# 编程语言的发展史和发展主线

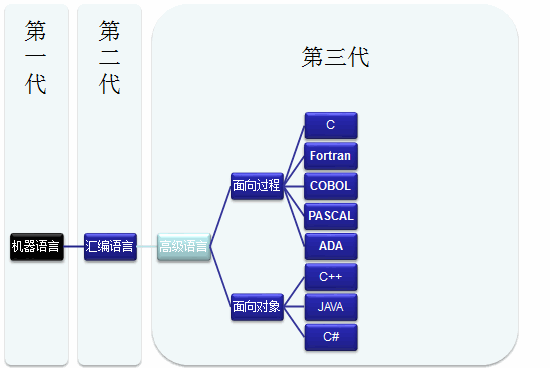
     计算机语言如果你将它当做一个产品，就像我们平时用的电视机、剃须刀、电脑、手机等， 他的发展也是有规律的。

任何一个产品的发展规律都是：向着人更加容易使用、功能越来越强大的方向发展。

那么，我们的计算机语言的发展也是这样，向着人更加容易使用，即更加容易写代码、更加容易实现现实逻辑的方向发展。套用奥林匹克的口号“更高、更快、更强”。那么计算机发展的主线可以总结为四个字：“更易、更强”。

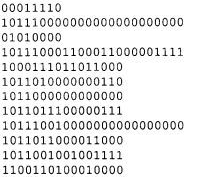
这个”更易、更强”的主线，也适用于任何的产品。如果，读者朋友以后要开公司创造自己的产品或者服务(服务也是一种产品)。你的产品是否符合”更易、更强”这个规律，也决定了产品能否取得成功，能否击败现有市场上同类型的产品！当然，也直接决定了你能否发财，能否养得起老婆孩子啦！

计算机语言一般意义上划分为：



## 第一代语言：机器语言

    我们都知道计算机的基本计算方式都是基于二进制的方式。如下就是一段典型的机器语言代码：

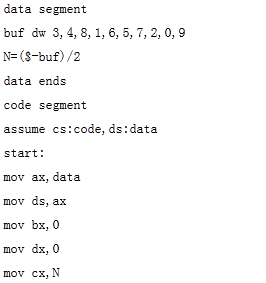


这种代码是直接输入给计算机使用的，不经过任何的转换！当然，在计算机刚发明的时候，计算机也是稀罕物，人却有很多。物以稀为贵嘛，人就只能迁就计算机，采用了这样一种计算机可以直接识别的代码。当然，这种代码计算机非常受用，不需要做任何转换，直接执行，太爽了！

可是，人就不爽了！这样的代码，又有几个人能写，又有几个人能读懂！如果用来解决最简单的计算问题，凑合着用，没问题。如果要解决复杂问题，需要写几千行、几万行这样的代码，估计谁写谁疯。如果现在我们仍然使用第一代语言，我相信所有程序员最终只能将精神病院作为最后的归宿了。

## 第二代语言：汇编语言

    为了解决机器语言无法读懂的问题。很自然，我们就想到能不能使用人可以读懂的单词来代替一些二进制指令和数据，方便我们人去读、编写代码。于是，汇编语言诞生了。他使用了大量的助记符来代替二进制指令，方便我们人去编写代码。如下是一段典型的汇编语言代码：



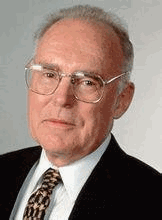
显然，这样的代码，要比起机器语言来，对于我们程序员来说，爽了很多，至少不需要再去背诵那些难记的二进制指令了！于是，有了更好的武器，意味着我们可以解决更复杂的代码，写个几百行、几千行汇编语言代码也成为不那么复杂的事情。

【注】现在仍然有很多领域在使用汇编语言，比如：操作工业机器人、单片机编程、甚至某些计算机病毒的编写等。

## 第三代语言

    随着晶体管、集成电路的出现，计算机的功耗越来越小、速度越来越快，计算机很快深入到了各行各业。有个很著名的定律叫做摩尔定律，他深刻的展示了计算机发展的规律：

“当价格不变时，[集成电路](http://baike.baidu.com/view/1355.htm)上可容纳的[晶体管](http://baike.baidu.com/view/30363.htm)数目，约每隔18个月便会增加一倍，性能也将提升一倍。换言之，每一美元所能买到的[电脑](http://baike.baidu.com/view/2358.htm)性能，将每隔18个月翻两倍以上”

         英特尔公司创始人---戈登·摩尔

摩尔定律在我们身边有着实实在在的体验，我们现在手边用的电脑、手机比起十年前强大的很多倍，但是价格实际上确越来越低！十年前，北京房价是3000一平的地方，现在涨到了4万，涨了十多倍。十年前，一个笔记本需要两万人民币，现在只需要几千元，但是性能确更好。我们多么希望，房价也能遵循摩尔定律啊！

这样，随着计算机深入更多的企业、单位、甚至更人。我们很自然的就希望使用计算机解决我们身边的问题。这种需求越来越明确，越来越复杂。汇编语言在体系结构设计上的缺陷，根本无法胜任解决复杂问题。使用汇编语言写几万行代码，几十万行代码，那真的可以称之为”噩梦”.

于是，各种各样，结构化设计更加优良的第三代语言诞生了。例如：C语言、Fortran、cobol、pascal、ada、c++、java、c#。

第三代语言数量众多，大体上分为：面向过程和面向对象两大类。C语言是典型的面向过程的语言。C++、JAVA是典型的面向对象的语言。

大家在学习过程中，不要有这样的误解。学了面向过程会对学习面向对象有不好的影响，影响思维。那纯属扯淡，只能说明你面向过程根本没入门! 事实上，面向过程和面向对象是相辅相成的。 在面向对象编程中，也离不开面向过程。关于，面向过程和面向对象的区别，我们将在面向对象章节中详细说明。

### C语言

第三代语言中，C语言是最重要的！可以称之为“现代语言的鼻祖”。其他很多语言都跟C语言有千丝万缕的联系！

C语言是一种[计算机程序设计语言](http://baike.baidu.com/view/2561555.htm)，它既具有[高级语言](http://baike.baidu.com/view/14900.htm)的特点，又具有[汇编语言](http://baike.baidu.com/view/49.htm)的特点。它由美国贝尔研究所的D.M.Ritchie于1972年推出，1978年后，C语言已先后被移植到大、中、小及微型机上，它可以作为工作系统设计语言，编写系统[应用程序](http://baike.baidu.com/view/330120.htm)，也可以作为应用程序设计语言，编写不依赖计算机硬件的应用程序。它的应用范围广泛，具备很强的数据处理能力，不仅仅是在软件开发上，而且各类科研都需要用到C语言，适于编写系统软件，三维，二维图形和动画，具体应用比如单片机以及[嵌入式系统开发](http://baike.baidu.com/view/657641.htm)。

### C++语言

显然就是C语言的升级版，看名字我们就知道了，就是C语言面向对象的版本！在C基础上，一九八三年又由[贝尔实验室](http://baike.baidu.com/view/37115.htm)的Bjarne Strou-strup推出了C++。 C++进一步扩充和完善了C语言，成为一种面向 对象的程序设计语言。

### JAVA语言

SUN公司的JAVA也是以C/C++为原型开发的，甚至有人将JAVA称之为“C++-”(C加加减).看名字就知道是将C++中某些不太好用的地方砍掉！

### Object-C语言

现在IOS和Mac开发中的主流语言。 根据C语言所衍生出来的语言，继承了C语言的特性，是扩充C的[面向对象](http://baike.baidu.com/view/125370.htm)[编程语言](http://baike.baidu.com/view/552871.htm)。它主要使用于[Mac OS X](http://baike.baidu.com/view/24778.htm)和GNUstep这两个使用OpenStep标准的系统，而在NeXTSTEP和OpenStep中它更是基本语言。

### C#语言

微软发明的一种希望替代JAVA的语言。盖茨曾经非常认可JAVA,，“JAVA是我见过最好的语言”，因此，微软跟JAVA曾经有一段蜜月期，合作非常好。但是后来由于版权纠纷，闹掰了。盖茨一怒之下，就要搞一个比JAVA更好的语言。因此，我们在C#中可以看到大量的JAVA中有的特性，非常明显的”山寨”，当然也可以称为”微创新”。创新都是源自”山寨”。但是，微软山寨了JAVA的细节，但是没有学到JAVA的精华，即跨平台。当然，这跟微软本身的公司特性有关，微软名字叫做：Microsoft，但是我们知道他的核心产品是windows，所以他希望将他所有的产品都跟windows绑定再一起，这样C#就失去了JAVA的跨平台的优势。



### 附：贝尔实验室(IT行业的圣地)

[晶体管](http://baike.baidu.com/view/30363.htm)、[激光器](http://baike.baidu.com/view/131593.htm)、[太阳能](http://baike.baidu.com/view/21294.htm)电池、[发光二极管](http://baike.baidu.com/view/84213.htm)、数字[交换机](http://baike.baidu.com/view/1077.htm)、[通信卫星](http://baike.baidu.com/view/19448.htm)、电子数字计算机、蜂窝[移动通信设备](http://baike.baidu.com/view/3067174.htm)、长途电视传送、仿真语言、[有声电影](http://baike.baidu.com/view/60070.htm)、[立体声录音](http://baike.baidu.com/view/252214.htm)，以及[通信网](http://baike.baidu.com/view/834903.htm)、C语言、C++语言等许多重大发明的诞生地。

详情请参考百度百科：<http://baike.baidu.com/view/37115.htm>

### 附：历史上最古老的IT公司IBM

    IBM（International Business Machines Corporation），[国际商业机器公司](http://baike.baidu.com/view/324670.htm)，或万国商业机器公司的简称。总公司在[纽约州](http://baike.baidu.com/view/190078.htm)阿蒙克市，1911年创立于[美国](http://baike.baidu.com/view/2398.htm)，是全球最大的信息技术和业务[解决方案](http://baike.baidu.com/view/1038216.htm)公司。



 详情请参考百度百科：http://baike.baidu.com/view/1937.htm

### 附：微软的前景堪忧

        微软的核心产品是windows. 微软的成长基本上跟个人电脑PC的普及和发展是同步的。但是，随着手机、平板电脑的发展，我们发现个人电脑的地位明显的在下降，这也意味着 windows的地位在显著的下降。“成也PC，败也PC”. 如果，微软不在手机和平板电脑上发力，将不可避免的沦为”第二个诺基亚”。

### 附：山寨和创新

        任何一种创新都不是凭空天上来的， 都是基于以前的旧事物发展的。都是一种”山寨”. 我们看到计算机语言的创新史其实就是一部山寨史。当然，如果把别人的跌出拿来只换个名字那叫耍流氓、不要脸，是真的”山寨”。 如果把别人的东西拿来进行改进，做的更好,那才就叫”创新”。

### 附：到底应该学哪门语言

       很多童鞋会有疑问，到底应该学什么语言呢? 纠结了很长时间，仍然没有下手。我的说法是，JUST DO IT!  学了再说。 其实，天下文章一大抄，语言之间固然有差异，但是隐藏在语言背后的编程思维方式没有差别。如果你精通了一门语言再去学其他语言，很快就可以上手。

纠结是个坏毛病。小时候，老是纠结于以后做爱因斯坦呢还是盖茨呢?纠结了很多年。长大了，才发现真的是想太多了！

# JAVA语言的发展史

## SUN公司介绍

       SUN公司全称是：Stanford University Network.  SUN公司是从斯坦福大学孵化器出的高科技公司。创立后，曾经市值名列IT公司第一位。在2001年时，市值2100亿美元，远超过google公司、IBM公司。SUN公司不仅打败了包括 IBM 在内的全部工作站（Work Station）和小型机（Mini Computer）公司，而且依靠它的 Solaris（一种Unix）和风靡世界的 Java 程序语言，成为在[操作系统](http://baike.baidu.com/view/880.htm)上最有可能挑战[微软](http://baike.baidu.com/view/2353.htm)的公司。太阳公司不乏能人，它不仅为 Google 培养了 CEO[埃里克.施密特](http://baike.baidu.com/view/1829276.htm)和首任工程部副总裁韦恩.罗森（Wayne Rosen），并且在一定程度上奠定了今天 Google 工程部门的基础。



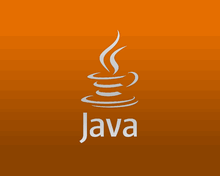
当然，今天的SUN公司已经一落千丈，并且被oracle公司收购。sun公司从成立到市值2000亿美元经过了20年，但是走下坡路只用了1年时间！这就是IT行业的残酷性！

## JAVA的诞生和核心特点

       上世纪90年代初，SUN公司的James Gosling(高司令，^\_^)领导的Green项目，着力打造一个分布式的、可以在消费类电子产品（如电视机顶盒）使用的系统架构。

这种架构中，要求写的代码可以运行在各种消费类电子产品中，即要求代码的中立性、跨平台性。

最初，Green项目使用的C++，但是在研发过程中发现C++并不能很好的适应新的需求。于是，这些高人对C++编译器做了改写，仍然不太理想，不太符合跨平台的特性。 牛人有牛人特有的牛脾气，于是，大家决定重起炉灶，开始研发一种新的语言，即JAVA.



经过一年半的研发，整个产品研发完成。但是，很遗憾，消费类电子产品并没有高速的发展起来，JAVA进入蛰伏期。

【注】跨平台特性，始终是JAVA的核心优势！

## JAVA的飞速发展

      无心插柳柳成荫！90年代中末期，有一个改变世界的大事出现了：互联网的爆发式增长。在传统产业，例如钢铁产业要想取得产量翻番，那需要很多年的时间。但是，在互联网行业都是呈指数级增长，爆炸式的增长。google可以在短短几年就增长到上千亿美元市值。

    互联网将各种各样的电脑连到了一起，这时，我们发现电 脑的操作系统各种不同。因此，急需一种跨平台的语言，通过它编写的程序可以在各种平台上运行。JAVA正好符合这种需求。使用JAVA编写的applet程序给了90年代的人们极大的震撼，于是JAVA迎来了他的黄金时代！

JAVA诞生后，经历了如下版本的发展变化：

•    1991年，Sun公司的Green项目，Oak

•    1995年，推出Java测试版

•    1996年，JDK1.0

•    1997年，JDK1.1

•    1998年，JDK1.2，大大改进了早期版本的缺陷，是一个革命性的版本，更名为Java2

•    2004年，J2SE 5.0 (1.5.0)  Tiger老虎

•    2006年，J2SE 6.0 (1.6.0)  Mustang野马

•    2011年，JAVASE7.0

## JAVA的特性和优势

简单性

        就是c++语法的纯净版。没有头文件，指针运算，结构，联合，操作符重载，虚基类等等。由于语法基于c，因此学习起来完全不费力。

面向对象

        面向对象是一种程序设计技术，他将重点放在数据（即对象）和对象之间的接口上。

可移植性(跨平台性)

       这是JAVA的一个重要的优势。JAVA代码或者说字节码、二进制码可以跨平台的移植，而不用管具体的操作系统和硬件环境。JAVA本身就是面向网络的，只有在网络环境中才能显示出他的优势，比如：现在我有一个网络环境，要让我的笔记本和手机交互，笔记本环境是windows发出一个让手机定时录音的JAVA代码，手机只有简单Linux内核操作系统，照样可以完美的执行这个代码。

“一次编写，随处运行”：“write once, run anywhere”

       JAVA在设计时就很注重移植和跨平台性。比如：JAVA的Int永远都是32位。不像c++可能是16，32，可能是根据编译器厂商规定的变化。这样的话程序的移植就会非常麻烦。

高性能

       JIT(JUST IN TIME)即时编译。将一些“热点”字节码编译成本地机器码，并将结果缓存起来，在需要的时候重新调用。这样的话，使JAVA程序的执行效率大大提高，某些代码甚至接待c++的效率。随着这种技术的一天天完善，也许有一天会超越编译代码的效率。

分布式

       JAVA是为internet的分布式环境设计的，因为他能够处理tcp/ip协议。事实上，通过url访问一个网络资源和访问本地文件是一样简单的。Java还支持远程方法调用(RMI,remote method Invocation)，使程序能够通过网络调用方法。

动态性

       就是在需要时将某些代码添加到正在运行的程序中。反射机制。当需要把某些代码添加到正在运行的程序中时，动态性是一个非常重要的特性。Java的动态特性是其面向对象设计方法的扩展。它允许程序动态地装入运行过程中所需要的类，这是C++语言进行面向对象程序设计所无法实现的

多线程

       多线程的使用可以带来更好的交互响应和实时行为。多线程的简单性是Java成为主流服务器端开发语言的主要原因之一。

安全性

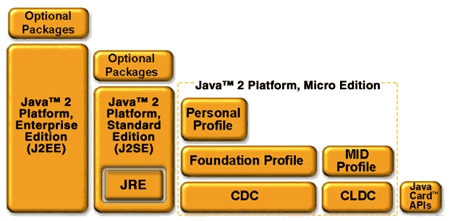
       Java适合于网络/分布式环境，为了达到这个目标，在安全性方面投入了很大的精力，使Java可以构建防病毒，防篡改的系统。

健壮性

       Java是一种健壮的语言，吸收了C/C++ 语言的优点，但去掉了其影响程序健壮性的部分（如：指针、内存的申请与释放等）。Java程序不可能造成计算机崩溃。Java系统仔细检测对内存的每次访问，确认它是合法的，而且不致引起任何问题。不过，即使Java程序也可能有错误。如果出现某种出乎意料之事，程序也不会崩溃，而是把该例外抛弃。再通过异常处理机制，程序就会发现这类例外，并加以处理。

## JAVA三大版本的区别

     SUN公司推出了JAVAEE、JAVASE、JAVAME适应各种不同平台上的开发和运行工作。我们以后的学习会围绕他们开展起来。



•    J2EE （JAVAEE）

–    [Java](http://www.javadby.com/) 2 Enterprise Edition：

–    定位在服务器端的应用

•    J2SE  (JAVASE)

–    [Java](http://www.javadby.com/) 2  Standard  Edition

–    定位在个人计算机上的应用

•    J2ME  (JAVAME)

–    [Java](http://www.javadby.com/)  2  Micro  Edition

–    定位在消费性电子产品的应用上。

## JAVA语言为什么能够成功

        一个产品的成功和一个人的成功是一个道理。除了自身具备过硬的优势外，还需要那么一点点狗屎运，甚至狗屎运非常关键。就是我们所讲的天时地利人和。JAVA的成功除了自己具备跨平台特性外，更重要的是正好踩中了互联网发展的节奏。微软的成功除了windows好用外，也是正好踩中了个人电脑发展的节奏。所以，我们除了自己努力外，也需要出去走走努力才踩一坨狗屎才行！

### 附：斯坦福大学的房产生意经

       斯坦福大学依靠自身庞大的校区，创建了斯坦福科技园(硅谷的前身)，初期采用房租免费等方式，鼓励自己的毕业生在科技园创业、鼓励社会企业入驻。由于免费的方式再加上其他优惠的政策，吸引了大量的创业者加入进来。最终大家添柴加火，造就了今天的硅谷。这才是最高明的房地产商啊！相比我国的房产商杀鸡取卵式的发展，无语！北京五环外房价都3万多了，一个精英人才就算是年薪20万，买个100平的房子，需要不吃不喝15年！如果要吃喝的话，基本一辈子都在为这100平的房子打工！赤裸裸的剥削至此，还有什么话说！

### 附：跟斯坦福大学有关的IT公司

牛逼的大学，不可思议的大学。看看这个大学的毕业生和教授创立的公司：

1.    惠普公司

2.    SUN公司

3.    思科公司

4.    雅虎公司

5.    Google公司

6.    英特尔公司

7.    NVidia(最大的显卡公司)

8.    耐克(卖鞋的，斯坦福除了IT还有耐克)

斯坦福大学毕业生创立的小于10亿美金的中小公司就不说了。这八个大公司加起来的市值，估计远远超过大部分国家的鸡的屁了吧?

