，也是设计和实现 Java 程序的基础，本部分将深入介绍类的相关知识。

8.2.1 类的概念

类是一组事物共有特征和功能的描述。类是对于一组事物的总体描述，是按照面向对象技术进行设计时最小的单位，也是组成项目的最基本的模块。

类的概念是抽象的，类似于建筑设计中的图纸，是对于现实需要代表的具体内容的抽象。类只包含框架结构，而不包含具体的数据。所以类代表的是总体，而不代表某个特定的个体。

例如设计电脑(computer)这个类，电脑是一组事物，则该类中包含的常见特征如下：

l 类型：笔记本或台式机

l 内存容量

l 硬盘容量

l CPU 类型

l 屏幕尺寸

l 主板类型

对于每一台具体的电脑来说，每个特征都有自己具体的数值，或者说是将特征数据具体化，而类需要代表的是总体特征，只需要具备特征的类型和结构，不需要具有具体的数值，因为一般一组事物的某个特征的数值都是不尽相同的，但是都统一的具备该特征。

同理，如果设计登录模块中的用户(user)类，则该类中包含的常见特征如下：

l 用户名

l 密码

对于每个具体的用户来说，都有自己特定的用户名和密码，但是对于用户这个类来说，只需要具备用户名和密码这两个特征的类型和结构即可。从这里也可以很直观的体会到，类是抽象的，是一组事物共有特征的描述。

上面是对于类结构具体特征的描述，其实类中除了包含特征的描述以外，还可以包含该类事物共有的功能，这些功能也是类的核心内容。

例如电脑这个类，包含的基本功能有：
l 打开
l 关闭
用户这个类，包含的基本功能有：

1　登录
通过在类的内部包含共有的功能，使得每个类都可以在内部实现一些规定的功能，从而减少和外部的交互，降低整个项目的复杂度。

这就是面向对象技术中类的概念的基本描述，每个类就代表一组事物，通过基本的特征和功能实现该类事物在项目内部的表达。

以上是从设计角度理解类的概念，其实从语法角度理解类的概念也很简单，类就是一种复合数据类型，或者说是一种程序员设计的新类型。因为在实际开发中，程序员可以根据需要声明新的类，所以在面向对象的开发中，程序员可以根据需要设计新的数据类型——类，从而实现项目要求的功能。

把设计角度中类的概念，转换为语法角度类的概念，是每个面向对象技术初学者都必须经历的阶段，通过进行该转换，可以把虚拟的类的概念转换成具体的类的概念，也是面向对象技术入门的标志。

对于一组事物来说，共有的特征和功能有很多，在实际抽象成类时，只需要抽象出实际项目中的需要的属性和功能即可。

8.2.2 类的声明

类是一种复合数据类型，则声明一个类就相当于创建了一种新的数据类型,在面向对象技术中就通过不断的创建新的数据类型来增强程序可以代表的数据的能力。

类声明总体的语法格式如下：

```
访问控制符　[修饰符]　class　　类名{

        [属性声明]

        [方法声明]

        [构造方法声明]

    }
```

说明：该语法格式中中括号内部的部分为可选。

其中访问控制符用于限定声明的类在多大范围内可以被其它的类访问，主要有默认访问控制符(无关键字)和 public；修饰符用于增强类的功能，

使声明的类具备某种特定的能力；class 是声明类时使用的关键字；类名是一个标识符，用于作为新声明的类的名称，要求必须符合标识符的命名规范。注：在 Java 语言的编码规范中，类名第一个字母要求大写。

例如如下示例：

```
public class Computer{}

class User{}
```

接着的大括号内部用于声明类的内部结构，类内部一般包括三类声明，且这三类声明都是可选的。说明如下：

l 属性声明

用于代表共有特征

l 方法声明

用于代码共有功能

l 构造方法声明

用于初始化类的变量

下面是这些声明的详细说明。

8.2.2.1 属性声明

属性，有些翻译为域、字段等，属性是类内部代表共有特征的结构，或者可以把属性理解为类的某个具体特征，类通过一系列的属性来代表一种新的数据类型。对于类比较基础的理解就是通过多个属性组合成的新的数据类型，这也是复合数据类型的由来。

属性声明的语法格式如下：

```
访问控制符 [修饰符] 数据类型 属性名[=值];
```

属性的访问控制符限定该属性被访问的范围，包含如下四种：public、protected、默认的(无关键字)和 private，分别代表不同的访问限制，具体的限制范围后续将有详细说明。

修饰符用于使属性具备某种特定的功能。

数据类型为该属性的类型，可以是 Java 语言中的任意数据类型，也就是说，既可以是基本数据类型也可以是复合数据类型。

属性名是一个标识符，用于代表该属性的名称，在声明属性时的同时可以为该属性进行赋值。

示例格式为：

```
public  int  cpuType;

public  char  sex = ‘男’;
```

在实际声明属性时，也可以一次声明多个属性,例如：

```
public  int  x = 10, y = 20;
```

不过为了程序结构的清晰，一般书写为如下格式：

```
public  int  x = 10;

public  int  y = 20;
```

另外，属性的作用范围是类的内部，可以在类内部的任何位置引用属性，包括在方法和构造方法的内部，而不论属性是否声明在方法的上面。

总得来说，类就是通过一系列属性的组合成为一种新的数据类型，从而可以代表一种更复杂的结构，也相当于为程序员提供了一种组合已有数据类型形成新数据类型的方法，从而更直观的去代表代表需要表达的数据。

8.2.2.2 方法声明

方法在类的内部代表该类具有的共有功能，将这些功能以方法的形式放置在类的内部，可以在需要时进行调用。

方法的声明和前面讲解的方法一致，只是在类内部增加了属性以后，可以在方法内部直接进行访问，而不需要进行参数传递了。

关于方法的声明示例如下：

```
public class Box{
```

```
        int width;
        int height;
        int length;
        public int vol(){
            return width * height * length;
        }
    }
```

在该示例中，声明了一个名字为 Box 的类，假设使用这个类来代表箱子，其中包含三个属性：length、width 和 height，依次代表箱子的长宽高，则在该类内部包含一个基本的功能，求箱子体积的 vol 方法，该方法的功能是计算箱子的体积。

下面是关于上面提到的用户类的基本实现,示例代码如下：

```
    public class User{
        public String username;
        public String password;
        public boolean login(){
            逻辑代码
        }
    }
```

在User类的内部，包含用户名和密码这两个属性，根据需要选择 Java 语言提供的 String 字符串类型进行代表，然后在该类的内部声明登录的 login 方法，在方法内部根据逻辑书写对应的实现代码。

Java 编程那些事儿 59——类(二)

郑州游戏学院 陈跃峰
出自：**http://blog.csdn.net/mailbomb**

8.2.2.3 构造方法声明

构造方法(Constructor)，也称作构造函数、构建器等，是初学者在学习时最容易混淆的概念之一。下面首先讲述构造方法和方法的区别。

构造方法和方法无任何关系，只是在翻译时名称比较类似罢了。而且构造方法和方法的功能不同，声明和调用的语法也不相同。

构造方法的功能：实现类这种数据类型的变量的初始化。由于类是一种复合数据类型，而复合数据类型的变量也比较复杂，所以专门需要对该类变量进行初始化，则语法上提供了专门的结构——构造方法。这就是构造方法出现的原因。而方法实现的是逻辑的功能，对应于逻辑上的算法，更多的是实现程序逻辑。所以构造方法是语法结构，而方法是逻辑功能，两者之间根本无关。

构造方法声明的语法格式：

```
访问控制符 构造方法名称(参数列表){

        构造方法体;

}
```

在该语法中，访问控制符指声明的构造方法被访问的权限，构造方法名称是标识符，语法上要求构造方法的名称必须类名相同，后续小括号内部是参数列表，其语法格式和方法参数列表的语法格式相同。

下面是构造方法的示例：

```
public class Box{

        int length;

        int width;

        int height;
```

```
    public Box(){
        length = 10;
        width = 10;
        height = 10;
    }

    public Box(int l,int w,int h){
        length = l;
        width = w;
        height = h;
    }
}
```

在该Box类中，声明了两个构造方法，一个没有参数，一个包含三个int类型的参数。在没有参数的构造方法中，将三个属性的值都初始化为10。带参数的构造方法中，可以传递进来三个参数，然后在构造方法内部依次把参数的值赋值给属性。

通常情况下，构造方法的声明放在属性声明和方法声明的中间。

一个类内部的构造方法可以有任意多个，但是要求这些构造方法不能相同。因为在一个类内部构造方法的名称都是相同的，所以只要参数列表相同的构造方法都是相同的构造方法。例如以下两个构造方法是相同的：

```
public  Test(int a,int[] b){}
public  Test(int b,int[] a){}
```

如果一个类声明中不包含构造方法的声明，则系统会自动为该类添加一个构造方法，当然如果类中已声明了构造方法则系统不会添加，这个系统自动添加的构造方法一般被称为默认构造方法，其声明格式如下：

```
public 构造方法名称(){}
```

所以以下两个代码是相同的：

```
public class DefaultConstructor{
        int  a;
}
```

和

```
public class DefaultConstructor{
        int  a;
        public DefaultConstructor(){}
}
```

在第一个代码中，没有声明构造方法，则系统自动添加默认的构造方法，而第二个代码中声明的构造方法和默认构造方法的结构一致，所以两个代码在功能上是完全相同的。

总之，构造方法是系统提供的一个结构，该结构的功能是实现对于类的变量的初始化，可以根据逻辑的需要声明对应的构造方法，并在构造方法内部根据需要进行具体的初始化。

8.2.2.4 面向对象基础使用

下面以一个简单的示例来描述面向对象的基本使用，主要是类声明的相关语法，以及基础的类设计的知识。

使用面向对象的方式来描述房屋的结构，要求如下：

(1) 门：(颜色为红色、可以被推开和关闭)

(2) 窗户：(颜色为白色、有一块玻璃(透明色、可以卸下)、可以被推开和关闭)

(3) 地：(由 100 块地板砖组成)

(4) 地板砖：(黄色、长 1 米、宽 1 米)

说明：其中红色用 1 代替，白色用 2 代替，黄色用 3 代替 透明色用 0 代替。

在使用面向对象描述时，将其中的名词转换为类，将该类内部的特征转换为属性，将该类内部的功能转换为方法，在构造方法内部实现对于属性的初始化。

则在该要求中，抽象的类一共有 5 个：门、窗户、玻璃、地和地板砖。

其中的颜色作为对应类的属性，推开和关闭作为对应类的方法。前面介绍过类是一种数据类型，则可以声明类类型的变量，并可以将该变量作为类的属性，这种类和类的关系在面向对象中称作使用关系。则按照该思路实现的代码如下：

```
                /**
                 * 门的类
                 * 文件名：Door.java
                 */
public class Door {
                        /**颜色*/
                        int color;
                        public Door(){
                                color = 1;  //初始化颜色
                        }
                        public void open(){}
                        public void close(){}
}
/**
 * 窗的类
 * 文件名：Window.java
 */
public class Window {
        /**颜色*/
        int color;
        public Window(){
                color = 2;  //初始化颜色
        }
public void open(){}
        public void close(){}
}

/**
 * 玻璃类
 * 文件名：Glass.java
 */
public class Glass {
        /**颜色*/
        int color;
        public Glass(){
                color = 0; //初始化颜色
        }
        public void remove(){}
```

```
}
/**
 * 地板类
 * 文件名：Floor.java
 */
public class Floor {
	FloorBrick[] fb;
	public Floor(){
		fb = new FloorBrick[100];
		for(int i = 0;i < fb.length;i++){
			fb[i] = new FloorBrick();  //初始化每个元素
		}
	}
}

/**
 * 地板砖类
 * 文件名：FloorBrick.java
 */
public class FloorBrick {
	/**颜色*/
	int color;
	/**长度*/
	int length;
	/**宽度*/
	int width;
	public FloorBrick(){
		color = 3; //初始化颜色
		length = 1; //初始化长度
		width = 1; //初始化宽度
	}
}
```

说明：在Floor类的代码中涉及到对于类类型的变量初始化的问题，相关语法将在后续详细介绍。

在该示例中，使用面向对象的思想描述了要求的房屋结构，并以Java语言语法的格式将
面向对象的思想转化为具体的代码，从而实现对于面向对象技术的基本使用。

Java 编程那些事儿 60——对象

郑州游戏学院 陈跃峰
出自：**http://blog.csdn.net/mailbomb**

8.3 对象

对象(Object)是面向对象技术的核心，按照面向对象的思考方式，现实中的每个实体都是一个对象，比如一个人、数据库中的一张表等，总结起来，就是面向对象技术中的经典语句——万事万物皆对象。

8.3.1 什么是对象？

其实面向对象技术只是提供了一种思考的方式，其思考方式就是把一个复杂的结构看成是一个整体,这样可以降低认知的复杂性。比如认识一个电脑，按照面向对象的认知方式，就是先把电脑分成一个个的对象：显示器对象、硬盘对象、CPU 对象等等，然后再一个一个的进行认知。

同时面向对象技术也是一种设计方式，其设计方式是把一个负责的模块划分为一个个小的模块分开进行设计，这样可以降低设计的复杂性。比如设计一个电脑，按照面向对象的设计方式，就是把电脑分成一个个的对象：显示器对象、硬盘对象、CPU 对象等等，然后再一个一个的进行设计。

正因为面向对象无论在认知和设计方面都降低了复杂性，所以在程序设计语言中得到了广泛的应用。其实也就是是对现实已存在的内容的升华，所以面向对象存在于生活的很多方面，并不是计算机程序设计领域里的“阳春白雪”。

在语法角度来看，对象就是一个变量，只是该变量比较复杂，其内部既包含属性(数据)，也包含方法(功能)。在 Java 语言中，把复合数据类型(包括数组、类和接口)的变量都称作对象。所以对象的概念相对来说，就显得跟具体了。

每个对象中存储类中对应属性的数值，或者从数据角度来理解对象的话，也可以把对象看作是类似 C 语言中结构体变量类似的结构。

下面来介绍一下 Java 语言中对象相关的语法。

8.3.2 对象的语法

对象相关的语法主要包含四个部分：对象的声明、对象的初始化、引用对象中的属性和引用对象中的方法。

例如，有如下一个类的代码：

```
public  class  Box{
        /**长度*/
        int length;
        /**宽度*/
        int width;
        /**高度*/
        int height;
        public Box(){
                length = 10;
                width = 10;
                height = 10;
        }

        public Box(int l,int w,int h){
                length = l;
                width = w;
                height = h;
        }

        /**求体积的方法*/
        public int volume(){
                return length * width * height;
        }
}
```

8.3.2.1 对象的声明

对象的声明，就是声明一个变量，其语法格式和变量声明的语法完全相同，格式如下：

```
数据类型 对象名;
```

这里要求数据类型必须为复合数据类型，基本数据类型声明的结构只能称为变量，而不能称为对象。

示例代码：

```
Box b;
```

这里声明了一个Box类型的对象b，该对象在内存中不占用存储空间，其值为null。当然声明对象时也可以采用如下格式：

```
Box b,b1;
```

8.3.2.2 对象的初始化

由于只声明的对象在内存中没有存储空间，所以需要为对象在内存中申请空间，并初始化各个属性的值，这个过程称作对象的初始化。

对象的初始化，都是通过直接或间接调用构造方法实现。对象的初始化可以和对象的声明写在一起，也可以分开进行书写，其语法格式如下：

```
对象名 = new 构造方法(参数);
```

例如：

```
Box b = new Box();

Box b1;

b1 = new Box(2,3,4);
```

其中对象b使用Box类中不带参数的构造方法进行初始化，按照Box类的结构，对象b中每个属性的值被都被初始化为10。而对象b1使用Box类中带参数的构造方法进行初始化，依据构造方法的结构，依次指定对象b1中的长、宽、高依次是2、3、4。

在初始化对象时，调用的构造方法必须在类中声明过，否则不能调用。因为类名和构造方法的名称相同，所以名称一般不容易发生错误，在实际使用时注意参数列表的结构也需要匹配。

有些时候，因为某些原因，把构造方法隐藏起来，这个时候可以使用其它的途径来创建对象，例如使用某些方法的返回值进行初始化等。

对象在初始化以后就可以进行使用了。

8.3.2.3 引用对象中的属性

对象是一个复合变量，很多时候需要引用对象内部的某一个属性，其语法格式为：

对象名.属性名

该语法中“.”代表引用，使用该表达式可以引用对象内部的属性，则该表达式的类型和该属性在类中声明的类型一致。

例如：

```
b.width = 5;
```

该语法中，b 是对象名，width 是对象 b 中的属性，因为在类 Box 中 width 属性的类型是 int 型，则该表达式的类型也是 int 类型，在程序中可以把该表达式看成是 int 类型的变量进行实际使用。

而在实际的面向对象程序中，一般都避免使用对象直接引用属性(使用访问控制符实现访问限制)，而替代的以 getter 和 setter 方法进行访问。

8.3.2.4 引用对象中的方法

如果需要执行对象内部的功能，则需要引用对象中的方法，也就是面向对象术语中的“消息传递”，其语法格式如下：

对象名.方法名(参数)

这里“.”也代表引用，使用该代码可以引用对象内部的方法，该语法中的参数必须和引用方法的声明结构匹配。

例如：

```
int v = b.volume();
```

这里引用对象 b 中的 volume 方法，实现的功能是求对象 b 的体积，并且把求得的体积赋值给变量 v。

在实际的项目中，通过引用对象中的方法实现项目中信息的传递以及功能的实现，通过对象和对象之间的关联，构造成一个有机的系统，从而代表实际项目中标的各种需求。

对象相关的语法就介绍这么多，在后续的学习中将经常用到。

8.3.3 对象的存储形式

对象是一个复合数据类型的变量，其存储方式和一般变量的存储方式也不相同。在 Java 的执行环境中，存储区域一般分为两类：

1　栈内存

该区域存储基本数据类型

1　堆内存

存储实际的对象内容。

而实际的对象存储比一般变量复杂，对象的存储分为两部分：对象的内容、对象内容的起始地址。

Java 编程那些事儿 61—面向对象设计方法

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

8.4　面向对象设计方法

前面介绍了面向对象技术的两个最基本、最重要的概念——类和对象，下面介绍一下面向对象技术的设计思路。

对于初学者来说，面向对象是学习 Java 语言时的第一个难点，其实面向对象只是一种思考问题的方式，或者理解为组织数据和功能的方式而已，当系统中的数据和功能都实现以后，按照数据和功能的相关性进行组织。

在使用面向对象技术设计项目时，一般的步骤如下：

1、　抽象类

2、　抽象类的属性和方法

3、　通过对象的关联构造系统

其中步骤 1 和 2 是设计需要实现的功能，步骤 3 更多的和业务逻辑相关，体现设

计的结构不是很多。

1　抽象类

抽象类最基本的方式是——将名词转换为类。

在一个系统中会存在很多的名词，如果这些名词需要频繁的在系统中进行使用时，则可以将这些名词抽象成类，便于后续的时候。例如在一个学生成绩管理系统中，则名词：学生、课程等则可以抽象成类。

而实际在抽象时，由于有一定的主观性，所以在系统设计时，不同人设计的系统会存在一些不同。

1　抽象类的属性和方法

把系统中的类抽象出来了以后，就可以设计每个类的内部结构了，而每个类内部最重要的结构就是属性和方法了。

抽象属性最基本的方式是——将数据抽象为属性。

抽象方法最基本的方式是——将功能抽象为方法。

在一个类内部会存在很多的数据和功能，在实际抽象时，只需要抽象自己需要的数据和功能即可。例如在学生成绩管理系统中，学生的姓名、班级和各个科目的成绩都是系统中需要使用的数据，而学生的家庭住址，联系电话则不会必须的属性，可以根据实际的需要取舍数据的值。

抽象功能时，只需要把该类在系统中需要执行的动作提取出来，然后使用方法的语法进行描述即可。

当然，面向对象设计还涉及很多其它的知识，这里讲解的只是一些基础的入门知识，更多的有关面向对象的知识可以阅读关于面向对象技术的专门书籍，并且在项目开发中逐步体会这些知识。

8.5　面向对象三大特性

面向对象技术在实际开发中有很多的特性，总结起来最核心的特性主要有三个：封装、继承和多态。

8.5.1 封装性

封装性指在实际实现时，将复杂的内部结构隐藏起来，并为这组复杂的结构取一个统一的名称进行使用。在现实世界中，大量的存在封装的例子，例如电脑的硬盘，将多组复杂的线路和存储信息的磁片封装起来，并将该组结构取名为硬盘，以后就可以使用硬盘来代表该结构，而不需要更多的了解内部的信息。

在面向对象技术中，类是典型的封装性的体现，类将一组属性和功能组合成一个统一的结构，并使用类名来代表该结构。

封装性的最大优势在于隐藏每个类的内部实现(内部结构)，从而既方便项目的分解也降低了项目的难度。

例如以设计汽车为例，我们可以把汽车看作是软件开发中的整个项目，在实际设计时，首先可以将设计汽车分解为设计汽车的每个组件，然后具体对每个组件进行设计，而组件和组件的设计之间关联性很小，例如设计发动机的设计小组不需要很详细的了解轮胎设计小组的工作。而这里的每个组件可以看作实际面向对象设计中的类，每个类都把自己的内部实现隐藏起来，只通过名称使其它类了解该类的作用，并开放一些基本的功能供其它的类使用即可。

这样可以在实际设计时，每个类都更注重自身的实现，而对于其它类的实现不需要深入了解，这样可以在总体上降低交流的频率，从而降低整个项目的复杂度。

通常情况下，一般把类和类之间的关联性又称作耦合性，类和类之间的关联性比较低也称作耦合性比较低。在实际设计项目时，低耦合的项目是程序设计人员设计系统的目标之一。

8.5.2 继承性

在我们认知现实世界时，一般会把事物进行分类，而每一类内部又划分出很多的小类，生物学中将该方式体现的很彻底。例如猩猩属于动物中的哺乳类灵长目，这里的动物、哺乳类和灵长目都是一个特定的类别，和以前不同的是这些类别之间存在包含关系(is-a)，换句话说，也就是哺乳类是动物类的一种，灵长目是哺乳类的一种。

其实在程序设计中，很多设计出来的类也存在这样的包含关系，这样一个类的内部会包含和其它类类似的特征和属性，如果在设计时可以以另外一个类为基础进行设计，那将是多么激动人心的特性，这个特性就是面向对象设计中的继承性。

在一个项目中，如果类和类之间存储包含关系，即一个类是另外一个类的一种，就可以使用继承。

继承性提供了全新的类设计方式，可以充分利用了已有类内部的结构和功能，极大的降低了类内部的代码重复，是设计类的一种显著的变革，对于大型的项目设计十分有用。另外很多技术的应用中也包含大量的继承成分，使整个技术体系比较固定。

8.5.2.1 继承语法

在 Java 语言中，继承的语法格式比较简单，如下所述：

```
访问控制符 [修饰符] class 类名 extends 父类名{
    ……
}
```

在声明类时，声明该类的继承关系，使用 extends 关键字实现，其中 extends 关键字前面是声明出的新类名，extends 关键字后面的类名是被继承的类名，要求被继承的类名已存在。Java 语言采用的是单重继承，也就是说一个类只能有一个直接父类。在类声明时，如果没有使用 extends 关键字声明父类，则自动继承 Object 类。说明：Object 类是系统提供的类，该类已存在。

示例代码如下：

```
//Animal.java
public class Animal {
    /**类型名称*/
    String name;
    /**移动方式*/
    int moveType;
}
```

```
//Mammalia.java

public class Mammalia extends Animal{

        /**哺育时间*/

        int fosterTime;

}
```

这里 Mammalia 类就是 Animal 类的子类，Animal 类就是 Mammalia 类的父类，子类和父类具有相对性，正如一个祖孙三代的家庭内部，每个人相对于不同的人扮演不同的角色一样。同时类和类之间的继承具备传递性，就如现实中的血缘关系一样。

8.5.2.2 继承说明

两个类之间如果存在了继承关系以后，将带来哪些不同呢?下面依次来进行说明：

l 子类拥有父类的所有属性

子类中继承父类中所有的属性，在父类中声明的属性在子类内部可以直接调用。说明：如果访问控制符限制则无法访问。

l 子类拥有父类的所有方法

子类中继承父类中所有的方法，在父类中声明的方法在子类内部可以直接调用。说明：如果访问控制符限制则无法访问。

l 子类不拥有父类的构造方法

子类不继承父类的构造方法，如果需要在子类内部使用和父类传入参数一样的构造方法，则需要在子类内部重新声明这些构造方法。

l 子类类型是父类类型

子类类型的对象可以自动转换为父类类型的对象，父类类型的对象则需要强制转换为子类的对象，转换的语法个基本数据类型转换的语法相同。

Java 编程那些事儿 62——继承(二)

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

8.5.2.3 方法覆盖

前面介绍了继承的一些基础知识，现在介绍一些在使用继承时需要注意的问题。熟悉这些问题将更好的解决项目中的实际问题。

例如在实际的游戏中，会按照怪物的种类实现设计。首先设计一个基础类Monster，然后按照怪物类别设计Monster的子类，如Boss、NormalMonster等。则在实际实现时，每个怪物都有移动(move)的功能，但是在Boss和NormalMonster的移动规则存在不同。这样就需要在子类的内部重新编写移动的功能，从而满足实际的移动要求。该示例的实现代码如下：

```
//Monster.java
public class Monster{
        public void move(){
                //移动功能
        }
}
//Boss.java
public class Boss extends Monster{
        public void move(){
                //Boss类的移动规则
        }
}
//NormalMonster.java
```

```
public class NormalMonster extends Monster{

    public void move(){

        // NormalMonster 类的移动规则

    }

}
```

这样在 Monster 的每个子类内部都重新书写了 move 方法的功能，这种在子类内部重新父类中的方法的语法现象，称作方法覆盖(override)。

方法覆盖在实际中保持了类的结构的统一，在实际使用时将极大的方便程序开发人员的使用，使项目的整体结构保持统一，便于项目的维护。

在使用子类的对象时，子类内部的方法将覆盖从父类继承过来的方法，也就是说子类的对象调用的是子类的功能方法，而不是父类的方法。

在进行方法覆盖时，子类内部的方法和父类的方法声明相同，而且子类方法的限制不能比父类的方法严格。例如不能使用比父类限制更大的访问控制符或抛出比父类更多的异常等，这个在实际使用方法覆盖时需要特别的注意。

在实际的项目中大量的存在需要在子类内部重写父类的功能方法的地方，恰当的使用方法覆盖将为项目开发带来很大的便利。

8.2.2.4 需要注意的问题

除了方法覆盖以外，在实际使用继承时还有很多需要注意的问题。下面就这些问题进行一一说明。

1、 属性覆盖没有必要

方法覆盖可以重写对应的功能，在实际继承时在语法上也支持属性覆盖(在子类内部声明和父类属性名相同的属性)，但是在实际使用时修改属性的类型将导致类结构的混乱，所以在继承时不能使用属性覆盖。

2、 子类构造方法的书写

该项是继承时书写子类最需要注意的问题。在子类的构造方法内部必须调用父类的构造方法，为了方便程序员进行开发，如果在子类内部不书写调用父类构造方法的代码时，则子类构造方法将自动调用父类的默认构造方法。而如果父类不存

在默认构造方法时，则必须在子类内部使用 super 关键字手动调用，关于 super 关键字的使用将在后续进行详细的介绍。

说明：子类构造方法的参数列表和父类构造方法的参数列表不必完全相同。

3、 子类的构造过程

在构造子类时由于需要父类的构造方法，所以实际构造子类的过程就显得比较复杂了。其实在实际执行时，子类的构造过程遵循：首先构造父类的结构，其次构造子类的结构，无论构造父类还是子类的结构，都是首先初始化属性，其次执行构造方法。则子类的构造过程具体如下：

如果类 A 是类 B 的父类，则类 B 的对象构造的顺序如下：

a) 类 A 的属性初始化

b) 类 A 的构造方法

c) 类 B 的属性

d) 类 B 的构造方法

由于任何一个类都直接或间接继承自 Object 类，所以 Object 类的属性和构造方法都是首先执行的。

4、 不要滥用继承

在实际的项目设计中，继承虽然很经常使用，但是还是不能滥用，使用继承的场合以及相关问题参看下面的说明。

8.5.2.5 如何设计继承

在实际的项目中，类和类之间的关系主要有三种：

1、 没有关系

项目中的两个类之间没有关联，不需要进行消息传递，则这两个类之间就没有关系，可以互相进行独立的设计。

2、 使用关系(has-a)