# JDK是什么?JRE是什么?JDK和JRE的区别?

Java Runtime Environment (JRE) 包含：

Java虚拟机、库函数、运行Java应用程序和Applet所必须文件

 Java 2 SDK (Development Kit)包含：

JRE的超集，包含编译器和调试器等用于程序开发的文件

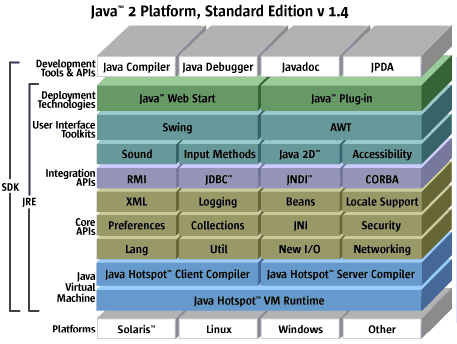
Java运行环境的三项主要功能：

    n  加载代码：由class loader 完成；

    n  校验代码：由bytecode verifier 完成；

    n  执行代码：由 runtime interpreter完成。

JDK 和JRE:



sdk（也就是jdk）是jre的超集，是在jre的基础上增加了编译器及其他一些开发工具。

jre就是java运行时环境，包括了jvm和其它一些java核心api,任何一台电脑，只有安装了jre才可以运行java程序.

总结：

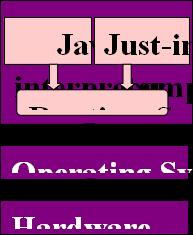
1.  如果只是要运行JAVA程序，之需要JRE就可以。 JRE通常非常小，也包含了JVM.

2.  如果要开发JAVA程序，就需要安装JDK。

# JVM是什么?(JAVA VIRTUAL MACHINE)

 JVM是一种规范，可以使用软件来实现，也可以使用硬件来实现，就是一个虚拟的用于执行bytecodes字节码的计算机。他也定义了指令集、寄存器集、结构栈、垃圾收集堆、内存区域。

JVM负责将java字节码解释运行，边解释边运行，这样，速度就会受到一定的影响。JAVA提供了另一种解释运行的方法JIT（just in time），可以一次解释完，再运行特定平台上的机器码，高级的JIT可以只能分析热点代码，并将这些代码转成本地机器码，并将结果缓存起来，下次直接从内存中调用，这样就大大提高了执行JAVA代码的效率。这样就实现了跨平台、可移植的功能。



1.  JVM是指在一台计算机上由软件或硬件模拟的计算机；它类似一个小巧而高效的CPU。

2.  byte-code代码是与平台无关的是虚拟机的机器指令。

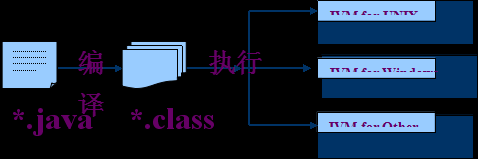
3.  java字节代码运行的两种方式:

   1)方式interpreter(解释)

   2)Just-in-time(即时编译):有代码生成器将字节代码转换成本机的机器代码,然后可以以较高速度执行。

   JAVA的跨平台实现的核心是不同平台使用不同的虚拟机

不同的操作系统有不同的虚拟机。Java 虚拟机机制屏蔽了底层运行平台的差别，实现了“一次编译，随处运行”。



# JAVA语言的应用范围

1.  桌面应用编程

2.  WEB客户端编程

3.  WEB服务器编程

4.  手机编程(J2ME或者现在的Android系统上开发应用程序)

5.  机器人编程

# JAVA开发环境的安装和配置详解

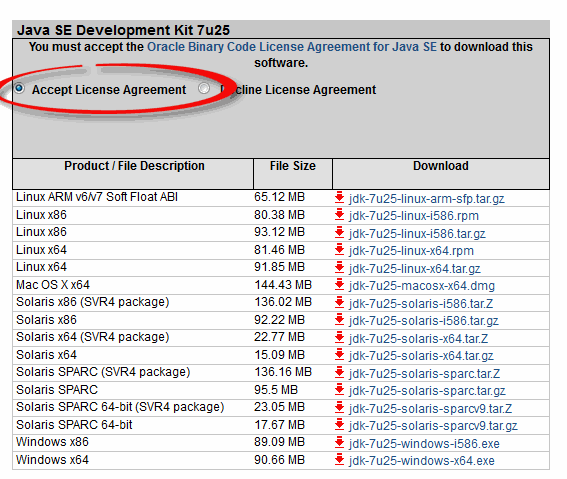
## 下载JDK

1.  在哪里下载JDK, 网址：

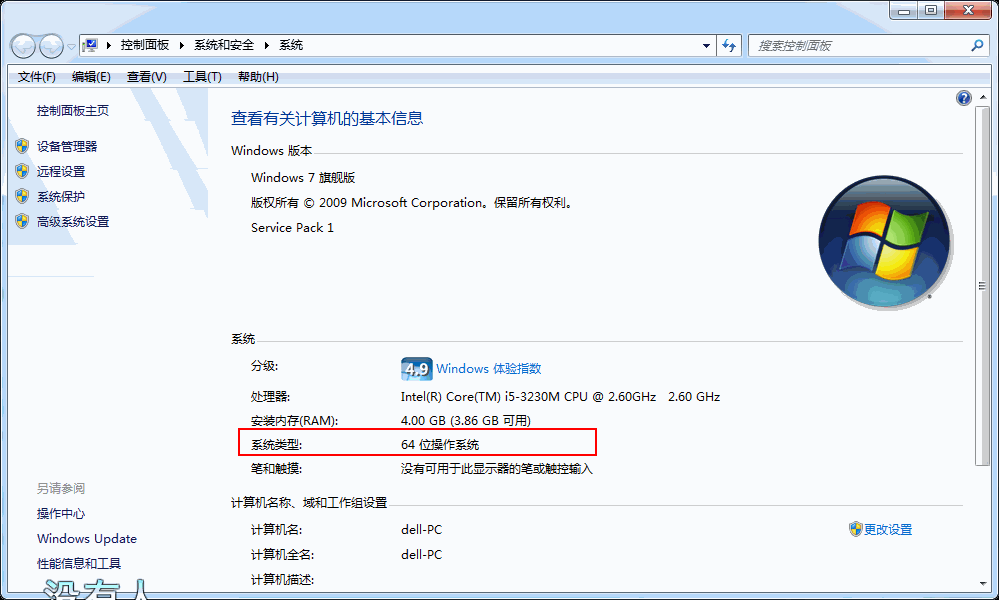
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

2.  JDK7的下载地址：

<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>



根据自己的系统下载不同的版本。如果系统是64位则可以下载64位系统。如何查看自己的操作系统到底是32位还是64位呢?，右键单击桌面的计算机，出现如下图所示信息：



## 安装JDK



其他步骤不解释，跟普通软件安装一样，点击下一步就OK！



## 环境变量配置

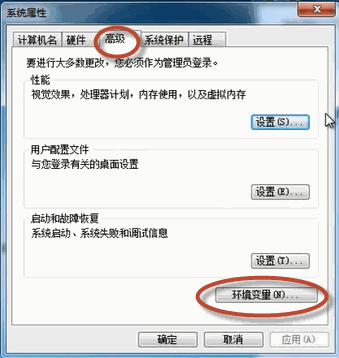
我们以win7系统为例，xp下的配置基本一致。

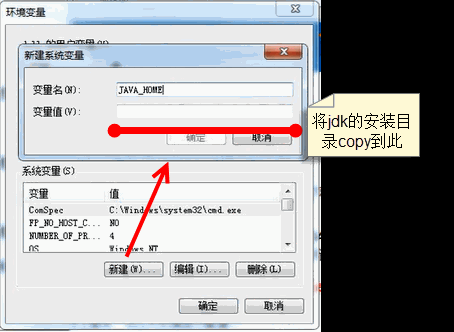
1.  右键单击计算机，点击属性：



2.  配置JAVA\_HOME

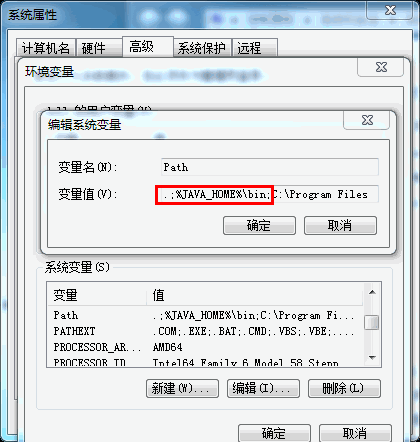






注意：不要在目录值后面再添加分号，逗号之类的符号！

3.  配置path

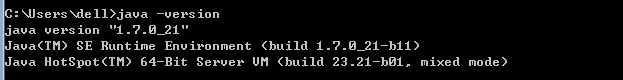


修改系统环境变量Path，在最前面追加% JAVA\_HOME %\bin，并以；和原路径分隔。再增加：“.;”。这是表示当前目录的意思。

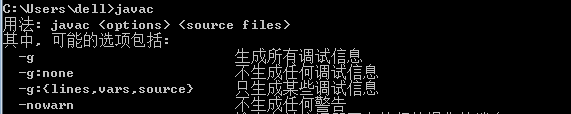
classpath问题：如果使用jdk1.5以上就不需要配置这个环境变量！JRE会自动搜索当前路径下的类文件及相关jar文件。

## 测试安装环境是否安装成功

运行àcmdà输入命令：java  –version: 出现如下结果，则通过测试。



输入javac，出现如下结果，则通过测试。



# 

# 第一个JAVA程序的编写和运行

1.使用记事本编辑：

public class Welcome {

    public static void main(String[] args) {

        System.out.println("我是高淇,欢迎大家加入www.bjsxt.cn社区,共同学习java！");

        }

}

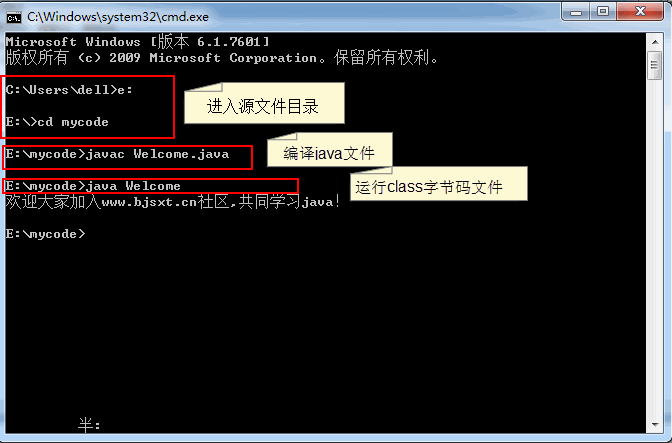
2.保存为：d://mycode/Welcome.java

3.进入控制台，进入：d://mycode/目录。

4.javac Welcome.java

5.java Welcome

6.出现运行结果：



# JAVA程序运行机制分析

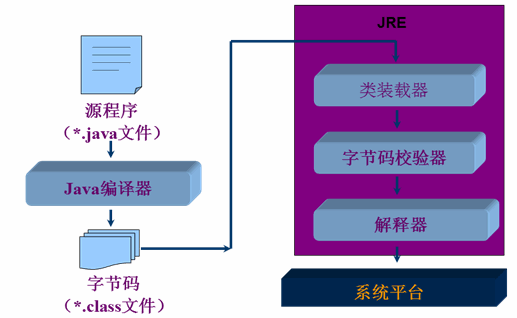
计算机的高级编程语言类型: 编译型 ，解释型.  Java 语言是两种类型的结合；

第一步：编译

利用编译器（javac）将源程序编译成字节码à 字节码文件名：源文件名.class

第二部：运行

利用虚拟机（解释器，java）解释执行class字节码文件。



# 第一个JAVA程序的总结和提升

 n  Java对大小写敏感，如果出现了大小写拼写错误，程序无法运行。

    n  关键字public被称作访问修饰符（access modifier）,用于控制程序的其它部分对这段代码的访问级别。

    n  关键字class 表明Java 程序中的全部内容都包含在类中，Java是一种纯面向对象的语言。

    n  一个源文件中至多只能有一个public的类声明，其它类的个数不限，如果源文件中包含一个public 类，源文件名必须和它中定义的public的类名相同，且以“java”为扩展名。

    n  一个源文件可以有多个class

    n  正确编译后的源文件，会得到相应的字节码文件，编译器为每个类生成独立的字节码文件，且将字节码文件自动命名为类的名字且以“class”为扩展名。

    n  main方法是Java应用程序的入口方法，它有固定的书写格式：

    n  public static void main(String[]  args) {…}

    n  在Java中，用花括号划分程序的各个部分，任何方法的代码都必须以“{”开始，以“}”结束， 由于编译器忽略空格，所以花括号风格不受限制。

    n  Java中每个语句必须以分号结束，回车不是语句的结束标志，所以一个语句可以跨多行。

编程风格:

    1.  注意缩进!

    2.  成对编程!

# 常用Java开发工具

文本编辑器

    n  UltraEdit

    n  EditPlus

    n  notepad++

集成开发环境（IDE：Integrated Development Environment）

    n  JBuilder （http://www.borland.com）

    n  Eclipse （[http://www.eclipse.org](http://www.eclipse.org/)）以后工作和学习的重点！

    n  NetBeans（http://java.sun.com）

    n  WSAD （http://www.ibm.com）

# 常用dos命令

1.  cd   进入一个目录

2.  cd ..

3.  dir

4.  上下键：查找敲过的命令

5.  Tab键：自动补齐命令

# 注释

        n  注释就是程序员为读者作的说明，是提高程序可读性的一种手段

        n  在Java中，有2种表示注释的方法

            q   // 单行注释----注释内容从//到本行结尾

        q  /\*

             单行或多行注释

       \*/

     n  /\* \*/注释不能嵌套

n  Java中的注释不会出现在字节码文件中。即JAVA编译器编译时会跳过注释语句。

|  |
| --- |
| public class Welcome{      public static void main(String[] /\*a\*/rgs){    //args是arguments的缩写          System.out.println("de");  }  }  class  Man{  }  class Car{  }  class Computer{  }  /\*  我是 的高淇  这是我的代码  欢迎大家来www.bjsxt.cn学习讨论，下载新资料  \*/ |

【注】文档注释，后面会有专门的课程进行讲授。如何写文档注释，如何生成java的api文档。

# 标识符

 用作给变量、类和方法命名。注意：

表示类名的标识符用大写字母开始。如：Man, GoodMan

表示方法和变量的标识符用小写字母开始，后面的描述性词以大写开始。eat(),eatFood()

Java 标识符有如下命名规则：

        n  标识符必须以字母、下划线\_ 、美元符$开头。

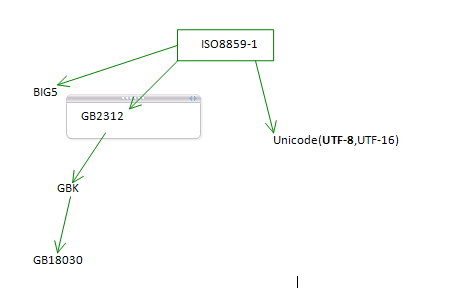
        n  标识符其它部分可以是字母、下划线“\_”、美元符“$”和数字的任意组合。

        n  Java 标识符大小写敏感，且长度无限制。

        n  不可以是Java的关键字。

        JAVA不采用通常语言使用的ASCII字符集，而是采用unicode这样的标准的国际字符集。因此，这里的字母的含义：英文、汉字等等。(不建议大家使用汉字来定义标识符！)

各种字符集的关系？



合法的标识符：

        int a = 3;

        int \_123 = 3;

        int $12aa = 3;

        int 变量1 = 55;

不合法的标识符：

        int 1a = 3;   //不能用数字开头

        int a# = 3;   //不能包含#这样的特殊字符

        int int = 3;  //不能使用关键字

# JAVA中的关键字/保留字

        Java关键字是Java语言保留供内部使用的,如class用于定义类。 关键字也可以称为保留字,它们的意思是一样的。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Abstract | Assert | boolean | break | byte | case |
| Catch | char(character) | class | const | continue | default |
| Do | Double | else | extends | final | finally |
| float | For | goto | if | implements | import |
| instanceof | Int | interface | long | native | new |
| Null | Package | private | protected | public | return |
| Short | Static | strictfp | super | switch | synchronized |
| This | Throw | throws | transient | try | void |
| Volatile | While |  |  |  |  |

不能使用关键字作为变量名或方法名。

代码：

|  |
| --- |
| /\*  \*测试标识符的写法  \*/  public class Welcome{          public static void main(String[] args){              int  $abc = 3;              int $ = 5;              int \_123=5;              //int 123abc = 6;    //标识符不能以数字开头              //int  abc# = 3;    //标识符不能包含除了字母、数字、下划线、$之外的其他字符              //int  class = 3;              int   = 10;   //java内部采用了Unicode字符集，universal。          }  } |

# 

# 基本数据类(primitive data type)

        .  Java是一种强类型语言，每个变量都必须声明其类型。

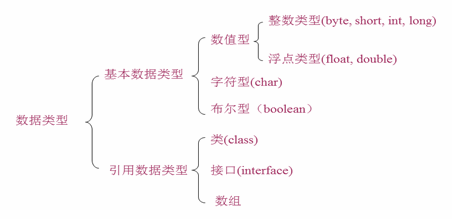
        .  Java的数据类型分为两大类：基本类型（primitive type）和引用类型    （reference type）

        .  Java中定义了3类8种基本数据类型

        .  逻辑型－boolean

        q  文本型－ char

        q  数值型－ byte,  short,  int,  long, float,  double



注：引用数据类型的大小统一为4个字节，记录的是其引用对象的地址！

# 整型变量

    整型用于表示没有小数部分的数值，它允许是负数。

    整型的范围与运行Java代码的机器无关，这正是Java程序具有很强移植能力的原因之一。于此相反，C和C++程序需要针对不同的处理器选择最有效的整形。

|  |  |  |
| --- | --- | --- |
| 类型 | 占用存储空间 | 表数范围 |
| Byte | 1字节 | -128~127 |
| Short | 2字节 | -215 ~ 215-1 （-32768~32767） |
| Int | 4字节 | -231 ~ 231-1  (-2147483648~2147483647) 约21亿 |
| Long | 8字节 | -263 ~ 263-1 |

ü  Java 语言整型常数的三种表示形式：

    q  十进制整数，如：99, -500, 0。

    q  八进制整数，要求以 0 开头，如：015。

    q  十六进制数，要求 0x 或 0X 开头，如：0x15 。

ü  Java语言的整型常数默认为int型，声明long型常量可以后加‘ l ’或‘ L ’ ，如：

        long a = 55555555;  //不出错，在Int表示的范围内(21亿内)。

        long b = 55555555555;//不加l出错，已经超过int表示的范围。报错：

        The literal 55555555555 of type int is out of range

        ü  Java中没有无符号类型

# 代码

|  |
| --- |
| //测试整数类型：byte,short,int,long。以及进制之间的转换问题  public class TestDataType {      public static void main(String[] args){          int a = 10;          int a2 = 010;          int a3 = 0xf;          // byte b = 200;          // System.out.println(b);          System.out.println(a);          System.out.println(a2);          System.out.println(a3);          System.out.println(Integer.toBinaryString(a));          System.out.println(Integer.toOctalString(a));          System.out.println(Integer.toHexString(a));          t a5 = 10;          long a6 = 200;          byte b2 = 100;   //如果数据的大小没有超过byte/short/char的表述范围，则可以自动转型。          long  a7 = 11123213232L;          long l = 3;          long l2 = l+3;    //L问题          }  } |

# 浮点型

|  |  |  |
| --- | --- | --- |
| **类型** | **占用存储空间** | **表数范围** |
| **Float** | **4字节** | **-3.403E38~3.403E38** |
| **Double** | **8字节** | **-1.798E308~1.798E308** |