如果一个类的对象是另外一个类的属性，则这两个类之间的关系是使用关系。例

如把房屋(House)看作是一个类，把门(Door)看成另外一个类，则房屋有一个门，代码的实现如下：

```
            //House.java
public class House{
                    public Door door;
            }
            //Door.java
            public class Door{


            }
```

则这里 Door 的对象是 House 类的属性，则 Door 和 House 类之间的关系就是使用关系，House 使用 Door 类来制作自身。

使用关系提供了使用已有类来声明新类的方式，可以以组合的方式来构建更复杂的类，这是项目中使用类的常见方式之一。

判断是否是使用关系的依据就是：has-a，一个类具备另外一个类的对象，例如一个 House 有一个门。

3、 继承关系(is-a)

如果一个类是另外一个类的一种，也就是在分类上存在包含关系，则应该使用继承来实现。例如 Boss 是怪物的一种，则使 Boss 继承 Monster 类。

下面简单介绍一些项目中继承的设计方法。在实际设计继承时，一般有两种设计的方法：

1、 自上而下的设计

在实际设计时，考虑类的体系结构，先设计父类，然后根据需要来增加子类，并在子类的内部实现或添加对应的方法。

2、 自下而上的设计

在实际设计时，首先不考虑类的关系，每个类都分开设计，然后从相关的类中把重复的属性和方法抽象出来形成父类。

对于初学者来说，第二种设计方式相对来说比较容易实现，所以一般初学者都按照第二种设计方式进行设计，设计完成以后再实现成具体的代码。

Java 编程那些事儿 63—多态性

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

8.5.3 多态性

多态性是面向对象技术中最灵活的特性，主要是增强项目的可扩展性，提高代码的可维护性。

多态性依赖继承特性，可以把多态理解为继承性的扩展或者深入。

在这里把多态性分为两方面来进行介绍，对象类型的多态和对象方法的多态。

为了方便后续的讲解，首先给出一个继承结构的示例。

```
          //文件名：SuperClass.java

 public class SuperClass{
        public void test(){
               System.out.println(“SuperClass”);
        }
 }

          // 文件名：SubbClass1.java

public class SubbClass1 extends SuperClass{
       public void test(){
             System.out.println(“SubbClass1”);
       }
}
```

```
                // 文件名：SubbClass2.java

public class SubbClass2 extends SuperClass{
        public void test(){
                System.out.println(“SubbClass2”);
        }
}
```

在该示例代码中，SubbClass1 和 SubbClass2 是 SuperClass 的子类，并且在子类的内部都覆盖父类中的 test 方法。由于这三个类中都书写构造方法，则按照默认构造方法的约定，每个类中都会被自动添加一个默认的构造方法。

8.5.3.1 对象类型的多态

对象类型的多态是指声明对象的类型不是对象的真正类型，而对象的真正类型由创建对象时调用的构造方法进行决定。例外，按照继承性的说明，子类的对象也是父类类型的对象，可以进行直接赋值。

例如如下代码：

```
SuperClass sc = new SubbClass1();
```

这里声明了一个 SuperClass 类型的对象 sc，然后使用 SuperClass 的子类 SubbClass1 的构造方法进行创建，因为子类类型的对象也是父类类型的对象，所以创建出来的对象可以直接赋值给父类类型的对象 sc。除了对象的赋值以外，另外一个更重要的知识是 sc 对象虽然使用 SuperClass 声明的类型，但是内部存储的却是 SubbClass1 类型的对象。这个可以 Java 语言的中 instanceof 运算符进行判断。

instanceof 是一个运算符，其作用是判断一个对象是否是某个类类型的对象，如果成立则表达式的值为 true，否则为 false。语法格式如下：

```
对象名 instanceof 类名
```

需要注意的是:这里的类名必须和声明对象时的类之间存储继承关系，否则将出现语法错误。

测试类型的代码如下：

```
/**
 * 测试对象类型的多态
 */
```

```
public class TestObjectType {
        public static void main(String[] args) {
                SuperClass sc = new SubbClass1();
                boolean b = sc instanceof SuperClass;
                boolean b1 = sc instanceof SubbClass1;
                System.out.println(b);
                System.out.println(b1);
        }
}
```

该测试程序的输出结果是：

```
true
true
```

由程序运行结果可以看出，sc 既是 SuperClass 类型的对象，也是 SubbClass1 类型的对象，而 SubbClass1 的类型被隐藏起来了，这就是对象的多态。其实 sc 对象不仅仅在类型上是 SubbClass1 类型的，其存储的内容也是 SubbClass1 的内容，具体参看后面介绍的对象方法的多态。

对象类型的多态有很多的用途，极大的方便了对象的存储和传递，使代码很方便的进行扩展，对于已有代码不产生影响。下面介绍两个基本的使用。

1. 对象的存储

在存储一系列不同子类的对象时，可以使用父类的结构来进行声明，这样可以方便数据的存储，例如需要存储多个 SubbClass1 和 SubbClass2 的对象时，则可以声明一个 SuperClass 类型的数组进行存储，示例代码如下：

```
SuperClass sc[] = new SuperClass[3];

sc[0] = new SubbClass1();

sc[1] = new SubbClass2();

sc[2] = new SubbClass1();
```

则这里的数组 sc，可以存储各个类型子类的对象，而数组中每个元素的值都是存储的对应子类的对象，而只是在名义上的类型(语法上的类型)是 SuperClass 类型的，这样将方便程序的控制，当增加新的子类类型时，已有的代码不需要进行改造就可以自动适应新的子类的结构。

例如新增了一个 SuperClass 的子类 SubbClass3，则该数组的代码可以修改成如下：

```
SuperClass sc[] = new SuperClass[3];

sc[0] = new SubbClass1();

sc[1] = new SubbClass2();

sc[2] = new SubbClass3();
```

其它的代码都需要进行修改，就可以适应新的结构，这是多态性最主要的用途。

2.对象的传递

在方法的传入参数传递，以及返回值处理方面都从对象类型的多态中受益。在向方法中传入参数时，如果该方法需要处理各个子类的对象，则只需要书写一个接受父类类型对象的方法即可。例如：

```
public void testObjectTypeMethod(SuperClass sc){}
```

则该在调用该方法时，可以传入 SuperClass 的对象，也可以传入其子类的对象，如果传入的是子类的对象，则子类对象中的内容不会丢失。例如调用的示例代码如下：

```
SuperClass sc = new SuperClass();

SubbClass1 sc1 = new SubbClass1();

SubbClass2 sc2 = new SubbClass2();

testObjectTypeMethod(sc);

testObjectTypeMethod(sc1);

testObjectTypeMethod(sc2);
```

这里说明的只是调用时的语法结构，这样的特性将使我们只需要书写一个方法，就可以处理所有子类的对象，简化代码的书写，降低代码的重复，从而降低维护的难度。

另外，方法的返回值也可以利用到该特性，例如如下方法：

```
public SuperClass testObjectTypeMethod2(){}
```

则在该方法的内部，既可以返回 SuperClass 类型的对象，也可以返回其子类的对象，也能简化代码的书写，便于代码的阅读和维护。

关于对象类型的多态，就简单的说明这么多，具体在项目中如何进行使用，还需要一定的技巧和方法。

8.5.3.2 对象方法的多态

对象方法的多态基于方法的覆盖，也就是该对象调用的方法具体是子类的方法还是父类的方法，由创建对象时使用的构造方法决定，而不是由声明对象时声明的类型决定。

示例代码如下：

```
/**
 * 测试对象方法的多态
 */

public class TestObjectMethod {
        public static void main(String[] args) {
                SuperClass sc = new SuperClass();
                SubbClass1 sc1 = new SubbClass1();
                SubbClass2 sc2 = new SubbClass2();
                SuperClass sc3 = new SubbClass1();
                testObjectTypeMethod(sc);
                testObjectTypeMethod(sc1);
                testObjectTypeMethod(sc2);
                testObjectTypeMethod(sc3);
        }
        public static void testObjectTypeMethod(SuperClass sc){
                sc.test();  //调用被覆盖的方法
        }
}
```

该代码的执行结果如下：

```
SuperClass
SubbClass1
SubbClass2
```

SubbClass1

则从代码的执行结果看，虽然 testObjectTypeMethod 方法接收的是 SuperClass 类型的对象，但是传入子类对象时，子类对象的内容没有丢失，所以在调用 test 方法时，还是调用的对应对象中对应的 test 方法。

这样就在功能上实现了对象的传递，从而保留了对象的内容，极大的方便了代码的扩展性。

但是，由于 Java 在执行程序时，在程序运行的过程中，需要判断对象调用的具体是父类的方法还是子类的方法，所以程序的执行速度会稍微有所降低。

Java 编程那些事儿 64——访问控制符、修饰符和其它关键字

作者：陈跃峰
出自：http://blog.csdn.net/mailbomb

8.6 访问控制符

访问控制符的作用是说明被声明的内容(类、属性、方法和构造方法)的访问权限，就像发布的文件一样，在文件中标注机密，就是说明该文件可以被那些人阅读。

访问控制在面向对象技术中处于很重要的地位，合理的使用访问控制符，可以通过降低类和类之间的耦合性(关联性)来降低整个项目的复杂度，也便于整个项目的开发和维护。具体的实现就是通过访问控制符将类中会被其它类调用的内容开放出来，而把不希望别人调用的内容隐藏起来，这样一个类开放的信息变得比较有限，从而降低了整个项目开放的信息，另外因为不被别人调用的功能被隐藏起来，在修改类内部隐藏的内容时，只要最终的功能没有改变，即使改变功能的实现方式，项目中其它的类不需要更改，这样可以提高了代码的可维护性，便于项目代码的修改。

在 Java 语言中访问控制权限有 4 种，使用三个关键字进行表达，依次如下：

<!--[if !supportLists]-->l <!--[endif]-->public——公共的

<!--[if !supportLists]-->l <!--[endif]-->protected——受保护的

<!--[if !supportLists]-->l <!--[endif]-->无访问控制符——默认的

<!--[if !supportLists]-->l <!--[endif]-->private——私有的

其中无访问控制符是指不书写任何的关键字，也代表一种访问权限，访问控制符的使用示例如下所示：

```
public class AccessControl {

                int n;

                public AccessControl(){

                        init();

                }

                private void init(){}

                 protected void test(int k){}

}
```

该示例代码中演示了各个访问控制符的实际使用示例，其中属性 n 的访问控制符就是默认的。

在实际使用时，类声明的访问控制符只有 2 个：public 和无访问控制符，属性声明、构造方法声明和方法声明的访问控制符可以是以上 4 种中的任何一个。

这 4 个访问控制符的权限作用如下表所示：

访问控制符	同一个类内部	同一个包内部	不同包中的子类	不同包中的非子类
public	Yes			

Yes
Yes
Yes

protected
Yes
Yes
Yes
No

无访问控制符
Yes
Yes
No
No

private
Yes
No
No
No

说明：在该表中，Yes 代表具备对应的权限，No 代表不具备对应的权限。

在 4 种访问控制中，public 一般称作公共权限，其限制最小，也可以说没有限制，使用 public 修饰的内容可以在其它所有位置访问，只要能访问到对应的类，就可以访问到类内部 public 修饰的内容，一般在项目中开放的方法和构造方法使用 public 修饰，开放给项目使用的类也使用 public 修饰。protected 一般称作继承权限，使用 protected 修饰的内容可以被同一个包中的类访问也可以在不同包内部的子类中访问，一般用于修饰只开放给子类的属性、方法和构造方法。无访问控制符一般称作包权限，无访问控制符修饰的内容可以被同一个包中的类访问，一般用于修饰项目中一个包内部的功能类，这些类的功能只是辅助其它的类实现，而为包外部的类提供功能。private 一般称作私有权限，其限制最大，类似于文件中的绝密，使用 private 修饰的内容只能在当前类中访问，而不能被类外部的任何内容访问，一般修饰不开放给外部使用的内容，修改 private 的内容一般对外部的实现没有影响。

下面以两个基本的示例来说明访问控制符在实际项目中的使用方式。

第一个使用示例：在项目中，一般不会将类的属性开放给其它的类，

也就是不允许外部的类直接访问属性，而是使用对应的存取方法来进行访问。例如在学校的学员管理系统中，需要实现的学生类，按照访问控制符的一般使用规则，实现的代码如下：

```
/**
 * 学员类，演示访问控制符的使用
 */

public class Student {

                    /**年龄*/

                    private int age;

                    /**学员 ID*/

                    private int id;

                    public int getAge() {

                              return age;

                    }

                    public void setAge(int age) {

                              if(age < 0){

                                        //处理代码，未实现

                              }

                              this.age = age;

                    }

                    public int getId() {

                              return id;

                    }

                    public void setId(int id) {
```

```
                                        //校验 id 是否合法的代码，未实现
                                        this.id = id;
                                    }
}
```

通过将属性的访问权限设定为 private，限制所有类外部对属性的访问，而为了让外部可以访问这些属性，专门声明对应的 get/set 方法来读取/存储数据，这样在设置属性值的 set 方法中，可以对于参数做出基本的校验，在上面的示例代码中，留出了校验参数的位置，具体的代码未在示例代码中实现。

小技巧：在 eclipse 中，get 和 set 方法可以在选中对应类的代码以后，使用“Source”菜单中的“Generate Getters and Setters…”菜单实现。

第二个使用示例：在项目中，一般为了设计的需要实现一些特定的功能，下面介绍一下使用访问控制符实现的一个功能——使一个类既不能创建对象也不能被继承。实现的方法如下：该类中只实现一个构造方法，而且将该构造方法的访问权限设置为私有的。具体实现代码如下：

```
/**
 * 不能创建对象且不能被继承的子类
 */

public class PrivateDemo {

                                    private PrivateDemo(){}

}
```

在该示例中，PrivateDemo 类只有一个构造方法，且该构造方法为私有。按照以前的介绍，创建对象时需要调用构造方法，而 private 修饰的构造方法无法在类的外部进行访问，所以无法创建对象。另外，在子类的构造方法中也需要调用父类的构造方法，由于 private 的构造方法无法得到调用，所以该类也不可以有对应的子类。

这里说明的只是两个基本的用途，在实际的项目中，可以根据需要灵活的使用访问控制符实现对应的功能。

总之，访问控制符通过控制声明的内容的访问权限，实现对于内容的

隐藏，从而降低使代码的耦合性降低，降低项目的复杂度，并且方便实际项目中代码的维护。

Java 编程那些事儿 65——static 修饰符

郑州游戏学院 陈跃峰
出自：http://blog.csdn.net/mailbomb

8.7 修饰符

修饰符的作用是让被修饰的内容具备特定的功能，在程序中合理使用修饰符可以在语法和功能上实现很多需要的效果。Java 语言中的修饰符主要有 5 个：static、final、native、abstract 和 synchronized。这里首先讲解 static、final 和 native 的作用。

8.7.1 static 修饰符

static 关键字的中文意思是静态的，该修饰符可以修饰成员变量，成员常量和成员方法。使用该关键字修饰的内容，在面向对象中 static 修饰的内容是隶属于类，而不是直接隶属于对象的，所以 static 修饰的成员变量一般称作类变量，而 static 修饰的方法一般称作类方法。另外，static 还可以修饰代码块，下面进行详细的介绍。

8.7.1.1 静态变量

static 修饰的变量称作静态变量。静态变量和一般的成员变量不同，一个类在加载到内存时，静态变量只初始化一次，也就是说所有对象的静态变量在内存中都只有一个存储位置，每个对象中的静态变量都指向内存中同一个地址，它是在所有的对象之间共享的数据。另外静态变量在引用时比较方便。所以一般在需要实现以下两个功能时使用静态变量：

l 在对象之间共享值时

l 方便访问变量时

下面首先说一下非静态变量(没有 static 修饰符修饰的成员变量)在内存中如何存储的。示例代码如下：

```
//文件名 Box.java
public class Box{
        int length;
        int width;
        int height;
        public Box(int l,int w,int h){
                length = l;
                width = w;
                height = h;
        }
}
//文件名 TestBox.java
public class TestBox{
        public static void main(String[] args){
                Box a = new Box(10,20,30);
                Box b = new Box(40,20,10);
        }
}
```

则对象 a 和对象 b 在内存中的存储格式如下图所示：

对象 a　　对象 b

从上面的图可以看出，非静态变量的值在每个对象中都有独立的存储空间，不同

对象间这些值之间没有管理，也就是说每个对象都为内部的每个非静态的变量分配独立的存储空间，所以每个对象中非静态变量是隶属于对象，也就是说在每个对象中可能是不同的。

简单介绍了非静态变量在对象中的存储以后，下面再来看一下静态变量是如何进行存储的。示例代码如下：

```
//文件名 StaticVar.java

public class StaticDemo{

        static int m;

        int n;

        char c;

}

//文件名 TestStaticVar.java

public class TestStaticVar{

        public static void main(String[] args){

                StaticVar sv1 = new StaticVar();

                StaticVar sv2 = new StaticVar();

        }

}
```

则对象 sv1 和对象 sv2 在内存中存储的格式如下图所示：

对象 sv1　　对象 sv2

对于 StaticDemo 类型的对象 sv1 和 sv2 来说，由于使用默认的构造方法进行构造，所以每个成员变量都被初始化为对应数据类型的默认值，int 的默认值为 0，char 的默认值为编号为 0 的字符，所以 sv1 和 sv2 对象中存储的值如上图所示。

而静态变量的存储和非静态变量的存储不同，在 Java 虚拟机内部，第一次使用类时初始化该类中的所有静态变量，以后就不再进行初始化，而且无论创建多少个该类的对象，静态变量的存储在内存中都是独立于对象的，也就是 Java 虚拟机单独为静态变量分配存储空间，所以导致所有的对象内部的静态变量在内存中存储时只有一个空间。这样就导致使用任何一个对象对该值的修改都是使该存储空间中的值发生改变，而其它对象在后续引用时就跟着发生了变化。静态变量就是使用这样的方式在所有的对象之间进行数值共享的。

静态变量在实际使用时，可以通过只存储一次来节约存储空间，这个特性导致在类内部定义的成员常量一般都做成静态的，因为常量的值在每个对象中都是相同的，而且使用 static 修饰也便于对成员常量的引用。

在类外部访问某类中静态变量(常量)的语法格式为：

类名.成员变量(常量)

例如：

StaticDemo.m

这样方便对于成员变量的访问。当然，语法上也不禁止使用：对象.成员变量，这样的语法格式进行访问，但是一般不推荐这样使用，而且有些类是无法创建对象的。

注意：static 关键字不能修饰成员方法或构造方法内部的变量。

8.7.1.2 静态方法

static 修饰的方法称作静态方法。静态方法和一般的成员方法相比，不同的地方有两个：一是调用起来比较方便，二是静态方法内部只能使用静态的成员变量。所以一般静态方法都是类内部的独立的功能方法。例如为了方便方法的调用，Java API 中的Math类中所有的方法都是静态的，而一般类内部的static方法也是方便其它类对该方法的调用。

示例代码如下：

```
//文件名 MyMath.java

public class MyMath{

        public static int max(int a,int b){
```

```
                return (a > b ? a : b);
        }
}
//文件名 TestMyMath.java
public class TestMyMath{
        public static void main(String[] args){
                int m = 10;
                int n = 20;
                int k = MyMath.max(m,n);
        }
}
```

静态方法在类的外部进行调用时不需要创建对象，使用类名.方法名(参数)这样的语法格式进行调研，简化了代码的编写。

使用静态方法时，需要特别注意的是静态方法内部使用该类的非静态成员变量，否则将出现语法错误。

静态方法是类内部的一类特殊方法，只有在需要时才将对应的方法声明成静态的，一个类内部的方法一般都是非静态的。

8.7.1.3 静态代码块

静态代码块指位于类声明的内部,方法和构造方法的外部,使用static修饰的代码块。静态代码块在该类第一次被使用时执行一次，以后再也不执行。在实际的代码中,如果需要对类进行初始化的代码,可以写在静态代码块的内部。

示例代码如下：

```
//文件名 StaticBlock.java
```

```
public class StaticBlock{
        static{
            System.out.println(“静态代码块!”);
        }
}
```

静态代码块是一种特殊的语法，熟悉该语法的特点，在实际程序中根据需要使用。

Java 编程那些事儿 66——final 修饰符

陈跃峰
出自：http://blog.csdn.net/mailbomb

8.7.2 final

final 关键字是最终的、最后的意思，在程序中可以用来修饰类、成员变量和方法的声明，由该关键字修饰的内容都是不可变的。

8.7.2.1 final 数据

final 修饰的数据是常量，常量既可以出现在类的内部，也可以出现在方法或构造方法的内部。在程序中常量只能赋值一次。

其它说明可以参看前面的常量介绍。

在程序中，一般类内部的成员常量为了方便调用，一般都使用 static 修饰符进行修饰。示例代码如下：

```
/**
 * 常量使用
 */

public class Student {

                    /**性别*/
```

```
    int sex;

    /**男性*/

    public final static int MALE = 0;

    /**女性*/

    public final static int FEMALE = 1;

}
```

8.7.2.2 final 方法

final 关键字也可以修饰方法，final 修饰的方法称作最终方法，最终方法不能被覆盖，也就是不能在子类的内部重写该方法。

使用 final 修饰方法，可以在一定程度上提高该方法的执行速度，应为在调用该方法时，就不需要进行覆盖的判断了。

8.7.2.3 final 类

final 关键字也可以修饰类，final 修饰的类称作最终类，最终类不能被继承，也就是该类不能有子类。

final 类内部的每个方法都是 final 方法。

8.7.3 native

native 关键字是“本地的”意思，native 修饰的方法，只有方法的声明使用 java 语言实现，而方法内部的代码都是在 Java 虚拟机内部使用其它语言实现。

一般 native 的方法，都是和系统操作有关的方法，或者是基于底层实现效率比较高的方法，常见于系统类中。例如 System 类的 arraycopy 方法等。

Java 编程那些事儿 67——this 和 super

陈跃峰
出自：http://blog.csdn.net/mailbomb

8.8 this 和 super

下面再来介绍一下 this 和 super 关键字的使用。在程序中通过使用 this 和 super 关键字，可以实现对于类内部很多内容方便的引用，也有助于理解面向对象的实现原理，更方便的理解面向对象技术的内部实现。

8.8.1 this 关键字

this 关键字代表自身，在程序中主要的使用用途有以下几个方面：

l 使用 this 关键字引用成员变量

l 使用 this 关键字在自身构造方法内部引用其它构造方法

l 使用 this 关键字代表自身类的对象

l 使用 this 关键字引用成员方法

8.8.1.1 引用成员变量

在一个类的方法或构造方法内部，可以使用“this.成员变量名”这样的格式来引用成员变量名，有些时候可以省略，有些时候不能省略。首先看一下下面的代码：

```
/**
 * 使用 this 引用成员变量
 */

public class ReferenceVariable {

                        private int a;
                        public ReferenceVariable(int a){
                                this.a = a;
                        }
                        public int getA(){
                                return a;
                        }
                        public void setA(int a){
                                this.a = a;
                        }
```

```
}
```

在该代码的构造方法和 setA 方法内部，都是用 this.a 引用类的成员变量。因为无论在构造方法还是 setA 方法内部，都包含 2 个变量名为 a 的变量，一个是参数 a，另外一个是成员变量 a。按照 Java 语言的变量作用范围规定，参数 a 的作用范围为构造方法或方法内部，成员变量 a 的作用范围是类的内部，这样在构造方法和 setA 方法内部就存在了变量 a 的冲突，Java 语言规定当变量作用范围重叠时，作用域小的变量覆盖作用域大的变量。所以在构造方法和 setA 方法内部，参数 a 起作用。

这样需要访问成员变量 a 则必须使用 this 进行引用。当然，如果变量名不发生重叠，则 this 可以省略。

但是为了增强代码的可读性，一般将参数的名称和成员变量的名称保持一致，所以 this 的使用频率在规范的代码内部应该很多。

8.8.1.2 引用构造方法

在一个类的构造方法内部，也可以使用 this 关键字引用其它的构造方法，这样可以降低代码的重复，也可以使所有的构造方法保持统一，这样方便以后的代码修改和维护，也方便代码的阅读。

下面是一个简单的示例：

```
/**
 * 使用 this 关键字引用构造方法
 */

public class ReferenceConstructor {
        int a;
        public ReferenceConstructor(){
                this(0);
        }
        public ReferenceConstructor(int a){
                this.a = a;
        }
}
```

这里在不带参数的构造方法内部，使用 this 调用了另外一个构造方法，其中 0 是根据需要传递的参数的值，当一个类内部的构造方法比较多时，可以只书写一个构造方法的内部功能代码，然后其它的构造方法都通过调用该构造方法实现，这样既保证了所有的构造是统一的，也降低了代码的重复。

在实际使用时，需要注意的是，在构造方法内部使用 this 关键字调用其它的构造方法时，调用的代码只能出现在构造方法内部的第一行可执行代码。这样，在构造方法内部使用 this 关键字调用构造方法最多会出现一次。

8.8.1.3 代表自身对象

在一个类的内部，也可以使用 this 代表自身类的对象，或者换句话说，每个类内部都有一个隐含的成员变量，该成员变量的类型是该类的类型，该成员变量的名称是 this，实际使用 this 代表自身类的对象的示例代码如下：

```
/**
 * 使用 this 代表自身类的对象
 */

public class ReferenceObject {
        ReferenceObject instance;
        public ReferenceObject(){
                instance = this;
        }
        public void test(){
                System.out.println(this);
        }
}
```

在构造方法内部，将对象 this 的值赋值给 instance，在 test 方法内部，输出对象 this 的内容，这里的 this 都代表自身类型的对象。

8.8.1.4 引用成员方法

在一个类的内部，成员方法之间的互相调用时也可以使用“this.方法名(参数)”来进行引用，只是所有这样的引用中 this 都可以省略，所以这里就不详细介绍了。

8.8.2 super 关键字

super 关键字的中文意思是超级的，使用 super 关键字可以在子类中引用父类中的内容。主要的使用形式有以下几种：

1　在子类的构造方法内部引用父类的构造方法

l 在子类中调用父类中的成员方法

l 在子类中调用父类中的成员变量

8.8.2.1 引用父类构造方法

在构造子类对象时，必须调用父类的构造方法。而为了方便代码的编写，在子类的构造方法内部会自动调用父类中默认的构造方法。但是如果父类中没有默认的构造方法时，则必须手动进行调用。

使用 super 可以在子类的构造方法内部调用父类的构造方法。可以在子类的构造方法内部根据需要调用父类中的构造方法。

使用 super 关键字调用父类构造方法的示例代码如下：

```
        //文件名：SuperClass.java
public class SuperClass {
        public SuperClass(){}
        public SuperClass(int a){}
}
        //文件名：SubClass.java
public class SubClass extends SuperClass {
        public SubClass(){
                super();  //可省略
        }
        public SubClass(int a){
                super(a);
        }
        public SubClass(String s){
                super();  //可省略
        }
}
```

在该示例代码中，SubClass 继承 SuperClass 类，在 SubClass 类的构造方法内部可以使用 super 关键字调用父类 SubClass 的构造方法，具体调用哪个构造方法没有限制，可以在子类内部根据需要进行调用，只是根据调用的构造方法不同传入适当的参数即可。

由于 SubClass 类的父类 SuperClass 内部有默认的构造方法，所以 SubClass 的构造方法内部 super()的代码可以省略。

和使用 this 关键字调用构造方法一样，super 调用构造方法的代码只能出现在

子类构造方法中的第一行可执行代码。这样 super 调用构造方法的代码在子类的构造方法内部则最多出现一句，且不能和 this 调用构造方法的代码一起使用。

8.8.2.2 引用父类成员方法

在子类中继承了父类中的成员方法，一般可以直接通过方法名使用，但是如果在子类中覆盖了父类的成员方法以后，如果需要在子类内部调用父类中被覆盖的成员方法时则不能直接调用了，这样就又需要使用 super 关键字了。

示例代码如下：

```
                //文件名：SuperClass2.java

public class SuperClass2 {
        public void test(){}
        public void print(int a){
                System.out.println("SuperClass2: " + a);
        }
}

//文件名:SubClass2

public class SubClass2 extends SuperClass2 {
        public void print(int a){
                super.print(a);
                System.out.println("SubClass2");
        }
        public void t(){
                super.test();  //super 可省略
                super.print(0); //不可省略
        }
}
```

8.8.2.3 引用父类成员变量

在子类中如果引用父类的成员变量，也可以使用“super.成员变量”来引用，只是一般成员变量的覆盖是没有意义的，所以这个时候都可以直接使用成员变量名进行引用，所以这里的 super 都可以省略。

8.8.3 注意的问题

最后，在实际使用 this 和 super 时，除了上面介绍到的需要注意的问题以外，还需要特别注意的是，this 和 super 都是非静态的，所以这两个关键

字都无法在静态方法内部进行使用。

Java 编程那些事儿 68——抽象类和接口(一)

陈跃峰
出自：http://blog.csdn.net/mailbomb

8.9 抽象类和接口

在实际的项目中，整个项目的代码一般可以分为结构代码和逻辑的代码。就像建造房屋时，需要首先搭建整个房屋的结构，然后再细化房屋相关的其它的结构，也像制造汽车时，需要首先制作汽车的框架，然后才是安装配件以及美化等工作。程序项目的实现也遵循同样的道理。

在项目设计时，一个基本的原则就是——“设计和实现相分离”。也就是说结构代码和逻辑代码的分离，就像设计汽车时只需要关注汽车的相关参数，而不必过于关心如何实现这些要求的制作。程序设计时也是首先设计项目的结构，而不用过多的关系每个逻辑的代码如何进行实现。

前面介绍的流程控制知识，主要解决的是逻辑的代码的编写，而类和对象的知识，则主要解决结构代码的编写。那么还有一个主要的问题：如何设计结构代码呢？这就需要使用下面介绍的抽象类和接口的知识了。

8.9.1 抽象类

抽象类(Abstract Class)是指使用 abstract 关键字修饰的类，也就是在声明一个类时加入了 abstract 关键字。抽象类是一种特殊的类，其它未使用 abstract 关键字修饰的类一般称作实体类。例如：

```
public abstract class A{
        public A(){}
}
```

抽象方法(Abstract Method)是指使用 abstract 关键字修饰的方法。抽象方法是一种特殊的方法，其它未使用 abstract 关键字修饰的方法一般称作实体方法。

```
public abstract void test();
```

抽象类和实体类相比，主要有以下两点不同：

l 抽象类不能使用自身的构造方法创建对象(语法不允许)

例如下面的语法是错误的：

```
A a = new A();
```

但是抽象类可以声明对象，例如下面的代码是正确的：

```
A a;
A a1,a2;
```

只是声明出的对象默认都是 null 的，无法调用其内部的非静态属性和非静态方法。

说明：抽象类可以使用子类的构造方法创建对象。

l 抽象类内部可以包含任意个(0 个、1 个或多个)抽象方法

抽象类内部可以包含抽象方法，也可以不包含抽象方法，对于包含的个数没有限制。而实体类内部不能包含抽象方法。

在抽象类内部，可以和实体类一样，包含构造方法、属性和实体方法，这点和一般的类一样。

抽象方法和实体方法相比，主要有以下几点不同：

l 抽象方法没有方法体

也就是说在声明抽象方法时，不能书写方法体的{}，而只能以分号结束方法。下面是实体方法和抽象方法声明的比较：

抽象方法声明：

```
public abstract void test(int a);
```

实体方法声明：

```
public  void test(int a){
        方法体
}
```

l 抽象方法所在的类必须为抽象类

也就是说，如果抽象方法声明在一个类内部，则该类必须为抽象类。(说明：抽象方法也可以出现在接口内部，这个将在后续进行介绍)。

这样，在继承时，如果继承的类是抽象类，而该抽象类中还包含抽象方法时，则该子类必须声明成抽象类,否则将出现语法错误。如果子类需要做成实体类的话，则必须覆盖继承的所有抽象方法。这个是抽象类最核心的语法功能——强制子类覆盖某些方法。

介绍了这么多抽象类和抽象方法的知识以后，那么抽象类有什么用途呢？

抽象类的用途主要有两个：

- **严禁直接创建该类的对象**

如果一个类内部包含的所有方法都是 static 方法，那么为了避免其它程序员误用，则可以将该类声明为 abstract，这样其它程序员只能使用类名.方法名调用对应方法，而不能使用对象名.方法名进行调用。这样的类例如 API 中的 Math 类

说明：配合 final 关键字使用，将必须该类被继承，这样将获得更加完美的效果。

- **强制子类覆盖抽象方法**

这样可以使所有的子类在方法声明上保持一致，在逻辑上也必须将方法的功能保持一致。例如游戏中设计类时，设计了怪物类以及相关的子类，每个怪物类都有移动方法，但是每种怪物的移动规则又不相同，这样通过使每个怪物类的移动方法的声明保持一致，方便调用。可以参看前面多态部分的介绍获得更多的关于调用统一知识。

这是抽象类最主要的用途。就像现实社会中，各种银行网点保持统一的装修风格，各种快餐店(肯德基、麦当劳等)保持统一的装修甚至风味，这样便于生活中的识别。通过让存在继承关系的类中功能一样(但是内部实现规则不同)的方法声明成一样的，方便多态的使用。

那么什么时候在设计时使用抽象类呢？这个问题参看一下抽象类的用途自然就知道了。关于抽象类的知识先介绍这么多，下面介绍接口的知识，最终将对抽象类和接口进行一下比较。

Java 编程那些事儿 69——抽象类和接口(二)

陈跃峰
出自：http://blog.csdn.net/mailbomb

8.9.2 接口

接口(Interface)是一种复合数据类型。

至此，Java语言的所有数据类型介绍完了，下面进行一个简单的总结。Java语言的数据类型分为两大类：基本数据类型和复合数据类型，其中基本数据类型有8种，复合数据类型包括数组、类和接口，由于开发过程中可以根据需要声明新的复合数据类型，所以复合数据类型的数量有无限个。

接口的概念，现实中使用的也很多，例如大家经常使用的U盘，则需要和计算机上的USB接口匹配使用，而且USB设备中除了U盘以外还有很多，例如USB风扇、USB数据线、USB鼠标、USB键盘等，他们都使用计算机上统一的USB接口，这样设备的通用性很强。简化了计算机接口的设计，使计算机不需要具备鼠标接口、键盘接口等专用的结构。

广义上来说，两个人说不同的方言，互相之间无法听懂另一方表达的意义，我们也可以称之为双方使用的接口不统一，CPU无法和主板匹配，我们也可以称之为接口不统一，例如AMD和Intel的CPU采用不同的针脚结构，甚至同一厂商不同型号的CPU针脚结构也不统一，这样很不方便设备之间的匹配，使用专业的技术术语叫作兼容性差。

那么什么是接口呢？其实接口就是一套规范。

例如USB接口，分为两套规范：公接口和母接口。例如U盘、USB鼠标上的USB接口为公接口，而电脑上的USB接口为母接口。规范中只规定公接口有4个通道，那些用来传输数据、那些用来进行供电，母接口规范只规定也有4个通道，那些用来传输数据，那些用来进行供电，电压是多少电流多大等。所有的这些规范都只规定了必须实现那些功能，但是却没有规定如何进行实现。

这种只规定实现什么功能，而不限制如何进行实现的结构，在程序设计领域中称作“设计和实现相分离”，其中规定实现的功能属于设计部分，而如何实现功能则是实现部分。这样进行程序项目制作，可以让一部分人专门进行项目设计，而由另一部分人进行项目实现。这点，很类似汽车的制造，由设计人员设计汽车，由制造人员进行制造。

这种“设计和实现相分离”的结构将极大的简化程序项目的设计和管理，使得整个项目的分工更加细致，也就是使程序设计完全独立出来，而在设计完成以后再进行代码编写。

接口就是一个纯粹用来设计的数据类型，在接口这种数据类型中，只

能书写两类声明的结构：

- **常量数据**

所有的常量数据都是public static的。如果声明时不书写则系统将自动添加这两个修饰符。

- **抽象方法**

接口中的所有方法都只在逻辑上规定该方法的作用，而不能书写方法体。所有接口中的方法都是public abstract的，如果声明时不书写则系统将自动添加这两个修饰符。

其中接口中的数据是常数，以后不能改变，而方法只是规定要做什么，而不去规定如何进行实现。这样接口就很方便设计人员进行设计，而不必过多的关系对应的方法如何在逻辑上进行实现。
接口声明的语法格式如下：

访问控制符 interface 接口名 [extends 父接口名1,父接口名2.....]{
常量声明
方法声明
}

和类的声明一样，访问控制符只能使用public和默认的。声明时使用interface关键字声明接口，接口可以继承其它的接口，使用extends关键字进行继承，多个接口名之间使用逗号进行分隔。和类的集成一样，子接口继承父接口中所有的常量数据和方法，子接口的对象也是父接口的对象。

注意：和抽象类一样，接口只能声明对象，而不能创建对象。

接口声明的示例代码如下，例如声明一个USB接口来代表实际使用中的USB结构：

```
public interface USB{
        /**电压*/
        public static final int V = 5;
        /**读取数据*/
        public abstract byte[] readData();
        /**写入数据*/
        public abstract void writeData(byte[] data);
}
```

该接口中规定电压常量为5V，声明了两个方法，要求实现USB时必须

实现这样两个方法，至于如何实现这里不做规定。这样这个数据类型就只是设计上的说明，而不牵扯具体的实现，这样在项目中使用时则比较通用。

从这点来看，接口类似于现实中使用的各个国家标准，标准中只规定该类型最终需要达到的标准，而不规定如何实现，各个厂商可以根据自己的产品工艺实现该要求即可。

在实际的项目中，设计接口需要对于项目的整体有比较深刻的了解和认识，这样才可以设计出需要的接口结构，关于接口的设计这里不作太深入的论述。如果需要更深刻的了解设计的结构，可以参阅 OReilly 的《Designing.Interfaces》一书。

接口设计完成以后，还需要再项目中实现接口规范中对应的要求，一般声明对应的类来实现接口，实现接口的语法为：

访问控制符 [修饰符] class 类名 [extends 父类名] implements 父接口名 1,父接口名 2……

实现接口的语法位于类声明中,位于继承声明的后面,使用 implements 关键字代表实现,后续是需要实现的接口的名称,一个类可以实现任意多个接口。

实现接口和继承类很类似，声明的类称作接口的子类，接口为该类的父接口，子类中继承父接口中所有的数据和方法，因为接口中所有的方法都是抽象方法，所以如果子类中不实现(覆盖)父接口中的方法，则该类必须声明为抽象类。

例如计算机实现了 USB 接口，则示例代码如下：

```
public class Computer implements USB{
        /**内存容量*/
        int memorySize;
        public abstract byte[] readData(){
                    //读数据的逻辑
        }
        public abstract void writeData(byte[] data){
                    //写数据的逻辑

        }
}
```

这里，Computer 类实现了前面的 USB 接口，在 Computer 类内部可以书写和 Computer 类相关的属性、方法和构造方法，这里对于实现接口没有影响，

而因为实现了 USB 接口，则必须覆盖 USB 接口中的 readData 和 writeData 抽象方法，至于方法内部的代码，则根据逻辑的需要进行实现，从而实现接口中要求实现的功能。

类似的，也可以使一个数码相机实现 USB 接口，则实现的示例代码为：

```
public class DigitalCamera implements USB{

        /**厂商名称*/
        String vendorName;
        public abstract byte[] readData(){
                //读数据的逻辑
        }
        public abstract void writeData(byte[] data){
                //写数据的逻辑
        }
}
```

在该类中，也可以根据该类的需要实现 USB 接口中规定的功能，至于如何实现则很可能和 Computer 类不同。

这样，虽然 Computer 类和 DigitalCamera 类自身原来的功能不同，但是都可以通过实现 USB 接口而实现同样的功能，这样单纯的从是否支持 USB 功能来看，这两个类的实现是一样的。按照面向对象的术语来说，这被称作屏蔽了类的类型之间的不同，保证了程序的通用性。

由于实现接口时，不限制实现的接口的数量，则任何一个类都可以实现任意多个接口，这样就使类的通用性获得了极大的增强，很方便类的对象之间的匹配。就像现实中 USB 接口规范方便了多种不同设备之间的互联一样。

在语法上，实现了接口的对象可以使用子类的构造方法进行创建，这样又很适合多态的结构，可以说接口的出现，使多态的特性更容易的进行实现。

在实际项目中，通过使用一定的接口，使得很多类的对象在实现某种类型的功能时，方法的声明是统一的，便于程序的调用和管理，利于程序项目的扩展。所以在现在的面向对象编程领域中，存在着另外的一个方向——面向接口的编程，其实很多 Java 的技术都是这样进行实现的，例如 JDBC 部分。

由于抽象类和接口的功能比较类似，后续将对于抽象类和接口进行一系列的比较，方便项目设计时的取舍。

Java 编程那些事儿 70——抽象类和接口(三)

陈跃峰
出自：**http://blog.csdn.net/mailbomb**

8.9.3 抽象类和接口的比较

抽象类和接口都是进行面向对象设计时专用的设计结构，在实际进行项目设计时，经常需要考虑的问题就是——“使用抽象类还是接口”？下面通过对于抽象类和接口进行简单的比较，熟悉两者之间的区别和联系，从而在实际设计时使用恰当的结构。

1.什么时候使用抽象类或接口？

当设计中为了规范类中方法声明的结构(即类的行为)时，使用抽象类或接口。也就是强制子类对外部提供统一的方法声明时，使用抽象类或接口。

2.抽象类和接口的区别(不同点)

a)抽象类是类，而接口是接口。

因为抽象类是一个类，所以类内部可以包含的内容(构造方法、方法和属性等)在抽象类内部都可以存在，当然抽象类也受到类的单重继承的限制。而接口是接口类型，所以接口内部只能包含常量属性和抽象方法，但是一个类可以实现多个接口，所以接口不受类的单重继承的限制。

b)抽象类内部可以包含实体方法，而接口不能

抽象类是一个类，所以在抽象类内部既可以包含抽象方法也可以包含实体方法，而接口内部的每个方法都必须是抽象方法。

c)抽象类可以继承类，而接口不能

抽象类是一个类，所以在设计时可以使抽象类继承其它的类，在已有类的基础上进行设计，但是接口不能继承类。

3.抽象类和接口的联系(相同点)

a)抽象类和接口都可以声明对象，但是都只能使用子类的构造方法进行创建。

b)抽象类和接口内部都可以包含抽象方法。

按照 Java 语言的语法，子类在继承抽象类或实现接口时，都必须覆盖这些抽象方法，否则必须声明为抽象类。

c)抽象类和接口都可以代表一种类型，从而可以统一子类对象的类型，获得良好的可扩展性。

4.什么时候使用抽象类？

当满足以下的条件时，最好使用抽象类进行设计：

a)子类不继承其它父类

b)子类中存在完全相同的功能实现的方法

c)子类中存在相同的属性

d)设计出的结构需要继承其它类

当需要满足 d 条件时，只能使用抽象类，否则也可以考虑使用接口实现。

5.什么时候使用接口？

当满足以下的条件时，最好使用接口进行设计：

a)子类已经继承了其它父类

b)子类中不存在完全相同的功能实现方法

c)子类中不存在相同的属性

d)设计出的结构不需要继承其它类

当需要满足 a 条件时，只能使用接口，否则也可以考虑使用抽象类实现。

6.抽象类和接口的其它用途

a)禁止创建该类的对象时，可以把该类声明为抽象类。
b)当需要存储大量的常量数据，而这些常量数据将会在项目中的多个类之间使用时，可以使用接口。
c)当需要统一具有某种功能的类的对象时，可以使用接口。例如 Serializable 接口。

当然，只有经过大量的系统设计训练以后，才可以更加深刻的理解抽象类和接口的区别和联系，从而更加自如的进行选择。

另外，需要说明的是，不是每个项目中都必须使用抽象类或接口的。

Java 编程那些事儿 71——内部类简介

陈跃峰
出自：http://blog.csdn.net/mailbomb

8.10 内部类(Inner Class)

内部类是 Java 语言中的一种特殊的语法，简单的来说，就是在一个类的内部再声明一个类，这些声明在类内部的类就被称作内部类。在实际声明时，内部类可以声明在类的内部、类的方法内部，也可以声明在类的构造方法内部，内部类声明的语法格式和一般类的声明一样,只是内部类声明时可以使用static修饰符进行修饰。

对于内部类的相关使用，本部分不做深入的介绍，只是进行简单的说明，辅助建立内部类的概念，能够进行一些基本的使用。

下面是一个简单的内部类的示例代码：

```
/**
 * 内部类基本使用示例代码
 */

public class OutClass {
        int i = 0;
        public class InnerClass{
                public void test(){
                        i++;
                }
        }
}
```

在该示例代码中，类 InnerClass 声明在了类 OutClass 的内部，所以 InnerClass 被称为内部类，而 OutClass 则被称为 InnerClass 的外部类。

该代码编译以后，将生成两个 class 文件，一个是 OutClass.class，

另外一个是 OutClass$InnerClass.class。这里需要说明的是，内部类也被编译成一个独立的类文件，类文件的名称为：外部类类名$内部类类名.class。

在内部类中，可以很方便的访问外部类的属性 i，而外部类不能直接引用内部类中的属性和方法，语法中提供了一套专门的格式来访问内部类中的属性和方法，这些语法这里就不作介绍了。

内部类是 Java 语言诞生以后新增的一个语法，设计该语法的初衷主要有两个：

1、隐藏内部类的实现

也就是将只有被外部类使用的功能隐藏在内部类的内部，其它类访问这个内部类时的语法会比较复杂，从而在一定程度了避免了其它类对于内部类的访问。

2、内部类可以访问外部类的所有属性和方法，避免了参数传递

在内部类中，可以访问外部类中的所有属性和方法，private 访问控制符修饰的属性和方法也可以被内部类访问，这样将方便内部类的编写，避免了参数传递，也减少了外部类需要向其它类开放的属性和方法的数量。

当然，内部类除了带来一系列的好处以外，还带来了一系列的不足，主要的不足有如下几点：

1、增加了语法的复杂度

在一个类的内部声明一个类，使 Java 语言的复杂度增加了很多，也在一定程度上损害了 Java 语言的编程风格，同时使阅读代码的难度大幅度的增加。

2、使整个项目的结构变得复杂

内部类通过增加每个类的复杂度，使得项目的结构变得更加复杂，而且没有原来的结构那么清晰。

当前内部类的语法在以下编程技术中的应用比较多：

1、图形用户界面编程中的事件处理

2、Java 语言 API 中部分功能类的实现

综合以上说明，在实际的开发过程中，如果不是十分必要，不建议编写内部类，对于内部类实现的功能可以使用类和类之间的调用进行实现，这样可以保留 Java 语言清晰的编程风格。

最后需要特别说明的是：在 Java 语法中，一个源代码内部可以编写多个类，例如：

```
/**
 * 一个类内部声明多个类
 */

public class TwoClassInOneFile {
        int n = 0;
        public void method(){}
}
class AnotherClass {
        char c;
        public void t(){}
}
```

这样的源代码，在编译时，将生成两个独立的 class 的文件：TwoClassInOneFile.class 和 AnotherClass.class，这样的结构不是内部类。

在实际项目中，不推荐进行这样的编码，主要原因是不方便类源代码的查找，而推荐每个源文件内部只声明一个类的形式。

Java 编程那些事儿 72——包的概念

陈跃峰
出自：http://blog.csdn.net/mailbomb

8.11 包的概念

随着项目复杂度的增加，一个项目中需要实现的类和接口的数量也将快速增长，为了方便对于代码的管理和阅读，需要将这些类和接口按照一定的规则进行分类，这就是程序设计中命名空间(name space)概念出现的原因。

在 Java 语言中，为了对同一个项目中的多个类和接口进行分类和管理，专门设计了包(Package)的概念，使用包管理和类和接口。

其实出现包的概念最开始的原因是避免类名重复，现在这个作用也得到了广泛的应用。例如各个公司会为同一个功能实现一套类，这些类名之间大量的存在重复，使用包的概念就可以很方便的解决类名重复的问题。

那么包到底是什么呢？其实包是一个逻辑的概念，就是给类名加了一个前缀，就像北京路这个路名上海和南京都有，再说这个路名时，就可以使用南京市北京路和上海市北京路进行区分，这里的北京路就相当于类，而前面的前缀，

如南京市或上海市，就相当于包名。

在物理上包被转换成一个文件夹，操作系统通过文件夹来管理各个类和接口，从而实现对类的分类，或者称之为按照类的功能对这些类进行封装。

下面介绍一下和包有关的两个语法：打包和引入包。

8.11.1 打包

打包的作用就是将声明的类放入包中，也就是为类指定包名。在实际的项目中，一般根据类的功能来设定包，例如设置界面类包，逻辑类包，网络类包等结构。

打包的语法格式为：

package 包名 1[.包名 2[.包名 3……]];

在该语法中，包名可以设置多个，包名和包名之间使用“.”进行分割，包名的个数没有限制。其中前面的包名包含后面的包名。按照 Java 语言的编码规范，包名所有字母都小写，由于包名将转换为文件夹的名称，所以包名中不能包含特殊字符。

例如：

```
package test;
package demo.chapter8;
package game.bubble.ui;
```

在示例中的最后一个，package game.bubble.ui，该包名将转换为路径\game\bubble\ui，也就是前面的包名是后面包名的父目录。

需要特别注意的是：

1、打包的语句必须是程序代码中的第一行可执行代码，也就是说打包语句的上面只能包含空行或注释。
2、打包的语句最多只有一句。

如果在代码中没有书写 package 语句，则该类将被打入到默认包中，默认包无法被其它的包引用。

示例代码：

```
//文件名：PackageClass.java
package chapter8;
public class PackageClass{
        //类内部的代码
}
```

通过打包语法，可以为每个类增加一个前缀，也就是说以前使用类，

使用类名即可，有了包名的概念以后，类的全名就是包名.类名了，通过在类名前面加入包名，可以使不同包中存在系统的类名，但是一般不建议一个项目中的类名存在重复。

当一个类有了包名以后，在编译和运行时将和前面的操作产生一些变化，下面介绍一下 JDK 和 Eclipse 中相关操作的变化。

a)JDK 编译和运行打包的源文件

编译源代码的命令：javac –d 类文件路径 源代码名称

例如：javac –d d:\ PackageClass.java

该命令的作用是将 PackageClass 类编译成 class，并将生成的 class 文件存储到 d 盘根目录下。则 class 文件的存储路径是：d:\chapter8\PackageClass.class。使用该命令编译时自动将包名转换为文件夹。

如果想将编译生成的 class 文件生成在源代码所在的目录，则可以使用如下格式：**javac –d . 源文件名**

这里的源文件没可以使用*.java 代表当前目录下的所有源文件。

说明：执行 javac 命令必须将目录切换到源代码所在的目录。

运行类文件的格式： java 包名.类名

例如：java chapter8.PackageClass

该命令的作用是运行该类，按照上面的编译命令，运行该程序时，不应该切换到 class 文件所在的目录，而是切换到最顶层的包名所在的目录，所以执行以上命令时，在控制台下面只需要切换到 d 盘根目录即可。

b)eclipe 编译和运行打包的源代码

在新建源代码时，可以在新建类向导中，可以在 package 栏中添加包名，在 eclipse 编译时会自动生成包名的结构。

运行带包名的代码也和以前的运行步骤一样。

需要特别注意的是：在 eclipse 中，源代码的组织也要求将包名转换为文件夹，否则将出现语法错误，这点在复制代码到 eclipse 项目中时需要特别小心。

8.11.2 引入包

当类进行了打包操作以后，同一个包内部的类默认引入，当需要使用其它包中的类时，则必须首先引入对应的类，然后才可以使用该类。

引入包的语法格式为：

import 包名 1.[.包名 2[.包名 3……]].类名|*;

在该语法中，import 关键字后面是包名，包名和包名之间使用“.”分割，最后为类名或“*”，如果书写类名则代表只引入该类，如果书写星号则代表引入该包中的所有类、接口、异常和错误等。

例如：

```
import chapter8.*;
import java.io.*;
import chapter8. PackageClass;
```

引入包的代码书写在类声明语句的上面，打包语句的下面，import 语句在一个代码中可以书写任意多句，例如：

```
package chapter8;
import java.io.*;
import java.net.*;
public class Test{
  //其它代码
}
```

说明：这里的 java.io 包和 java.net 包都是 JDK 中提供的包名。

每引入一个类或接口，将在内存中占有几十个字节的空间，所以使用星号引入一个包中所有的类或接口虽然编写起来比较简单，但是却浪费了一些内存，而一个一个的引入类名编写起来又比较麻烦。

小提示：使用 eclipse 中的 source 菜单下的 Orgnize Imports 可以将星号的引用转换为对类名的引用。

最后需要特别注意的是，import 只引入当前包下面的类，而不引入该包下面的子包里面的类。例如：

```
//文件名：A.java
package a;
public class A{}
//文件名：B.java
package a.b;
public class B{}

//文件名：Test.java
import a.*;
public class Test{
        public static void main(String[] args){
                B b = new B();   //语法错误，B 类未引入
        }
}
```

说明：在该示例的 Test 类中，使用 import a.*;引入了包 a 中的所有类，也就是类 A，而包 a 的子包 a.b 中的类将不会被引入，所以在 Test 类中使用 B 类将出现语法错误。

8.11.3 包的概念小结

总之，包的概念(也就是命名空间)使得类名冲突的问题获得良好的解决，另外也可以使用包来分类管理一个项目中的多个类，使得项目中的类数量比较多时，结构显得比较清晰。

到现在为止，综合以前学习的所有语法知识，规范的类结构的代码如下：

```
/*
 *  版权信息
 */
打包语句
引入语句
/**
 * 类注释
 */
类声明
```

JDK 中的源代码代码(位置在 JDK 安装目录下的 src.zip 中)均以这样的格式组织类，这也是规范编写类结构代码的基础。

8.12 总结

面向对象(OO)是一种思想，该思想通过改变数据和功能的组织方式，从而彻底改变了整个项目中代码的结构，也使得每个类的功能单纯，从而极大的降低了项目的复杂度，使得项目开发更方便的进行模块化，从而可以为开发更复杂的项目提供了机会。

本章虽然使用了不少的篇幅来介绍面向对象的知识，但是这些还只是一些基础，更多的知识需要在实际的编程实践中去进行体会。

当掌握了本章的知识以后，就可以比较轻松的去阅读更专业的面向对象知识的书籍，比如面向对象理论以及设计模式相关的书籍，从而提高对于面向

对象的认识。

面向对象的知识就介绍这么多，将在下一章中介绍 JDK 中类文档的使用，并结合文档的使用介绍如何使用 JDK 中提供的类，从而进入 Java 开发技术的学习。

Java 编程那些事儿 73——JDK 文档使用

陈跃峰
出自：http://blog.csdn.net/mailbomb

第九章 JDK 文档使用

前面提到过，一个程序设计语言主要包含三个部分：语言的语法、开发工具和一套基础的功能。Java 语言的语法以及开发工具 JDK 前面都已经介绍过了，本章就将介绍 Java 语言提供的这套基础的功能。

对于任何一个程序设计语言来说，都将提供一套已经编写完成的基础功能，这种和语言一起发布的这套基础功能一般被称作 API(Application Programming Interface,应用编程接口)，在面向过程的语言中也被称为函数库(Function Library)，在面向对象的语言中也被称作类库(Class Library)。

这套功能都会以专门的文档来进行提供，在 J2SE 开发中，这套文档称之为 JDK 文档。例如 Windows 操作系统的基本功能称为 Windows API，而和 VB、VC 开发相关的功能组成的文档称之为 MSDN 等。

这套功能就是在学习程序设计过程中需要学习的基本开发技术。

9.1 JDK 文档概述

JDK 文档中包含了 JDK 中开放给程序员的所有通用类结构的说明，该文档是学习 J2SE 编程的必备参考资料，该文档由 SUN 公司以免费的 HTML 文档的形式进行提供，当然也可以下载到其他人制作的 CHM 格式的文档。