* float类型又被称作单精度类型，**尾数可以精确到7位有效数字**，在很多情况下，float类型的精度很难满足需求。
* double表示这种类型的数值精度是float类型的两倍，又被称作双精度，绝大部分应用程序都采用double类型。
* Java 浮点类型常量有两种表示形式
  + 十进制数形式，例如:

3.14 314.0 0.314

* + 科学记数法形式，如

314e2 314E2 314E-2

**double** f = 314e2; //314\*10^2-->31400.0

**double** f2 = 314e-2; //314\*10^(-2)-->3.14

* Float类型的数值有一个后缀F/f ，**没有后缀F/f的浮点数值默认为double类型**。也可以在浮点数值后添加后缀D/d, 以明确其为double类型：

|  |
| --- |
| 浮点类型float, double的数据不适合在不容许舍入误差的金融计算领域。         如果需要进行不产生舍入误差的**精确数字计算，需要使用BigDecimal类**。  注：           主要理由：**由于字长有限，浮点数能够精确表示的数是有限的，因而也是离散的。浮点数一般都存在舍入误差，很多数字无法精确表示(例如0.1)**，其结果只能是接近， 但不等于。  二进制浮点数不能精确的表示0.1,0.01,0.001这样10的负次幂。并不是所有的小数都能可以精确的用二进制浮点数表示。  **最好完全避免使用浮点数比较：**  **float** f = 0.1f;  **double** d = 1.0/10;  System.*out*.println(f==d); //false  **float** d1 = 423432423f;  **float** d2 = d1+1;  **if**(d1==d2){  System.*out*.println("d1==d2");  }**else**{  System.*out*.println("d1!=d2");  }  大数值：  Java.math下面的两个有用的类：BigInteger和BigDecimal，这两个类可以处理任意长度的数值。BigInteger实现了任意精度的整数运算。BigDecimal实现了任意精度的浮点运算。  **浮点数使用总结：**   1. **默认是double** 2. **浮点数存在舍入误差，很多数字不能精确表示。如果需要进行不产生舍入误差的精确数字计算，需要使用BigDecimal类。** 3. **避免比较中使用浮点数** |

# 代码

|  |
| --- |
| //测试浮点数  **public** **class** TestFloatType {    **public** **static** **void** main(String[] args){    //double d = 3.14; //浮点数常量默认类型是double。  //float f = 6.28F;  **double** d2 = 314e-2; //采用科学计数法的写法  System.*out*.println(d2);    **float** f = 0.1f;  **double** d = 1.0/10;  System.*out*.println(f==d); //false      }  } |

# 字符型（2个字节）

     .单引号用来表示字符常量。例如‘A’是一个字符，它与“A”是不同的，“A”表示含有一个字符的字符串。

    .  char 类型用来表示在Unicode编码表中的字符。

      Unicode编码被设计用来处理各种语言的所有文字，它占2个字节，可允许有65536个字符；ASCII码占1个字节，可允许有128个字符，是Unicode编码表中前128个字符。

                char eChar = 'a';

char cChar ='中';

      Unicode具有从0到65535之间的编码，他们通常用从’\u0000’到’\uFFFF’之间的十六进制值来表示（前缀为u表示Unicode）

                 char c = ‘\u0061;

    .Java 语言中还允许使用转义字符 ‘\’ 来将其后的字符转变为其它的含义，

          char c2 = '\n';  //代表换行符

|  |  |  |
| --- | --- | --- |
| 转义符 | 含义 | Unicode值 |
| \b | 退格（backspace） | \u0008 |
| \n | 换行 | \u000a |
| \r | 回车 | \u000d |
| \t | 制表符（tab） | \u0009 |
| \“ | 双引号 | \u0022 |
| \‘ | 单引号 | \u0027 |
| \\ | 反斜杠 | \u005c |

 注：以后我们学的String类，其实是字符序列(char sequence)。

boolean类型（一位，不是一个字节）

    .boolean类型有两个值，true和false,不可以 0 或非 0 的整数替代 true 和 false ，这点和C语言不同。

      boolean 类型用来判断逻辑条件，一般用于程序流程控制 。

    boolean flag ;

        flag = ………;

       if(flag) {

      // true分支

     } else {

     //  false分支

    }

|  |
| --- |
| 实践：Less is More！！请不要这样写：if ( is == true && done == false ) ，只有新手才那么写。 对于任何程序员 if ( whether && !done ) 都不难理解吧。所以去掉所有的==fasle 和 ==true。 |

# 代码

|  |
| --- |
| //测试char  public class TestCharType {      public static void main(String[] args){  /\*          char c1 = 'a';          char c2 = '尚';   //unicode  2: 0-65535          char c3 = '\n';          System.out.print(c1);          System.out.print(c3);          System.out.print(c2);  \*/  /\*          char c4 = 'a';          int i = c4 + 2;          char c5 = (char)i;   //强制转型          System.out.println(c5);  //循环打印a-z      for(int j=0;j<26;j++){      char temp = (char)(c4+j);      System.out.print(temp);      }  //java里面的字符串，是定义成：String类了。  String  str = "abcdefghijklmnopqrstuvwxyz";  System.out.println("\n"+str);  \*/  boolean b = false;   //false  if(b){  System.out.println("true");  }  }  } |

# 自动类型转换

    自动类型转换：容量小的数据类型可以自动转换为容量大的数据类型。在图中，黑色的实线表示无数据丢失的自动类型转换，而红色的虚线表示在转换时可能会精度的损失。

    特例: 可以将整型常量直接赋值给byte, short, char等类型变量，而不需要进行强制类型转换，只要不超出其表数范围

            Short  b = 12;     //合法

            short  b = 1234567;    //非法

# 强制类型转换（Cast）

  强制类型转换，又被称为造型，用于显式的转换一个数值的类型. 在有可能丢失信息的情况下进行的转换是通过造型来完成的，但可能造成精度降低或溢出。

        强制类型转换的语法格式：“(type)var”，运算符“()”中的type表示将值var想要转换成的目标数据类型。

    例如：double x  = 3.14；

          int nx = (int)x;   //值为3

          char c = 'a';

           int d = c+1;

          System.out.println(d);

          System.out.println((char)d);

     当将一种类型强制转换成另一种类型，而又超出了目标类型的表示范围，就会被截断成为一个完全不同的值。

     例如： int x = 300;

            byte bx = (byte)x;    //值为44

    不能在布尔类型和任何数值类型之间做强制类型转换。

    如果需要把布尔类型转换成一个数字该怎么办呢?

# 基本类型转化时常见错误和问题

1.操作比较大的数时，要留意是否溢出，尤其是整数操作时。

|  |
| --- |
| int money = 1000000000;  //10亿      int years = 20;      int total = money\*years;   //返回的是负数      long total1 = money\*years;  //返回的仍然是负数。默认是int，因此结果会转成int值，再转成long。但是已经发生了数据丢失  long total2 = money\*((long)years);   //先将一个因子变成long，整个表达式发生提升。全部用long来计算。 |

2. L和l 的问题：

    1.    不要命名名字为l的变量

    2.    long类型使用大写L不要用小写。

|  |
| --- |
| int l = 2;          long a = 23451l;          System.out.println(l+1); |

# 代码

|  |
| --- |
| //测试自动转型和强制转型  public class TestCast {      public static void main(String[] args){  /\*      byte b = 123;       //byte b2 = 300;       //char c = -3;      char c2 = 'a';      int i = c2;      long d01 = 123213;      float f = d01;      //测试强制转型      int i2 = -100;      char c3 = (char)i2;   //-100超过char的表数范围，所以转换成完全不同的值，无意义的值！      System.out.println(c3);  \*/  /\*  //表达式中的类型提升问题      int a = 3;      long b = 4;      double d = 5.3;      int c = (int)(a+b);   //做所有的二元运算符(+-/\*%)，都会有类型提升的问题！      float f = (float)(a + d);  \*/      int money = 1000000000;  //10亿      int years = 20;      long total = (long)money\*years;   //返回的是负数      System.out.println(total);  //一个人70年心跳多少次      long times = 70L\*60\*24\*365\*70;      System.out.println(times);      }  } |

# JDK7新特性: 二进制整数

由于我们在开发中也经常使用二进制整数，因此JDK7为我们直接提供了二进制整数的类型。我们只要以：0b开头即可。

int a = 0b0101：

# JDK7新特性：下划线分隔符

在实际开发和学习中，如果遇到特别长的数字，读懂它令人头疼！JDK7为我们提供了下划线分隔符，可以按照自己的习惯进行分割，如：

int b = 1\_2312\_3131;

我们很容易就知道这是1亿2312万3131啦！ 非常符合国人的习惯！

# 代码

|  |
| --- |
| public class TestBinaryNum {      public static void main(String[] args){      int a = 0b0000\_0000\_0000\_0000\_0000\_0000\_0000\_0011;      int b = 1\_2312\_3131;      System.out.println(a);      System.out.println(b);      }  } |

# 

# 变量(variable)

     我们通过变量来操纵存储空间中的数据，变量就是指代这个存储空间！空间位置是确定的，但是里面放置什么值不确定！

Java是一种强类型语言，每个变量都必须声明其类型。

Java变量是程序中最基本的存储单元，其要素包括变量名，变量类型和作用域。

变量在使用前必须对其声明, 只有在变量声明以后，才能为其分配相应长度的存储单元，声明格式为：

    type  varName  [=value] [{,varName[=value]}] ；

注意事项：

      每个变量都有类型，类型可以是基本类型，也可以是引用类型。

      变量名必须是合法的标识符。

变量声明是一条完整的语句，因此每一个声明都必须以分号结束

变量声明举例：

    double salary ；   boolean done；

    long earthPopulation ；   int age ；

  可以在一行中声明多个变量：

      int i ,j; // both are integers

q  不提倡这种风格，逐一声明每一个变量可以提高程序可读性。

n  可以将变量的声明和初始化放在同一行中，例如：

     int age = 18;    float e = 2.718281828f;

n  变量可划分为：

q  局部变量(lacal  variable)：方法或语句块内部定义的变量。生命周期是从声明位置开始到”}”为止。

在使用前必须先声明和初始化(赋初值)。

|  |
| --- |
| public void incorrectLocalV() {      int i;      i=10;      int j = i+5 ; // 编译出错，变量i还未被初始化  } |

      实例变量（成员变量 member variable）：方法外部、类的内部定义的变量。从属于对象，生命周期伴随对象始终。

如果不自行初始化，他会自动初始化成该类型的默认初始值（数值型变量初始化成0或0.0，字符型变量的初始化值是16位的0，布尔型默认是false）

    静态变量（类变量 static variable）：使用static定义。 从属于类，生命周期伴随类始终，从类加载到卸载。 (注：讲完内存分析后我们再深入！先放一放这个概念！)

如果不自行初始化，他会自动初始化成该类型的默认初始值（数值型变量初始化成0或0.0，字符型变量的初始化值是16位的0，布尔型默认是false）

课堂练习1：

|  |
| --- |
| public class LocalVariableTest {      public static void main(String[ ] arg) {      boolean flag = true;  // 声明boolean型变量并赋值      char c1, c2;   // 声明char型变量      c1 = '\u0041';   // 为char型变量赋值      c2 = 'B';   // 为char型变量赋值      int x;   // 声明int型变量      x = 9;  //为int型变量赋值      int y = x;  // 声明并初始化int型变量      float f = 3.15f;   // 声明float型变量并赋值      double d = 3.1415926;  //声明double型变量并赋值      }  } |

**常量(Constant)**

·  常量：初始化([initialize](app:ds:initialize))后不能再改变值！

|  |
| --- |
| public class Constants {      public static void main(String[] args) {      final double PI = 3.14;      // PI = 3.15;    //error      double r = 4;      double area = PI \* r \* r;      double circle = 2 \* PI \* r;      System.out.println("area = " + area);      System.out.println("circle = " + circle);      }  } |

# 变量命名规则(规范)

    1.  所有变量、方法、类名：见名知意

    2.  类成员变量：首字母小写和驼峰原则  :  monthSalary

    3.  局部变量：首字母小写和驼峰原则

    4.  常量：大写字母和下划线：MAX\_VALUE

    5.  类名：首字母大写和驼峰原则:  Man, GoodMan

    6.  方法名：首字母小写和驼峰原则: run(), runRun()

# 代码

|  |
| --- |
| public class TestVariable {      int t;    //实例变量，成员变量，属性      public static void main(String[] args){      int a;      int b = a+3;      int x,y,z;      final int C=34;      C = 35;      final int  MAX\_SPEED = 120;      }  } |

# 

# 运算符(operator)

* Java 语言支持如下运算符：
  + 算术运算符: +，-，\*，/，%，++，--
  + 赋值运算符 =
  + 关系运算符: >，<，>=，<=，==，!= instanceof
  + 逻辑运算符: &&，||，!
  + 位运算符: &，|，^，~ ， >>，<<，>>> (了解！！！)
  + 条件运算符 ？：
  + 扩展赋值运算符:+=，-=，\*=，/=

## 二元运算符

整数运算：

如果两个操作数有一个为Long, 则结果也为long

没有long时，结果为int。即使操作数全为shot,byte，结果也是int.

浮点运算：

如果两个操作数有一个为double, 则结果为double.

只有两个操作数都是float, 则结果才为float.

## 取模运算

其操作数可以为浮点数,一般使用整数。如：5.9%3.9=2.000000004

要点：

  负数%负数＝负数；   
  负数%正数＝负数；   
  正数%负数＝正数；

注：一般都是正整数运算，不用考虑这么细！

## 一元运算符(++, --)

**int** a = 3;

**int** b = a++; //执行完后,b=3。先给b赋值，再自增。

**int** c = ++a; //执行完后,c=5。先自增,再给b赋值

**注意：java中的乘幂处理：**

**int** a = 3^2; //java中不能这么处理， ^是异或符号。

**double** b = Math.*pow*(3, 2);

Math类提供了很多科学和工程计算需要的方法和常数。

## 布尔逻辑表达符

逻辑与：&&和&，逻辑或：||和|，逻辑非：！。

逻辑与和逻辑或采用**短路的方式**。从左到右计算，如果确定值则不会再计算下去。

逻辑与只要有一个为false, 则直接返回false.

逻辑或只要有一个为true, 则直接返回true;

**boolean** c = 1>2&&2>(3/0);

## 位运算符

（了解）

~ -- 取反 & -- 按位与

| -- 按位或 ^ -- 按位异或

<<：左移运算符， >>：右移运算符 >>>：无符号移位运算符

右移一位相当于除2取商。

左移一位相当于乘2。

**int** a = 3\*2\*2;

**int** b = 3<<2; //相当于：3\*2\*2;

**int** a = 12/2/2;

**int** b = 12>>2;

## 扩展运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 用法举例 | 等效的表达式 |
| += | a += b | a = a+b |
| -= | a -= b | a = a-b |
| \*= | a \*= b | a = a\*b |
| /= | a /= b | a = a/b |
| %= | a %= b | a = a%b |

## 字符串连接符

* “+”运算符两侧的操作数中**只要有一个是字符串(String)类型**，系统会自动将另一个操作数转换为字符串然后再进行连接。

int c = 12;

System.out.println("c=" + c);

## 三目条件运算符

* 三目条件运算符，语法格式：

**x ? y : z**

* 其中 x 为 boolean 类型表达式，先计算 x 的值，若为true，则整个三目运算的结果为表达式 y 的值，否则整个运算结果为表达式 z 的值。
* 举例：

int score = 80; int x = -100;

String type = score < 60 ? "不及格" : "及格";

**int flag = x > 0 ? 1 : (x == 0 ? 0 : -1);**

System.out.println("type= " + type);

System.out.println("flag= "+ flag);

## 运算符优先级的问题

|  |  |
| --- | --- |
| R to L | . ( ) { } ; , ++ -- ~ ! (data type) |
| L to R | \* / % |
| L to R | + - |
| L to R | << >> >>> |
| L to R | < > <= >= instanceof |
| L to R | == != |
| L to R | & |
| L to R | ^ |
| L to R | | |
| L to R | && |
| L to R | || |
| R to L | ? : |
| R to L | = \*= /= %=   += -= <<= >>=  >>>= &= ^= |= |

大家不需要去刻意的记住他，表达式里面优先使用小括号来组织！！

# 代码

|  |
| --- |
| **public** **class** TestOperator {    **public** **static** **void** main(String[] args){  /\*  double d = 10.2%3;  System.out.println(d);    int a = 3;  int b = a++; //执行完后,b=3。先给b赋值，再自增。  int c = ++a; //执行完后,c=5。先自增,再给b赋值  System.out.println(a);  System.out.println(b);  System.out.println(c);  \*/  // int c = 3/0;    /\*  boolean c = 1<2&&2>(3/0);  System.out.println(c);  \*/  /\*  //测试位运算  int m = 8;  int n = 4;  System.out.println(m&n);  System.out.println(m|n);  System.out.println(~m);  System.out.println(m^n);      int a = 3\*2\*2;  int b = 3<<3; //相当于：3\*2\*2;  int c = 12/2/2;  int d = 12>>2;  System.out.println(a);  System.out.println(b);  System.out.println(c);  System.out.println(d);    boolean b1 = true&false;  System.out.println(b1);  \*/    /\*  //扩展运算符  int a = 3;  a +=5; //a = a+5;  \*/    /\*  //字符串相连：加号两边只要有一个为字符串，则变为字符串连接符，整个结果为字符串。  System.out.println(4+"5");  \*/    **int** a=3;  **int** b=5;  String str= "";  /\*  if(a<b){  str = "a<b";  }else{  str = "a>b";  }  \*/  str = (a<b)?"a<b":"a>=b";  System.*out*.println(str);    }  } |

# 

# eclipse介绍

工作和学习中使用的最多的JAVA集成开发环境(IDE)。

Eclipse 是一个开放源代码的、基于Java的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。

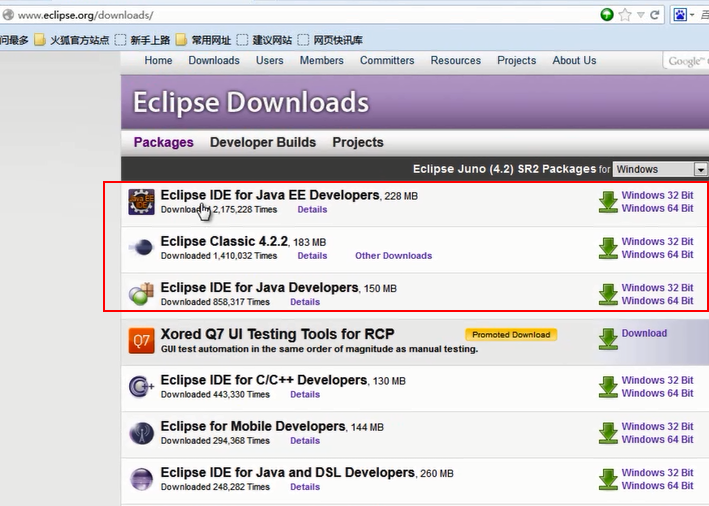
尽管 Eclipse 是使用[Java](http://baike.baidu.com/view/29.htm)语言开发的，但它的用途并不限于 Java 语言；例如，支持诸如[C/C++](http://baike.baidu.com/view/1107274.htm)、[COBOL](http://baike.baidu.com/view/90933.htm)、[PHP](http://baike.baidu.com/view/99.htm)等[编程语言](http://baike.baidu.com/view/552871.htm)的插件已经可用，或预计将会推出。Eclipse 框架还可用来作为与软件开发无关的其他应用程序类型的基础，比如[内容管理系统](http://baike.baidu.com/view/857578.htm)。