JDK 文档没有随 JDK 一起发布，如果需要使用该份文档，则需要到 SUN 公司的网站专门进行下载，最新版本的 JDK 文档下载地址为：

http://java.sun.com/javase/downloads/?intcmp=1281

选择“Java SE 6 Documentation”项目下载即可。

在 06 年初，SUN 公司组织专人将 JDK5.0 的文档翻译成了中文，如果英文阅读比较困难，则可以使用中文文档，其下载地址为：

http://gceclub.sun.com.cn/chinese_java_docs.html

说明：建议阅读英文文档，因为这样既可以锻炼英文阅读能力，也可以阅读到最新的文档。

注意：使用的文档的版本最好和使用的 JDK 的版本匹配。

JDK 的文档以网页文件压缩包的形式提供，下载完成以后，只需要解压缩即可，可以根据需要解压缩到任意的路径。

9.2 JDK 文档结构

打开 JDK 文档所在目录下 api 目录中的 index.html 即可打开文档结构，打开以后的界面如下图所示：

JDK API 文档图

在该文档中，页面的左上角区域显示 JDK API 中所有包名的列表，右下角区域默认显示所有类结构，如果选择对应的包名，则只显示该包中的接口、类、异常和错误等信息。页面右侧区域显示详细信息，当选择对应的类时，则显示该类的详细信息。

例如选择 java.lang 包中的 String 类时，显示的界面如下图：

String 类的文档结构

在类的文档中主要包含以下几部分内容：类的继承和被继承关系，类的声明、类的功能说明、属性列表、构造方法列表和方法列表等。类内部结构中的属性、构造方法和方法每个都包含一个超链接，通过点击该链接可以查看更详细的说明。

9.3 类和接口使用语法

在 JDK 文档中，可以很方便的查阅到每个类的作用和类内部的结构，如构造方法、属性和方法等，在实际的开发过程中，则需要根据项目的要求使用对应类。

具体该使用哪个类或接口，以及如何使用类或接口，这些是开发技术的要求，本部分只是总结一下和类和接口使用相关的语法知识。

无论使用类还是接口，首先需要引入该结构，使用的语法格式为：

import 包名.类名|*;

说明：这里的“|”表示或者的关系。

注意：java.lang 包中的类和接口，系统会自动引入，所以该包中的类和接口无需引用即可使用。

9.3.1 类使用语法

对于从 JDK 文档中查阅出的类，使用的方法一般只有三种：继承该类、使用该类的对象和调用该类中的静态属性或静态方法。下面一一介绍这三种使用方式相关的语法。

9.3.1.1 继承类的语法

如果以继承的方式使用 JDK 文档中提供的类，则对应的语法格式如下：

1、构造方法

首先需要说明的是，构造方法不被继承。也就是说子类中的构造方法和父类中的构造方法的参数列表没有关联。

如果被继承的类，也就是 JDK 文档中提供的类，有默认的构造方法(也就是无参数的构造方法)，则子类的构造方法内部会自动调用，不需要编写特殊的代码。

如果被继承的类中没有默认的构造方法，则必须在子类中书写构造方法，而且需要在子类的构造方法中的第一行使用 super 关键字调用父类中对应的构造方法，在使用 super 调用时也必须传入对应的参数。

如果父类中没有提供 public 或 protected 修饰的构造方法，则该类不能被继承。

2、属性和方法

继承了一个类以后，该类中的所有属性和方法都被继承下来，在子类中可以像使用自己声明的属性和方法一样使用这些结构。

说明：在 JDK 文档中只将被覆盖的方法单独列举在文档中，未被覆盖的属性和方法则显示在属性和方法列表的下面。

最后一个和继承有关的语法是，如果继承的类是抽象类，则必须覆盖父类中的抽象方法，否则声明的类则必须声明为抽象类。

9.3.1.2 使用类的对象

如果通过创建对象对类进行使用，这种方式是最常用的方式，则相关的语法格式如下：

1、构造方法

首先声明对象，然后使用 new 关键字和构造方法来创建对象。当构造方法比较多时，根据需要调用合适参数列表的构造方法，调用时仔细

阅读参数列表即可。

格式为：

类名 对象名 = new 构造方法(参数);

2、属性和方法

当对象创建完成以后，则可以通过该对象调用类提供的属性和方法了。

调用属性和方法的语法格式为：

对象名.属性 或 对象名.方法(参数)

9.3.1.3 调用类的静态属性和静态方法

调用类中的静态属性和静态方法，语法中提供了简单的格式：

类名.静态属性名 和 类名.静态方法名(参数)

注意：对于类内部的静态属性和静态方法，不推荐使用对象名.静态属性名和对象名.静态方法名进行调用。

9.3.2 接口使用语法

对于 JDK 文档中提供的接口，其使用方式一般有两种：声明该接口的对象、实现该接口和继承接口。接口的使用比类要简单一些，下面一一介绍相关的语法格式。

9.3.2.1 声明接口对象

对于一部分接口，需要声明一个接口对象，然后使用 JDK 中提供的方法或对应的子类构造方法进行创建，但是需要首先声明接口的对象。

语法格式为：

接口名 对象名;

9.3.2.2 实现接口

另外一种更常见的使用接口的方式是声明一个类实现接口，一个类可以实现任意多个接口，使用 implements 关键字实现，如果接口中包含抽象方法，则必须覆盖抽象方法，否则必须声明为抽象类。

9.3.2.3 继承接口

也可以声明一个接口，继承 JDK 文档中已有的接口，接口继承接口支持多重继承，使用 extends 关键字继承即可。

Java 编程那些事儿 74——java.lang 包介绍 1

陈跃峰
出自：**http://blog.csdn.net/mailbomb**

9.4 JDK API 包名综述

在整个 JDK API 中，大约包含 1、200 个包，总体来看，包名第一个名称主要有三种：java、javax 和 org。其中以 java 开头的包名是 JDK 的基础语言包，以 javax 开头的属于 JDK 扩展包(其中 x 是 extend 的简写)，而以 org 开头的则是第三方组织提供的功能包(org 是 organization 的简写)。而在 JDK API 中还包含了一些以 com.sun 开头的包名，这些是 SUN 公司提供的一些功能包，由于这些包中的类随着 JDK 版本的更改变化很大，不具备兼容性，所以未在标准的 JDK API 文档中进行公开。

在本章接下来的内容中，首先介绍常用的类的功能以及基本使用，这些介绍主要涵盖 java.lang 包和 java.util 包中的内容。

9.5 java.lang 包

java.lang 包是 Java 基础语言包(其中 lang 是 language(语言)的简写)，该包中包含 Java 语言所需要的基本的功能类、接口等信息，是进行 Java 语言编程的基础。

由于在进行 Java 语言编程时，该包的使用特别频繁，所以在 Java 语言中，该包是被默认引入的。

下面就介绍一下该包中常见类的功能以及相关使用。

9.5.1 Object

Object 类 Java 语言的灵魂，因为所有的类(除了 Object 类)，都是该类的子类，即使不书写继承，系统也会自动继承该类，所以 Object 是整个 Java 语言继承树的唯一一个根，这就是 Java 语言特色的单根继承体系。包括数组也实现了该类中的方法。

由于 Java 语言的这种单根继承体系，所以整个 Java 语言的结构中很方便的实现了很多复杂的特性，例如多线程等控制，也可以很方便的对于整个 Java 语言体系进行更新。

由于Object类是Java语言中所有类的父类，所以Object类中的方法将出现在每个类的内部，熟悉该类中的常见方法中的使用，是每个程序员学习的基础。

1、equals方法

equals方法实现的功能是判断两个对象的内容是否相同。Object类中该方法的实现很简单，Object类中equals方法实现的代码如下(说明：该代码可以从JDK安装目录下的src.zip中找到)：

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

在Object类中方法的实现比较简单，如果真正需要在项目中进行比较时，这个equals方法的作用是无法达到实际的要求的。所以如果在项目中涉及的类需要比较该类型的对象时，则必须覆盖equals方法。

下面以一个简单的类为示例，编写一个简单的equals方法，源代码如下：

```
/**
 * equals方法编写示例
 */

public class MyEquals {

        /**对象成员变量*/
        String name;
        /**基本数据类型成员变量*/
        int n;
        /**
         * 判断对象内容是否相同
         * @param obj 需要比较的对象
         */
         public boolean equals(Object obj){
            //如果比较的内容是自身
            if(obj == this){
                return true;
             }
             //对象类型不同
            if(!(obj instanceof MyEquals)){
                    return false;
```

```
            }
            //转换成当前类类型
            MyEquals m = (MyEquals)obj;
            /*依次比较对象中每个变量*/
            //name 属性不同

            if(!name.equals(m.name)){
                return false;
            }
            //n 属性不同
            if(!(n == m.n)){
                  return false;
            }
           //如果都相同，则返回 true
            return true;
        }
}
```

在实际比较时，首先判断是否是自身，然后再判断对象的类型是否符合要求，可以使用 instanceof 关键字进行判断，该运算符的语法格式为：

对象名 instanceof 类名

如果对象名是后续类名类型的对象，则结果为 true，否则为 false。

如果类型符合要求，然后就可以依次比较对象中每个属性的值是否相同了，如果有一个属性的值不相同则不相等。

2、finalize 方法

finalize 方法的作用和前面介绍的构造方法的概念刚好相反，构造方法的作用是初始化一个对象，而 finalize 方法的作用是释放一个对象占用的内存空间时，会被 JVM 自动调用的方法。

说明：finalize 方法的作用和 C++中析构函数的作用一样。

如果在对象被释放时，需要执行一些操作的话，则可以在该类中覆盖 finalize 方法，然后在方法内部书写需要执行的代码即可。

3、hashcode 方法

hashcode 方法的作用是获得一个数值，该数值一般被称作散列码，使用这个数

值可以快速判断两个对象是否不相同，主要应用于集合框架中类的快速判断。

两个内容相同的对象，其 hashcode 方法的返回值必须相同，而两个不相同的对象其 hashcode 的值可能相同。

如果自己编写的类需要存储到集合类中，则覆盖该方法可以提高集合类的执行效率。

4、toString 方法

toString 方法是显示对象内容时会被系统自动调用的方法，当输出一个对象的内容时，系统会自动调用该类的 toString 方法，例如输出 Object 类型的对象 obj，则以下两组代码的功能是一样的：

```
System.out.println(obj);

System.out.println(obj.toString());
```

而 Object 类中的 toString 类实现比较简单，其源代码为：

```
public String toString() {

        return getClass().getName() +
"@" + Integer.toHexString(hashCode());

}
```

如果需要自己的类的对象按照一定的格式进行输出，则可以在自己设计的类内部覆盖 toString 方法，然后设计需要的输出格式即可。

至于 Object 类中的其它方法，下面做一个基本的介绍：

- **clone 方法：**复制对象。也就是创建一个和该对象的内容完全一样的对象，新的对象拥有独立的内存空间。

- **getClass 方法：**主要获得对象的类型，该方法主要用于反射技术的实现。

另外的 wait、notify 和 notifyAll 等方法在是为了实现多线程的需要而实现的，将在后续的多线程技术中进行详细的介绍。

9.5.2 Math

Math 类是一个数学工具类，在 Java 语言中，常用的数学常数和数学方法都在该类内部，而且无论是常数还是方法都是 static 类型的，方便程序员进行实际的使用。

下面示例代码是调用Math类中的abs方法实现求数字的绝对值的实现代码：

```
/**
 * Math 类基本使用
 */
public class MathDemo {

                public static void main(String[] args) {
                        int m = -10;
                        int n = Math.abs(m);
                        System.out.println("绝对值是：" + n);

                 }
}
```

由于Math类的方法作用比较简单，这里就不一一举例了，具体的方法请参阅 JDK API 文档。

Java 编程那些事儿 75——String 类使用

陈跃峰
出自：http://blog.csdn.net/mailbomb

9.5.3 String 和 StringBuffer

String 和 StringBuffer 类都是代表字符串，也就是任意多个字符组成的序列。程序需要存储的大量文字一般都使用字符串进行代表。在这两个类中，包含了大部分关于字符串操作的方法，在实际操作字符串时，可以首先查阅这两个类中的方法。

String 和 StringBuffer 虽然都代表字符串，但是由于两个类内部实现的区别，所以一般把 String 看成不可变字符串，而把 StringBuffer 看成可变

字符串，对于String的每次改变(例如字符串连接等)都会生成一个新的字符串，比较浪费内存，而StringBuffer每次都改变自身，不生成新的对象，比较节约内存。

下面就详细介绍一下String和StringBuffer的实际使用。

9.5.3.1 String类

1、String对象的初始化

由于String对象特别常用，所以在对String对象进行初始化时，Java提供了一种简化的特殊语法，格式如下：

```
String s = “abc”;
s = “Java 语言”;
```

其实按照面向对象的标准语法，其格式应该为：

```
String s = new String(“abc”);
s = new String(“Java 语言”);
```

只是按照面向对象的标准语法，在内存使用上存在比较大的浪费。例如String s = new String(“abc”);实际上创建了两个String对象，一个是”abc”对象，存储在常量空间中，一个是使用new关键字为对象s申请的空间。

其它的构造方法的参数，可以参看String类的API文档。

2、字符串的常见操作

a、charAt方法

该方法的作用是按照索引值(规定字符串中第一个字符的索引值是0，第二个字符的索引值是1，依次类推)，获得字符串中的指定字符。例如：

```
String s = “abc”;
char c = s.chatAt(1);
```

则变量c的值是’b’。

b、compareTo方法

该方法的作用是比较两个字符串的大小，比较的原理是依次比较每个字符的字符编码。首先比较两个字符串的第一个字符，如果第一个字符串的字符编码大于第二个的字符串的字符编码，则返回大于0的值，如果小于则返回小于0的值，如果相等则比较后续的字符，如果两个字符串中的字符编码完全相同则返回0。

例如：

```
String s = “abc”;
```

```
String s1 = “abd”;
int value = s.compareTo(s1);
```

则 value 的值是小于 0 的值，即-1。

在 String 类中还存在一个类似的方法 compareToIgnoreCase，这个方法是忽略字符的大小写进行比较，比较的规则和 compareTo 一样。例如：

```
String s = “aBc”;
String s1 = “ABC”;
int value = s. compareToIgnoreCase (s1);
```

则 value 的值是 0，即两个字符串相等。

c、concat 方法

该方法的作用是进行字符串的连接，将两个字符串连接以后形成一个新的字符串。例如：

```
String s = “abc”;
String s1 = “def”;
String s2 = s.concat(s1);
```

则连接以后生成的新字符串 s2 的值是”abcdef”，而字符串 s 和 s1 的值不发生改变。如果需要连接多个字符串，可以使用如下方法：

```
String s = “abc”;
String s1 = “def”;
String s2 = “1234”;
String s3 = s.concat(s1).concat(s2);
```

则生成的新字符串 s3 的值为”abcdef1234”。

其实在实际使用时，语法上提供了一种更简单的形式，就是使用“+”进行字符串的连接。例如：

```
String s = “abc” + “1234”;
```

则字符串 s 的值是”abc1234”，这样书写更加简单直观。

而且使用“+”进行连接，不仅可以连接字符串，也可以连接其他类型。但是要求进行连接时至少有一个参与连接的内容是字符串类型。而且“+”匹配的顺序是从左向右，如果两边连接的内容都是基本数字类型则按照加法运算，如果参与连接的内容有一个是字符串才按照字符串进行连接。

例如：

```
int a = 10;
String s = “123” + a + 5;
```

则连接以后字符串 s 的值是“123105”，计算的过程为首先连接字符串”123”和变量 a 的值，生成字符串”12310”，然后使用该字符串再和数字 5 进行连接生成最终的结果。

而如下代码：

```
int a = 10;
String s = a + 5 + “123”;
```

则连接以后字符串 s 的值是”15123”，计算的过程为首先计算 a 和数字 5，由于都是数字型则进行加法运算或者数字值15,然后再使用数字值15和字符串”123”进行连接获得最终的结果。

而下面的连接代码是错误的：

```
int a = 12;
String s = a + 5 + ‘s’;
```

因为参与连接的没有一个字符串，则计算出来的结果是数字值，在赋值时无法将一个数字值赋值给字符串 s。

d、endsWith 方法

该方法的作用是判断字符串是否以某个字符串结尾，如果以对应的字符串结尾，则返回 true。

例如：

```
String s = “student.doc”;
boolean b = s.endsWith(“doc”);
```

则变量 b 的值是 true。

e、equals 方法

该方法的作用是判断两个字符串对象的内容是否相同。如果相同则返回 true，否则返回 false。例如：

```
String s = “abc”;
String s1 = new String(“abc”);
boolean b = s.equals(s1);
```

而使用“==”比较的是两个对象在内存中存储的地址是否一样。例如上面的代码中，如果判断：

boolean b = (s == s1);

则变量 b 的值是 false，因为 s 对象对应的地址是”abc”的地址，而 s1 使用 new 关键字申请新的内存，所以内存地址和 s 的”abc”的地址不一样，所以获得的值是 false。

在 String 类中存在一个类似的方法 equalsIgnoreCase，该方法的作用是忽略大小写比较两个字符串的内容是否相同。例如：

```
String s = “abc”;
String s1 =”ABC”;
boolean b = s. equalsIgnoreCase (s1);
```

则变量 b 的值是 true。

f、getBytes 方法

该方法的作用是将字符串转换为对应的 byte 数组，从而便于数据的存储和传输。例如：

```
String s = “计算机”;
byte[] b = s.getBytes();    //使用本机默认的字符串转换为 byte 数组
 byte[] b = s.getBytes(“gb2312”); //使用 gb2312 字符集转换为 byte 数组
```

在实际转换时，一定要注意字符集的问题，否则中文在转换时将会出现问题。

g、indexOf 方法

该方法的作用是查找特定字符或字符串在当前字符串中的起始位置，如果不存在则返回-1。例如：

```
String s = “abcded”;
int index = s.indexOf(‘d’);
int index1 = s.indexOf(‘h’);
```

则返回字符 d 在字符串 s 中第一次出现的位置，数值为 3。由于字符 h 在字符串 s 中不存在，则 index1 的值是-1。

当然，也可以从特定位置以后查找对应的字符，例如：

int index = s.indexOf(‘d’,4);

则查找字符串 s 中从索引值 4(包括 4)以后的字符中第一个出现的字符 d，则 index 的值是 5。

由于 indexOf 是重载的，也可以查找特定字符串在当前字符串中出现的起始位置，使用方式和查找字符的方式一样。

另外一个类似的方法是 lastIndexOf 方法，其作用是从字符串的末尾开始向前查找第一次出现的规定的字符或字符串，例如：

String s = “abcded”;
int index = s. lastIndexOf(‘d’);
则 index 的值是 5。

h、length 方法

该方法的作用是返回字符串的长度，也就是返回字符串中字符的个数。中文字符也是一个字符。例如：

String s = “abc”;
String s1 = “Java 语言”;
int len = s.length();
int len1 = s1.length();

则变量 len 的值是 3，变量 len1 的值是 6。

i、replace 方法

该方法的作用是替换字符串中所有指定的字符，然后生成一个新的字符串。经过该方法调用以后，原来的字符串不发生改变。例如：

String s = “abcat”;
String s1 = s.replace(‘a’,’1’);

该代码的作用是将字符串 s 中所有的字符 a 替换成字符 1，生成的新字符串 s1 的值是”1bc1t”，而字符串 s 的内容不发生改变。

如果需要将字符串中某个指定的字符串替换为其它字符串，则可以使用 replaceAll 方法，例如：

String s = “abatbac”;
String s1 = s.replaceAll(“ba”,”12”);

该代码的作用是将字符串 s 中所有的字符串”ab”替换为”12”，生成新的字符串”a12t12c”，而字符串 s 的内容也不发生改变。

如果只需要替换第一个出现的指定字符串时，可以使用 replaceFirst 方法，例如：

String s = “abatbac”;
String s1 = s. replaceFirst (“ba”,”12”);

该代码的作用是只将字符串 s 中第一次出现的字符串”ab”替换为字符串”12”，则字符串 s1 的值是”a12tbac”，字符串 s 的内容也不发生改变。

j、split 方法

该方法的作用是以特定的字符串作为间隔，拆分当前字符串的内容，一般拆分以后会获得一个字符串数组。例如：

String s = “ab,12,df”;
String s1[] = s.split(“,”);

该代码的作用是以字符串”,”作为间隔，拆分字符串 s，从而得到拆分以后的字符串数字 s1，其内容为：｛“ab”,”12”,”df”｝。

该方法是解析字符串的基础方法。

如果字符串中在内部存在和间隔字符串相同的内容时将拆除空字符串，尾部的空字符串会被忽略掉。例如：

String s = “abbcbtbb”;
String s1[] = s.split(“b”);

则拆分出的结果字符串数组 s1 的内容为：｛“a”,””,”c”,”t”｝。拆分出的中间的空字符串的数量等于中间间隔字符串的数量减一个。例如：

String s = “abbbcbtbbb”;
String s1[] = s.split(“b”);

则拆分出的结果是：｛“a”,””,””,”c”,”t”｝。最后的空字符串不论有多少个，都会被忽略。

如果需要限定拆分以后的字符串数量，则可以使用另外一个 split 方法，例如：

String s = “abcbtb1”;
String s1[] = s.split(“b”,2);

该代码的作用是将字符串 s 最多拆分成包含 2 个字符串数组。则结果为:{“a”,”cbtb1”}。

如果第二个参数为负数，则拆分出尽可能多的字符串，包括尾部的空字符串也将被保留。

k、startsWith 方法

该方法的作用和 endsWith 方法类似，只是该方法是判断字符串是否以某个字符串作为开始。例如:

String s = “TestGame”;
boolean b = s.startsWith(“Test”);

则变量 b 的值是 true。

l、substring 方法

该方法的作用是取字符串中的“子串”，所谓“子串”即字符串中的一部分。例如“23”是字符串“123”的子串。

字符串“123”的子串一共有 6 个:”1”、”2”、”3”、”12”、”23”、”123”。而”32”不是字符串”123”的子串。
例如:

String s = “Test”;
String s1 = s.substring(2);

则该代码的作用是取字符串 s 中索引值为 2(包括)以后的所有字符作为子串，则字符串 s1 的值是”st”。

如果数字的值和字符串的长度相同，则返回空字符串。例如:

String s = “Test”;
String s1 = s.substring(4);

则字符串 s1 的值是””。

如果需要取字符串内部的一部分，则可以使用带 2 个参数的 substring 方法，例

如：

```
String s = “TestString”;
String s1 = s.substring(2,5);
```

则该代码的作用是取字符串 s 中从索引值 2(包括)开始，到索引值 5(不包括)的部分作为子串，则字符串 s1 的值是”stS”。

下面是一个简单的应用代码，该代码的作用是输出任意一个字符串的所有子串。代码如下：

```
String s = “子串示例”;
int len = s.length();  //获得字符串长度
for(int begin = 0;begin < len – 1;begin++){   //起始索引值
        for(int end = begin + 1;end <= len;end++){   //结束索引值
            System.out.println(s.substring(begin,end));
    }
}
```

在该代码中，循环变量 begin 代表需要获得的子串的起始索引值，其变化的区间从第一个字符的索引值 0 到倒数第二个字符串的索引值 len -2，而 end 代表需要获得的子串的结束索引值，其变化的区间从起始索引值的后续一个到字符串长度。通过循环的嵌套，可以遍历字符串中的所有子串。

m、toCharArray 方法

该方法的作用和 getBytes 方法类似，即将字符串转换为对应的 char 数组。例如：

```
String s = “abc”;
char[] c = s.toCharArray();
```

则字符数组 c 的值为：{‘a’,’b’,’c’}。

n、toLowerCase 方法

该方法的作用是将字符串中所有大写字符都转换为小写。例如：

```
String s = “AbC123”;
String s1 = s.toLowerCase();
```

则字符串 s1 的值是”abc123”，而字符串 s 的值不变。

类似的方法是 toUpperCase，该方法的作用是将字符串中的小写字符转换为对应的大写字符。例如：

```
String s = “AbC123”;
String s1 = s. toUpperCase ();
```

则字符串 s1 的值是”ABC123”，而字符串 s 的值也不变。

o、trim 方法

该方法的作用是去掉字符串开始和结尾的所有空格，然后形成一个新的字符串。该方法不去掉字符串中间的空格。例如：

```
String s = “   abc   abc   123   “;
String s1 = s.trim();
```

则字符串 s1 的值为:” abc abc 123”。字符串 s 的值不变。

p、valueOf 方法

该方法的作用是将其它类型的数据转换为字符串类型。需要注意的是，基本数据和字符串对象之间不能使用以前的强制类型转换的语法进行转换。

另外，由于该方法是 static 方法，所以不用创建 String 类型的对象即可。例如：

```
int n = 10;
String s = String.valueOf(n);
```

则字符串 s 的值是”10”。虽然对于程序员来说，没有发生什么变化，但是对于程序来说，数据的类型却发生了变化。

介绍一个简单的应用，判断一个自然数是几位数字的逻辑代码如下：

```
int n = 12345;
String s = String.valueOf(n);
int len = s.length();
```

则这里字符串的长度 len，就代表该自然数的位数。这种判断比数学判断方法在逻辑上要简单一些。

关于 String 类的使用就介绍这么多，其它的方法以及这里到的方法的详细声明可以参看对应的 API 文档。

Java 编程那些事儿 76——StringBuffer 类和 System 类

陈跃峰
出自：http://blog.csdn.net/mailbomb

9.5.3.2 StringBuffer 类

StringBuffer 类和 String 一样，也用来代表字符串，只是由于 StringBuffer 的内部实现方式和 String 不同，所以 StringBuffer 在进行字符串处理时，不生成新的对象，在内存使用上要优于 String 类。

所以在实际使用时，如果经常需要对一个字符串进行修改，例如插入、删除等操作，使用 StringBuffer 要更加适合一些。

在 StringBuffer 类中存在很多和 String 类一样的方法，这些方法在功能上和 String 类中的功能是完全一样的。

但是有一个最显著的区别在于，对于 StringBuffer 对象的每次修改都会改变对象自身，这点是和 String 类最大的区别。

另外由于 StringBuffer 是线程安全的，关于线程的概念后续有专门的章节进行介绍，所以在多线程程序中也可以很方便的进行使用，但是程序的执行效率相对来说就要稍微慢一些。

1、StringBuffer 对象的初始化

StringBuffer 对象的初始化不像 String 类的初始化一样，Java 提供的有特殊的语法，而通常情况下一般使用构造方法进行初始化。

例如：

```
StringBuffer s = new StringBuffer();
```

这样初始化出的 StringBuffer 对象是一个空的对象。
如果需要创建带有内容的 StringBuffer 对象，则可以使用：

```
StringBuffer s = new StringBuffer("abc");
```

这样初始化出的 StringBuffer 对象的内容就是字符串”abc”。
需要注意的是，StringBuffer 和 String 属于不同的类型，也不能直接进行强制类型转换，下面的代码都是错误的：

```
StringBuffer s = "abc";                    //赋值类型不匹配
StringBuffer s = (StringBuffer)"abc";      //不存在继承关系，无法进行强转
```

StringBuffer 对象和 String 对象之间的互转的代码如下：

```
String s = "abc";
StringBuffer sb1 = new StringBuffer("123");
StringBuffer sb2 = new StringBuffer(s);      //String 转换为 StringBuffer
```

String s1 = sb1.toString();　　//StringBuffer 转换为 String

2、StringBuffer 的常用方法

StringBuffer 类中的方法主要偏重于对于字符串的变化，例如追加、插入和删除等，这个也是 StringBuffer 和 String 类的主要区别。

a、append 方法

public StringBuffer append(boolean b)

该方法的作用是追加内容到当前 StringBuffer 对象的末尾，类似于字符串的连接。调用该方法以后，StringBuffer 对象的内容也发生改变，例如：

```
StringBuffer sb = new StringBuffer("abc");
sb.append(true);
```

则对象 sb 的值将变成”abctrue”。

使用该方法进行字符串的连接，将比 String 更加节约内容，例如应用于数据库 SQL 语句的连接，例如：

```
StringBuffer sb = new StringBuffer();
String user = "test";
String pwd = "123";
sb.append("select * from userInfo where username=").append(user).append(
" and pwd=").append(pwd);
```

这样对象 sb 的值就是字符串“select * from userInfo where username=test and pwd=123”。

b、deleteCharAt 方法

public StringBuffer deleteCharAt(int index)

该方法的作用是删除指定位置的字符,然后将剩余的内容形成新的字符串。例如：

```
StringBuffer sb = new StringBuffer("Test");
sb. deleteCharAt(1);
```

该代码的作用删除字符串对象 sb 中索引值为 1 的字符，也就是删除第二个字符，剩余的内容组成一个新的字符串。所以对象 sb 的值变为”Tst”。

还存在一个功能类似的 delete 方法：

public StringBuffer delete(int start,int end)

该方法的作用是删除指定区间以内的所有字符，包含 start，不包含 end 索引值的区间。例如：

StringBuffer sb = new StringBuffer(“TestString”);
sb. delete (1,4);

该代码的作用是删除索引值 1(包括)到索引值 4(不包括)之间的所有字符，剩余的字符形成新的字符串。则对象 sb 的值是”TString”。

c、insert 方法

public StringBuffer insert(int offset, boolean b)

该方法的作用是在 StringBuffer 对象中插入内容，然后形成新的字符串。例如：

StringBuffer sb = new StringBuffer(“TestString”);
sb.insert(4,false);

该示例代码的作用是在对象 sb 的索引值 4 的位置插入 false 值，形成新的字符串，则执行以后对象 sb 的值是”TestfalseString”。

d、reverse 方法

public StringBuffer reverse()

该方法的作用是将 StringBuffer 对象中的内容反转，然后形成新的字符串。例如：

StringBuffer sb = new StringBuffer(“abc”);
sb.reverse();

经过反转以后，对象 sb 中的内容将变为”cba”。

e、setCharAt 方法

public void setCharAt(int index, char ch)

该方法的作用是修改对象中索引值为 index 位置的字符为新的字符 ch。例如：

StringBuffer sb = new StringBuffer(“abc”);
sb.setCharAt(1,’D’);

则对象 sb 的值将变成”aDc”。

f、trimToSize 方法

public void trimToSize()

该方法的作用是将 StringBuffer 对象的中存储空间缩小到和字符串长度一样的长度，减少空间的浪费。

总之，在实际使用时，String 和 StringBuffer 各有优势和不足，可以根据具体的使用环境，选择对应的类型进行使用。

9.5.4 System

System类代表系统，系统级的很多属性和控制方法都放置在该类的内部。该类位于 java.lang 包。

由于该类的构造方法是 private 的，所以无法创建该类的对象，也就是无法实例化该类。其内部的成员变量和成员方法都是 static 的，所以也可以很方便的进行调用。

1、成员变量

System类内部包含 in、out 和 err 三个成员变量，分别代表标准输入流(键盘输入)，标准输出流(显示器)和标准错误输出流(显示器)。

例如：

System.out.println(“Test”);

该行代码的作用是将字符串”Test”输出到系统的标准输出设备上，也就是显示在屏幕上。

后续在学习完 IO 相关的知识以后，可以使用 System 类中的成员方法改变标准输入流等对应的设备，例如可以将标准输出流输出的信息输出到文件内部，从而形成日志文件等。

2、成员方法

System 类中提供了一些系统级的操作方法，这些方法实现的功能分别如下：

a、arraycopy 方法

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

该方法的作用是数组拷贝，也就是将一个数组中的内容复制到另外一个数组中的指定位置，由于该方法是 native 方法，所以性能上比使用循环高效。

使用示例：

```
int[] a = {1,2,3,4};
int[] b = new int[5];
System.arraycopy(a,1,b,3,2);
```

该代码的作用是将数组 a 中，从下标为 1 开始，复制到数组 b 从下标 3 开始的位置，总共复制 2 个。也就是将 a[1]复制给 b[3]，将 a[2]复制给 b[4]，这样经过复制以后数组 a 中的值不发生变化，而数组 b 中的值将变成{0,0,0,2,3}。

b、currentTimeMillis 方法

public static long currentTimeMillis()

该方法的作用是返回当前的计算机时间，时间的表达格式为当前计算机时间和 GMT 时间(格林威治时间)1970 年 1 月 1 号 0 时 0 分 0 秒所差的毫秒数。例如：

```
long l = System. currentTimeMillis();
```

则获得的将是一个长整型的数字，该数字就是以差值表达的当前时间。

使用该方法获得的时间不够直观，但是却很方便时间的计算。例如，计算程序运行需要的时间则可以使用如下的代码：

```
long start = System. currentTimeMillis();
```

```
for(int i = 0;i < 100000000;i++){
        int a = 0;
}
long end = System. currentTimeMillis();
long   time = end – start;
```

则这里变量 time 的值就代表该代码中间的 for 循环执行需要的毫秒数，使用这种方式可以测试不同算法的程序的执行效率高低，也可以用于后期线程控制时的精确延时实现。

c、exit 方法

public static void exit(int status)

该方法的作用是退出程序。其中 status 的值为 0 代表正常退出，非零代表异常退出。使用该方法可以在图形界面编程中实现程序的退出功能等。

d、gc 方法

public static void gc()

该方法的作用是请求系统进行垃圾回收。至于系统是否立刻回收，则取决于系统中垃圾回收算法的实现以及系统执行时的情况。

e、getProperty 方法

public static String getProperty(String key)

该方法的作用是获得系统中属性名为 key 的属性对应的值。系统中常见的属性名以及属性的作用如下表所示。

属性名列表

属性名	属性说明
java.version	Java 运行时环境版本
java.home	Java 安装目录
os.name	操作系统的名称
os.version	操作系统的版本
user.name	用户的账户名称

user.home
用户的主目录

user.dir
用户的当前工作目录
例如：

```
String osName = System.getProperty("os.name");
String user = System.getProperty("user.name");
System.out.println("当前操作系统是:" + osName);
System.out.println("当前用户是:" + user);
```

使用该方法可以获得很多系统级的参数以及对应的值。

Java 编程那些事儿 77——包装类

陈跃峰
出自：**http://blog.csdn.net/mailbomb**

9.5.5 包装类

Java 语言是一个面向对象的语言，但是 Java 中的基本数据类型却是不面向对象的，这在实际使用时存在很多的不便，为了解决这个不足，在设计类时为每个基本数据类型设计了一个对应的类进行代表，这样八个和基本数据类型对应的类统称为包装类(Wrapper Class)，有些地方也翻译为外覆类或数据类型类。

包装类均位于 java.lang 包，包装类和基本数据类型的对应关系如下表所示：

包装类对应表

基本数据类型	包装类
byte	Byte
boolean	Boolean
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double

在这八个类名中，除了 Integer 和 Character 类以后，其它六个类的类名和基本数据类型一直，只是类名的第一个字母大写即可。

对于包装类说，这些类的用途主要包含两种：

a、作为和基本数据类型对应的类类型存在，方便涉及到对象的操作。

b、包含每种基本数据类型的相关属性如最大值、最小值等，以及相关的操作方法。

由于八个包装类的使用比较类似，下面以最常用的 Integer 类为例子介绍包装类的实际使用。

1、实现 int 和 Integer 类之间的转换

在实际转换时，使用 Integer 类的构造方法和 Integer 类内部的 intValue 方法实现这些类型之间的相互转换，实现的代码如下：

```
int n = 10;
Integer in = new Integer(100);
//将 int 类型转换为 Integer 类型
Integer in1 = new Integer(n);
//将 Integer 类型的对象转换为 int 类型
int m = in.intValue();
```

2、Integer 类内部的常用方法

在 Integer 类内部包含了一些和 int 操作有关的方法，下面介绍一些比较常用的方法：

a、parseInt 方法

```
public static int parseInt(String s)
```

该方法的作用是将数字字符串转换为 int 数值。在以后的界面编程中，将字符串转换为对应的 int 数字是一种比较常见的操作。使用示例如下：

```
String s = “123”;
int n = Integer.parseInt(s);
```

则 int 变量 n 的值是 123，该方法实际上实现了字符串和 int 之间的转换，如果字符串都包含的不是都是数字字符，则程序执行将出现异常。(说明：异常的概念将在下一章进行讲述)

另外一个 parseInt 方法：

```
public static int parseInt(String s, int radix)
```

则实现将字符串按照参数 radix 指定的进制转换为 int，使用示例如下：

```
//将字符串”120”按照十进制转换为 int，则结果为 120
int n = Integer.parseInt(“120”,10);
//将字符串”12”按照十六进制转换为 int，则结果为 18
int n = Integer.parseInt(“12”,16);
//将字符串”ff”按照十六进制转换为 int，则结果为 255
int n = Integer.parseInt(“ff”,16);
```

这样可以实现更灵活的转换。

b、toString 方法

```
public static String toString(int i)
```

该方法的作用是将 int 类型转换为对应的 String 类型。

使用示例代码如下：

```
int m = 1000;
String s = Integer.toString(m);
```

则字符串 s 的值是”1000”。

另外一个 toString 方法则实现将 int 值转换为特定进制的字符串：

```
public static int parseInt(String s, int radix)
```

使用示例代码如下：

```
int m = 20;
String s = Integer.toString(m);
```

则字符串 s 的值是”14”。

其实，JDK 自从 1.5(5.0)版本以后，就引入了自动拆装箱的语法，也就是在进行基本数据类型和对应的包装类转换时，系统将自动进行，这将大大方便程序员的代码书写。使用示例代码如下：

```
//int 类型会自动转换为 Integer 类型
int m = 12;
Integer in = m;
//Integer 类型会自动转换为 int 类型
int n = in;
```

所以在实际使用时的类型转换将变得很简单，系统将自动实现对应的转换。

Java 编程那些事儿 78——时间和日期处理

陈跃峰
出自：http://blog.csdn.net/mailbomb

9.6 java.util 包

java.util 包是 Java 语言提供的工具类包，该包中包含了如日期、随机数和数据结构实现类等一系列的工具类实现，是学习 Java 语言的基础包之一。

本部分就以 Java 语言中常用的时间和日期处理、随机数处理以及集合框架为基础进行介绍。

9.6.1 时间和日期处理

使用程序进行时间和日期处理，是程序员必须的一种常用技能，在不同的程序设计语言中提供了不同的格式进行实现，现在就介绍一下 Java 语言中的实现方式，以及基本的应用，使得对于 Java 语言的时间和日期处理技术有比较全面的认识。

在程序中，某个固定的时间代表的都是一个时间点，也就是一个时间的瞬间，例如2009年3月8日15点50分0秒，在实际的应用中，经常需要对于两个时间进行比较或计算时间之间的差值，这些功能在Java语言中都可以很方便的实现。

在Java语言中，时间的表达单位是毫秒。也就是说，Java语言中的时间处理可以精确到毫秒。

在Java语言中，表达时间的方式有两种：

a、绝对时间。以直观的形式来表达某个时间点，例如2009年10月10号0点0分0秒。使用这种形式表达时间，使用起来比较直观，但是不方便进行时间之间的计算。例如无法很直观的计算2020年1月1号0点0分0秒和上面这个时间之间相差多少天。绝对时间以对象的形式进行表达，Java API中提供了java.util包中的Date类和Calendar类的对象进行表达。

b、相对时间。以一个long型的数字表达某个时间点。例如102847423468。使用这种方式的优缺点和绝对时间刚好相反。这种方式很方便时间之间的计算，但是阅读起来很不直观。在Java API中以需要表达的时间点，例如2009年10月10号0点0分0秒，和GMT(格林威治时间，也就是伦敦时间)1970年1月1号0点0分0秒之间相差的毫秒数作为相对时间的数值,如果该时间在这个时间只好，则相对时间为正数，否则相对时间为负数。Java API中提供了java.lang包中的System类的currentTimeMillis方法，获得以相对时间形式描述的当前系统时间。

在实际使用时，绝对时间和相对时间之间可以很方便的进行转换。

9.6.1.1 Date类

在JDK1.0中，Date类是唯一的一个代表时间的类，但是由于Date类不便于实现国际化,所以从JDK1.1版本开始,推荐使用Calendar类进行时间和日期处理。这里简单介绍一下Date类的使用。

1、使用Date类代表当前系统时间

```
Date d = new Date();
System.out.println(d);
```

使用Date类的默认构造方法创建出的对象就代表当前时间，由于Date类覆盖了toString方法，所以可以直接输出Date类型的对象，显示的结果如下：

```
Sun Mar 08 16:35:58 CST 2009
```

在该格式中，Sun代表Sunday(周日)，Mar代表March(三月)，08代表8号，CST代表China Standard Time(中国标准时间，也就是北京时间(东八区))。

2、使用Date类代表指定的时间

```
Date d1 = new Date(2009-1900,3-1,9);
System.out.println(d1);
```

使用带参数的构造方法，可以构造指定日期的Date类对象，Date类中年份的参数应该是实际需要代表的年份减去1900，实际需要代表的月份减去1以后的值。例如上面的示例代码代表就是2009年3月9号。

实际代表具体的年月日时分秒的日期对象，和这个类似。

3、获得 Date 对象中的信息

```
Date d2 = new Date();
//年份
int year = d2.getYear() + 1900;
//月份
int month = d2.getMonth() + 1;
//日期
int date = d2.getDate();
//小时
int hour = d2.getHours();
//分钟
int minute = d2.getMinutes();
//秒
int second = d2.getSeconds();
//星期几
int day = d2.getDay();
System.out.println("年份：" + year);
System.out.println("月份：" + month);
System.out.println("日期：" + date);
System.out.println("小时：" + hour);
System.out.println("分钟：" + minute);
System.out.println("秒：" + second);
System.out.println("星期：" + day);
```

使用 Date 类中对应的 get 方法，可以获得 Date 类对象中相关的信息，需要注意的是使用 getYear 获得是 Date 对象中年份减去 1900 以后的值，所以需要显示对应的年份则需要在返回值的基础上加上 1900，月份类似。在 Date 类中还提供了 getDay 方法，用于获得 Date 对象代表的时间是星期几，Date 类规定周日是 0，周一是 1，周二是 2，后续的依次类推。

4、Date 对象和相对时间之间的互转

```
Date d3 = new Date(2009-1900,3-1,10);
long time = 1290876532190L;
//将 Date 类的对象转换为相对时间
long t = d3.getTime();
System.out.println(t);
//将相对时间转换为 Date 类的对象
Date d4 = new Date(time);
System.out.println(d4);
```

使用 Date 对象中的 getTime 方法，可以将 Date 类的对象转换为相对时间，使用 Date 类的构造方法，可以将相对时间转换为 Date 类的对象。经过转换以后，既方便了时间的计算，也使时间显示比较直观了。

9.6.1.2 Calendar 类

从 JDK1.1 版本开始，在处理日期和时间时，系统推荐使用 Calendar 类进行实现。在设计上，Calendar 类的功能要比 Date 类强大很多，而且在实现方式上也比 Date 类要复杂一些，下面就介绍一下 Calendar 类的使用。

Calendar 类是一个抽象类，在实际使用时实现特定的子类的对象，创建对象的过程对程序员来说是透明的，只需要使用 getInstance 方法创建即可。

1、使用 Calendar 类代表当前时间

```
Calendar c = Calendar.getInstance();
```

由于 Calendar 类是抽象类，且 Calendar 类的构造方法是 protected 的，所以无法使用 Calendar 类的构造方法来创建对象，API 中提供了 getInstance 方法用来创建对象。

使用该方法获得的 Calendar 对象就代表当前的系统时间，由于 Calendar 类 toString 实现的没有 Date 类那么直观，所以直接输出 Calendar 类的对象意义不大。

2、使用 Calendar 类代表指定的时间

```
Calendar c1 = Calendar.getInstance();
c1.set(2009, 3 - 1, 9);
```

使用 Calendar 类代表特定的时间，需要首先创建一个 Calendar 的对象，然后再设定该对象中的年月日参数来完成。

set 方法的声明为：

```
public final void set(int year,int month,int date)
```

以上示例代码设置的时间为 2009 年 3 月 9 日，其参数的结构和 Date 类不一样。Calendar 类中年份的数值直接书写，月份的值为实际的月份值减 1，日期的值就是实际的日期值。

如果只设定某个字段，例如日期的值，则可以使用如下 set 方法：

```
public void set(int field,int value)
```

在该方法中，参数 field 代表要设置的字段的类型，常见类型如下：

Calendar.YEAR——年份
Calendar.MONTH——月份
Calendar.DATE——日期
Calendar.DAY_OF_MONTH——日期，和上面的字段完全相同
Calendar.HOUR——12 小时制的小时数
Calendar.HOUR_OF_DAY——24 小时制的小时数
Calendar.MINUTE——分钟
Calendar.SECOND——秒
Calendar.DAY_OF_WEEK——星期几

后续的参数 value 代表，设置成的值。例如：

```
c1.set(Calendar.DATE,10);
```

该代码的作用是将 c1 对象代表的时间中日期设置为 10 号，其它所有的数值会被重新计算，例如星期几以及对应的相对时间数值等。

3、获得 Calendar 类中的信息

```
Calendar c2 = Calendar.getInstance();
```

```
//年份
int year = c2.get(Calendar.YEAR);
//月份
int month = c2.get(Calendar.MONTH) + 1;
//日期
int date = c2.get(Calendar.DATE);
//小时
int hour = c2.get(Calendar.HOUR_OF_DAY);
//分钟
int minute = c2.get(Calendar.MINUTE);
//秒
int second = c2.get(Calendar.SECOND);
//星期几
int day = c2.get(Calendar.DAY_OF_WEEK);
System.out.println("年份：" + year);
System.out.println("月份：" + month);
System.out.println("日期：" + date);
System.out.println("小时：" + hour);
System.out.println("分钟：" + minute);
System.out.println("秒：" + second);
System.out.println("星期：" + day);
```

使用 Calendar 类中的 get 方法可以获得 Calendar 对象中对应的信息，get 方法的声明如下：

```
public int get(int field)
```

其中参数field代表需要获得的字段的值,字段说明和上面的set方法保持一致。需要说明的是，获得的月份为实际的月份值减 1，获得的星期的值和 Date 类不一样。在 Calendar 类中，周日是 1，周一是 2，周二是 3，依次类推。

4、其它方法说明

其实 Calendar 类中还提供了很多其它有用的方法，下面简单的介绍几个常见方法的使用。

a、add 方法

```
public abstract void add(int field,int amount)
```

该方法的作用是在 Calendar 对象中的某个字段上增加或减少一定的数值，增加是 amount 的值为正，减少时 amount 的值为负。

例如在计算一下当前时间 100 天以后的日期，代码如下：

```
Calendar c3 = Calendar.getInstance();
c3.add(Calendar.DATE, 100);
int year1 = c3.get(Calendar.YEAR);
//月份
int month1 = c3.get(Calendar.MONTH) + 1;
//日期
int date1 = c3.get(Calendar.DATE);
```

```
System.out.println(year1 + "年" + month1 + "月" +
date1 + "日");
```

这里 add 方法是指在 c3 对象的 Calendar.DATE，也就是日期字段上增加 100，类内部会重新计算该日期对象中其它各字段的值，从而获得 100 天以后的日期，例如程序的输出结果可能为：

2009 年 6 月 17 日

b、after 方法

```
public boolean after(Object when)
```

该方法的作用是判断当前日期对象是否在 when 对象的后面，如果在 when 对象的后面则返回 true，否则返回 false。例如：

```
Calendar c4 = Calendar.getInstance();
c4.set(2009, 10 - 1, 10);
Calendar c5 = Calendar.getInstance();
c5.set(2010, 10 - 1, 10);
boolean b = c5.after(c4);
System.out.println(b);
```

在该示例代码中对象 c4 代表的时间是 2009 年 10 月 10 号，对象 c5 代表的时间是 2010 年 10 月 10 号，则对象 c5 代表的日期在 c4 代表的日期之后，所以 after 方法的返回值是 true。

另外一个类似的方法是 before，该方法是判断当前日期对象是否位于另外一个日期对象之前。

c、getTime 方法

```
public final Date getTime()
```

该方法的作用是将 Calendar 类型的对象转换为对应的 Date 类对象，两者代表相同的时间点。

类似的方法是 setTime，该方法的作用是将 Date 对象转换为对应的 Calendar 对象，该方法的声明如下：

```
public final void setTime(Date date)
```

转换的示例代码如下：

```
Date d = new Date();
Calendar c6 = Calendar.getInstance();
//Calendar 类型的对象转换为 Date 对象
Date d1 = c6.getTime();
//Date 类型的对象转换为 Calendar 对象
Calendar c7 = Calendar.getInstance();
c7.setTime(d);
```

5、Calendar 对象和相对时间之间的互转

```
Calendar c8 = Calendar.getInstance();
long t = 1252785271098L;
//将 Calendar 对象转换为相对时间
long t1 = c8.getTimeInMillis();
//将相对时间转换为 Calendar 对象
```

```
Calendar c9 = Calendar.getInstance();
c9.setTimeInMillis(t1);
```

在转换时，使用Calendar类中的getTimeInMillis方法可以将Calendar对象转换为相对时间。在将相对时间转换为Calendar对象时，首先创建一个Calendar对象，然后再使用Calendar类的setTimeInMillis方法设置时间即可。

9.6.1.3 应用示例

下面以两个简单的示例介绍时间和日期处理的基本使用。

1、计算两个日期之间相差的天数

例如计算2010年4月1号和2009年3月11号之间相差的天数，则可以使用时间和日期处理进行计算。

该程序实现的原理为：首先代表两个特定的时间点，这里使用Calendar的对象进行代表，然后将两个时间点转换为对应的相对时间，求两个时间点相对时间的差值，然后除以1天的毫秒数(24小时X60分钟X60秒X1000毫秒)即可获得对应的天数。实现该示例的完整代码如下：

```
import java.util.*;
/**
 * 计算两个日期之间相差的天数
 */

public class DateExample1 {
        public static void main(String[] args) {
                //设置两个日期
                //日期：2009年3月11号
                Calendar c1 = Calendar.getInstance();
                c1.set(2009, 3 - 1, 11);
                //日期：2010年4月1号
                Calendar c2 = Calendar.getInstance();
                c2.set(2010, 4 - 1, 1);
                //转换为相对时间
                long t1 = c1.getTimeInMillis();
                long t2 = c2.getTimeInMillis();
                //计算天数
                long days = (t2 - t1)/(24 * 60 * 60 * 1000);
                System.out.println(days);
        }
}
```

2、输出当前月的月历

该示例的功能是输出当前系统时间所在月的日历，例如当前系统时间是2009年3月10日，则输出2009年3月的日历。

该程序实现的原理为：首先获得该月 1 号是星期几，然后获得该月的天数，最后使用流程控制实现按照日历的格式进行输出即可。即如果 1 号是星期一，则打印一个单位的空格，如果 1 号是星期二，则打印两个单位的空格，依次类推。打印完星期六的日期以后，进行换行。实现该示例的完整代码如下：

```
import java.util.*;
/**
 * 输出当前月的日历
 */

public class DateExample2{
        public static void main(String[] args){
                //获得当前时间
                Calendar c = Calendar.getInstance();
                //设置代表的日期为 1 号
                c.set(Calendar.DATE,1);
                //获得 1 号是星期几
                int start = c.get(Calendar.DAY_OF_WEEK);
                //获得当前月的最大日期数
                int maxDay = c.getActualMaximum(Calendar.DATE);
                //输出标题
                  System.out.println("星期日 星期一 星期二 星期三
星期四  星期五   星期六");
                //输出开始的空格
                for(int i = 1;i < start;i++){
                        System.out.print("      ");
                }
                //输出该月中的所有日期
                for(int i = 1;i <= maxDay;i++){
                        //输出日期数字
                        System.out.print(" " + i);
                        //输出分隔空格
                        System.out.print("     ");
                        if(i < 10){
                                System.out.print(' ');
                        }
                        //判断是否换行
                        if((start + i - 1) % 7 == 0){
                                System.out.println();
                        }
                }
                //换行
                System.out.println();
```

```
        }
}
```

关于时间和日期的处理就介绍这么多，更多的实现方法还需要根据具体问题进行对应的实现。

Java 编程那些事儿 79——Random 随机处理

陈跃峰
出自：http://blog.csdn.net/mailbomb

9.6.2 Random 类

在实际的项目开发过程中，经常需要产生一些随机数值，例如网站登录中的校验数字等，或者需要以一定的几率实现某种效果，例如游戏程序中的物品掉落等。

在 Java API 中，在 java.util 包中专门提供了一个和随机处理有关的类，这个类就是 Random 类。随机数字的生成相关的方法都包含在该类的内部。

Random 类中实现的随机算法是伪随机，也就是有规则的随机。在进行随机时，随机算法的起源数字称为种子数(seed)，在种子数的基础上进行一定的变换，从而产生需要的随机数字。

相同种子数的 Random 对象，相同次数生成的随机数字是完全相同的。也就是说，两个种子数相同的 Random 对象，第一次生成的随机数字完全相同，第二次生成的随机数字也完全相同。这点在生成多个随机数字时需要特别注意。

下面介绍一下 Random 类的使用，以及如何生成指定区间的随机数组以及实现程序中要求的几率。

1、Random 对象的生成

Random 类包含两个构造方法，下面依次进行介绍：

a、public Random()

该构造方法使用一个和当前系统时间对应的相对时间有关的数字作为种子数，然后使用这个种子数构造 Random 对象。

b、public Random(long seed)

该构造方法可以通过制定一个种子数进行创建。

示例代码：

```
Random r = new Random();
Random r1 = new Random(10);
```

再次强调：种子数只是随机算法的起源数字，和生成的随机数字的区间无关。

2、Random 类中的常用方法

Random 类中的方法比较简单，每个方法的功能也很容易理解。需要说明的是，Random 类中各方法生成的随机数字都是均匀分布的，也就是说区间内部的数字生成的几率是均等的。下面对这些方法做一下基本的介绍：

a、public boolean nextBoolean()

该方法的作用是生成一个随机的 boolean 值，生成 true 和 false 的值几率相等，也就是都是 50%的几率。

b、public double nextDouble()

该方法的作用是生成一个随机的 double 值，数值介于[0,1.0)之间，这里中括号代表包含区间端点，小括号代表不包含区间端点，也就是 0 到 1 之间的随机小数，包含 0 而不包含 1.0。

c、public int nextInt()

该方法的作用是生成一个随机的 int 值，该值介于 int 的区间，也就是-231 到 231-1 之间。

如果需要生成指定区间的 int 值，则需要进行一定的数学变换，具体可以参看下面的使用示例中的代码。

d、public int nextInt(int n)

该方法的作用是生成一个随机的 int 值，该值介于[0,n)的区间，也就是 0 到 n 之间的随机 int 值，包含 0 而不包含 n。

如果想生成指定区间的 int 值，也需要进行一定的数学变换，具体可以参看下面的使用示例中的代码。

e、public void setSeed(long seed)

该方法的作用是重新设置 Random 对象中的种子数。设置完种子数以后的 Random 对象和相同种子数使用 new 关键字创建出的 Random 对象相同。

3、Random 类使用示例

使用 Random 类，一般是生成指定区间的随机数字，下面就一一介绍如何生成对应区间的随机数字。以下生成随机数的代码均使用以下 Random 对象 r 进行生成：

```
Random r = new Random();
```

a、生成[0,1.0)区间的小数

```
double d1 = r.nextDouble();
```

直接使用 nextDouble 方法获得。

b、生成[0,5.0)区间的小数

```
double d2 = r.nextDouble() * 5;
```

因为 nextDouble 方法生成的数字区间是[0,1.0)，将该区间扩大 5 倍即是要求的区间。

同理，生成[0,d)区间的随机小数，d 为任意正的小数，则只需要将 nextDouble 方法的返回值乘以 d 即可。

c、生成[1,2.5)区间的小数

```
double d3 = r.nextDouble() * 1.5 + 1;
```

生成[1,2.5)区间的随机小数，则只需要首先生成[0,1.5)区间的随机数字，然后

将生成的随机数区间加 1 即可。
同理，生成任意非从 0 开始的小数区间[d1,d2)范围的随机数字(其中 d1 不等于 0)，则只需要首先生成[0,d2-d1)区间的随机数字，然后将生成的随机数字区间加上 d1 即可。

d、生成任意整数

int n1 = r.nextInt();