# eclipse下载

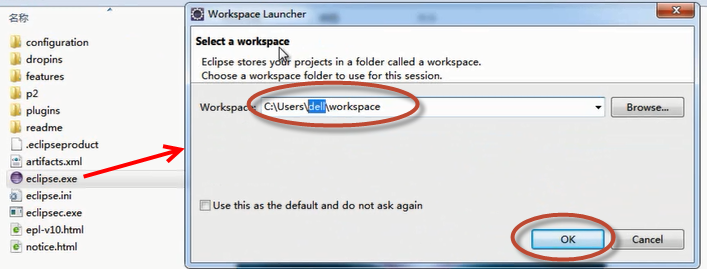
eclipse下载地址：<http://www.eclipse.org/downloads/>

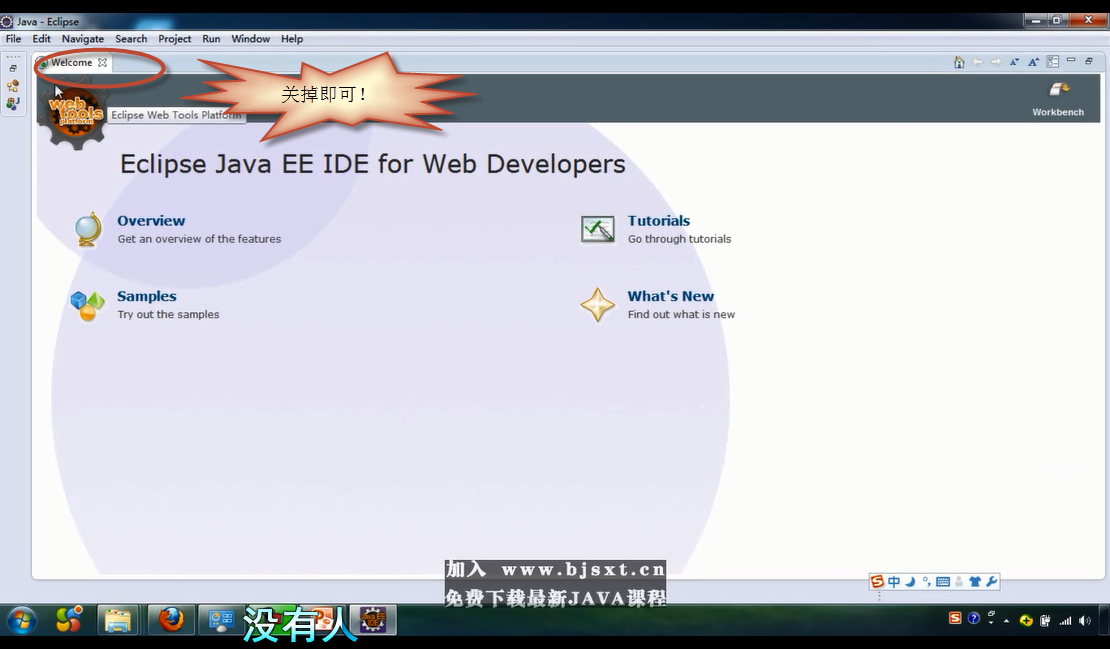
注意：下载eclipse时，根据你安装的JDK是64还是32，下载不同的版本！

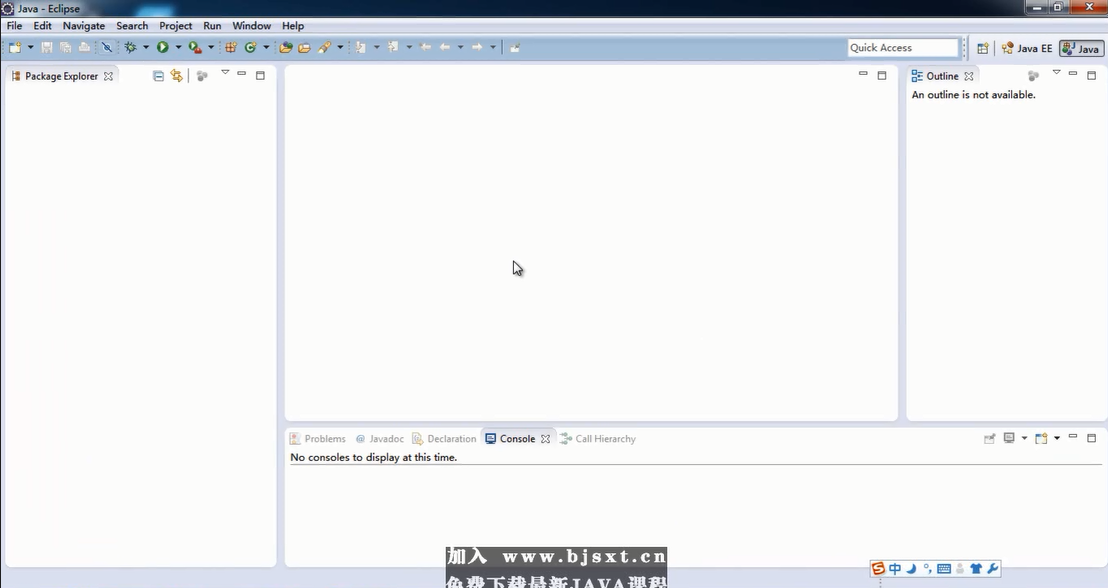


# eclipse使用步骤

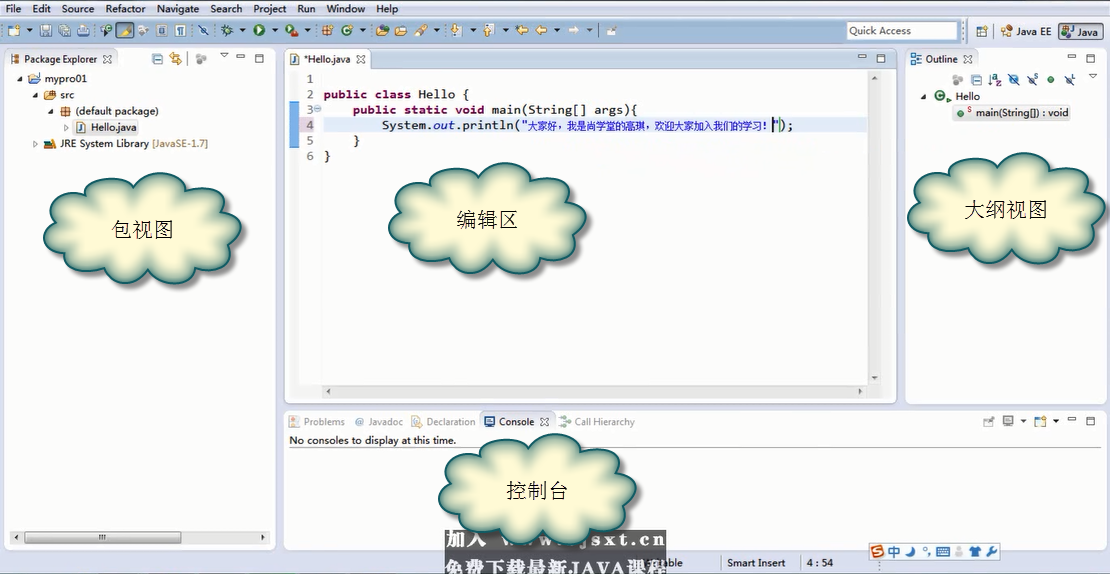
1. 解压eclipse压缩包
2. 双击：eclipse.exe
3. 选择work space(工作空间，说白了就是你所建立的java项目的存储目录)
4. 开始使用eclipse建立和开发项目





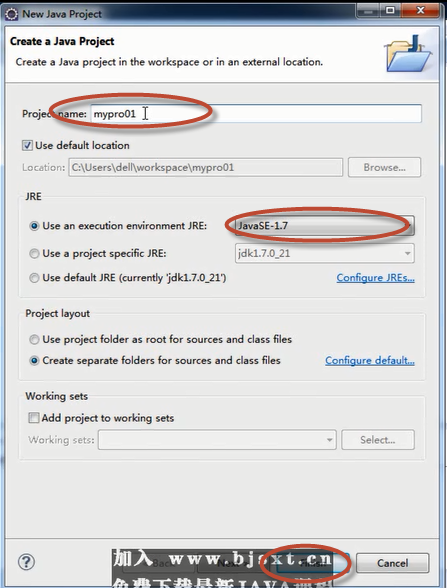


# eclipse视图功能简单介绍

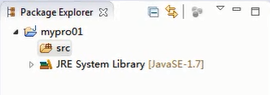


# eclipse中建立java项目

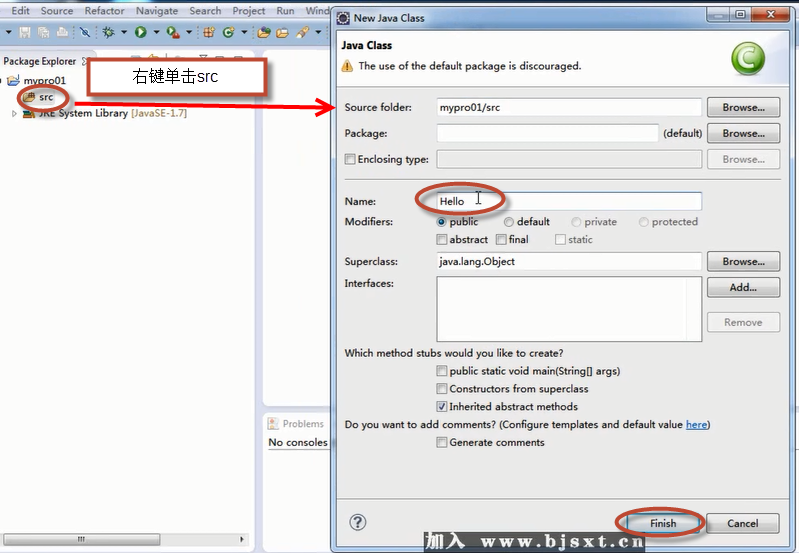
右键单击包视图🡪new 🡪Java Project 🡪输入项目名称



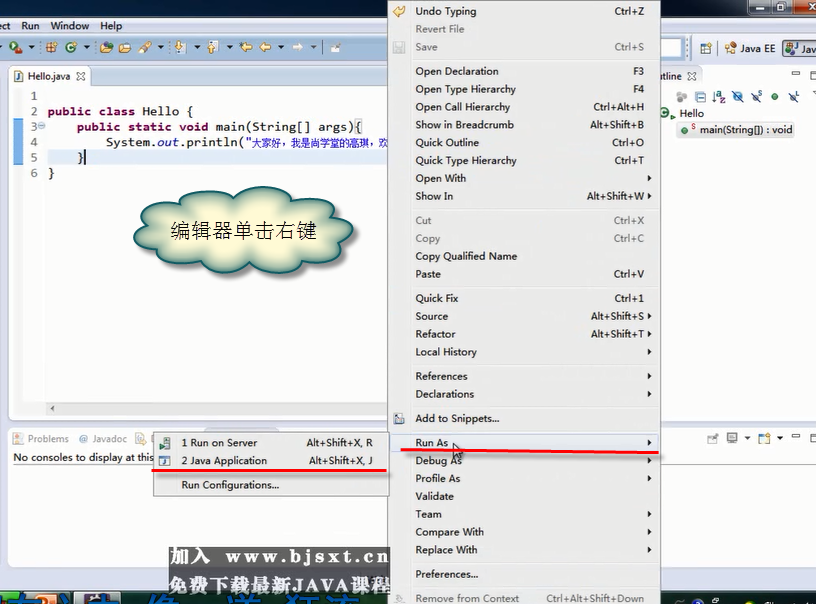
在包视图中出现java项目：



右键单击src，新建java文件：



# eclipse中如何运行java类



**附：**

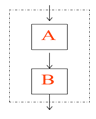
**eclipse常用快捷键(暂时大家可以不关注，随着学习的深入慢慢体会！)**

* 1. 命令提示：alt+/ (main, syso)
  2. 文件选择：
     1. Shift+ 上下左右，home，end键
     2. Ctrl+Shift+ 左右 单词选中
  3. 删除一行：ctrl+d
  4. 复制一行到下面或上面： ctrl+alt+上下
  5. 换行：alt+上下
  6. 编辑文件切换：ctrl+F6，或者ctrl+E，上下
  7. 视图切换：ctrl+F7
  8. 查看错误建议：ctrl+1
  9. 导入类：ctrl+shift+o
  10. 重命名：alt+shift+r

# 

# 顺序结构

    JAVA的基本结构就是顺序结构，除非特别指明，否则就按照顺序一句一句执行顺序结构是最简单的算法结构，

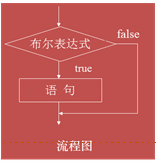


语句与语句之间，框与框之间是按从上到下的顺序进行的，它是由若干个依次执行的处理步骤组成的，它是任何一个算法都离不开的一种基本算法结构。顺序结构在程序流程图中的体现就是用流程线将程序框自上而地连接起来，按顺序执行算法步骤。

# 选择结构

if单选择结构

如果条件为true执行一个操作：



|  |
| --- |
| 意义：if语句对条件表达式进行一次测试，若测试为真，则执行下面的语句，否则跳过该语句      double i = 6 \* Math.random();      double j = 6 \* Math.random();      double k = 6 \* Math.random();      int count = (int) (i + j + k);      if(count > 15) {      System.out.println("今天手气不错");      }      if(count >= 10 && count <= 15) {   //错误写法：10<count<15      System.out.println("今天手气很一般");      }      if(count < 10) {      System.out.println("今天手气不怎么样");      }  System.out.println("得了" + count + "分"); |

     Math类的使用：

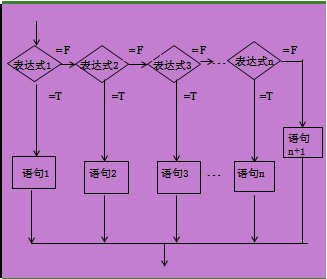
    int i = (int) (6 \* Math.random()); //产生：[0,5]

  f-else双选择结构：

     如果条件为true执行一个操作，为false执行另一个操作：

|  |
| --- |
| 意义：    当条件表达式为真时，执行语句块1，否则，执行语句块2。也就是else部分。  http://www.sxt.cn/editor/attached/image/20151029/20151029212042_496.png      double r = 4 \* Math.random();      double area = Math.PI \* Math.pow(r, 2);      double circle = 2 \* Math.PI \* r;      System.out.println("半径为： " + r);      System.out.println("面积为： " + area);      System.out.println("周长为： " + circle);      if(area >= circle) {      System.out.println("面积大于等于周长");      } else {      System.out.println("周长大于面积");  } |

**If-elseif-else多选择结构**

****

|  |
| --- |
| if(布尔表达式1) {      语句块1；      } else if(布尔表达式2) {      语句块2；      }………      else if(布尔表达式n){      语句块n;      } else {      语句块n+1;      }  逐条if语句进行判断，条件匹配，进入语句体  否则对if语句继续匹配  public class IfTest3 {      public static void main(String[] args) {      int age = (int) (100 \* Math.random());      System.out.print("年龄是" + age + ", 属于");   if (age < 15) {      System.out.println("儿童, 喜欢玩！");  } else if (age < 25) {      System.out.println("青年, 要学习！");  } else if (age < 45) {      System.out.println("中年, 要工作！");  } else if (age < 65) {      System.out.println("中老年, 要补钙！");  } else if (age < 85) {      System.out.println("老年, 多运动！");  } else {      System.out.println("老寿星, 古来稀！");  }      }  } |

# 代码

|  |
| --- |
| /\*   \* 测试if语句   \*/  public class TestIf {      public static void main(String[] args) {      double d = Math.random();      int e =1+(int)(d\*6);      System.out.println(e);  if(e>3) {      System.out.println("大数！");      System.out.println("大数！!!!");  }else{      System.out.println("小数！");  }      System.out.println("测试多选择结构");  if(e==6){      System.out.println("运气非常好！");  }else if(e>=4){      System.out.println("运气不错！");  }else if(e>=2){      System.out.println("运气一般吧");  }else{      System.out.println("运气不好！");  }      }  } |

# 

# switch多选择结构

根据表达式值的不同执行许多不同的操作

Switch语句：case标签必须是整数(byte,short,char,int)或者枚举，不能是字符串。

【注：JDK7.0可以使用字符串！】

|  |
| --- |
| 根据表达式的值，从一系列代码选出一段执行。格式如下：  switch (表达式) {  case 值1:  语句序列;  [break];  case 值2:  语句序列;  [break];  … … … … …  [default:  默认语句;]  }  switch语句会根据表达式的值**从相匹配的case标签处开始执行，一直执行到break语句处或者是switch语句的末尾。**与任一case值不匹配，则进入default语句(如果有的话) |

# 代码

|  |
| --- |
| /\*  \* 测试swtich语句  \*/  **public** **class** TestSwitch {  **public** **static** **void** main(String[] args) {  **double** d = Math.*random*();  **int** e =1+(**int**)(d\*6);  System.*out*.println(e);    System.*out*.println("测试多选择结构");  **if**(e==6){  System.*out*.println("运气非常好！");  }**else** **if**(e==5){  System.*out*.println("运气很不错！");  }**else** **if**(e==4){  System.*out*.println("运气还行吧");  }**else**{ //1,2,3  System.*out*.println("运气不好！");  }    System.*out*.println("#########################");  **switch**(e){ //int,或者自动可以转为int的类型(byte,char,short)。枚举。//JDK7中可以放置字符串。  **case** 6:  System.*out*.println("运气非常好！");  **break**; //一般在每个case后面都要加break，防止出现case穿透现象。  **case** 5:  System.*out*.println("运气很不错！");  **break**;  **case** 4:  System.*out*.println("运气还行吧");  **break**;  **default**:  System.*out*.println("运气不好！");  **break**;  }    System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*下面例子反过来利用了case穿透现象！");  **char** c = 'a';  **int** rand =(**int**) (26\*Math.*random*());  **char** c2 = (**char**)(c+rand);  System.*out*.print(c2 + ": ");  **switch** (c2) {  **case** 'a':  **case** 'e':  **case** 'i':  **case** 'o':  **case** 'u':  System.*out*.println("元音");  **break**;  **case** 'y':  **case** 'w':  System.*out*.println("半元音");  **break**;  **default**:  System.*out*.println("辅音");  }        }  } |

# 

# JDK7.0增强switch语句

可以使用字符串作为表达式结果和case值。

# 代码

|  |
| --- |
| /\*\*  \* 测试JDK7中的switch新特性  \* **@author** dell  \*  \*/  **public** **class** TestSwitch02 {  **public** **static** **void** main(String[] args) {  String a = "高琪";    **switch** (a) { //JDK7的新特性，表达式结果可以是字符串！！！  **case** "马士兵":  System.*out*.println("输入的马士兵");  **break**;  **case** "高琪":  System.*out*.println("输入的高琪");  **break**;  **default**:  System.*out*.println("大家好！");  **break**;  }    }  } |

# 

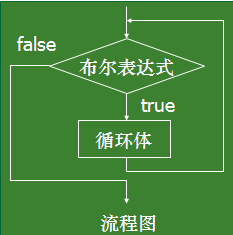
# 循环结构

1.    当型：当P条件成立时(T),反复执行A,直到P为“假”时才停止循环.

2.    直到型：先执行A, 再判断P,若为T,再执行A,如此反复,直到P为F.