直接使用 nextInt 方法即可。

e、生成[0,10)区间的整数

int n2 = r.nextInt(10);
n2 = Math.abs(r.nextInt() %10);

以上两行代码均可生成[0,10)区间的整数。

第一种实现使用 Random 类中的 nextInt(int n)方法直接实现。

第二种实现中，首先调用 nextInt()方法生成一个任意的 int 数字，该数字和 10 取余以后生成的数字区间为(-10,10)，因为按照数学上的规定余数的绝对值小于除数，然后再对该区间求绝对值，则得到的区间就是[0,10)了。

同理，生成任意[0,n)区间的随机整数，都可以使用如下代码：

```
int n2 = r.nextInt(n);
n2 = Math.abs(r.nextInt() % n);
```

f、生成[0,10]区间的整数

int n3 = r.nextInt(11);
n3 = Math.abs(r.nextInt() %11);

相对于整数区间，[0,10]区间和[0,11)区间等价，所以即生成[0,11)区间的整数。

g、生成[-3,15)区间的整数

int n4 = r.nextInt(18) - 3;
n4 = Math.abs(r.nextInt() %18) - 3;

生成非从 0 开始区间的随机整数，可以参看上面非从 0 开始的小数区间实现原理的说明。

h、几率实现

按照一定的几率实现程序逻辑也是随机处理可以解决的一个问题。下面以一个简单的示例演示如何使用随机数字实现几率的逻辑。

在前面的方法介绍中，nextInt(int n)方法中生成的数字是均匀的，也就是说该区间内部的每个数字生成的几率是相同的。那么如果生成一个[0,100)区间的随机整数，则每个数字生成的几率应该是相同的，而且由于该区间中总计有 100 个整数，所以每个数字的几率都是 1%。按照这个理论，可以实现程序中的几率问题。

示例：随机生成一个整数，该整数以 55%的几率生成 1，以 40%的几率生成 2，以 5%的几率生成 3。实现的代码如下：

```
int n5 = r.nextInt(100);
    int m;  //结果数字
    if(n5 < 55){ //55 个数字的区间，55%的几率
```

```
            m = 1;
        }else if(n5 < 95){//[55,95)，40 个数字的区间，40%的几率
            m = 2;
        }else{
            m = 3;
        }
```

因为每个数字的几率都是 1%，则任意 55 个数字的区间的几率就是 55%，为了代码方便书写，这里使用[0,55)区间的所有整数，后续的原理一样。

当然，这里的代码可以简化，因为几率都是 5%的倍数，所以只要以 5%为基础来控制几率即可，下面是简化的代码实现：

```
int n6 = r.nextInt(20);
    int m1;
    if(n6 < 11){
            m1 = 1;
    }else if(n6 < 19){
            m1= 2;
    }else{
            m1 = 3;
    }
```

在程序内部，几率的逻辑就可以按照上面的说明进行实现。

4、其它问题

a、相同种子数 Random 对象问题

前面介绍过，相同种子数的 Random 对象，相同次数生成的随机数字是完全相同的，下面是测试的代码：

```
Random r1 = new Random(10);
Random r2 = new Random(10);
for(int i = 0;i < 2;i++){
        System.out.println(r1.nextInt());
        System.out.println(r2.nextInt());
}
```

在该代码中，对象 r1 和 r2 使用的种子数都是 10，则这两个对象相同次数生成的随机数是完全相同的。

如果想避免出现随机数字相同的情况，则需要注意，无论项目中需要生成多少个随机数字，都只使用一个 Random 对象即可。

b、关于 Math 类中的 random 方法

其实在 Math 类中也有一个 random 方法，该 random 方法的工作是生成一个

[0,1.0)区间的随机小数。

通过阅读 Math 类的源代码可以发现，Math 类中的 random 方法就是直接调用 Random 类中的 nextDouble 方法实现的。

只是 random 方法的调用比较简单，所以很多程序员都习惯使用 Math 类的 random 方法来生成随机数字。

Java 编程那些事儿 80——集合框架简述

陈跃峰
出自：http://blog.csdn.net/mailbomb

9.6.3 集合框架简述

在 JDK API 中专门设计了一组类，这组类的功能就是实现各种各样方式的数据存储，这样一组专门用来存储其它对象的类，一般被称为对象容器类，简称容器类，这组类和接口的设计结构也被统称为集合框架(Collection Framework)。

这组类和接口都包含在 java.util 包中。

为了使整个集合框架中的类便于使用，在设计集合框架时大量的使用接口，实际实现的功能类实现对应的接口，这样可以保证各个集合类的使用方式保持统一。

在集合框架中，提供的存储方式共有两种：

1、按照索引值操作数据

在这种存储方式中，为每个存储的数据设定一个索引值，存储在容器中的第一个元素索引值是 0，第二个索引值是 1，依次类推。在操作数据时按照索引值操作对应的数据，实现这种方式的集合类都实现 java.util.Collection 接口。

2、按照名称操作数据

在这种存储方式中，为每个存储的数据设定一个名称(任意非 null 的对象都可以作为名称)，以后按照该名称操作该数据，要求名称不能重复，每个名称对应唯一的一个值。这种存储数据的方式也称作名称-数值对，也就是名值对存储。实现这种方式的几个类都实现 java.util.Map 接口。

这里“按照索引值操作数据”的存储方式，又按照容器内部是否能够存储重复的元素，划分成两类：

1、允许存储重复元素。

这种存储方式中，所有的类都实现了 java.util.List 接口。

2、不允许存储重复元素。

这种存储方式中，所有的类都实现了 java.util.Set 接口。

这样，集合框架中的类就分成了三大类：

1、List 系列

该系列中的类按照索引值来操作数据，允许存放重复的元素。

2、Set 系列

该系列中的类按照索引值来操作数据，不允许存放重复的元素。

3、Map 系列

该系列中的类按照名称来操作数据，名称不允许重复，值可以重复，一个名称对应一个唯一的值。

而在数据结构中，实现数据的存储又可以使用不同的数据结构类型进行存储，例如数组、链表、栈、队列和树等，则以上三类集合框架可以使用不同的数据结构类进行实现，使用每种数据结构则具备该中数据结构的特点。例如使用数组则访问速度快，使用链表则便于动态插入和删除等，这样就造成了集合框架的复杂性。

另外，在将对象存储到集合类中，为了加快存储的速度，要求被存储对象的类中必须覆盖 equals 方法和 hashCode 方法。

对于这些集合类，下面按照以上三个系列的顺序一一进行说明。

9.6.3.1 List 系列

List 系列的类均实现 List 接口，大部分的类都以 List 作为类名的后缀，也有部分该体系中的类命名比较特殊。

该系列中的类，比较常见的有 ArrayList 和 LinkedList 两个。其中 ArrayList 是以数组为基础实现的 List，而 LinkedList 则是以链表为基础实现的 List，ArrayList 拥有数组的优点，而 LinkedList 拥有链表的优点。

由于该体系中的类均实现 List 接口，所以在这些类的内部，相同的功能方法声明是保持一致的，下面进行一一介绍：

a、add 方法

boolean add(Object o)

该方法的作用是追加对象 o 到已有容器的末尾。

另外一个 add 方法：

void add(int index, Object element)

该方法的作用是将对象 element 插入到容器中索引值为 index 的位置，原来位于该位置的对象以及后续的内容将依次向后移动。

b、addAll 方法

boolean addAll(Collection c)

该方法的作用是将容器对象 c 中的每个元素依次添加到当前容器的末尾。

另外一个 addAll 方法：

boolean addAll(int index, Collection c)

该方法的作用是将容器对象 c 中的第一个元素插入到当前容器中索引值为 index 的位置，第二个元素插入到当前容器中索引值为 index+1 的位置，依次类推。而当前容器中原来位于 index 以及 index 索引值以后的元素则依次向后移动。

c、get 方法

Object get(int index)

该方法的作用是返回当前容器对象中索引值为 index 的元素的内容。

d、indexOf 方法

int indexOf(Object o)

该方法的作用是查找当前容器中是否存在对象 o，如果存在则返回该对象第一次出现位置的索引值，如果不存在则返回-1。

另外一个方法 lastIndexOf 则是从末尾向前查找，返回从末尾向前第一次出现位置的索引值，如果不存在则返回-1。

e、remove 方法

Object remove(int index)

该方法的作用是删除索引值为 index 的对象的内容，如果删除成功则返回被删除对象的内容。

另外一个 remove 方法：

boolean remove(Object o)

该方法的作用是删除对象内容为 o 的元素，如果相同的对象有多个，则只删除索引值小的对象。如果删除成功则返回 true，否则返回 false。

无论使用哪一个 remove 方法，类内部都自动移动将被删除位置后续的所有元素向前移动，保证索引值的连续性。

f、set 方法

Object set(int index, Object element)

该方法的作用是修改索引值为 index 的内容，将原来的内容修改成对象 element 的内容。

g、size 方法

int size()

该方法的作用是返回当前容器中已经存储的有效元素的个数。

h、toArray 方法

Object[] toArray()

该方法的作用是将当前容器中的元素按照顺序转换成一个 Object 数组。

下面是一个简单的以 ArrayList 类为基础实现的 List 系列中类基本使用的示例，代码如下：

```
import java.util.*;
/**
 * 以 ArrayList 类为基础演示 List 系列类的基本使用
 */

public class ArrayListUse {
        public static void main(String[] args) {
                //容器对象的初始化
                List list = new ArrayList();
                //添加数据
                list.add("1");
                list.add("2");
```

```
                    list.add("3");
                    list.add("1");
                    list.add("1");
                    //插入数据
                    list.add(1,"12");
                    //修改数据
                    list.set(2, "a");
                    //删除数据
                    list.remove("1");
                    //遍历
                    int size = list.size();  //获得有效个数
                    //循环有效索引值
                    for(int i = 0;i < size;i++){
                            System.out.println((String)list.get(i));
                    }
          }
}
```

该程序的运行结果为：

```
12
a
3
1
1
```

在 List 系列中，还包含了 Stack(栈)类和 Vector(向量)类，Stack 类除了实现 List 系列的功能以外，还实现了栈的结构，主要实现了出栈的 pop 方法和入栈的 push 方法。

而 Vector 类由于需要兼容老版本 JDK 中缘故，所以在实现的方法中需要提供老版本 Vector 类中对应的方法，这样导致 Vector 类中相同或类似的功能方法一般是成对出现的。

9.6.3.2 Set 系列

Set 系列中的类都实现了 Set 接口，该系列中的类均以 Set 作为类名的后缀。该系列中的容器类，不允许存储重复的元素。也就是当容器中已经存储一个相同的元素时，无法实现添加一个完全相同的元素，也无法将已有的元素修改成和其它元素相同。

Set 系列中类的这些特点，使得在某些特殊场合的使用比较适合。该系列中常见的类有：

1、CopyOnWriteArraySet

以数组为基础实现的 Set 类。

2、HashSet

以哈希表为基础实现的 Set 类。

3、LinkedHashSet

以链表为基础实现的 Set 类。

4、TreeSet

以树为基础实现的 Set 类。

以不同的数据结构类型实现的 Set 类，拥有不同数据结构带来的特性，在实际使用时，根据逻辑的需要选择合适的 Set 类进行使用。

Set 系列中的类的方法和 List 系列中的类的方法要比 List 系列中少很多，例如不支持插入和修改，而且对于 Set 系列中元素的遍历也需要转换为专门的 Iterator(迭代器)对象才可以进行遍历，遍历时顺序和 Set 中存储的顺序会有所不同。

下面是以 HashSet 类为基础实现的示例代码，代码如下：

```
import java.util.*;
/**
 * 以 HashSet 为基础演示 Set 系列类的基本使用
 */
public class HashSetUse {
        public static void main(String[] args) {
                //容器对象的初始化
                Set set = new HashSet();
                //添加元素
                set.add("1");
                set.add("2");
                set.add("3");
                set.add("1");
                set.add("1");
                //删除数据
                //set.remove("1");
                //遍历
                Iterator iterator = set.iterator();
                while(iterator.hasNext()){
                    System.out.println((String)iterator.next());
                }
        }
}
```

该程序的运行结果为：

```
3
2
1
```

9.6.3.3 Map 系列

Map 系列中的类都实现了 Map 接口，该系列中的部分类以 Map 作为类名的后缀。该系列容器类存储元素的方式和以上两种完全不同。

Map 提供了一种使用“名称：值”这样的名称和数值对存储数据的方法，在该存储方式中，名称不可以重复，而不同的名称中可以存储相同的数值。具体这种存储的格式将在示例代码中进行实现。

在这种存储结构中，任何不为 null 的对象都可以作为一个名称(key)来作为存储的值(value)的标识，使用这种形式更利于存储比较零散的数据，也方便数据的查找和获得。Map 类中存储的数据没有索引值，系统会以一定的形式索引存储的名称，从而提高读取数据时的速度。

该系列中常见的类有：

1、HashMap

以 Hash(哈希表)为基础实现的 Map 类。

2、LinkedHashMap

以链表和 Hash(哈希表)为基础实现的 Map 类。

3、TreeMap

以树为基础实现的 Map 类。

和上面的结构类似，以不同的数据结构实现的 Map 类，拥有不同数据结构的特点，在实际的项目中使用时，根据需要选择合适的即可。

该系列的类中常见的方法如下：

a、get 方法

Object get(Object key)

该方法的作用是获得当前容器中名称为 key 的结构对应的值。

b、keySet 方法

Set keySet()

该方法的作用是返回当前容器中所有的名称，将所有的名称以 Set 的形式返回。使用这个方法可以实现对于 Map 中所有元素的遍历。

c、put 方法

Object put(Object key, Object value)

该方法的作用是将值 value 以名称 key 的形式存储到容器中。

d、putAll 方法

void putAll(Map t)

该方法的作用是将 Map 对象 t 中的所有数据按照原来的格式存储到当前容器类中，相当于合并两个 Map 容器对象。

e、remove 方法

Object remove(Object key)

该方法的作用是删除容器中名称为 key 的值。

f、size 方法

int size()

该方法的作用是返回当前日期中存储的名称：值数据的组数。

g、values 方法

Collection values()

该方法的作用是返回当前容器所有的值组成的集合，以 Collection 对象的形式

返回。

下面是一个简单的示例，在该示例中演示 Map 系列类的基本使用，代码如下：

```
import java.util.*;
/**
 * 以 HashMap 为基础演示 Map 系列中类的使用
 */
public class HashMapUse {
	public static void main(String[] args) {
		//容器对象的初始化
		Map map = new HashMap();
		//存储数据
		map.put("苹果", "2.5");
		map.put("桔子", "2.5");
		map.put("香蕉", "3");
		map.put("菠萝", "2");
		//删除元素
		map.remove("桔子");
		//修改元素的值
		map.put("菠萝", "5");
		//获得元素个数
		int size = map.size();
		System.out.println("个数是：" + size);
		//遍历 Map
		Set set = map.keySet();
		Iterator iterator = set.iterator();
		while(iterator.hasNext()){
			//获得名称
			String name = (String)iterator.next();
			//获得数值
			String value = (String)map.get(name);
			//显示到控制台

			System.out.println(name + ":" + value);
		}
	}
}
```

该程序的运行结果为：

```
个数是：3
香蕉:3
菠萝:5
苹果:2.5
```

9.6.3.4 使用示例

如前所述，集合框架中的类只是提供了一种数据存储的方式，在实际使用时，可以根据逻辑的需要选择合适的集合类进行使用。

下面以一个字符串计算的示例演示集合类的实际使用。

该程序的功能为计算一个数字字符串，例如”1+2*31-5”、”12*30/34-450”等，的计算结果，在该示例中支持四则运算，但是不支持括号。本示例中计算的字符串要求合法。

该程序实现的原理是：首先按照运算符作为间隔，将字符串差分为数字字符串和运算符字符串的序列，由于分拆出的字符串数量不固定，所以存储到List系列的Vector容器中，然后按照运算符的优先级进行计算。

该程序的代码如下：

```
import java.util.*;
/**
 * 计算字符串的值
 */
public class CalcStr {
        public static void main(String[] args) {
                String s = "1+20*3/5";
                double d = calc(s);
                System.out.println(d);
        }
        /**
         * 计算字符串的值
         * @param s 需要计算的字符串
         * @return 计算结果
         */
        public static double calc(String s){
                //拆分字符串
                Vector v = split(s);
                //print(v); //测试代码
                //计算字符串
                double d = calcVector(v);
                return d;
        }

        /**
         * 将字符串拆分为数字和运算符。
         * 例如："1+23*4"则拆分为："1"、"+"、"23"、"*"和"4"
         * @param s 需要拆分的字符串
         * @return 拆分以后的结果
         */
        private static Vector split(String s){
```

```
        Vector v = new Vector();
        String content = "";
        int len = s.length(); //字符串长度
        char c;
        for(int i = 0;i < len;i++){
                c = s.charAt(i);
                //判断是否为运算符
                if(c == '+' ||
                        c == '-' ||
                         c == '*' ||
                          c == '/'){
                        //存储数字
                        v.add(content);
                        //存储运算符
                        v.add("" + c);
                        //清除已有字符串
                        content = "";
                }else{
                        content += c; //连接字符串
                }
        }
        v.add(content); //添加最后一个数字
        return v;
}

/**
 * 测试代码，输出拆分以后的结果
 * @param v 需要打印的Vector对象
 */
private static void print(Vector v){
        int size = v.size();
        for(int i = 0;i < size;i++){
                System.out.println((String)v.get(i));
        }
}

/**
 * 计算Vector中的数据
 * @param v 存储拆分后字符串的Vector
 * @return 计算结果
 */
private static double calcVector(Vector v){
```

```
                int index1;
                int index2;
                //计算乘除
                while(true){
                        index1 = v.indexOf("*"); //乘号索引值
                        index2 = v.indexOf("/"); //除号索引值
                        //无乘除符号
                        if(index1 == - 1 && index2 == -1){
                                break;  //结束循环
                        }
                        //如果有乘号
                        if(index1 != -1){
                              //没有除号或乘号在前
                             if(index2 == -1 || index1 < index2){
                                                        String s1 =
(String)v.get(index1 - 1); //第一个数字
                                                      String opr =
(String)v.get(index1); //运算符
                                                        String s2 =
(String)v.get(index1 + 1); //第二个数字
                                                //计算
                                                      String answer =
calc(s1,s2,opr);
                                                //计算以后的处理
                                                handle(answer,index1 -
1,v);
                                }
                        }
                        //有除号
                        if(index2 != -1){
                                //没有乘号或除号在前
                                    if(index1 == -1 || index2 <
index1){
                                                        String s1 =
(String)v.get(index2 - 1); //第一个数字
                                                      String opr =
(String)v.get(index2); //运算符
                                                        String s2 =
(String)v.get(index2 + 1); //第二个数字
                                                //计算
                                                      String answer =
calc(s1,s2,opr);
```

```
                                        //计算以后的处理
                                        handle(answer,index2 -
1,v);
                                }
                        }
                }
                //计算加
                int index3 = v.indexOf("+");
                while(index3 != -1){ //有加号
                        String s1 = (String)v.get(index3 - 1); //
第一个数字
                         String opr = (String)v.get(index3); //运
算符
                        String s2 = (String)v.get(index3 + 1); //
第二个数字
                        //计算
                        String answer = calc(s1,s2,opr);
                        //计算以后的处理
                        handle(answer,index3 - 1,v);
                        //获得下一个加号的位置
                        index3 = v.indexOf("+");
                }

                //计算减
                index3 = v.indexOf("-");
                while(index3 != -1){ //有加号
                        String s1 = (String)v.get(index3 - 1); //
第一个数字
                       String opr = (String)v.get(index3); //运算
符
                        String s2 = (String)v.get(index3 + 1); //
第二个数字
                        //计算
                        String answer = calc(s1,s2,opr);
                        //计算以后的处理
                        handle(answer,index3 - 1,v);
                        //获得下一个减号的位置
                        index3 = v.indexOf("-");
                }
                //反馈结果
                String data = (String)v.get(0);
                return Double.parseDouble(data);
```

```
        }

        /**
         * 计算两个字符串类型的值运算结果
         * @param number1 数字 1
         * @param number2 数字 2
         * @param opr 运算符
         * @return 运算结果
         */
              private static String calc(String number1,String
number2,String opr){
                  //将字符串转换为数字
                  double d1 = Double.parseDouble(number1);
                  double d2 = Double.parseDouble(number2);
                  //判断运算符
                  if(opr.equals("+")){
                           return "" + (d1 + d2);
                  }
                  if(opr.equals("-")){
                           return "" + (d1 - d2);
                  }
                  if(opr.equals("*")){
                           return "" + (d1 * d2);
                  }
                  if(opr.equals("/")){
                           return "" + (d1 / d2);
                  }
                  return "0";  //运算符错误时返回 0
        }

        /**
         * 计算以后的处理
         * @param answer 计算结果
         * @param index 参与计算的三个字符串中第一个字符串的起始位置
         * @param v 存储字符串的容器
         */
       private static void handle(String answer,int index,Vector v){
                  //删除计算过的字符串
                  for(int i = 0;i < 3;i++){
                           v.remove(index);
                  }
                  //将计算结果插入到 index 位置
```

```
                v.insertElementAt(answer, index);
        }
}
```

该程序的运行结果为：

```
13.0
```

9.7 总结

在本章中，主要介绍了 java.lang 包和 java.util 包中比较常见的类的使用，熟悉了 JDK API 的基本使用，掌握了文档的查阅以及方法调用等一些基本的技能，为后续章节的学习打下坚实的基础。

Java 编程那些事儿 81——异常处理概述

陈跃峰
出自：http://blog.csdn.net/mailbomb

第十章 异常处理

在实际的项目中，程序执行时经常会出现一些意外的情况，这些意外的情况会导致程序出错或者崩溃，从而影响程序的正常执行，如果不能很好的处理这些意外情况，将使项目的稳定性不强。

对于这些程序执行时出现的意外情况，在 Java 语言中被称作异常(Exception)，出现异常时相关的处理则称之为异常处理。

异常处理是 Java 语言中的一种机制，恰当使用异常处理可以使整个项目更加稳定，也使项目中正常的逻辑代码和错误处理的代码实现分离，便于代码的阅读和维护。

本章就将深入介绍 Java 语言中的异常处理机制。

10.1 概述

在实际的项目中，并不是所有的情况都是那样理想，例如不可能有使用不尽的内存，也不可能有熟练的软件使用人员等，这样就会导致项目在执行时会出现各种各样不可预料的情况，这样的情况如果处理不好，则会导致程序崩溃或者中止执行，例如 Windows 操作系统的蓝屏。

在一个完善的项目，这些不可预料的情况必须得到正确的处理，才能使整个项目具有比较强的稳定性，这才是符合要求的项目。为了实现这种处理结构，在 Java 语言中新增了一套完善的语法，通过处理这些情况来增强项目的稳定性，这套语法就是异常处理的语法。

这些程序执行时出现的不可预料的情况，也就是执行时的意外情况，在 Java 语言的语法中被称作异常。

其实简单的进行异常处理很多程序设计语言都是可以实现的，就是根据情况判断，不同的情况作出不同的处理。Java 语言的异常处理机制最大的优势之一就是可以将异常情况在方法调用中进行传递，通过传递可以将异常情况传递到合适的位置再进行处理。这种机制就类似于现实中你发现了火灾，你一个人是无法扑灭大火的，那么你就将这种异常情况传递给 119，然后 119 再将这个情况传递给附近的消防队，消防队及时赶到进行灭火。使用这种处理机制，使得 Java 语言的异常处理更加灵活。

另外，使用异常处理机制，可以在源代码级别将正常执行的逻辑代码，和进行异常情况处理的代码相分离，更加便于代码的阅读。

异常处理机制是 Java 语法的一个特色功能，通过恰当的使用该机制，可以使得 Java 语言编写的项目更加稳定。

当然，异常处理机制也存在一些弊端，例如使用异常处理将降低程序的执行效率，增加语法的复杂度等。

下面简单的看一个执行时将出现异常的代码：

```
/**
 * 异常出现示例
 */
public class ExceptionDemo {
        public static void main(String[] args) {
                String s = null;
                int len = s.length();
        }
}
```

在运行该程序时，在控制台的输出结果如下：

```
Exception in thread "main" java.lang.NullPointerException at ExceptionDemo.main(ExceptionDemo.java:7)
```

从这个程序执行时的输出可以看出，提示在 main 线程(thread)中出现了异常，异常的类型为 java.lang.NullPointerException，异常出现在 ExceptionDemo 的 main 方法中，出现异常的代码在 ExceptionDemo.java 代码中

的第 7 行。在 JDK 文档查阅 java.lang 包，在异常部分可以查阅到关于 NullPointerException 的详细说明。这里出现该异常是因为对象 s 没有创建造成的。将程序中的 String s = null;代码替换为 String s = “abc”;即可避免出现该异常。

在程序执行中，会出现各种各样的异常情况，这些异常情况在 Java 体系将代表常见的异常情况，这就是下面的异常类体系中需要介绍的知识。

10.2 异常类体系

在 Java 语言以前，代表各种异常情况一般使用数字，例如常见的浏览器中的 404 错误，以及 Windows 中的错误编号等，使用这些数字可以代表各种异常情况，但是最大的不足在于这些数字不够直观，无法很直接的从这些数字中知道异常出现的原因。

所以在 Java 语言中代表异常时，不再使用数字来代表各种异常的情况，而是使用一个专门的类来代表一种特定的异常情况，在系统中传递的异常情况就是该类的对象，所有代表异常的类组成的体系就是 Java 语言中的异常类体系。

为了方便对于这些可传递对象的管理，Java API 中专门设计了 java.lang.Throwable 类，只有该类子类的对象才可以在系统的异常传递体系中进行。该类的两个子类分别是：

1、Error 类

该类代表错误，指程序无法恢复的异常情况。对于所有错误类型以及其子类，都不要求程序进行处理。常见的 Error 类例如内存溢出 StackOverflowError 等。

2、Exception 类

该类代表异常，指程序有可能恢复的异常情况。该类就是整个 Java 语言异常类体系中的父类。使用该类，可以代表所有异常的情况。

在 Java API 中，声明了几百个 Exception 的子类分别来代表各种各样的常见异常情况，这些类根据需要代表的情况位于不同的包中，这些类的类名均以 Exception 作为类名的后缀。如果遇到的异常情况，Java API 中没有对应的异常类进行代表，也可以声明新的异常类来代表特定的情况。

在这些异常类中，根据是否是程序自身导致的异常，将所有的异常类分为两种：

1、RuntimeException 及其所有子类

该类异常属于程序运行时异常，也就是由于程序自身的问题导致产生的异常，例

如数组下标越界异常 ArrayIndexOutOfBoundsException 等。
该类异常在语法上不强制程序员必须处理，即使不处理这样的异常也不会出现语法错误。

2、其它 Exception 子类

该类异常属于程序外部的问题引起的异常，也就是由于程序运行时某些外部问题导致产生的异常，例如文件不存在异常 FileNotFoundException 等。

该类异常在语法上强制程序员必须进行处理，如果不进行处理则会出现语法错误。

熟悉异常类的分类，将有助于后续语法中的处理，也使得在使用异常类时可以选择恰当的异常类类型。

由于异常类的数量非常多，在实际使用时需要经常查阅异常类的文档，下面列举一些常见的异常类，如下表所示：

异常类类名	功能说明
java.lang.NullPointerException	空指针异常，调用 **null** 对象中的非 **static** 成员变量或成员方法时产生该异常
java.lang.ArrayIndexOutofBoundsException	数组下标越界异常，数组下标数值小于 **0** 或大于等于数组长度时产生该异常
java.lang.IllegalArgumentException	非法参数异常，当参数不合法时产生该异常

下面来介绍一下异常处理的相关语法。

Java 编程那些事儿 82——异常处理语法 1

10.3 异常处理语法

为了方便程序员进行异常的处理，在 Java 语言中创建了一套语法，这些语法主要分为以下几个部分：

1、抛出异常

当程序运行时，如果发现异常的情况，通过生成对应的异常对象，并将该异常对象传递给 Java 的运行时系统，使得系统中包含该异常信息，这样的过程被称作抛出异常。

抛出异常是整个异常处理机制的起点，也是异常的发源地，一般出现在项目底层的代码中。

2、声明异常

当一个方法在执行时,除了能够完成正常的功能以外,还可以出现一些异常情况,为了提醒调用该方法的程序员注意处理这些异常情况,需要在方法的声明中将这些异常声明出来，这就是声明异常。

声明异常的语法使得异常处理更加容易进行实现。

3、捕获异常及异常处理

当异常被抛出以后,如果不进行处理,则异常会在方法调用过程中一直进行传递,直到最后一个方法，在 J2SE 中也就是 main 方法，最终将显示在控制台。

在实际项目中，当异常被抛出以后，需要首先捕获到该异常，按照异常的种类不同，分别进行处理。

4、声明自定义异常类

虽然在 JDK API 中提供了几百个异常类,但是这些异常所代表的还只是常见的异常情况，在实际使用时还是无法代表所有的异常情况，所以 Java 语言运行声明自定义的异常类,使用这些自定义的异常类来代表实际项目中 JDK API 无法代表的异常情况。

下面依次详细介绍一下这些语法的相关规则。

10.3.1 抛出异常

在书写项目中相关的底层基础代码时，相关的方法除了实现应该实现的功能以外，还需要考虑到各种异常情况，如果出现该代码所在的方法无法处理的异常情况时，则应该在该方法内部抛出对应类型的异常时，使得整个方法的逻辑比较严谨。

例如，下面是一个实现将十进制数字转换为二进制或 8 进制字符串的方法：

```
/**
 * 将自然数转换为二进制或八进制字符串
 * @param value 需要转换的自然数
 * @param radix 基数，只能取 2 或 8
 * @return 转换后的字符串
 */
public static String toString(int value,int radix){
        if(value == 0){
                return "0";
        }
        StringBuffer s = new StringBuffer();
        int temp; //余数
        while(value != 0){ //未转换结束
                temp = value % radix;  //取余数
                s.insert(0,temp);  //添加到字符串缓冲区
```

```
                value /= radix;  //去掉余数
            }
            return s.toString();
        }
```

在该方法中使用除 n 取余的方法，将参数 value 转换为对应的字符串，当在 main 方法中以书写如下代码时：

```
System.out.println(toString(12,2));
```

则程序的运行结果是：1100

这样在正常的情况下，程序获得了正确的结果，但是该方法由于逻辑的限制，只能实现将“自然数”转换为“二进制或八进制”字符串，如果在其它程序员误传入非法的参数时，则程序会获得不正常的结果，例如书写如下调用的代码时：

```
System.out.println(toString(12,16));
```

则程序的运行结果是： 12

这个结果在转换的逻辑上就是错误的。这样就因为其它程序员误传入非法参数而出现了错误的结果。如果该方法作为实际项目的一个逻辑存在，则会由于该方法的问题导致后续其它的功能也发生错误，这是每个程序员都不希望看到的。

所以该方法虽然在功能上达到了要求，但是逻辑还是不严谨的，还需要在其它程序员调用该方法时传入非法参数这样的异常情况时，将这种异常报告出来，这就需要抛出异常的的代码了。

抛出异常的语法格式为：

```
throw 异常对象;
```

例如：

```
throw new NullPointerException();
```

或

```
IllegalArgumentException e = new IllegalArgumentException();
throw e;
```

该代码书写在方法或构造方法的内部。该语法中，使用 throw 关键字，后续为代表对应异常情况的异常类类型的对象。当系统执行到该代码时，将中止当前方法的执行，而直接返回到调用该方法的位置。所以在该代码下面不能直接书写其它的代码，因为这些代码将永远无法执行到。例如：

```
throw new NullPointerException();
int n = 10;  //语法错误，该代码无法到达
```

按照该语法，则上面的转换方法改造以后的代码如下：

```
/**
 * 将自然数转换为二进制或八进制字符串
 * @param value 需要转换的自然数
```

```
     * @param radix 基数，只能取 2 或 8
     * @return 转换后的字符串
     */
    public static String toString(int value,int radix){
            //判断异常的代码
            if(value <0){
                    throw new IllegalArgumentException("需要
转换的数字不是自然数！");
            }
            if(radix != 2 && radix != 8){
                    throw new IllegalArgumentException("进制
参数非法");
            }
            if(value == 0){
                    return "0";
            }
            StringBuffer s = new StringBuffer();
            int temp; //余数
            while(value != 0){ //未转换结束
                    temp = value % radix;  //取余数
                    s.insert(0,temp);  //添加到字符串缓冲区
                    value /= radix;  //去掉余数
            }
            return s.toString();
    }
```

这里，当 value 的值小于 0 时，则抛出非法参数异常，当 radix 的值不是 2 或 8 时，则抛出非法参数异常。

这样在执行如下代码：

```
    System.out.println(toString(12,2));
    System.out.println(toString(12,16));
```

则程序的执行结果是： 1100

```
Exception in thread "main" java.lang.IllegalArgumentException: 进制参
数非法
```

at ThrowException.toString(ThrowException.java:22)

at ThrowException.main(ThrowException.java:7)

这里当参数符合要求时，则输出正确结果 **1100**，如果参数不合法，则抛出异常，由于异常没有得到处理，则将终止程序的执行，则控制台输出异常的信息，并显示异常的类型以及异常出现的位置。

这样，就通过抛出异常的语法，使得该方法的逻辑比较严谨，在方法的参数不合法，即出现异常情况时，将这个异常报告出来，使得该方法不会出现错误的结果。另外，抛出的异常将传递给运行时系统，这样就将这种异常的情况传递出来，提醒其它的结构进行处理。

10.3.2 声明异常

异常虽然被抛出了，但是由于抛出异常的代码是在方法或构造方法的内部的，在调用方法或构造方法时一般是无法看到方法或构造方法的源代码的，这样调用的程序员就无法知道该方法或构造方法将出现怎样的异常情况，所以需要有一种语法，可以使得调用的程序员可以看到被调用的结构可能出现的异常情况，这就是声明异常的语法。

声明异常的语法类似于药品上的副作用说明，在患者服用药品时，知道药品的正常功能，但是无法详细了解药品的成分以及每种成分的含量(类似于源代码)，但是在药品的说明上都有副作用的说明，例如过敏者不能服用等，这些和声明异常的语法在功能上是类似。

声明异常的语法格式为：

throws 异常类名

例如：

```
public static void test(int n) throws IllegalArgumentException,IOException
public Test()throws IllegalArgumentException
```

该语法使用在方法和构造方法的声明以后，在 **throws** 关键字以后，书写该方法或构造方法可能出现的异常，在这里需要书写异常类的类名，如果有多个，则使用逗号分隔这些异常类名即可。

这里需要注意的是：

1、这些异常必须是该方法内部可能抛出的异常

2、异常类名之间没有顺序

3、属于 **RuntimeException** 子类的异常可以不书写在 **throws** 语句以后，但是另外一类异常如果可能抛出则必须声明在 **throws** 语句之后

通过在对应的方法或构造方法声明中书写 **throws** 语句，使得调用该方法或构造方法的程序员可以在调用时看到对应结构可能出现的异常情况，从而提示对于这些异常情况进行处理，从而增强程序的健壮性。

而且即使在程序由于未处理对应的异常而导致程序在运行时出现错误时，也可以使程序员可以通过对应的提示获得错误的原因，并指导程序员进行逻辑的修正，这样都可以提高程序编写时的效率，也可以使程序员更加容易的编写出逻辑严谨的代码，从而增加项目的质量，提高程序的稳定性。

但是声明异常以后，异常还是存在的，异常还没有获得处理，在异常体系中最重要的还是捕获到异常，然后针对异常的类型不同作出对应的处理。

Java 编程那些事儿 83——异常处理语法 2

陈跃峰

出自：http://blog.csdn.net/mailbomb

10.3.3 捕获异常及异常处理

在整个异常处理机制中，异常在系统中进行传递，传递到程序员认为合适的位置，就捕获到该异常，然后进行逻辑处理，使得项目不会因为出现异常而崩溃。

为了捕获异常并对异常进行处理，使用的捕获异常以及处理的语法格式为：

```
try{
    //逻辑代码
}catch(异常类名 参数名){
    //处理代码
}
```

在该语法中，将正常的程序逻辑代码书写在 **try** 语句块内部进行执行，这些代码为可能抛出异常的代码，而 **catch** 语句中书写对应的异常类的类名，在 **catch** 语句块内部书写出现该类型的异常时的处理代码。

程序执行到 **try-catch** 语句时，如果没有发生异常，则完整执行 **try** 语句块内部的所有代码，而 **catch** 语句块内部的代码不会执行，如果在执行时发生异常，则从发生异常的代码开始，后续的 **try** 语句块代码不会执行，而跳转到该类型的异常对应的 **catch** 语句块中。

示例代码如下：

```
String s = "123";
try{
    int n = Integer.parseInt(s);
    System.out.println(n);
}catch(NumberFormatException e){
    System.out.println("该字符串无法转换！");
}
```

在该示例代码中，**Integer** 类的 **parseInt** 方法可能会抛出 **NumberFormatException**，因为 **parseInt** 方法的声明如下：

```
public static int parseInt(String s) throws NumberFormatException
```

这里字符串 **s** 转换为 **int** 没有发生异常，则程序执行完 **try** 语句块内部的代码，程序的运行结果为：**123**

如果将字符串对象 **s** 的值修改为**"abc"**，则运行上面的代码，则 **parseInt** 方法执行时将抛出 **NumberFormatException**，则调用 **parseInt** 方法语句后续的 **try** 语句块中的代码不会执行，程序的执行流程跳转到捕获 **NumberFormatException** 异常的 **catch** 语句块内部，然后执行该 **catch** 语句块内部的代码，则程序的执行结果是：

```
该字符串无法转换！
```

这就是最基本的捕获异常和异常处理的代码结构。使用 **try** 语句捕获程序执行时抛出的异常，使用 **catch** 语句块匹配抛出的异常类型，在 **catch** 语句块内部书写异常处理的代码。

在实际程序中，也可以根据异常类型的不同进行不同的处理，这样就需要多

个 **catch** 语句块，其结构如下：

```
try{
    //逻辑代码
} catch(异常类名 1  参数名 1){
    //处理代码 1
} catch(异常类名 2  参数名 2){
    //处理代码 2
}
……
}catch(异常类名 n  参数名 n){
    //处理代码 n
}
```

例如：

```
String s = "123";
try{
    int n = Integer.parseInt(s);
    System.out.println(n);
    char c = s.charAt(4);
    System.out.println(c);
}catch(NumberFormatException e){
    System.out.println("该字符串无法转换！ ");
}catch(StringIndexOutOfBoundsException e){
    System.out.println("字符串索引值越界");
}
```

在执行时，按照 **catch** 语句块书写的顺序从上向下进行匹配，直到匹配到合适的异常就结束 **try-catch** 语句块的执行。

在实际执行时，就可以根据捕获的异常类型不同，书写不同的异常处理的代码了。使用该语法时需要注意，如果这些异常类直接存在继承关系，则子类应该书写在上面，父类应该书写在下面，否则将出现语法错误。例如：

```
String s = "123";
try{
    int n = Integer.parseInt(s);
    System.out.println(n);
    char c = s.charAt(4);
    System.out.println(c);
}catch(Exception e){

}catch(NumberFormatException e){ //语法错误，异常已经被处理
    System.out.println("该字符串无法转换！ ");
}catch(StringIndexOutOfBoundsException e){ //语法错误，异常已经被处理
    System.out.println("字符串索引值越界");
```

```
}
```

这里 **Exception** 类是所有异常类的父类，在匹配时可以匹配到所有的异常，所有后续的两个异常处理的代码根本不会得到执行，所以将出现语法错误。正确的代码应该为：

```
String s = "123";
try{
    int n = Integer.parseInt(s);
    System.out.println(n);
    char c = s.charAt(4);
    System.out.println(c);
}catch(NumberFormatException e){
    System.out.println("该字符串无法转换！");
}catch(StringIndexOutOfBoundsException e){
    System.out.println("字符串索引值越界");
}catch(Exception e){


}
```

如果在程序执行时，所有的异常处理的代码都是一样的，则可以使用 **Exception** 代表所有的异常，而不需要一一进行区分，示例代码如下：

```
String s = "123";
try{
    int n = Integer.parseInt(s);
    System.out.println(n);
    char c = s.charAt(4);
    System.out.println(c);
}catch(Exception e){
    //统一的处理代码
}
```

在实际使用时，由于 **try-catch** 的执行流程，使得无论是 **try** 语句块还是 **catch** 语句块都不一定会被完全执行，而有些处理代码则必须得到执行，例如文件的关闭和网络连接的关闭等，这样如何在 **try** 语句块和 **catch** 语句块中都书写则显得重复，而且容易出现问题，这样在异常处理的语法中专门设计了 **finally** 语句块来进行代码的书写。语法保证 **finally** 语句块内部的代码肯定获得执行，即使在 **try** 或 **catch** 语句块中包含 **return** 语句也会获得执行，语法格式如下：

```
finally{
    //清理代码
}
```

该语法可以和上面的两种 **try-catch** 语句块匹配，也可以和 **try** 语句匹配使用，和 **try** 语句块匹配的语法格式如下：

```
try{
    //逻辑代码
}finally{
```

```
        //清理代码
    }
```

这样在执行时，无论 **try** 语句块中的语句是否发生异常，**finally** 语句块内部的代码都会获得执行。

而只书写 **finally** 而不书写 **catch** 则会导致异常的丢失，所以最常用的异常处理的语法格式还是如下语法：

```
    try{
        //逻辑代码
    }catch(异常类名 参数){
        //异常处理代码
    }finally{
        //清理代码
    }
```

这样就是整个异常处理部分的逻辑代码、异常处理代码和清理的代码成为一个整体，使得程序代码更加显得紧凑，便于代码的阅读和使用。

最后，介绍一下使用异常处理语法时需要注意的问题：

1、书写在 **try** 语句块内部的代码执行效率比较低。所以在书写代码时，只把可能出现异常的代码书写在 **try** 语句块内部。

2、如果逻辑代码中抛出的异常属于 **RuntimeException** 的子类，则不强制书写在 **try** 语句块的内部，如果抛出的异常属于非 **RuntimeException** 的子类，则必须进行处理，其中书写在 **try** 语句块是一种常见的处理方式。

3、**catch** 语句块中只能捕获 **try** 语句块中可能抛出的异常，否则将出现语法错误。

10.3.4 声明自定义异常类

如果 **JDK API** 中提供的已有的异常类无法满足实际的使用需要，则可以根据需要声明自定义的异常类来代表项目中指定类型的异常。

异常类在语法上要求必须直接或间接继承 **Exception**，可以根据需要选择继承 **Exception** 或 **RuntimeException** 类，这样也设定了自定义异常类的类型。如果直接继承 **Exception**，则属于必须被处理的异常，如果继承 **RuntimeException**，则不强制必须被处理。当然，可以根据需要继承其它 **Exception** 的子类。

在编码规范上，一般将异常类的类名命名为 **XXXException**，其中 **XXX** 用来代表该异常的作用。

示例代码如下：

```
/**
 * 自定义异常类
 */
public class MyException extends RuntimeException {}
```

自定义异常类的用途，则完全由程序员进行规定，可以在出现该异常类型的条件时抛出该异常，则就可以代表该类型的异常了。

在实际的项目中，有些时候还需要设计专门的异常类体系，来代表各种项目中需要代表的异常情况。

10.4 异常处理方式

前面介绍了异常处理机制的相关语法，但是当出现异常时，如何进行处理是语法无法解决的问题，下面就介绍一下异常处理的方式。

异常处理，顾名思义就是将出现的异常处理掉，但是根据异常出现的位置以及异常的类型不同，会出现很多的方式，依次介绍如下：

1、不处理

该处理方式就是只捕获异常不进行处理。不推荐使用该方式。

例如：

```
String s = “abc”;
try{
    int n = Integer.parseInt(s);
}catch(Exception e){

}
```

对于这样的处理，该异常被忽略掉了，有可能会影响后续逻辑的执行执行。该种处理方式一般被初学者使用的比较多。

2、直接处理掉

如果具备处理异常的条件，则可以根据不同的异常情况将该异常处理掉，例如给出用户错误原因的提示等或根据逻辑上的需要修正对应的数值。

例如：

```
/**
 * 异常处理示例
 * 该程序的功能是将用户输出的命令行参数转换为数字并输出
 */
public class ExceptionHandle1 {
    public static void main(String[] args) {
        int n = 0;
        try{
            //将输入的第一个命令行参数转换为数字
            n = Integer.parseInt(args[0]);
            //输出转换后的结果
            System.out.println(n);
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("请输入命令行参数！");
        }catch(NumberFormatException e){
            System.out.println("命令行参数不是数字字符串！");
        }
    }
}
```

在执行该代码时，如果发生数组下标越界异常，则代表用户未输入命令行参数，

则提示用户输入命令行参数，如果发生数字格式化异常，则代表用户输入的第一个命令行参数不是数字字符串，则给出用户对应的提示信息。
该示例中就是根据异常出现的原因提示用户进行正确的操作，这是一种常见的异常处理的方式。

3、重新抛出

在实际的项目，有些时候也需要将某个方法内部出现的所有异常都转换成项目中自定义的某个异常类，然后再重新抛出。

例如：

```
try{
    //逻辑代码
}catch(Exception e){
    throw new MyException();
}
```

这样转换以后，比较容易发现异常出现的位置，也方便项目中逻辑上的处理。

4、在方法中声明

如果当前方法或构造方法中，或在当前方法或构造方法中调用的其它方法，会抛出某个异常，而在当前方法或构造方法中没有足够的信息对该异常进行处理，则可以将该异常进行传递，将该异常传递给调用当前方法的位置在进行处理。这就是异常在系统中传递的方式。

例如：

```
/**
 * 异常处理示例 2
 * 演示在方法中重新声明异常
 */
public class ExceptionHandle2 {
    public void test(String s) throws NumberFormatException{
        int n = Integer.parseInt(s);
        System.out.println(n);
    }
}
```

在 **test** 方法中，由于 **parseInt** 方法会抛出 **NumberFormatException** 异常，而该方法内部无法对此进行处理，所以则在 **test** 方法内部声明把 **NumberFormatException** 异常抛出，这样该异常就由调用 **test** 的方法再进行处理。这样异常就可以在方法之间进行传递了。

这里列举了异常处理中常见的一些处理方式，当然也可以根据需要，进行其它的处理。

10.5 总结

异常处理是 **Java** 语言中增强程序健壮性的一种机制，**Java** 语法为异常处理提供了一系列的语法格式，在实际的项目中，需要根据具体的情况对异常给出对应的处理，使得异常能够被正确的处理掉，从而保证项目的正常执行。

Java 编程那些事儿 84——IO 简介

陈跃峰
出自：**http://blog.csdn.net/mailbomb**

第十一章 I/O 处理

I/O(Input/Output)是输入和输出的简称，在 **Java** 语言中所有和输入输出有关的都属于 **IO** 处理技术，例如包含文件的读写和网络数据的发送等。

其实整个计算机实现功能时最主要的就是 **IO**，用户使用鼠标、键盘等输入设备向程序输入数据，程序进行一定的逻辑处理，然后再将处理结果以一定的形式显示到显示器等输出设备上。

I/O 处理技术是 **Java** 语言中实现文件操作、内存操作、控制台输入以及网络编程的基础，但是由于 **I/O** 技术本身的设计原因，也使得 **I/O** 处理技术的难度比较大，需要花费比较大的精力进行学习。

11.1 I/O 简介

IO 是输入和输出的简称，在实际的使用时，输入和输出是有方向的。就像现实中两个人之间借钱一样，例如 **A** 借钱给 **B**，相对于 **A** 来说是借出，而相对于 **B** 来说则是借入。所以在程序中提到输入和输出时，也需要区分清楚是相对的内容。

在程序中，输入和输出都是相对于当前程序而言的，例如从硬盘上读取一个配置文件的内容到程序中，则相当于将文件的内容输入到程序内部，因此输入和“读”对应，而将程序中的内容保存到硬盘上，则相当于将文件的内容输出到程序外部，因此输出和“写”对应。熟悉输入和输出的对应关系，将有助于后续内容的学习。

在 **Java** 语言中，输入和输出的概念要比其它语言的输入和输出的概念涵盖的内容广泛得多，不仅包含文件的读写，也包含网络数据的发送，甚至内存数据的读写以及控制台数据的接收等都由 **IO** 来完成。

为了使输入和输出的结构保持统一，从而方便程序员使用 **IO** 相关的类，在 **Java** 语言的 **IO** 类设计中引入了一个新的概念——**Stream(**流**)**。

由于在进行 **IO** 操作时，需要操作的种类很多，例如文件、内存和网络连接等，这些都被称作数据源**(data source)**，对于不同的数据源处理的方式是不一样的，如果直接交给程序员进行处理，对于程序员来说则显得比较复杂。

所以在所有的 **IO** 类设计时，在读数据时，**JDK API** 将数据源的数据转换为一种固定的数据序列，在写数据时，将需要写的数据以一定的格式写入到数据序列，由 **JDK API** 完成将数据序列中的数据写入到对应的数据源中。这样由系统完成复杂的数据转换以及不同数据源之间的不同的变换，从而简化程序员的编码。

IO 的这种设计就和城市中的供水和排水系统设计是一样的，在供水的时候，水源有江河水、湖水和地下水等不同类型，由自来水公司完成把水源转换为对应的水流。而在排水系统设计时，只需要将污水排入污水管道即可，至于这些污水是怎么被处理的，则不需要关心，这样也简化了家庭用水的处理。

IO 设计中这种数据序列被形象的称作流**(Stream)**。通过使用流的概念，使程序员面对不同的数据源时只需要建立不同的流即可，而底层流实现的复杂性则由系统完成，从而使程序员不必深入的了解每种数据源的读写方式，从而降低了 **IO** 编程的复杂度。

在整个 **IO** 处理中，读数据的过程分为两个步骤：**1**、将数据源的内容转换为流结构，该步骤由 **JDK API** 完成，程序员只需要选择合适的流类型即可。**2**、从流中读取数据，该步骤由程序员完成，流中数据的顺序和数据源中数据的存储顺序保持一致。

写数据的过程也分为两个步骤：**1**、为连接指定的数据源而建立的专门的流结构，该步骤由 **JDK API** 完成，程序员只需要选择合适的流类型捷克。**2**、将数据以一定的格式写入到流中，该步骤由程序员完成，写入流中的数据的顺序就是数据在数据源中的存储顺序。最后，当数据写入流中以后，可以通过一定的方式把流中的数据写入数据源，或者当流被关闭时，系统会自动将流中的数据写入数据源中。

这样，在整个 **IO** 类设计时，将最复杂的和数据源操作的部分由 **JDK API** 进行完成，而程序员进行编程时，只需要选择合适的流类型，然后进行读写即可。

和现实的结构一样，**IO** 中的流也是有方向的，用于读的流被称作输入流**(Input Stream)**，用于写的流被称作输出流**(Output Stream)**。则进行读写的时候需要选择合适的流对象进行操作。

流结构的示意图如下所示：

读数据流示意图

写数据流示意图

而由于 **Java** 语言使用面向对象技术，所以在实现时，每个流类型都使用专门的类进行代表，而把读或写该类型数据源的逻辑封装在类的内部，在程序员实际使用时创建对应的对象就完成了流的构造，后续的 **IO** 操作则只需要读或写流对象内部的数据即可。这样 **IO** 操作对于 **Java** 程序员来说，就显得比较简单，而且比较容易操作了。

Java 编程那些事儿 85——IO 类体系

陈跃峰
出自：http://blog.csdn.net/mailbomb

11.2 I/O 类体系

在 **JDK API** 中，基础的 **IO** 类都位于 **java.io** 包，而新实现的 **IO** 类则位于一

系列以 **java.nio** 开头的包名中，这里首先介绍 **java.io** 包中类的体系结构。

按照前面的说明，流是有方向的，则整个流的结构按照流的方向可以划分为两类：

1、输入流：

该类流将外部数据源的数据转换为流，程序通过读取该类流中的数据，完成对于外部数据源中数据的读入。

2、输出流：

该类流完成将流中的数据转换到对应的数据源中，程序通过向该类流中写入数据，完成将数据写入到对应的外部数据源中。

而在实际实现时，由于 **JDK API** 历史的原因，在 **java.io** 包中又实现了两类流：字节流**(byte stream)**和字符流**(char stream)**。这两种流实现的是流中数据序列的单位，在字节流中，数据序列以 **byte** 为单位，也就是流中的数据按照一个 **byte** 一个 **byte** 的顺序实现成流，对于该类流操作的基本单位是一个 **byte**，而对于字节流，数据序列以 **char** 为单位，也就是流中的数据按照一个 **char** 一个插入的顺序实现成流，对于该类流操作的基本单位是一个 **char**。

另外字节流是从 **JDK1.0** 开始加入到 **API** 中的，而字符流则是从 **JDK1.1** 开始才加入到 **API** 中的，对于现在使用的 **JDK** 版本来说，这两类流都包含在 **API** 的内部。在实际使用时，字符流的效率要比字节流高一些。

在实际使用时，字符流中的类基本上和字节流中的类对应，所以在开始学习 **IO** 类时，可以从最基础的字节流开始学习。

在 **SUN** 设计 **JDK** 的 **IO** 类时，按照以上的分类，为每个系列的类设计了一个父类，而实现具体操作的类都作为该系列类的子类，则 **IO** 类设计时的四个体系中每个体系中对应的父类分别是：

11.2.1 字节输入流 InputStream

该类是 **IO** 编程中所有字节输入流的父类，熟悉该类的使用将对使用字节输入流产生很大的帮助，下面做一下详细的介绍。

按照前面介绍的流的概念，字节输入流完成的是按照字节形式构造读取数据的输入流的结构，每个该类的对象就是一个实际的输入流，在构造时由 **API** 完成将外部数据源转换为流对象的操作，这种转换对程序员来说是透明的。在程序使用时，程序员只需要读取该流对象，就可以完成对于外部数据的读取了。

InputStream 是所有字节输入流的父类，所以在 **InputStream** 类中包含的每个方法都会被所有字节输入流类继承，通过将读取以及操作数据的基本方法都声明在 **InputStream** 类内部，使每个子类根据需要覆盖对应的方法，这样的设计可以保证每个字节输入流子类在进行实际使用时，开放给程序员使用的功能方法是一致的。这样将简化 **IO** 类学习的难度，方便程序员进行实际的编程。

默认情况下，对于输入流内部数据的读取都是单向的，也就是只能从输入流从前向后读，已经读取的数据将从输入流内部删除掉。如果需要重复读取流中同一段内容，则需要使用流类中的 **mark** 方法进行标记，然后才能重复读取。这种设计在使用流类时，需要深刻进行体会。

在 **InputStream** 类中，常见的方法有：

a、available 方法

public int available() throws IOException

该方法的作用是返回当前流对象中还没有被读取的字节数量。也就是获得流中数据的长度。

假设初始情况下流内部包含 **100** 个字节的数据，程序调用对应的方法读取了一个字节，则当前流中剩余的字节数量将变成 **99** 个。

另外，该方法不是在所有字节输入流内部都得到正确的实现，所以使用该方法获得流中数据的个数是不可靠的。

b、close 方法

public void close() throws IOException

该方法的作用是关闭当前流对象，并释放该流对象占用的资源。

在 **IO** 操作结束以后，关闭流是进行 **IO** 操作时都需要实现的功能，这样既可以保证数据源的安全，也可以减少内存的占用。

c、markSupported 方法

public boolean markSupported()

该方法的作用是判断流是否支持标记**(mark)**。标记类似于读书时的书签，可以很方便的回到原来读过的位置继续向下读取。

d、reset 方法

public void reset() throws IOException

该方法的作用是使流读取的位置回到设定标记的位置。可以从该位置开始继续向后读取。

e、mark 方法

public void mark(int readlimit)