## While循环

while循环的基本格式和流程图：

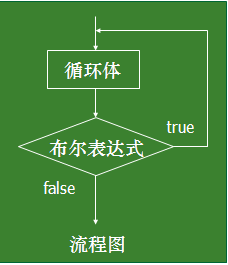


while (布尔表达式) {

  循环体；

}

 while循环详解：



    在循环刚开始时，会计算一次“布尔表达式”的值，若条件为真，执行循环体。而对于后来每一次额外的循环，都会在开始前重新计算一次。

        语句中应有使循环趋向于结束的语句，否则会出现无限循环–––"死"循环。

### while循环代码示例

|  |
| --- |
| public class WhileTest {      public static void main(String[] args) {      int  i = 0;      int  sum = 0;      while (i <= 100) {   // 1+2+3+…+100=?      sum += i;   //sum = sum+i;      i++;      }      System.out.println("Sum= " + sum);      }  } |

## Do-while循环

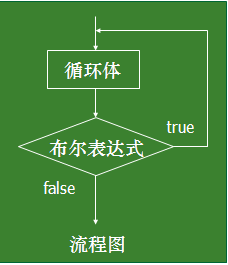
(用的很少，了解！！)

### do-while循环的基本格式：

do {

  循环体；

  } while(布尔表达式) ;



### do-while循环说明和流程图：

1.    先执行循环体

2.    后判断布尔表达式

3.    循环体至少执行一次

### do-while示例代码：

|  |
| --- |
| public class Test {      public static void main(String[] args) {      int i = 0;      int sum = 0;  do {      sum += i; // sum = sum + i      i++;  } while (i <= 100);      System.out.println("Sum= " + sum);  }  } |

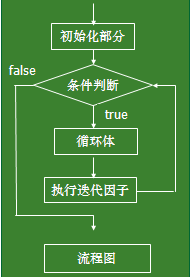
## While和do-While的区别：

while先判断后执行。dowhile是先执行后判断！

Dowhile总是保证循环体会被至少执行一次！这是他们的主要差别。

|  |
| --- |
| int a = 0;  while(a<0){      System.out.println(a);      a++;  }      System.out.println("-----");      a=0;  do{      System.out.println(a);      a++;  } while (a<0); |
| 执行结果：  -----  0 |

# For循环

* ****for循环语句是支持迭代的一种通用结构，是最有效、最灵活的循环结构
* 语法形式

for (初始表达式;布尔表达式;步进) {

循环体；

}

* for循环在第一次反复之前要进行初始化。随后，它会进行条件测试，而且在每一次反复的时候，进行某种形式的“步进”。
  1. 初始化部分设置循环变量的初值
  2. 条件判断部分为任意布尔表达式
  3. 迭代因子控制循环变量的增减

**【注意】for循环在执行条件测试后，先执行程序部分，再执行步进。**

# 代码

|  |
| --- |
| /\*\*  \* 测试For循环语句  \* **@author**  \*  \*/  **public** **class** TestFor {  **public** **static** **void** main(String[] args) {  **int** a = 1; //初始化  **while**(a<=100){ //条件判断  System.*out*.println(a); //循环体  a+=2; //迭代  }  System.*out*.println("while循环结束！");      **for**(**int** i = 1;i<=100;i++){ //初始化//条件判断 //迭代  System.*out*.println(i); //循环体  }  System.*out*.println("while循环结束！");  }  } |

循环综合练习01：

|  |
| --- |
| /\*\*  \* while和for循环后的练习题目  \* **@author** \*  \*/  **public** **class** TestWhileFor {  **public** **static** **void** main(String[] args) {  //计算0到100之间的奇数和偶数的和  **int** oddSum = 0; //用来保存奇数的和  **int** evenSum = 0; //用来存放偶数的和  **for**(**int** i=0;i<=100;i++){  **if**(i%2!=0){  oddSum += i;  }**else**{  evenSum += i;  }    }  System.*out*.println("奇数的和："+oddSum);  System.*out*.println("偶数的和："+evenSum);    System.*out*.println("#########################");  //用while和for循环输出1-1000之间能被5整除的数，并且每行输出3个  **for**(**int** j = 1;j<=1000;j++){  **if**(j%5==0){  System.*out*.print(j+"\t");  }  **if**(j%(5\*3)==0){  System.*out*.println();  }  }    }  } |

# 打印九九乘法表

    通过本练习，大家要体会如何分析问题、如何切入问题！在我们以后写代码的过程中，一定要学会将一个大问题分解成若干小问题，然后，由易到难，各个击破！这也是我们以后开发项目时的基本思维过程。希望大家好好体会！

# 代码

|  |
| --- |
| /\*\*   \* 测试控制语句练习\*   \* @author   \*/  public class TestWhileFor02 {      public static void main(String[] args) {      for (int m = 1; m <= 9; m++) {      for (int i = 1; i <= m; i++) {      System.out.print(i + "\*" + m + "=" + (i \* m) + "\t");  }      System.out.println();  }   }  } |

# 

# break语句

* 在任何循环语句的主体部分，均可用break控制循环的流程。break用于强行退出循环，不执行循环中剩余的语句。(break语句也在switch语句中使用)

|  |
| --- |
| **//循环输出随机数，直到输出88为止！** |
| **public** **class** BreakTest {  **public** **static** **void** main(String[] args) {  **int** total = 0;  System.*out*.println("Begin");  **while** (**true**) {  total++;  **int** i = (**int**) Math.*round*(100 \* Math.*random*());  **if** (i == 88)  **break**;  }  System.*out*.println("Game over, used " + total + " times.");  }  } |

# continue语句

* **continue 语句用在循环语句体中，用于终止某次循环过程，即跳过循环体中尚未执行的语句，接着进行下一次是否执行循环的判定。**

|  |
| --- |
| **//把100~150之间不能被3整除的数输出：** |
| **public** **class** ContinueTest {  **public** **static** **void** main(String[] args) {  **for** (**int** i = 100; i < 150; i++) {  **if** (i % 3 == 0)  **continue**;  System.*out*.println(i);  }  }  } |

# 带标签的break和continue语句

(了解即可)

1. goto关键字很早就在程序设计语言中出现。尽管goto仍是Java的一个保留字，但并未在语言中得到正式使用；Java没有goto。然而，在break和continue这两个关键字的身上，我们仍然能看出一些goto的影子---带标签的break和continue。
2. “标签”是指后面跟一个冒号的标识符，例如：label:
3. 对Java来说唯一用到标签的地方是在循环语句之前。而在循环之前设置标签的唯一理由是：我们希望在其中嵌套另一个循环，由于break和continue关键字通常只中断当前循环，但若随同标签使用，它们就会中断到存在标签的地方。
4. **带标签的break和continue的例子：**

|  |
| --- |
| **public class PrimeNumber { //打印101-150之间所有的质数**  **public static void main(String args[]) {**  **int** count = 0;  outer: **for** (**int** i = 101; i < 150; i ++) {  **for** (**int** j = 2; j < i / 2; j++) {  **if** (i % j == 0)  **continue** outer;  }  System.*out*.print(i+ " ");    }  **}**  **}** |

# 

# 方法

**设计方法的原则：方法的本意是功能块，就是实现某个功能的语句块的集合。 我们设计方法的时候，最好保持方法的原子性，就是一个方法只完成1个功能，这样利于我们后期的扩展。**

## 方法的要点

* **Java的方法类似于其它语言的函数，是一段用来完成特定功能的代码片段，声明格式：**

[修饰符1 修饰符2 …] 返回值类型 方法名(形式参数列表){

Java语句；… … …

}

* 形式参数：在方法被调用时用于接收外界输入的数据。
* 实参：调用方法时实际传给方法的数据。
* 返回值：方法在执行完毕后返还给调用它的环境的数据。
* 返回值类型：事先约定的返回值的数据类型，**如无返回值，必须给出返回值类型void**。
* Java语言中使用下述形式调用方法：对象名.方法名(实参列表)
* 实参的数目、数据类型和次序必须和所调用方法声明的形参列表匹配，
* **return 语句终止方法的运行并指定要返回的数据。**
* Java中进行方法调用中传递参数时，遵循值传递的原则：  
   基本类型传递的是该数据值本身。引用类型传递的是对对象的引用，而不是对象本身。
* **JAVA中只有值传递！**

# 代码

|  |
| --- |
| /\*\*  \* 测试方法  \*  \* **@author** \*  \*/  **public** **class** TestMethod {    //设计方法的原则：方法的本意是功能块，就是实现某个功能的语句块的集合。 我们设计方法的时候，最好保持方法的原子性，就是一个方法只完成1个功能，这样利于我们后期的扩展。  **public** **static** **void** test01(**int** a) {  **int** oddSum = 0; // 用来保存奇数的和  **int** evenSum = 0; // 用来存放偶数的和  **for** (**int** i = 0; i <= a; i++) {  **if** (i % 2 != 0) {  oddSum += i;  } **else** {  evenSum += i;  }  }  System.*out*.println("奇数的和：" + oddSum);  System.*out*.println("偶数的和：" + evenSum);  }    **public** **static** **void** test02(**int** a,**int** b,**int** c){  **for** (**int** j = 1; j <= a; j++) {  **if** (j % b == 0) {  System.*out*.print(j + "\t");  }  **if** (j % (b \* c) == 0) {  System.*out*.println();  }  }  }    **public** **static** **int** add(**int** a,**int** b){  **int** sum = a+b;  **if**(a==3){  **return** 0; //return 两个作用：结束方法的运行、返回值。  }  System.*out*.println("输出");  **return** sum;  }    **public** **static** **void** main(String[] args) {  *test01*(1000);  *test02*(100,6,3); //1-100之间，可以被6整除，每行输出3个。  System.*out*.println("###########");  **int** s = *add*(3,5);  System.*out*.println(s);  }  } |

# 

# 递归算法

    A方法调用B方法，我们很容易理解！递归就是：A方法调用A方法！就是自己调用自己，因此我们在设计递归算法时，一定要指明什么时候自己不调用自己。否则，就是个死循环！

## 递归算法要点

    递归是一种常见的解决问题的方法，即把问题逐渐简单化。递归的基本思想就是“自己调用自己”，一个使用递归技术的方法将会直接或者间接的调用自己。

利用递归可以用简单的程序来解决一些复杂的问题。比如：大多数排序使用的就是递归算法。

递归结构包括两个部分：

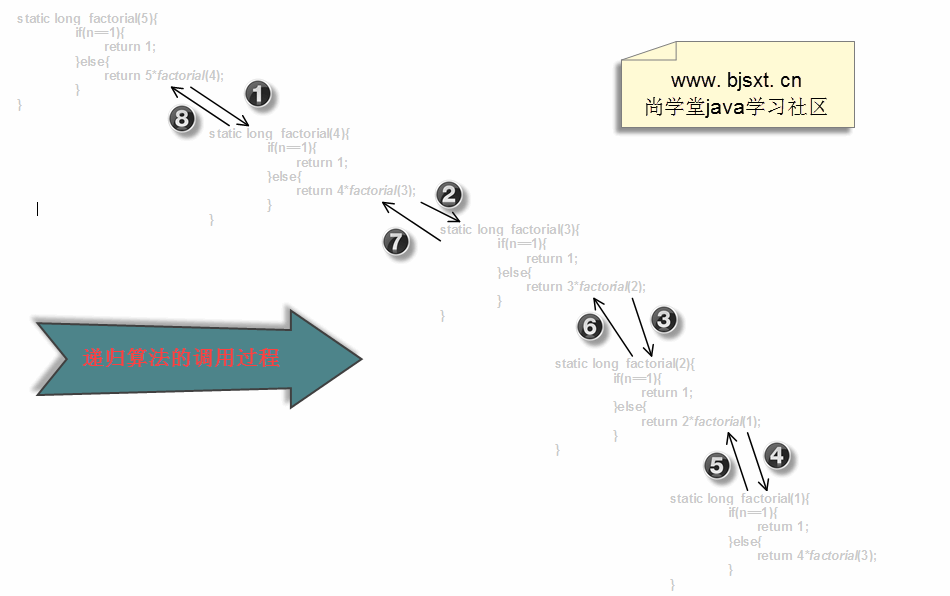
    1.    定义递归头。解答：什么时候不调用自身方法。如果没有头，将陷入死循环。

    2.    递归体。解答：什么时候需要调用自身方法。

# 代码

|  |
| --- |
| /\*\*   \* 测试递归算法   \* @author   \*   \*/  public class TestRecursion {      static int a = 0;      public static void test01(){      a++;      System.out.println("test01:"+a);      if(a<=10){  //递归头      test01();      }else{      //递归体      System.out.println("over");      }  }  public static void test02(){      System.out.println("TestRecursion.test02()");  }  public static long factorial(int n){      if(n==1){      return 1;      }else{      return n\*factorial(n-1);      }  }  public static void main(String[] args) {      test01();      System.out.println(factorial(10));      }  } |

  代码中阶乘代码的调用执行过程，参见下图，可以放大看：



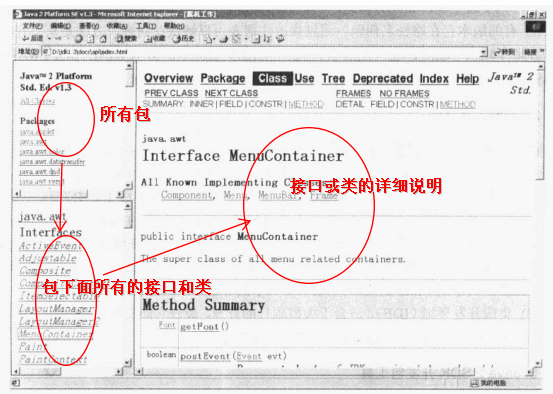
# 

# API是什么？

API：Application Programming Interface,应用程序编程接口。

# API文档是什么?

    DK为我们提供了很多类，如何使用这些类。我们可以通过查看API文档来获取信息。API文档就相当于我们学英语用的词典一样！



# package包的概念

## 为什么需要package?

    为了便于管理类：合适的类位于合适的包！

    为了解决类之间的重名问题。

## package怎么用?

    通常是类的第一句非注释性语句。

    包名：域名倒着写即可，再加上模块名，并与内部管理类。

    其实内部实现就是靠目录结构来做到的。

 com.sun.test

com.oracle.test

com.bjsxt.gao.test

com.bjsxt.gao.view

com.bjsxt.gao.view.model

 注意事项：

    写项目时都要加包，不要使用默认包。

com.gao和com.gao.car，这两个包没有包含关系，是两个完全独立的包。只是逻辑上看起来后者是前者的一部分。

## JDK中的主要包

       java.lang－包含一些Java语言的核心类，如String、Math、Integer、System和Thread，提供常用功能。

       java.awt－包含了构成抽象窗口工具集（abstract window toolkits）的多个类，这些类被用来构建和管理应用程序的图形用户界面(GUI)。

     java.net－包含执行与网络相关的操作的类。

     java.io－包含能提供多种输入/输出功能的类。

     java.util－包含一些实用工具类，如定义系统特性、使用与日期日历相关的函数。

# 如何生成自己项目的API文档?

    •    特殊的注释：

–    文档注释：/\*\*

–    使用JAVADOC生成API文档

–    解决问题：代码和文档的分离

    •    常用的java注释标签：

–    @Author 作者

–    @version 版本

–    @param  参数

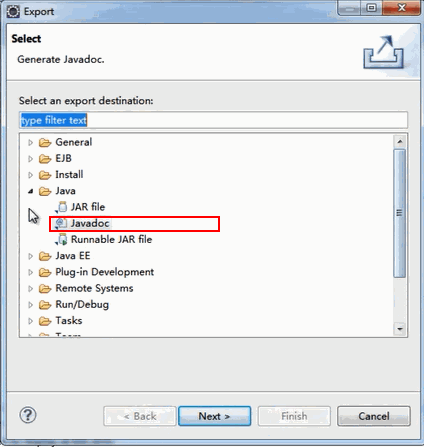
–    @return  返回值的含义

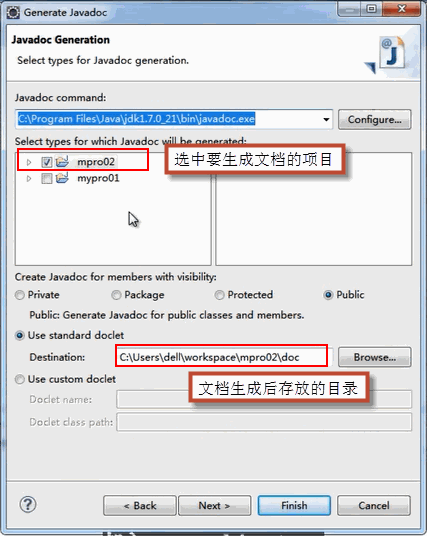
–    @throws 抛出异常描述

–    @deprecated 废弃。建议用户不在使用该方法

## 在eclipse中创建api文档的步骤

1.    右键单击项目，选中export





# import

## 为什么需要import?

    如果不适用import，我们如果用到其他包的类时，只能这么写：java.util.Date，代码量太大，不利于编写和维护。通过import可以导入其他包下面的类，从而可以在本类中直接通过类名来调用。

## import怎么使用?

import  java.util.Date;

import  java.util.\*;   //导入该包下所有的类。会降低编译速度，但不会降低运行速度。

 注意要点：

    1.    java会默认导入java.lang包下所有的类，因此这些类我们可以直接使用。

    2.    如果导入两个同名的类，只能用包名+类名来显示调用相关类：

    java.util.Date date  = new  java.util.Date();

# 静态导入

    static  import静态导入(JDK 1.5后增加, 了解！！！！)：

    静态导入的作用：用于导入指定类的静态属性

如何使用：

    import static java.lang.Math.\*;//导入Math类的所有静态属性

    import static java.lang.Math.PI;//导入Math类的PI属性

然后，我们可以在程序中直接使用：System.out.println(PI);

# 课堂中的代码

(通过Scanner类练习import以及了解如何获取键盘的输入信息)：

|  |
| --- |
| import java.util.Scanner;   /\*\*   \* 测试import导入Scanner类，以及Scanner类的使用，如何接收键盘的输入。   \* @author   \*   \*/  public class TestScanner {      public static void test01(){      Scanner s = new Scanner(System.in);      String str = s.next();  //程序运行到next会阻塞，等待键盘的输入！      System.out.println("刚才键盘输入："+str);  }  public static void test02(){      Scanner s = new Scanner(System.in);      System.out.println("请输入一个加数：");      int a = s.nextInt();      System.out.println("请输入被加数：");      int b = s.nextInt();      int sum =a+b;      System.out.println("计算结果，和为："+sum);  }  public static void main(String[] args) {      test02();  }  } |

# 面向对象和面向过程的本质区别

    语言的进化发展跟生物的进化发展其实是一回事，都是”物以类聚”。相近的感光细胞聚到一起变成了我们的眼睛，相近的嗅觉细胞聚到一起变成了我们的鼻子。

语句多了，我们将完成同样功能的相近的语句，聚到了一块儿，便于我们使用。于是，方法出现了！

变量多了，我们将功能相近的变量组在一起，聚到一起归类，便于我们调用。于是，结构体出现了！

再后来，方法多了，变量多了！结构体不够用了！我们就将功能相近的变量和方法聚到了一起，于是类和对象出现了！

寥寥数语，就深刻的展示了语言的进化历史！其实，都非常自然，”物以类聚”。希望大家能记住这句话。

    企业的发展也是”物以类聚”的过程，完成市场推广的人员聚到一起形成了市场部。完成技术开发的人员聚到一起形成了开发部！

# 面向过程的思维模式

    面向过程的思维模式是简单的线性思维，思考问题首先陷入第一步做什么、第二步做什么的细节中。这种思维模式适合处理简单的事情，比如：上厕所。

如果面对复杂的事情，这种思维模式会陷入令人发疯的状态！比如：如何造神舟十号！

# 面向对象的思维模式

     面向对象的思维模式说白了就是分类思维模式。思考问题首先会解决问题需要哪些分类，然后对这些分类进行单独思考。最后，才对某个分类下的细节进行面向过程的思索。

这样就可以形成很好的协作分工。比如：设计师分了10个类，然后将10个类交给了10个人分别进行详细设计和编码！

显然，面向对象适合处理复杂的问题，适合处理需要多人协作的问题！

如果一个问题需要多人协作一起解决，那么你一定要用面向对象的方式来思考！

  老毛显然是面向对象思维方式的高手，很好地对中国的阶层做了分类(大家可以看看老毛的著作，都是谈各个阶层的关系问题，面向对象啊！)，然后，让他手下的干将制定详细策略各个击破。老毛不出延安，天下事就已定！ 若老毛生在今天做了程序员，必然也将做到总监、架构师级别。

老蒋显然没有采用面向对象的思维方式处理问题，天天忙得团团转，做着飞机到处指挥，最后被赶到了一个小岛上。若老蒋生在今天做了程序员，嗯，老蒋会是个天天加班干的好程序员。

  对于描述复杂的事物，为了从宏观上把握、从整体上合理分析，我们需要使用面向对象的思路来分析整个系统。但是，具体到微观操作，仍然需要面向过程的思路去处理。

# 

# 面向对象编程(oop)的本质

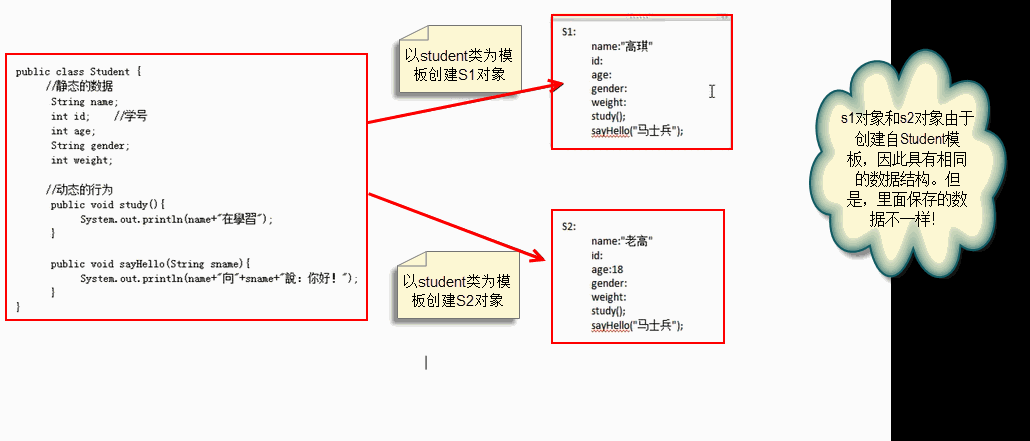
     面向对象编程的本质就是：以类的方式组织代码，以对象的组织(封装)数据。

   抽象是什么?就是将多个物体共同点归纳出来，就是抽出像的部分！

   从认识论角度考虑是先有对象后有类。对象，是具体的事物。类，是抽象的，是对对象的抽象。

从代码运行角度考虑是先有类后有对象。类是对象的模板。

   如下图深刻的揭示了类和对象的关系：



# 代码：

|  |
| --- |
| package cn.bjsxt.oop;    public class Student {      //静态的数据       String name;       int id;    //学号       int age;       String gender;       int weight;  //动态的行为   public void study(){   System.out.println(name+"在學習");   }    public void sayHello(String sname){   System.out.println(name+"向"+sname+"說：你好！");   }  } |
| public class Test1 {     public static void main(String[] args) {          Student s1 = new Student();          s1.name = "高琪";          s1.study();          s1.sayHello("馬士兵");          Student s2 = new Student();          s2.age = 18;          s2.name="老高";          }  } |

# 

# JAVA程序运行的内存分析

栈：

    1.    每个线程私有，不能实现线程间的共享！

    2.    局部变量放置于栈中。

    3.    栈是由系统自动分配，速度快！栈是一个连续的内存空间！

堆：

    1.    放置new出来的对象！

    2.    堆是一个不连续的内存空间，分配灵活，速度慢！

方法区(也是堆)：

    1.    被所有线程共享！

    2.    用来存放程序中永远是不变或唯一的内容。（类代码信息、静态变量、字符串常量）

     【注】本次内存分析，我们的主要目的是让大家了解基本的内存概念。类加载器、Class对象这些更加详细的内容，我们将在后面专门讲反射的课程里面讲。

**引用类型的概念**

    1.    java中，除了基本数据类型之外的其他类型称之为引用类型。

    2.    java中的对象是通过引用来操作的。(引用：reference)

    .    说白了，引用指的就是对象的地址！

**属性（field，或者叫成员变量）**

    1.    属性用于定义该类或该类对象包含的数据或者说静态属性。

    2.    属性作用范围是整个类体。

    3.    属性的默认初始化：

    在定义成员变量时可以对其初始化，如果不对其初始化，Java使用默认的值对其初始化。(数值：0,0.0   char:\u0000,  boolean:false,  所有引用类型:null)

    4.    属性定义格式：

[修饰符]  属性类型  属性名 = [默认值]

# 类的方法

        方法是类和对象动态行为特征的抽象。方法很类似于面向过程中的函数。面向过程中，函数是最基本单位，整个程序有一个个函数调用组成；面向对象中，整个程序的基本单位是类，方法是从属于类或对象的。

 方法定义格式：

[修饰符]  方法返回值类型  方法名(形参列表) {

// n条语句

}

**java对象的创建和使用**

    ³    必须使用 new 关键字创建对象。

    ²    Person person= new Person ();

    ³     使用对象（引用） . 成员变量来引用对象的成员变量。

    ²    person.age

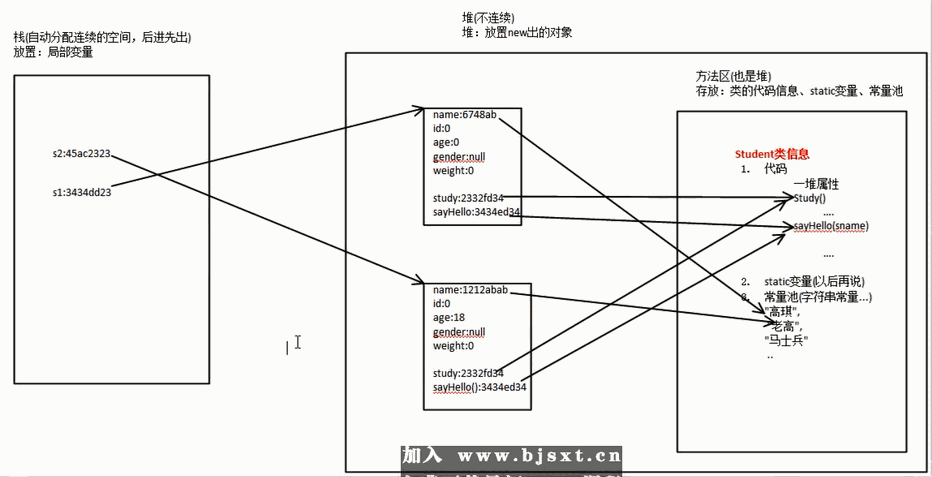
    ³     使用对象（引用） . 方法（参数列表）来调用对象的方法。

    ²    person. setAge(23)

# 代码和内存分析结果

|  |
| --- |
|  |
| package cn.bjsxt.oop;       public class Student {  //静态的数据       String name;       int id;    //学号       int age;       String gender;       int weight;    //动态的行为   public void study(){       System.out.println(name+"在學習");   }     public void sayHello(String sname){       System.out.println(name+"向"+sname+"說：你好！");   }  } |
| package cn.bjsxt.oop;   public class Test1 {       public static void main(String[] args) {   //通过类加载器Class Loader加载Student类。 加载后，在方法区中就有了Student类的信息！      Student s1 = new Student();      s1.name = "高琪";      s1.study();      s1.sayHello("馬士兵");  Student s2 = new Student();      s2.age = 18;      s2.name="老高";  }  } |

 上面Test1类执行的内存分析图如下(更加详细的分析流程，参见31集视频)：



# 

# JAVA程序运行的内存分析

栈：

    1.    每个线程私有，不能实现线程间的共享！

    2.    局部变量放置于栈中。

    3.    栈是由系统自动分配，速度快！栈是一个连续的内存空间！

堆：

    1.    放置new出来的对象！

    2.    堆是一个不连续的内存空间，分配灵活，速度慢！

方法区(也是堆)：

    1.    被所有线程共享！

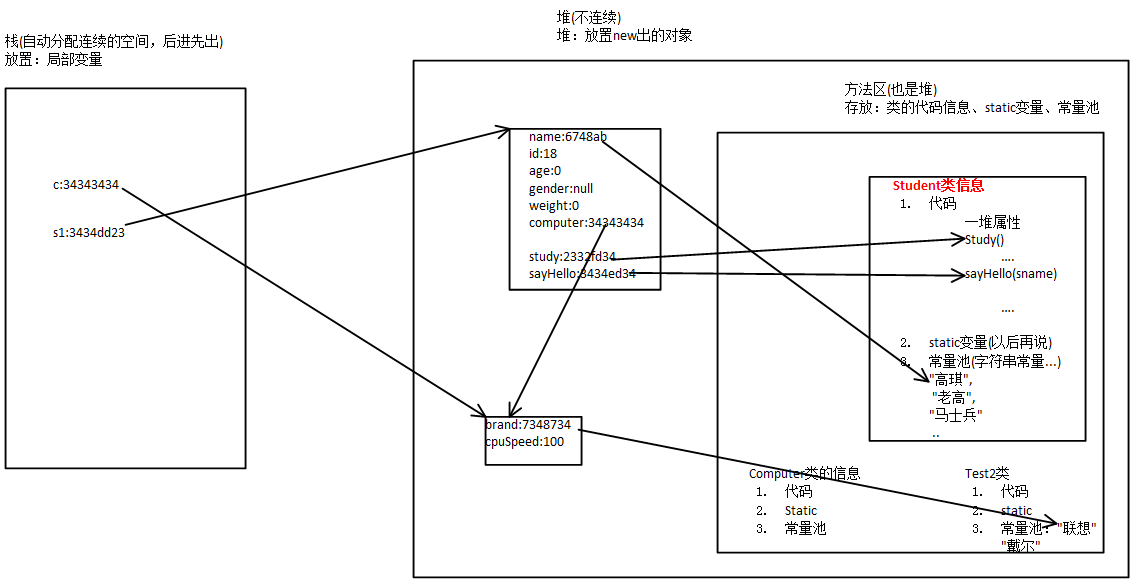
    2.    用来存放程序中永远是不变或唯一的内容。（类代码信息、静态变量、字符串常量）

 【注】本次内存分析，我们的主要目的是让大家了解基本的内存概念。类加载器、Class对象这些更加详细的内容，我们将在后面专门讲反射的课程里面讲。

**java中更加详细的内存分析**

# 代码和内存分析图

|  |
| --- |
| package cn.bjsxt.oop;   public class Computer {      String brand;      int cpuSpeed;  } |
| package cn.bjsxt.oop;     public class Student {      //静态的数据       String name;       int id;    //学号       int age;       String gender;       int weight;    Computer computer;   //动态的行为   public void study(){       System.out.println(name+"在學習");   }    public void sayHello(String sname){       System.out.println(name+"向"+sname+"說：你好！");   }  } |
| package cn.bjsxt.oop;   public class Test2 {      public static void main(String[] args) {      Student s1 = new Student();      s1.name = "高琪";      s1.age = 18;  Computer c = new Computer();      c.brand = "联想";      c.cpuSpeed = 100;      s1.computer = c;      c.brand = "戴尔";  System.out.println(s1.computer.brand);  }  } |



# 垃圾回收机制(Garbage Collection)

JAVA的内存管理实际上指的就是对象的管理，其中包括对象空间的分配和释放。

## 对象空间的分配

使用new关键字创建对象即可

## 对象空间的释放

将对象赋值null，即该对象没有引用指向时。垃圾回收器将负责回收所有”不可达”对象的内存空间。

## 要点

1. 程序员无权调用垃圾回收器。
2. 程序员可以通过System.gc()。通知GC运行，但是JAVA规范并不能保证立刻运行。
3. finalize方法，是JAVA提供给程序员用来释放对象或资源的方法，但是尽量少用。

# 

# 构造器(constructor，或者叫构造方法)

## 构造器作用：

    构造器用于构造该类的实例。

## 构造器调用：

Java通过new关键字来调用构造器，从而返回该类的实例。

## 构造器定义的格式：

[修饰符] 类名(形参列表){

//n条语句

}

## 构造器使用要点

    1.    通过new关键字调用！！

    2.    构造器虽然有返回值（返回该类的对象），但是不能定义返回类型 (返回值的类型肯定是本类)，不能在构造器里调用return。

    3.    如果我们没有定义构造器，则系统会自动定义一个无参的构造函数。如果已定义则编译器不会添加！

    4.    构造器的方法名必须和类名一致！

## 构造器简单代码测试

|  |
| --- |
| package cn.bjsxt.oop;   public class Car {      int speed;      public Car(){      System.out.println("构造一个车");      }  } |
| package cn.bjsxt.oop;   /\*\*   \* 测试构造方法   \* @author  \*   \*/  public class TestConstructor {      public static void main(String[] args) {      Car c = new Car();      }  } |

**课堂作业**

³    定义一个“点”（Point）类用来表示三维空间中的点（有三个坐标）。要求如下：

²    可以生成具有特定坐标的点对象。

²    提供可以设置三个坐标的方法。

²    提供可以计算该“点”距另外点距离的方法。

|  |
| --- |
| package cn.bjsxt.oop;    public class Point {  double x,y,z;  //通过构造方法初始化对象的属性  public Point(double x,double y, double z){      x =x;      y = y;      z = z;  }  public  void setX(double \_x){      x = \_x;  }  public  void setY(double \_y){      y = \_y;  }  public  void setZ(double \_z){      z = \_z;  }  public double distance(Point p){      double result =  Math.sqrt((x-p.x)\*(x-p.x)+(y-p.y)\*(y-p.y)+(z-p.z)\*(z-p.z));      return result;  }  public static void main(String[] args) {      Point p = new Point(3,4,8);      Point p2 = new Point(200,40,80);      p.setX(10);      System.out.println(p.x);      System.out.println(p.distance(p2));  }  } |

# 

# 方法的重载(overload)，构造方法的重载

方法的重载是指一个类中可以定义有相同的名字，但参数不同的多个方法。 调用时，会根据不同的参数表选择对应的方法。

     方法调用时，匹配离得最近的类型

    不同的含义：类型，个数，顺序不同

    只有返回值不同不构成方法的重载（int a(String str){}, void  a{int i},  调用：a()，谁能告诉我是调哪个方法?）

    只有形参的名称不同，不构成方法的重载

    与普通方法一样，构造方法也可以重载

|  |
| --- |
| package cn.bjsxt.oop;   /\*\*   \* 测试重载   \* @author  \*   \*/  public class TestOverload {      public int add(int a, int b){      return a+b;  }  public static void main(String[] args) {      MyMath m = new MyMath();      int result = m.add(4.2,8);      System.out.println(result);      }  }      class  MyMath {      int a;      int b;      public MyMath(){  }    public MyMath(int a){      this.a = a;  }  public MyMath(int b, int a){      this.b = b;      this.a = a;  }  public int add(int b, double a){      return (int)(a+b);  }  public int add(double a, int b){      return (int)(a+b);  }  public int add(int a, int b){      return a+b;  }  public int add(int a, int b,int c){      return a+b+c;  }  } |

# 

# static关键字

在类中，用static声明的成员变量为静态成员变量.

    1.它为该类的公用变量，属于类，被该类的所有实例共享，在类被载入时被显式初始化，

    2.对于该类的所有对象来说，static成员变量只有一份。被该类的所有对象共享！！

    3.可以使用”对象.类属性”来调用。不过，一般都是用“类名.类属性”。

    4.用static声明的方法为静态方法

    5.不需要对象，就可以调用(类名.方法名)

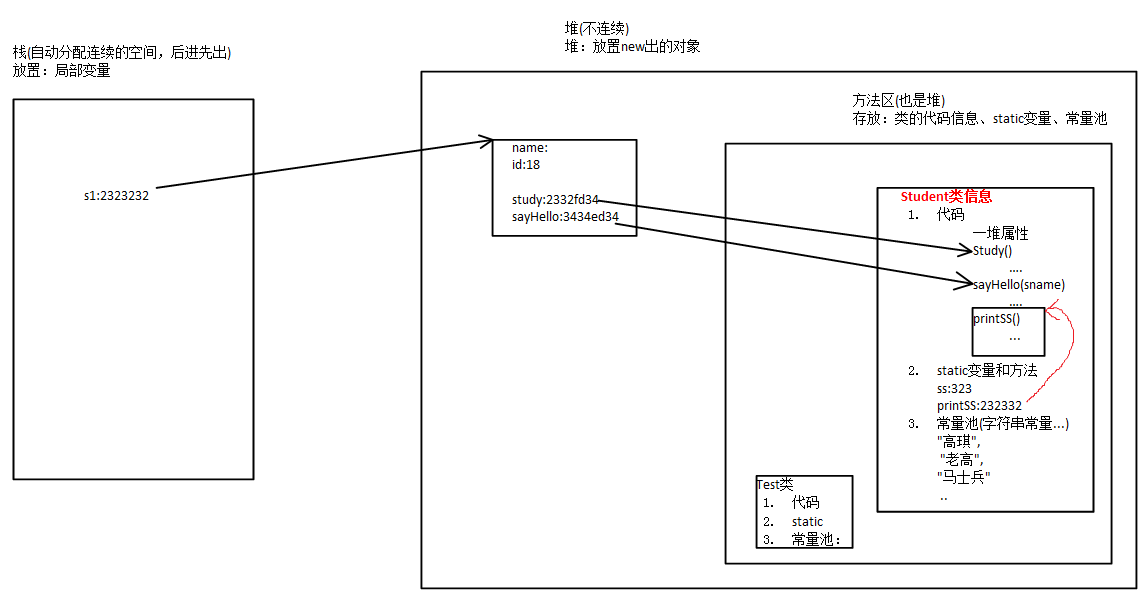
    在调用该方法时，不会将对象的引用传递给它，所以在static方法中不可访问非static的成员。

可以通过对象引用或类名（不需要实例化）访问静态成员。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.testStatic;   public class Student {       String name;       int id;      static  int  ss;    public static void printSS(){       System.out.println(ss);   }     public void study(){       printSS();       System.out.println(name+"在學習");   }     public void sayHello(String sname){       System.out.println(name+"向"+sname+"說：你好！");    }  } |
| package cn.bjsxt.oop.testStatic;  public class Test {      public static void main(String[] args) {      Student.ss = 323;      Student.printSS();      Student s1 = new Student();  }  } |

 static 代码的内存分析图如下（想查看具体怎么画的流程，请看相应的视频课程）:



# 

# this关键字

    普通方法中，this总是指向调用该方法的对象。

    构造方法中，this总是指向正要初始化的对象。

## this最常的用法：

    1.    让类中的一个方法，访问该类的另一个方法或属性。

    2.    使用this关键字调用重载构造方法。避免相同的初始化代码，只能在构造方法中用，并且必须位于构造方法的第一句。

## this使用时的注意事项：

    ·this不能用于static方法！(this指向当前对象，static方法跟对象没有一毛钱的关系)

# 代码

|  |
| --- |
| package cn.bjsxt.oop.testThis;   public class Student {       String name;       int id;    public  Student(String name,int id){       this(name);   //通过this调用其他构造方法，必须位于第一句！ Constructor call must be the first statement in a constructor       this.name = name;       this.id = id;   }     public Student(String name){       this.name = name;   }   public Student(){       System.out.println("构造一个对象");   }     public void setName(String name){       this.name = name;   }     public void study(){       this.name=  "张三";       System.out.println(name+"在學習");   }    public void sayHello(String sname){       System.out.println(name+"向"+sname+"說：你好！");       }  } |

# 

# 为什么需要继承？继承的作用？

    第一好处：继承的本质在于抽象。类是对对象的抽象，继承是对某一批类的抽象。

    第二好处：为了提高代码的复用性。

extands的意思是“扩展”。子类是父类的扩展。

【注】JAVA中类只有单继承没有多继承！ 接口可以多继承！

## 继承介绍

现实世界中的继承无处不在。比如：

动物：

   哺乳动物、爬行动物

哺乳动物：

  灵长目、鲸目等。

 继承的本质是对某一批类的抽象，从而实现对现实世界更好的建模。

## 如何实现继承？

使用extends关键字即可。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.inherit;   /\*\*   \* 测试继承   \* @author   \*   \*/  public class Animal {      String eye;      public void run(){  System.out.println("跑跑！");  }  public void eat(){      System.out.println("吃吃！");  }  public void sleep(){      System.out.println("zzzzz");  }  public  Animal(){      System.out.println("创建一个动物！");  }  }    class Mammal extends Animal {      public void taisheng(){      System.out.println("我是胎生");  }  }    class Bird  extends Animal {      public void eggSheng(){      System.out.println("卵生");  }  } |
| package cn.bjsxt.oop.inherit;    public class Test {  public static void main(String[] args) {      Bird b = new Bird();      b.run();  }  } |

# 

# 方法的重写（override）

³    在子类中可以根据需要对从基类中继承来的方法进行重写。

³     重写方法必须和被重写方法具有相同方法名称、参数列表和返回类型。

³     重写方法不能使用比被重写方法更严格的访问权限。（由于多态）

# 方法重写和方法重载的区别

    很多人问重写和重载什么区别? 他们两个完全是两回事。除了名字都带一个“重”字之外。

方法重载指的是：同一个类中，一个方法名对应了多个方法(形参列表不同)

方法的重写指的是：子类重写了父类的方法！

# 代码

|  |
| --- |
| package cn.bjsxt.oop.inherit;   /\*\*   \* 测试继承   \* @author   \*   \*/  public class Animal  {      String eye;      public void run(){  System.out.println("跑跑！");  }      public void eat(){      System.out.println("吃吃！");  }  public void sleep(){      System.out.println("zzzzz");  }  }    class Mammal extends Animal {      public void taisheng(){      System.out.println("我是胎生");  }  }    class Bird  extends Animal {  //重写了父类的run方法！  public void run(){      super.run();      System.out.println("我是一个小小小小鸟，飞呀飞不高");  }  public void eggSheng(){      System.out.println("卵生");  }  } |
| package cn.bjsxt.oop.inherit;  public class Test {  public static void main(String[] args) {      Bird b = new Bird();      b.run();  }  } |

# 

# Object类

³    Object类是所有Java类的根基类

³    如果在类的声明中未使用extends关键字指明其基类，则默认基类为Object类

     public class Person {

        ...