为流中当前的位置设置标志，使得以后可以从该位置继续读取。变量 **readlimit** 指设置该标志以后可以读取的流中最大数据的个数。当设置标志以后，读取的字节数量超过该限制，则标志会失效。

f、read 方法

read 方法是输入流类使用时最核心的方法，能够熟练使用该方法就代表 **IO** 基本使用已经入门。所以在学习以及后期的使用中都需要深刻理解该方法的使用。

在实际读取流中的数据时，只能按照流中的数据存储顺序依次进行读取，在使用字节输入流时，读取数据的最小单位是字节**(byte)**。

另外，需要注意的是，**read** 方法是阻塞方法，也就是如果流对象中无数据可以读取时，则 **read** 方法会阻止程序继续向下运行，一直到有数据可以读取为止。

read 方法总计有三个，依次是：

public abstract int read() throws IOException

该方法的作用是读取当前流对象中的第一个字节。当该字节被读取出来以后，则该字节将被从流对象中删除，原来流对象中的第二个字节将变成流中的第一个字节，而使用流对象的 **available** 方法获得的数值也将减少 **1**。如果需要读取流中的所以数据，只要使用一个循环依次读取每个数据即可。当读取到流的末尾时，该方法返回**-1**。该返回值的 **int** 中只有最后一个字节是流中的有效数据，所以在获得流中的数值时需要进行强制转换。返回值作成 **int** 的目的主要是处理好**-1** 的问题。

由于该方法是抽象的，所以会在子类中被覆盖，从而实现最基础的读数据的功能。

public int read(byte[] b) throws IOException

该方法的作用是读取当前流对象中的数据，并将读取到的数据依次存储到数组 **b (b** 需要提前初始化完成)中，也就是把当前流中的第一个字节的数据存储到 **b [0]**，第二个字节的数据存储到 **b[1]**，依次类推。流中已经读取过的数据也会被删除，后续的数据会变成流中的第一个字节。而实际读取的字节数量则作为方法的返回值返回。

public int read(byte[] b, int off, int len) throws IOException

该方法的作用和上面的方法类似，也是将读取的数据存储到 **b** 中，只是将流中的第一个数据存储到 **b** 中下标为 **off** 的位置，最多读取 **len** 个数据，而实际读取的字节数量则作为方法的返回值返回。

g、skip 方法

public long skip(long n) throws IOException

该方法的作用是跳过当前流对象中的 **n** 个字节，而实际跳过的字节数量则以返回值的方式返回。

跳过 **n** 个字节以后，如果需要读取则是从新的位置开始读取了。使用该方法可以跳过流中指定的字节数，而不用依次进行读取了。

从流中读取出数据以后，获得的是一个 **byte** 数组，还需要根据以前的数据格式，实现对于该 **byte** 数组的解析。

由于 **InputStream** 类是字节输入流的父类，所以该体系中的每个子类都包含以上的方法，这些方法是实现 **IO** 流数据读取的基础。

11.2.2 字节输出流 OutputStream

该类是所有的字节输出流的父类，在实际使用时，一般使用该类的子类进行编程，但是该类内部的方法是实现字节输出流的基础。

该体系中的类完成把对应的数据写入到数据源中，在写数据时，进行的操作分两步实现：第一步，将需要输出的数据写入流对象中，数据的格式由程序员进行设定，该步骤需要编写代码实现；第二步，将流中的数据输出到数据源中，该步骤由 **API** 实现，程序员不需要了解内部实现的细节，只需要构造对应的流对象即可。

在实际写入流时，流内部会保留一个缓冲区，会将程序员写入流对象的数据首先暂存起来，然后在缓冲区满时将数据输出到数据源。当然，当流关闭时，输出流内部的数据会被强制输出。

字节输出流中数据的单位是字节，在将数据写入流时，一般情况下需要将数据转换为字节数组进行写入。

在 **OutputStream** 中，常见的方法有：

a、close 方法

public void close() throws IOException

该方法的作用是关闭流，释放流占用的资源。

b、flush 方法

public void flush() throws IOException

该方法的作用是将当前流对象中的缓冲数据强制输出出去。使用该方法可以实现立即输出。

c、write 方法

write 方法是输出流中的核心方法，该方法实现将数据写入流中。在实际写入前，需要实现对应的格式，然后依次写入到流中。写入流的顺序就是实际数据输出的顺序。

write 方法总计有 **3** 个，依次是：

public abstract void write(int b) throws IOException

该方法的作用是向流的末尾写入一个字节的数据。写入的数据为参数 **b** 的最后一个字节。在实际向流中写数据时需要按照逻辑的顺序进行写入。该方法在 **OutputStream** 的子类内部进行实现。

public void write(byte[] b) throws IOException

该方法的作用是将数组 **b** 中的数据依次写入当前的流对象中。

public void write(byte[] b, int off, int len) throws IOException

该方法的作用是将数组 **b** 中从下标为 **off(**包含**)**开始，后续长度为 **len** 个的数据依次写入到流对象中。

在实际写入时，还需要根据逻辑的需要设定 **byte** 数值的格式，这个根据不同的需要实现不同的格式。

11.2.3 字符输入流 Reader

字符输入流体系是对字节输入流体系的升级，在子类的功能上基本和字节输入流体系中的子类一一对应，但是由于字符输入流内部设计方式的不同，使得字符输入流的执行效率要比字节输入流体系高一些，在遇到类似功能的类时，可以优先选择使用字符输入流体系中的类，从而提高程序的执行效率。

Reader 体系中的类和 **InputStream** 体系中的类，在功能上是一致的，最大的区别就是 **Reader** 体系中的类读取数据的单位是字符**(char)**，也就是每次最少读入一个字符**(**两个字节**)**的数据，在 **Reader** 体系中的读数据的方法都以字符作为最基本的单位。

Reader 类和 **InputStream** 类中的很多方法，无论声明还是功能都是一样的，但是也增加了两个方法,依次介绍如下：

a、read 方法

public int read(CharBuffer target) throws IOException

该方法的作用是将流内部的数据依次读入 **CharBuffer** 对象中，实际读入的 **char** 个数作为返回值返回。

b、ready 方法

public boolean ready() throws IOException

该方法的作用是返回当前流对象是否准备完成，也就是流内部是否包含可以被读取的数据。

其它和 **InputStream** 类一样的方法可以参看上面的介绍。

11.2.4 字符输出流 Writer

字符输出流体系是对字节输出流体系的升级，在子类的功能实现上基本上和字节输出流保持一一对应。但由于该体系中的类设计的比较晚，所以该体系中的类执行的效率要比字节输出流中对应的类效率高一些。在遇到类似功能的类时，可以优先选择使用该体系中的类进行使用，从而提高程序的执行效率。

Writer 体系中的类和 **OutputStream** 体系中的类，在功能上是一致的，最大的区别就是 **Writer** 体系中的类写入数据的单位是字符**(char)**，也就是每次最少写入一个字符**(**两个字节**)**的数据，在 **Writer** 体系中的写数据的方法都以字符作为最基本的操作单位。

Writer 类和 **OutputStream** 类中的很多方法，无论声明还是功能都是一样的，但是还是增加了一些方法，依次介绍如下：

a、append 方法

将数据写入流的末尾。总计有 **3** 个方法，依次是：

public Writer append(char c) throws IOException

该方法的作用和 **write(int c)**的作用完全一样，既将字符 **c** 写入流的末尾。

public Writer append(CharSequence csq) throws IOException

该方法的作用是将 **CharSequence** 对象 **csq** 写入流的末尾，在写入时会调用 **csq** 的 **toString** 方法将该对象转换为字符串，然后再将该字符串写入流的末尾。

public Writer append(CharSequence csq, int start, int end)
throws IOException

该方法的作用和上面的方法类似，只是将转换后字符串从索引值为 **start(**包含**)**到索引值为 **end(**不包含**)**的部分写入流中。

b、write 方法

除了基本的 **write** 方法以外，在 **Writer** 类中又新增了两个，依次是：

public void write(String str) throws IOException

该方法的作用是将字符串 **str** 写入流中。写入时首先将 **str** 使用 **getChars** 方法转换成对应的 **char** 数组，然后实现依次写入流的末尾。

public void write(String str, int off, int len)
throws IOException

该方法的作用是将字符串 **str** 中索引值为 **off(**包含**)**开始，后续长度为 **len** 个字符写入到流的末尾。

使用这两个方法将更方便将字符串写入流的末尾。

其它和 **OutputStream** 类一样的方法可以参看上面的介绍。

11.2.5 小结

在实际使用 **IO** 类时，根据逻辑上的需要，挑选对应体系中的类进行实际的使用，从而实现程序中 **IO** 的相关功能。

熟悉了 **IO** 类的体系以后，就可以首先熟悉基本的 **IO** 类的使用，然后再按照

IO 类体系中相关类的使用方式逐步去了解相关的 IO 类的使用，从而逐步熟悉 java.io 包中类的使用，然后再掌握 IO 编程。

在实际使用时，一般都使用这 4 个类中对应的子类，每个子类完成相关的功能。对于这些子类，也可以根据这些类是否直接连接数据源，将这些 IO 类分类为：

1、实体流

指直接连接数据源的 IO 流类

2、装饰流

指不直接连接数据源，而是建立在其它实体流对象的基础之上。

下面 IO 类的使用中将分别介绍这些体系中的类。在实际使用时也应该根据流属于实体流还是装饰流进行不同的使用。

Java 编程那些事儿 86——文件操作之 File 类使用

陈跃峰
出自：http://blog.csdn.net/mailbomb

11.3 I/O 类使用

由于在 IO 操作中，需要使用的数据源有很多，作为一个 IO 技术的初学者，从读写文件开始学习 IO 技术是一个比较好的选择。因为文件是一种常见的数据源，而且读写文件也是程序员进行 IO 编程的一个基本能力。本章 IO 类的使用就从读写文件开始。

11.3.1 文件操作

文件(File)是最常见的数据源之一，在程序中经常需要将数据存储到文件中，例如图片文件、声音文件等数据文件，也经常需要根据需要从指定的文件中进行数据的读取。当然，在实际使用时，文件都包含一个的格式，这个格式需要程序员根据需要进行设计，读取已有的文件时也需要熟悉对应的文件格式，才能把数据从文件中正确的读取出来。

文件的存储介质有很多，例如硬盘、光盘和 U 盘等，由于 IO 类设计时，从数据源转换为流对象的操作由 API 实现了，所以存储介质的不同对于程序员来说是透明的，和实际编写代码无关。

11.3.1.1 文件的概念

文件是计算机中一种基本的数据存储形式，在实际存储数据时，如果对于数据的读写速度要求不是很高，存储的数据量不是很大时，使用文件作为一种持久数据存储的方式是比较好的选择。

存储在文件内部的数据和内存中的数据不同，存储在文件中的数据是一种

“持久存储”，也就是当程序退出或计算机关机以后，数据还是存在的，而内存内部的数据在程序退出或计算机关机以后，数据就丢失了。

在不同的存储介质中，文件中的数据都是以一定的顺序依次存储起来，在实际读取时由硬件以及操作系统完成对于数据的控制，保证程序读取到的数据和存储的顺序保持一致。

每个文件以一个文件路径和文件名称进行表示，在需要访问该文件的时，只需要知道该文件的路径以及文件的全名即可。在不同的操作系统环境下，文件路径的表示形式是不一样的，例如在 **Windows** 操作系统中一般的表示形式为 **C:\windows\system**，而 **Unix** 上的表示形式为**/user/my**。所以如果需要让 **Java** 程序能够在不同的操作系统下运行，书写文件路径时还需要比较注意。

11.3.1.1.1 绝对路径和相对路径

绝对路径是指书写文件的完整路径，例如 **d:\java\Hello.java**，该路径中包含文件的完整路径 **d:\java** 以及文件的全名 **Hello.java**。使用该路径可以唯一的找到一个文件，不会产生歧义。但是使用绝对路径在表示文件时，受到的限制很大，且不能在不同的操作系统下运行，因为不同操作系统下绝对路径的表达形式存在不同。

相对路径是指书写文件的部分路径，例如**\test\Hello.java**，该路径中只包含文件的部分路径**\test** 和文件的全名 **Hello.java**，部分路径是指当前路径下的子路径，例如当前程序在 **d:\abc** 下运行，则该文件的完整路径就是 **d:\abc\test**。使用这种形式，可以更加通用的代表文件的位置，使得文件路径产生一定的灵活性。

在 **Eclipse** 项目中运行程序时，当前路径是项目的根目录，例如工作空间存储在 **d:\javaproject**，当前项目名称是 **Test**，则当前路径是：**d:\javaproject\Test**。在控制台下面运行程序时，当前路径是 **class** 文件所在的目录，如果 **class** 文件包含包名，则以该 **class** 文件最顶层的包名作为当前路径。

另外在 **Java** 语言的代码内部书写文件路径时，需要注意大小写，大小写需要保持一致，路径中的文件夹名称区分大小写。由于**'\'** 是 **Java** 语言中的特殊字符，所以在代码内部书写文件路径时，例如代表**“c:\test\java\Hello.java”**时，需要书写成**“c:\\test\\java\\Hello.java”**或**“c:/test/java/Hello.java”**，这些都需要在代码中注意。

11.3.1.1.2 文件名称

文件名称一般采用“文件名.后缀名”的形式进行命名，其中“文件名”用来表示文件的作用，而使用后缀名来表示文件的类型，这是当前操作系统中常见的一种形式，例如**“readme.txt”**文件，其中 **readme** 代表该文件时说明文件，而 **txt** 后缀名代表文件时文本文件类型，在操作系统中，还会自动将特定格式的后缀名和对应的程序关联，在双击该文件时使用特定的程序打开。

其实在文件名称只是一个标示，和实际存储的文件内容没有必然的联系，只是使用这种方式方便文件的使用。在程序中需要存储数据时，如果自己设计了特定的文件格式，则可以自定义文件的后缀名，来标示自己的文件类型。

和文件路径一样，在 **Java** 代码内部书写文件名称时也区分大小写，文件名称的大小写必须和操作系统中的大小写保持一致。

另外，在书写文件名称时不要忘记书写文件的后缀名。

11.3.1.2 File 类

为了很方便的代表文件的概念，以及存储一些对于文件的基本操作，在 **java.io** 包中设计了一个专门的类——**File** 类。

在 **File** 类中包含了大部分和文件操作的功能方法，该类的对象可以代表一个具体的文件或文件夹，所以以前曾有人建议将该类的类名修改成 **FilePath**，因为该类也可以代表一个文件夹，更准确的说是可以代表一个文件路径。

下面介绍一下 **File** 类的基本使用。

1、File 对象代表文件路径

File 类的对象可以代表一个具体的文件路径，在实际代表时，可以使用绝对路径也可以使用相对路径。

下面是创建的文件对象示例。

public File(String pathname)

该示例中使用一个文件路径表示一个 **File** 类的对象，例如：

```
File f1 = new File("d:\\test\\1.txt");
File f2 = new File("1.txt");
File f3 = new File("e:\\abc");
```

这里的 **f1** 和 **f2** 对象分别代表一个文件，**f1** 是绝对路径，而 **f2** 是相对路径，**f3** 则代表一个文件夹，文件夹也是文件路径的一种。

public File(String parent, String child)

也可以使用父路径和子路径结合，实现代表文件路径，例如：

```
File f4 = new File("d:\\test\\","1.txt");
```

这样代表的文件路径是：**d:\test\1.txt**。

2、File 类常用方法

File 类中包含了很多获得文件或文件夹属性的方法，使用起来比较方便，下面将常见的方法介绍如下：

a、createNewFile 方法

public boolean createNewFile() throws IOException

该方法的作用是创建指定的文件。该方法只能用于创建文件，不能用于创建文件夹，且文件路径中包含的文件夹必须存在。

b、delect 方法

public boolean delete()

该方法的作用是删除当前文件或文件夹。如果删除的是文件夹，则该文件夹必须为空。如果需要删除一个非空的文件夹，则需要首先删除该文件夹内部的每个文件和文件夹，然后在可以删除，这个需要书写一定的逻辑代码实现。

c、exists 方法

public boolean exists()

该方法的作用是判断当前文件或文件夹是否存在。

d、getAbsolutePath 方法

public String getAbsolutePath()

该方法的作用是获得当前文件或文件夹的绝对路径。例如 **c:\test\1.t** 则返回 **c:\test\1.t**。

e、getName 方法

public String getName()

该方法的作用是获得当前文件或文件夹的名称。例如 **c:\test\1.t**，则返回 **1.t**。

f、getParent 方法

public String getParent()

该方法的作用是获得当前路径中的父路径。例如 **c:\test\1.t** 则返回 **c:\test**。

g、isDirectory 方法

public boolean isDirectory()

该方法的作用是判断当前 **File** 对象是否是目录。

h、isFile 方法

public boolean isFile()

该方法的作用是判断当前 **File** 对象是否是文件。

i、length 方法

public long length()

该方法的作用是返回文件存储时占用的字节数。该数值获得的是文件的实际大小，而不是文件在存储时占用的空间数。

j、list 方法

public String[] list()

该方法的作用是返回当前文件夹下所有的文件名和文件夹名称。说明，该名称不是绝对路径。

k、listFiles 方法

public File[] listFiles()

该方法的作用是返回当前文件夹下所有的文件对象。

l、mkdir 方法

public boolean mkdir()

该方法的作用是创建当前文件文件夹，而不创建该路径中的其它文件夹。假设 **d** 盘下只有一个 **test** 文件夹，则创建 **d:\test\abc** 文件夹则成功，如果创建 **d:\a\b** 文件夹则创建失败，因为该路径中 **d:\a** 文件夹不存在。如果创建成功则返回 **true**，否则返回 **false**。

m、mkdirs 方法

public boolean mkdirs()

该方法的作用是创建文件夹，如果当前路径中包含的父目录不存在时，也会自动根据需要创建。

n、renameTo 方法

public boolean renameTo(File dest)

该方法的作用是修改文件名。在修改文件名时不能改变文件路径，如果该路径下已有该文件，则会修改失败。

o、setReadOnly 方法

public boolean setReadOnly()

该方法的作用是设置当前文件或文件夹为只读。

3、File 类基本示例

以上各方法实现的测试代码如下：

```
import java.io.File;
/**
 * File 类使用示例
 */
public class FileDemo {
    public static void main(String[] args) {
        //创建 File 对象
        File f1 = new File("d:\\test");
        File f2 = new File("1.txt");
        File f3 = new File("e:\\file.txt");
        File f4 = new File("d:\\","1.txt");
        //创建文件
        try{
            boolean b = f3.createNewFile();
        }catch(Exception e){
            e.printStackTrace();
        }
        //判断文件是否存在
        System.out.println(f4.exists());
        //获得文件的绝对路径
        System.out.println(f3.getAbsolutePath());
        //获得文件名
        System.out.println(f3.getName());
        //获得父路径
        System.out.println(f3.getParent());
        //判断是否是目录
        System.out.println(f1.isDirectory());
        //判断是否是文件
        System.out.println(f3.isFile());
        //获得文件长度
        System.out.println(f3.length());
        //获得当前文件夹下所有文件和文件夹名称
        String[] s = f1.list();
        for(int i = 0;i < s.length;i++){
            System.out.println(s[i]);
        }
        //获得文件对象
        File[] f5 = f1.listFiles();
        for(int i = 0;i < f5.length;i++){
            System.out.println(f5[i]);
        }
```

```
        //创建文件夹
        File f6 = new File("e:\\test\\abc");
        boolean b1 = f6.mkdir();
        System.out.println(b1);
        b1 = f6.mkdirs();
        System.out.println(b1);
        //修改文件名
        File f7 = new File("e:\\a.txt");
        boolean b2 = f3.renameTo(f7);
        System.out.println(b2);
        //设置文件为只读
        f7.setReadOnly();
    }
}
```

4、File 类综合示例

下面以两个示例演示 **File** 类的综合使用。第一个示例是显示某个文件夹下的所有文件和文件夹，原理是输出当前名称，然后判断当前 **File** 对象是文件还是文件夹，如果则获得该文件夹下的所有子文件和子文件夹，并递归调用该方法实现。第二个示例是删除某个文件夹下的所有文件和文件夹，原理是判断是否是文件，如果是文件则直接删除，如果是文件夹，则获得该文件夹下所有的子文件和子文件夹，然后递归调用该方法处理所有子文件和子文件夹，然后将空文件夹删除。则测试时谨慎使用第二个方法，以免删除自己有用的数据文件。示例代码如下：

```
        import java.io.File;
/**
 * 文件综合使用示例
 */
public class AdvanceFileDemo {
    public static void main(String[] args) {
        File f = new File("e:\\Book");
        printAllFile(f);
        File f1 = new File("e:\\test");
        deleteAll(f1);
    }

    /**
     * 打印 f 路径下所有的文件和文件夹
     * @param f 文件对象
     */
    public static void printAllFile(File f){
        //打印当前文件名
        System.out.println(f.getName());
        //是否是文件夹
```

```
        if(f.isDirectory()){
            //获得该文件夹下所有子文件和子文件夹
            File[] f1 = f.listFiles();
            //循环处理每个对象
            int len = f1.length;
            for(int i = 0;i < len;i++){
                //递归调用，处理每个文件对象
                printAllFile(f1[i]);
            }
        }
    }

    /**
     * 删除对象 f 下的所有文件和文件夹
     * @param f 文件路径
     */
    public static void deleteAll(File f){
        //文件
        if(f.isFile()){
            f.delete();
        }else{ //文件夹
            //获得当前文件夹下的所有子文件和子文件夹
            File f1[] = f.listFiles();
            //循环处理每个对象
            int len = f1.length;
            for(int i = 0;i < len;i++){
                //递归调用，处理每个文件对象
                deleteAll(f1[i]);
            }
            //删除当前文件夹
            f.delete();
        }
    }
}
```

关于 File 类的使用就介绍这么多，其它的方法和使用时需要注意的问题还需要多进行练习和实际使用。

Java 编程那些事儿 87——文件操作之读取文件

陈跃峰
出自：http://blog.csdn.net/mailbomb

11.3.1.3 读取文件

虽然前面介绍了流的概念，但是这个概念对于初学者来说，还是比较抽象的，下面以实际的读取文件为例子，介绍流的概念，以及输入流的基本使用。

按照前面介绍的知识，将文件中的数据读入程序，是将程序外部的数据传入程序中，应该使用输入流——**InputStream** 或 **Reader**。而由于读取的是特定的数据源——文件，则可以使用输入对应的子类 **FileInputStream** 或 **FileReader** 实现。

在实际书写代码时，需要首先熟悉读取文件在程序中实现的过程。在 **Java** 语言的 **IO** 编程中，读取文件是分两个步骤：**1**、将文件中的数据转换为流，**2**、读取流内部的数据。其中第一个步骤由系统完成，只需要创建对应的流对象即可，对象创建完成以后步骤 **1** 就完成了，第二个步骤使用输入流对象中的 **read** 方法即可实现了。

使用输入流进行编程时，代码一般分为 **3** 个部分：**1、创建流对象，2、读取流对象内部的数据，3、关闭流对象。**下面以读取文件的代码示例：

```
        import java.io.*;
/**
 *  使用 FileInputStream 读取文件
 */
public class ReadFile1 {
    public static void main(String[] args) {
        //声明流对象
        FileInputStream fis = null;
        try{
            //创建流对象
            fis = new FileInputStream("e:\\a.txt");
            //读取数据，并将读取到的数据存储到数组中
            byte[] data = new byte[1024]; //数据存储的数组
            int i = 0;   //当前下标
            //读取流中的第一个字节数据
            int n = fis.read();
            //依次读取后续的数据
            while(n != -1){ //未到达流的末尾
                //将有效数据存储到数组中
                data[i] = (byte)n;
                //下标增加
                i++;
                //读取下一个字节的数据
                n = fis.read();
            }

            //解析数据
            String s = new String(data,0,i);
            //输出字符串
```

```
            System.out.println(s);
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                //关闭流，释放资源
                fis.close();
            }catch(Exception e){}
        }
    }
}
```

在该示例代码中，首先创建一个 **FileInputStream** 类型的对象 **fis**：

```
fis = new FileInputStream("e:\\a.txt");
```

这样建立了一个连接到数据源 **e:\a.txt** 的流，并将该数据源中的数据转换为流对象 **fis**，以后程序读取数据源中的数据，只需要从流对象 **fis** 中读取即可。

读取流 **fis** 中的数据，需要使用 **read** 方法，该方法是从 **InputStream** 类中继承过来的方法，该方法的作用是每次读取流中的一个字节，如果需要读取流中的所有数据，需要使用循环读取，当到达流的末尾时，**read** 方法的返回值是**-1**。

在该示例中，首先读取流中的第一个字节：

```
int n = fis.read();
```

并将读取的值赋值给 **int** 值 **n**，如果流 **fis** 为空，则 **n** 的值是**-1**，否则 **n** 中的最后一个字节包含的时流 **fis** 中的第一个字节，该字节被读取以后，将被从流 **fis** 中删除。

然后循环读取流中的其它数据，如果读取到的数据不是**-1**，则将已经读取到的数据 **n** 强制转换为 **byte**，即取 **n** 中的有效数据——最后一个字节，并存储到数组 **data** 中，然后调用流对象 **fis** 中的 **read** 方法继续读取流中的下一个字节的数据。一直这样循环下去，直到读取到的数据是**-1**，也就是读取到流的末尾则循环结束。

这里的数组长度是 **1024**，所以要求流中的数据长度不能超过 **1024**，所以该示例代码在这里具有一定的局限性。如果流的数据个数比较多，则可以将 **1024** 扩大到合适的个数即可。

经过上面的循环以后，就可以将流中的数据依次存储到 **data** 数组中，存储到 **data** 数组中有效数据的个数是 **i** 个，即循环次数。

其实截至到这里，**IO** 操作中的读取数据已经完成，然后再按照数据源中的数据格式，这里是文件的格式，解析读取出的 **byte** 数组即可。

该示例代码中的解析，只是将从流对象中读取到的有效的数据，也就是 **data** 数组中的前 **n** 个数据，转换为字符串，然后进行输出。

在该示例代码中，只是在 **catch** 语句中输出异常的信息，便于代码的调试，在实际的程序中，需要根据情况进行一定的逻辑处理，例如给出提示信息等。

最后在 **finally** 语句块中，关闭流对象 **fis**，释放流对象占用的资源，关闭数据源，实现流操作的结束工作。

上面详细介绍了读取文件的过程，其实在实际读取流数据时，还可以使用其它的 **read** 方法，下面的示例代码是使用另外一个 **read** 方法实现读取的代码：

```
        import java.io.FileInputStream;
/**
 * 使用 FileInputStream 读取文件
 */
public class ReadFile2 {
    public static void main(String[] args) {
        //声明流对象
        FileInputStream fis = null;
        try{
            //创建流对象
            fis = new FileInputStream("e:\\a.txt");
            //读取数据，并将读取到的数据存储到数组中
            byte[] data = new byte[1024]; //数据存储的数组
            int i = fis.read(data);

            //解析数据
            String s = new String(data,0,i);
            //输出字符串
            System.out.println(s);
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
                //关闭流，释放资源
                fis.close();
            }catch(Exception e){}
        }
    }
}
```

该示例代码中，只使用一行代码：

```
        int i = fis.read(data);
```

就实现了将流对象 **fis** 中的数据读取到字节数组 **data** 中。该行代码的作用是将 **fis** 流中的数据读取出来，并依次存储到数组 **data** 中，返回值为实际读取的有效数据的个数。

使用该中方式在进行读取时，可以简化读取的代码。

当然，在读取文件时，也可以使用 **Reader** 类的子类 **FileReader** 进行实现，在编写代码时，只需要将上面示例代码中的 **byte** 数组替换成 **char** 数组即可。

使用 **FileReader** 读取文件时，是按照 **char** 为单位进行读取的，所以更适合于文本文件的读取，而对于二进制文件或自定义格式的文件来说，还是使用 **FileInputStream** 进行读取，方便对于读取到的数据进行解析和操作。

读取其它数据源的操作和读取文件类似，最大的区别在于建立流对象时选择的类不同，而流对象一旦建立，则基本的读取方法是一样，如果只使用最基本的 **read**

方法进行读取，则使用基本上是一致的。这也是 **IO** 类设计的初衷，使得对于流对象的操作保持一致，简化 **IO** 类使用的难度。