     }

³    等价于：

public class Person extends Object {

...

}

# toString方法

³    Object类中定义有public String toString()方法，其返回值是 String 类型，描述当前对象的有关信息。

³    在进行String与其它类型数据的连接操作时（如：System.out.println(“info”+person）），将自动调用该对象类的 toString()方法

³    Object类中toString方法的源代码是：

|  |
| --- |
| public String toString() {     return getClass().getName() + "@" + Integer.toHexString(hashCode());  } |

默认是打印：类名+哈希码(根据对象地址计算得出 )！

³ 可以根据需要在用户自定义类型中重写toString()方法。比如String类就重写了toString方法，源代码如下：

|  |
| --- |
| public String toString() {          return this;      } |

# equals方法

³    Object类中定义有：

²    public boolean equals(Object obj)方法

±    提供定义对象是否“相等”的逻辑。

²    Object 的 equals 方法 定义为：x.equals ( y ) 当 x 和 y是同一个对象的应用时返回 true 否则返回 false

查看Object类中的equals方法源代码可以看出：

|  |
| --- |
| public boolean equals(Object obj) {      return (this == obj);  } |

²    J2SDK提供的一些类，如String，Date等，重写了Object的equals方法，调用这些类的equals方法， x.equals (y) ,当x和y所引用的对象是同一类对象且属性内容相等时（并不一定是相同对象），返回 true 否则返回 false。

String类的equals方法显然是比较数组中的char值是否相等，源代码摘录如下：

|  |
| --- |
| public boolean equals(Object anObject) {          if (this == anObject) {              return true;          }          if (anObject instanceof String) {              String anotherString = (String) anObject;              int n = value.length;              if (n == anotherString.value.length) {                  char v1[] = value;                  char v2[] = anotherString.value;                  int i = 0;                  while (n-- != 0) {                      if (v1[i] != v2[i])                              return false;                      i++;                  }                  return true;              }          }          return false;      } |

²    可以根据需要在用户自定义类型中重写equals方法。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.testObject;   public class Mobile {  public String toString(){  return "我是一部移动电话";  }  } |
| package cn.bjsxt.oop.testObject;    public class TestObject {  public static void main(String[] args) {      Object obj = new Object();      Object obj2 = new Object();      System.out.println(obj.toString());      System.out.println(obj2.toString());      System.out.println(obj==obj2);      System.out.println(obj.equals(obj2));      String str;      Mobile m = new Mobile();      System.out.println(m.toString());  }  } |

# 

# super

    super是直接父类对象的引用。可以通过super来访问父类中被子类覆盖的方法或属性。

除了Object类之外其他所有类的构造方法第一句总是super(…)

    任何类的构造方法中，若是构造方法的第一行代码没有显式的调用super(...);那么Java默认都会调用super(); 所以你这里的super(); 加不加都无所谓。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.inherit;   /\*\*   \* 测试继承   \* @author   \*   \*/  public class Animal /\*extends Object\*/ {      String eye;      public void run(){      System.out.println("跑跑！");  }  public void eat(){      System.out.println("吃吃！");  }  public void sleep(){      System.out.println("zzzzz");  }  public  Animal(){      super();      System.out.println("创建一个动物！");  }  }    class Mammal extends Animal {      public void taisheng(){      System.out.println("我是胎生");  }  }    class Bird  extends Animal {      public void run(){      super.run();      System.out.println("我是一个小小小小鸟，飞呀飞不高");  }  public void eggSheng(){      System.out.println("卵生");  }  public Bird(){      super();      System.out.println("建一个鸟对象");  }  } |
| package cn.bjsxt.oop.inherit;   public class Test {  public static void main(String[] args) {      Bird b = new Bird();      b.run();  }  }  http://www.sxt.cn/editor/attached/image/20151031/20151031202127_231.png |

# 

# 继承和组合

共同点：都可以实现代码的复用

不同点：

is-a关系：使用继承

has-a关系：使用组合

# 代码

使用组合实现前几节课用继承实现的关系：Animal、Mammal、Bird：

|  |
| --- |
| package cn.bjsxt.oop.inherit;    /\*\*   \* 测试组合   \* @author   \*   \*/  public class Animal2 {      String eye;  public void run(){      System.out.println("跑跑！");  }  public void eat(){      System.out.println("吃吃！");  }  public void sleep(){      System.out.println("zzzzz");  }  public  Animal2(){      super();  System.out.println("创建一个动物！");  }  public static void main(String[] args) {      Bird2 b = new Bird2();      b.run();      b.animal2.eat();  }  }    class Mammal2  {      Animal2 animal2=new Animal2();  public void taisheng(){  System.out.println("我是胎生");  }  }    class Bird2  {      Animal2 animal2=new Animal2();      public void run(){      animal2.run();      System.out.println("我是一个小小小小鸟，飞呀飞不高");  }  public void eggSheng(){      System.out.println("卵生");  }  public Bird2(){      super();      System.out.println("建一个鸟对象");  }  } |

# 

# final的用法

修饰变量：

    被final修饰的变量不可改变。即初始化一次后，就不能再次被赋值！即，变为常量！

修饰方法：

    被final修饰的方法不可以被子类重写！

修饰类：

    被final修饰的类不能被继承!

# 代码

|  |
| --- |
| package cn.bjsxt.oop.testFinal;  public class TestFinal {  public static void main(String[] args) {      final int MAX\_VALUE= 200;    //常量。      double d = Math.PI;  }  } |
| package cn.bjsxt.oop.testFinal;   public /\*final\*/ class Animal {    //final修饰类则说明，这个类不能被继承！      public /\*final\*/ void run(){   //final加到方法前面，意味着该方法不能被子类重写！      System.out.println("跑跑！");  }  }      class Bird  extends Animal {      public void run(){      super.run();      System.out.println("我是一个小小小小鸟，飞呀飞不高");  }  } |

# 

面向对象三大特征：

继承、封装、多态

# 封装(encapsulation)

## 为什么需要封装？封装的作用和含义？

我要看电视，只需要按一下开关和换台就可以了。有必要了解电视机内部的结构吗？有必要碰碰显像管吗？

我要开车，….

制造厂家为了方便我们使用电视，方便我们开车，把复杂的内部细节全部封装起来，只给我们暴露简单的接口，比如：电源开关、比如：油门。具体怎么内部实现的，我们不需要操心。

需要让用户知道的暴露出来，不需要让用户了解的全部隐藏起来。这就是封装。

白话：该露的露，该藏的藏

专业：我们程序设计要追求“高内聚，低耦合”。高内聚就是类的内部数据操作细节自己完成，不允许外部干涉；低耦合：仅暴露少量的方法给外部使用。

## 编程中封装的具体意义

便于调用者调用。

良好的封装，便于修改内部代码，提高可维护性。

良好的封装，可进行数据完整性检测，保证数据的有效性。

## 使用访问控制符，实现封装

à范围越来越大

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 同一个类 | 同一个包中 | 子类 | 所有类 |
| private | \* |  |  |  |
| default（即不加修饰符的时候） | \* | \* |  |  |
| protected | \* | \* | \* |  |
| public | \* | \* | \* | \* |

## 封装的使用细节

1.    属性一般使用private.(除非本属性确定会让子类继承并且使用)

提供相应的get/set方法来访问相关属性. 这些方法通常是public，从而提供对属性的读取操作。  （注意：boolean变量的get方法是用：is开头!）

2.    一些只用于本类的辅助性方法可以用private，希望其他类调用的方法用public。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.encapsulation01;    public class Man {      private String name;      private String id;      private boolean man;      public static int cc;      public static final int MAX\_SPEED = 120;      public String getName(){      return name;  }  public void setName(String name){      this.name = name;  }  public String getId() {      return id;  }  public void setId(String id) {      this.id = id;  }  public boolean isMan() {      return man;  }  public void setMan(boolean man) {      this.man = man;  }  } |
| package cn.bjsxt.oop.encapsulation01;   public class Test01 {      private String str;  protected void print(){      String s = str;      System.out.println("Test01.print()");  }  }    class Test001 extends Test01 {      public void pp(){      super.print();  }  public static void main(String[] args) {      Test01 t = new Test01();      t.print();  }  } |
| package cn.bjsxt.oop.encapsulation01;    public class Test02 {  public static void main(String[] args) {      Test01 t = new Test01();      t.print();  }  } |
| package cn.bjsxt.oop.encapsulation02;   import cn.bjsxt.oop.encapsulation01.Test01;   public class Test0001 extends Test01 {      public void ttt(){      super.print();      print();  }  public static void main(String[] args) {      Test0001 t = new Test0001();      t.print();  }  } |
| package cn.bjsxt.oop.encapsulation02;   import cn.bjsxt.oop.encapsulation01.Test01;    public class Test03 {  public static void main(String[] args) {      Test01 t = new Test01();      //  t.print();  }  } |

# 

面向对象三大特征：

继承、封装、多态

# 多态(polymorphism)

     多态性是OOP中的一个重要特性，主要是用来实现动态联编的，换句话说，就是程序的最终状态只有在执行过程中才被决定而非在编译期间就决定了。这对于大型系统来说能提高系统的灵活性和扩展性。

     多态可以让我们不用关心某个对象到底是什么具体类型，就可以使用该对象的某些方法，从而实现更加灵活的编程，提高系统的可扩展性。

## 为什么需要多态?

    多态性是指允许不同类的对象对同一消息作出不同的响应。

## java中如何实现多态?使用多态的好处?

    引用变量的两种类型：

    编译时类型（模糊一点，一般是一个父类）

    由声明时的类型决定。

    运行时类型（运行时，具体是哪个子类就是哪个子类）

    由实际对应的对象类型决定。通过动态绑定调用该对象的方法。动态绑定会使我们的程序编写更加灵活，但是会减慢程序运行速度。这也是JAVA比C++/C慢的主要原因之一。

 多态：如果编译时类型和运行时类型不一致，就会造成多态。

## 方法绑定(method  binding)

    执行调用方法时，系统根据相关信息，能够执行内存地址中代表该方法的代码。分为静态绑定和动态绑定。

静态绑定：

在编译期完成，可以提高代码执行速度。

动态绑定：

    通过对象调用的方法，采用动态绑定机制。 这虽然让我们编程灵活，但是降低了代码的执行速度。这也是JAVA比C/C++速度慢的主要因素之一。 JAVA中除了final类、final方、static方法，所有方法都是JVM在运行期才进行动态绑定的。

 ?????

    应该针对成员变量、static方法等做一个专门的动态绑定的课程：

<http://blog.sina.com.cn/s/blog_600046120100wdza.html>

<http://blog.csdn.net/aaaaaaaa0705/article/details/6263213>

<http://www.doc88.com/p-389664843916.html>

## 多态的使用要点

    1.    多态是方法的多态，属性没有多态性。

    2.    编写程序时，如果想调用运行时类型的方法，只能进行类型转换。不然通不过编译器的检查。但是如果两个没有关联的类进行强制转换，会报：ClassCastException。 比如：本来是狗，我把它转成猫。就会报这个异常。

    3.    多态的存在要有3个必要条件：要有继承，要有方法重写，父类引用指向子类对象

## 对象的转型(casting)

为什么需要强制转换类型？

    引用变量只能调用它编译类型的方法，不能调用它运行类型的方法。这时，我们就需要进行类型的强制转换！

³    一个父类的引用类型变量可以“指向”其子类的对象。

³    一个父类的引用不可以访问其子类对象新增加的成员（属性和方法）。

³    可以使用 引用 变量 instanceof 类名 来判断该引用型变量所“指向”的对象是否属于该类或该类的子类。

³    子类的对象可以当作基类的对象来使用称作向上转型（upcasting），反之称为向下转型（downcasting）

# 代码

|  |
| --- |
| package cn.bjsxt.oop.polymorphism;   public class Animal {      String str;      public void voice(){      System.out.println("普通动物叫声！");  }  }    class Cat extends Animal {  public void voice(){      System.out.println("喵喵喵");  }  public void catchMouse(){      System.out.println("抓老鼠");  }  }    class Dog extends Animal {  public void voice(){      System.out.println("汪汪汪");  }  public void seeDoor(){      System.out.println("看门！");  }  }    class Tiger extends Animal {  public void voice(){      System.out.println("哇哇哇");  }    }    class Pig extends Animal {  public void voice(){      System.out.println("哼哼哼");  }  } |
| package cn.bjsxt.oop.polymorphism;    public class Test {  public static void testAnimalVoice(Animal c){      c.voice();      if(c instanceof Cat){      ((Cat) c).catchMouse();  }  }  /\*  public static void testAnimalVoice(Dog c){      c.voice();  }  public static void testAnimalVoice(Pig c){      c.voice();  }\*/  //javac Test.java  // java Test  public static void main(String[] args) {      Animal a = new Cat();      Cat a2 = (Cat)a;      testAnimalVoice(a);    //  a2.catchMouse();    //  Animal b = new Dog();  //  Animal c = new Tiger();  //  testAnimalVoice(b);  //  testAnimalVoice(c);  }  } |

# 

# 面向对象三大特征：

继承、封装、多态

# 代码

|  |
| --- |
| package cn.bjsxt.oop.polymorphism;   public class Animal {      String str;  public void voice(){      System.out.println("普通动物叫声！");  }  }    class Cat extends Animal {  public void voice(){      System.out.println("喵喵喵");  }  public void catchMouse(){      System.out.println("抓老鼠");  }  }    class Dog extends Animal {  public void voice(){      System.out.println("汪汪汪");  }  public void seeDoor(){      ystem.out.println("看门！");  }  }    class Tiger extends Animal {  public void voice(){      System.out.println("哇哇哇");  }    }    class Pig extends Animal {  public void voice(){      System.out.println("哼哼哼");  }  } |
| package cn.bjsxt.oop.polymorphism;   public class Test {      public static void testAnimalVoice(Animal c){      c.voice();  if(c instanceof Cat){  ((Cat) c).catchMouse();  }  }  public static void main(String[] args) {      Animal a = new Cat();      Cat a2 = (Cat)a;      testAnimalVoice(a);  }  }  20151031210424_394.png |

 注：this永远指向当前对象！

# 

# 面向对象三大特征：

继承、封装、多态

# 代码

|  |
| --- |
| package cn.bjsxt.oop.polymorphism.myServlet;    public class HttpServlet {  public void service(){      System.out.println("HttpServlet.service()");      this.doGet();  }  public void doGet(){      System.out.println("HttpServlet.doGet()");  }  } |
| package cn.bjsxt.oop.polymorphism.myServlet;   public class MyServlet extends HttpServlet {  public void doGet(){      System.out.println("MyServlet.doGet()");  }  } |
| package cn.bjsxt.oop.polymorphism.myServlet;   public class Test {  public static void main(String[] args) {      HttpServlet s = new MyServlet();      s.service();  }  }  20151031211050_855.png |

# 

# 抽象方法

    只声明了方法，同时使用了abstract修饰，没有方法体！必须要子类来重写！

# 抽象类

    包含抽象方法的类，必须设计成抽象类！

# 抽象类的使用要点

    1.    通过抽象类，可以避免子类设计的随意性。

    2.    有抽象方法的类只能定义能抽象类

    3.    抽象类不能实例化，及不能用new来实例化抽象类。

    4.    抽象类可以包含属性、方法、构造方法。但是构造方法不能用来new实例，只能用来被子类调用。

    5.    抽象类只能用来继承。

    6.    抽象方法必须被子类实现。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.abstractClass;  public abstract class Animal {      String str;      public abstract void run(); //抽象方法的意义在于：将方法的设计和实现分离！  public void breath(){      System.out.println("呼吸！");      run();  }  public Animal(){      System.out.println("创造一个动物！");  }  }    class Cat extends Animal {  public void run(){      System.out.println("猫步小跑");  }  }   class Dog extends Animal  {  public void run(){      System.out.println("狗跑");  }  } |
| package cn.bjsxt.oop.abstractClass;   public class Test {  public static void main(String[] args) {      Animal a = new Cat();      a.breath();  }  } |

# 

 普通类：只有具体实现

抽象类：具体实现和规范(抽象方法) 都有！

接口：只有规范！

# 为什么需要接口?

    为什么需要接口?接口和抽象类的区别?

    接口就是比“抽象类”还“抽象”的“抽象类”，可以更加规范的对子类进行约束。全面地专业地实现了：规范和具体实现的分离。

   抽象类还提供某些具体实现，接口不提供任何实现，接口中所有方法都是抽象方法。接口是完全面向规范的，规定了一批类具有的公共方法规范。

   从接口的实现者角度看，接口定义了可以向外部提供的服务。

   从接口的调用者角度看，接口定义了实现者能提供那些服务。

   接口是两个模块之间通信的标准，通信的规范。如果能把你要设计的系统之间模块之间的接口定义好，就相当于完成了系统的设计大纲，剩下的就是添砖加瓦的具体实现了。大家在工作以后，做系统时往往就是使用“面向接口”的思想来设计系统。

# 如何定义接口?

格式：

[访问修饰符]  interface 接口名   [extends  父接口1，父接口2…]  {

常量定义

方法定义

}

访问修饰符：只能是public或默认。

接口名：和类名采用相同命名机制

extends：接口可以多继承

常量：接口中的属性只能是常量，总是：public static final 修饰。不写也是。

方法：接口中的方法只能是：public abstract。 省略的话，也是public abstract.

# 接口的本质探讨

    接口就是规范，定义的是一组规则，体现了现实世界中“如果你是…则必须能…”的思想。如果你是天使，则必须能飞。如果你是汽车，则必须能跑。如果你好人，则必须干掉坏人；如果你是坏人，则必须欺负好人。

接口的本质是契约，就像我们人间的法律一样。制定好后大家都遵守。

     OO的精髓，是对对象的抽象，最能体现这一点的就是接口。为什么我们讨论设计 模式都只针对具备了抽象能力的语言（比如c++、java、c#等），就是因为设计模式所研究的，实际上就是如何合理的去抽象。

# 接口使用要点

    1.    子类通过implements来实现接口中的规范

    2.    接口不能创建实例，但是可用于声明引用变量类型。

    3.    一个类实现了接口，必须实现接口中所有的方法，并且这些方法只能是public的。

    4.    接口中只能包含静态常量、抽象方法，不能有普通属性、构造方法、普通方法。

    5.    常量必然是：public static final。  我们不加这些修饰符，也是这样。

    6.    方法必然是：public abstrac。  我们不加这些修饰符，也是这样。

    7.    接口完全支持多继承。和类的继承类似，子接口扩展某个父接口，将会获得父接口中所定义的一切。

# 代码

|  |
| --- |
| package cn.bjsxt.oop.testInterface;  public interface MyInteface {  //接口中只有常量和抽象方法  /\*public static final\*/ String MAX\_GREAD = "BOSS";  int MAX\_SPEED = 120;  /\*public abstract\*/ void test01();  public int test02(int a,int b);  } |
| package cn.bjsxt.oop.testInterface;    public interface Flyable {      int MAX\_SPEED = 11000;      int MIN\_HEIGHT = 1;      void fly();  }    interface Attack {  void attack();  }    class Plane  implements Flyable{  @Override  public void fly() {  System.out.println("飞机靠发动机飞！");  }  }    class Man  implements Flyable{  @Override  public void fly() {  System.out.println("跳起来，飞！");  }  }    class Stone implements Flyable, Attack{  int weight;  @Override  public void fly() {  System.out.println("被人扔出去，飞！");  }    @Override  public void attack() {  System.out.println("石头攻击!");  }  } |
| package cn.bjsxt.oop.testInterface;    public class Test {  public static void main(String[] args) {  Flyable f = new Stone();  f.fly();  }  } |
| package cn.bjsxt.oop.testInterface;    public interface InterfaceA {  void aaa();  }    interface InterfaceB {  void bbb();  }  interface InterfaceC extends InterfaceA,InterfaceB {  void ccc();  }    class TestClass implements InterfaceC{    @Override  public void aaa() {  // TODO Auto-generated method stub  }    @Override  public void bbb() {  // TODO Auto-generated method stub  }   @Override  public void ccc() {  // TODO Auto-generated method stub  }  } |

# 

# 代码

|  |
| --- |
| package cn.bjsxt.oop.callback;   public class PaintFrame {  public static void drawFrame(IMyFrame f){      System.out.println("启动线程");      System.out.println("增加循环");      System.out.println("查看消息栈");      //画窗口      f.paint();      System.out.println("启动缓存，增加效率");  }  public static void main(String[] args) {      drawFrame(new GameFrame01());  }  }    class GameFrame01 /\*extends MyFrame\*/ implements IMyFrame {  public void paint(){      System.out.println("GameFrame01.paint()");      System.out.println("画窗口");  }  }    class GameFrame02 extends MyFrame {  public void paint(){      System.out.println("GameFrame02.paint()");      System.out.println("画窗口");  }  } |
| package cn.bjsxt.oop.callback;   public abstract class MyFrame {  public void paint(){      System.out.println("把自己窗口画出来！");  }  }   interface IMyFrame {      void paint();  } |

# 

# 组合

使用组合，可以获得更多的灵活性，你甚至可以在运行的时候才决定把哪几个类组合在一起。     
  使用继承，他是一种高度的耦合，派生类和基类被紧紧的绑在一起，灵活性大大降低，而且，滥用继承，也会使继承树变得又大又复杂，很难理解和维护

如果是”is-a”关系，用继承。

如果是”has-a”关系，用组合。

|  |
| --- |
| class Computer {  Cpu cpu;  Mainboard  md;  Mouse  mouse;  }   class Cpu {  }   class Mainboard{  }   class Mouse {  } |

# 内部类(innerclasses)

    一般情况，我们把类定义成独立的单元。有些情况下，我们把一个类放在另一个类的内部定义，称为内部类。

## 内部类的作用：

    1.    内部类提供了更好的封装。只能让外部类直接访问，不允许同一个包中的其他类直接访问。

    2.    内部类可以直接访问外部类的私有属性，内部类被当成其外部类的成员。 但外部类不能访问内部类的内部属性。

 内部类的使用场合：

    由于内部类提供了更好的封装特性，并且可以很方便的访问外部类的属性。所以，通常内部类在只为所在外部类提供服务的情况下优先使用。

## 内部类的分类：

    1.    成员内部类（可以使用private、proteted、public任意进行修饰。 类文件：外部类$内部类.class）

    a)    非静态内部类（外部类里使用非静态内部类和平时使用其他类没什么不同）

i.    非静态内部类必须寄存在一个外部类对象里。因此，如果有一个非静态内部类对象那么一定存在对应的外部类对象。 非静态内部类对象单独属于外部类的某个对象。

ii.    非静态内部类可以使用外部类的成员，但是外部类不能直接访问非静态内部类成员。

iii.    非静态内部类不能有静态方法、静态属性、静态初始化块。

iv.    静态成员不能访问非静态成员：外部类的静态方法、静态代码块不能访问非静态内部类，包括不能使用非静态内部类定义变量、创建实例。

v.    成员变量访问要点：

    1.    内部类里方法的局部变量：变量名

    2.    内部类属性：this.变量名

    3.    外部类属性：外部类名.this.变量名

vi.    内部类的访问：

    1.    外部类中定义内部类：  new InnerClass()

    2.    外部类以外的地方使用非静态内部类：   
 Outer.inner  varname = OuterObject.new Inner()

|  |
| --- |
| Face.Nose nose =  new Face().new Nose();  Face f2 = new Face();  Face.Nose nose2 = f2.new Nose(); |

 b) 静态内部类

i.    定义方式：

static  class   ClassName {

//类体

}

ii.    使用要点：

    1.    当一个静态内部类对象存在，并不一定存在对应的外部类对象。 因此，静态内部类的实例方法不能直接访问外部类的实例方法。

    2.    静态内部类看做外部类的一个静态成员。 因此，外部类的方法中可以通过：静态内部类.名字  访问静态内部类的静态成员。通过 new 静态内部类()访问静态内部类的实例。

    3.    在外部类的外面创建静态内部类：

Face.TestStaticInner  aInner = new  Face.TestStaticInner();

 2. 匿名内部类

    适合那种只需要使用一次的类。比如：键盘监听操作等等。语法：

new  父类构造器(实参类表) 实现接口 () {

//匿名内部类类体！

}

|  |
| --- |
| this.addWindowListener(new WindowAdapter(){    @Override  public void windowClosing(WindowEvent e) {  System.exit(0);  }            }); |
| this.addKeyListener(new KeyAdapter(){  @Override  public void keyPressed(KeyEvent e) {  // TODO Auto-generated method stub  myTank.keyPressed(e);  }   @Override  public void keyReleased(KeyEvent e) {  // TODO Auto-generated method stub  myTank.keyReleased(e);  }  }); |

# 局部内部类

定义在方法内部。作用域只限于本方法。用的非常少。

# 字符串（java.lang.String类）的使用

     Java字符串就是Unicode字符序列，例如串“Java”就是4个Unicode字符J,a,v,a组成的。

n    Java没有内置的字符串类型，而是在标准Java类库中提供了一个预定义的类String，每个用双引号括起来的字符串都是String类的一个实例。

    String e = “” ; // an empty string

    String greeting = “Hello World”;

n    Java允许使用符号"+"把两个字符串连接起来

String s1 = “Hello”;String s2 = “World!”;

String s = s1 + s2; //HelloWorld!

n    符号“+”把两个字符串按给定的顺序连接在一起，并且是完全按照给定的形式。

n    当“+”运算符两侧的操作数中只要有一个是字符串(String)类型，系统会自动将另一个操作数转换为字符串然后再进行连接

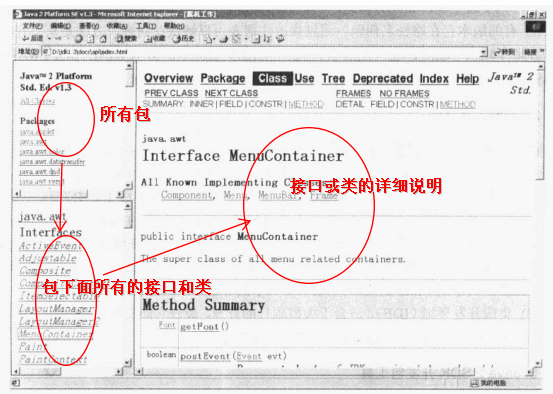
int age = 18;

String s = "age is" + age;  //s赋值为age is 18

这种特性通常被用在输出语句中：

System.out.println(“age is” + age）;

 开始学习阅读API文档：



字符串类常用的方法：

n    char charAt(int index)

返回字符串中第index个字符。

n    boolean equals(String other)

   如果字符串与other相等，返回true

n    boolean equalsIgnoreCase(String other)

如果字符串与other相等（忽略大小写），则返回true

n    int indexOf(String str)    lastIndexOf()

n    int indexOf(String str,int fromIndex)

   返回与str匹配的第一个字串的开始位置，该位置从0或fromIndex开始计算，如果原始串中不存在str，返回-1.

n    int length()

返回字符串的长度。

n    String replace(char oldChar,char newChar)

   返回一个新串，它是通过用 newChar 替换此字符串中出现的所有oldChar而生成的

|  |
| --- |
| public class StringTest1 {  public static void main(String[] args) {      String s1 = "core java";      String s2 = "Core Java";      System.out.println(s1.charAt(4));      System.out.println(s2.length());      System.out.println(s1.equals(s2));      System.out.println(s1.equalsIgnoreCase(s2));      System.out.println(s1.indexOf("java"));      System.out.println(s1.indexOf("apple"));      String s = s1.replace(' ', '&');      System.out.println("result is :" + s);  }  } |

n    boolean startsWith(String prefix)

   如果字符串以prefix开始，则返回true

n    boolean endsWith(String prefix)

   如果字符串以prefix结尾，则返回true

n    String substring(int beginIndex)

n    String substring(int beginIndex,int endIndex)

n    返回一个新字符串，该串包含从原始字符串beginIndex到串尾或endIndex-1的所以字符

n    String toLowerCase()

返回一个新字符串，该串将原始字符串中的所有大写字母改成小写字母

n    String toUpperCase()

返回一个新字符串，该串将原始字符串中的所有小写字母改成大写字母

n    String trim()

返回一个新字符串，该串删除了原始字符串头部和尾部的空格

|  |
| --- |
| public class StringTest2 {  public static void main(String[] args) {  String s = "";  String s1 = "How are you?";  System.out.println(s1.startsWith("How"));  System.out.println(s1.endsWith("you"));  s = s1.substring(4);       System.out.println(s);  s = s1.substring(4,7); System.out.println(s);  s = s1.toLowerCase();      System.out.println(s);  s= s1.toUpperCase(); System.out.println(s);  String s2 = "  How old are you!! " ;  s = s2.trim();   System.out.println(s);  System.out.println(s2); //???  }  } |

**字符串相等的判断**

n    equals方法用来检测两个字符串内容是否相等。如果字符串s和t内容相等，则s.equals(t)返回true,否则返回false.

n    s和t既可以是字符串变量，也可以是字符串常数，例如： “Hello”.equals(t);

n    要测试两个字符串除了大小写区别外是否是相等的，需要使用equalsIgnoreCase方法，例如：

       “Hello”.equalsIgnoreCase(“hellO”);//true

n    判断字符串是否相等不要使用“==”

|  |
| --- |
| String g1="Hello World!";  String g2="Hello World!";  boolean b1=g1.equals(g2);  System.out.println("\tg1.equals(g2) 是否相等：" + (b1 ? "相等" : "不相等") );  boolean b2=(g1==g2);  System.out.println("\tg1==g2 是否相等：" + (b2 ? "相等" : "不相等") );  System.out.println("=========================================");  String  s1=new String("Hello World!");  String  s2=new String("Hello World!");  boolean b3=s1.equals(s2);  System.out.println("\ts1.equals(s2) 是否相等：" + (b3 ? "相等" : "不相等") );  boolean b4=(s1==s2);  System.out.println("\ts1==s2 是否相等：" + (b4 ? "相等" : "不相等"));  }  }  /\*\*   \*   g1.equals(g2) 是否相等：相等   \*   g1==g2 是否相等：相等   \*======================================   \*   s1.equals(s2) 是否相等：相等   \*   s1==s2 是否相等：不相等   \*   \*   问: 为什么会产生以上不同的结果，你能说明其原因吗？   \*   (或者 画内存变化图来说明)   \*/ |

# 

# 数组概述和特点

        数组的定义：数组是相同类型数据的有序集合. 数组描述的是相同类型的若干个数据,按照一定的先后次序排列组合而成。其中,每一个数据称作一个数组元素,每个数组元素可以通过一个下标来访问它们.

 n 数组的四个基本特点：

    1.    其长度是确定的。数组一旦被创建，它的大小就是不可以改变的。

    2.    其元素必须是相同类型,不允许出现混合类型。

    3.    数组中的元素可以是任何数据类型，包括基本类型和引用类型。

    4.    数组变量属引用类型，数组也可以看成是对象，数组中的每个元素相当于该对象的成员变量。数组本身就是对象，Java中对象是在堆中的，因此数组无论保存原始类型还是其他对象类型，数组对象本身是在堆中的。

# 创建数组和初始化

n    声明（一维数组的声明方式有两种）

type[]   arr\_name;（推荐使用这中方式）

type   arr\_name[];

注：

    1.    声明的时候并没有实例化任何对象，只有在实例化数组对象时，JVM才分配空间，这时才与长度有关。因此，声明数组时不能指定其长度(数组中元素的个数)，例如： int a[5]; //非法

    2.    声明一个数组的时候并没有数组被真正的创建。

    3.    构造一个数组，必须指定长度。

n    构建数组：

创建基本类型一维数组示例：

|  |  |
| --- | --- |
| public class Test{           public static void main(String args[]){         int[] s = null;   // 1         s = new int[10];  //2         for ( int i=0; i<10; i++ ) {                        s[i] =2\*i+1;           System.out.println(s[i]);         }  //3             }      }  http://www.sxt.cn/editor/attached/image/20151101/20151101113616_277.png  http://www.sxt.cn/editor/attached/image/20151101/20151101113637_432.pnghttp://www.sxt.cn/editor/attached/image/20151101/20151101113647_491.png |  |

创建引用类型一维数组示例：

|  |  |
| --- | --- |
| class Man{      private int age;      private int id;  public Man(int id,int age) {      super();  this.age = age;  this.id = id;  }  }  http://www.sxt.cn/editor/attached/image/20151101/20151101113953_282.pnghttp://www.sxt.cn/editor/attached/image/20151101/20151101114014_544.png   public class AppMain {  public static void main(String[] args) {  Man[] mans;  mans = new Man[10];  //1  http://www.sxt.cn/editor/attached/image/20151101/20151101114047_233.png  Man m1 = new Man(1,11);  Man m2 = new Man(2,22);  //2  mans[0]=m1;  mans[1]=m2;  //3  }  }  http://www.sxt.cn/editor/attached/image/20151101/20151101114134_235.png |  |

n    初始化

     1.   静态初始化

除了用new关键字来产生数组以外,还可以直接在定义数组的同时就为数组元素分配空间并赋值。

int[] a = {1,2,3};

Man[] mans = {

new Man(1,1),

new Man(2,2)

};

    2.    动态初始化

数组定义与为数组元素分配空间并赋值的操作分开进行。

int[] a = new int[2];

a[0]=1;

a[1]=2;

    3.数组的默认初始化：

    数组是引用类型，它的元素相当于类的实例变量，因此数组一经分配空间，其中的每个元素也被按照实例变量同样的方式被隐式初始化。

int a[] = new int[2];  //0,0

boolean [] b = new boolean[2];  //false,false

String[] s = new String[2];     //null, null

    3.    数组的界限

1.    下标的合法区间：[0, length-1]

# 代码

|  |
| --- |
| package cn.bjsxt.array;   public class Car {  String name;  public Car(String name){  this.name=  name;  }  } |
| package cn.bjsxt.array;   /\*\*   \* 数组的基本概念   \* @author dell   \*   \*/  public class Test01 {  public static void main(String[] args) {  /\*\*   \* 1. 数组是相同数据类型(数据类型可以为任意类型)的有序集合   \* 2. 数组也是对象。数组元素相当于对象的成员变量(详情请见内存图)   \* 3. 数组长度的确定的，不可变的。如果越界，则报：ArrayIndexOutofBoundsException   \*/      int[] a = new int[3];      a[0] = 23;      a[1] = 28;      a[2] = 32;   Car[] cars = new Car[4];      cars[0] = new Car("奔驰");      System.out.println(a.length);      for(int i=0;i<a.length;i++){      System.out.println(a[i]);      }  }  } |

# 

# 初始化

    静态初始化

除了用new关键字来产生数组以外,还可以直接在定义数组的同时就为数组元素分配空间并赋值。

    int[] a = {1,2,3};

    Man[] mans = {

    new Man(1,1),

    new Man(2,2)

};

    动态初始化

数组定义与为数组元素分配空间并赋值的操作分开进行。

    int[] a = new int[2];

    a[0]=1;

    a[1]=2;

      数组的默认初始化：

    数组是引用类型，它的元素相当于类的实例变量，因此数组一经分配空间，其中的每个元素也被按照实例变量同样的方式被隐式初始化。

int a[] = new int[2];  //0,0

boolean [] b = new boolean[2];  //false,false

String[] s = new String[2];     //null, null

n    数组的界限

1.    下标的合法区间：[0, length-1]

# 代码

|  |
| --- |
| package cn.bjsxt.array;   public class Car {      String name;      public Car(String name){      this.name=  name;  }  } |
| package cn.bjsxt.array;   /\*\*   \* 数组的基本语法   \* @author dell   \*   \*/  public class Test02 {  public static void main(String[] args) {  //声明      int[] a;      int b[];  //创建数组对象      a = new int[4];      b = new int[5];  //初始化(对数组元素的初始化)  //默认初始化：数组元素相当于对象的成员变量，默认值跟成员变量的规则一样。数字0，布尔false,char\u0000,引用：null  //动态初始化：  for(int i=0;i<a.length;i++){      a[i] = i\*12;  }  //静态初始化      int c[] = {23,43,56,78};   //长度：4，索引范围：[0,3]      Car[] cars = {      new Car("奔驰"),      new Car("比亚迪"),      new Car("宝马")  };      Car c2 = new Car("奔驰");      System.out.println(c2==cars[0]);  }  } |

# 

# String类的常用方法

      String 类对象保存不可修改的Unicode字符序列

  String类的下述方法能创建并返回一个新的String对象: concat, replace,  substring, toLowerCase, toUpperCase, trim.

n    提供查找功能的有关方法: endsWith, startsWith, indexOf,，lastIndexOf.

n    提供比较功能的方法: equals, equalsIgnoreCase,  compareTo.

n    其它方法: charAt ，length.

n    public static String valueOf(…)可以将基本类型数据转换为字符串。

n    ？？？？查资料，列出所有常用方法的列表和解释

|  |  |  |
| --- | --- | --- |
| 方法  ( 高淇整理) | 中文和英文解释  ( 高淇整理) | 源代码和解释  ( 高淇整理) |
| length() | Returns the length of this string.  返回字符串的长度 | public int length() {  return value.length;  //返回字符数组的长度  } |
|  |  |  |

# String类的JDK源码分析

如下只列出核心源代码，跟我们的理解暂时不相关的直接省去，想看最初的源代码请自己查看JDK源码。

|  |
| --- |
| public final class String  {  /\*\* The value is used for character storage. \*/  private final char value[];  public String() {          this.value = new char[0];  }  public String(String original) {          this.value = original.value;  }  public String(char value[]) {          this.value = Arrays.copyOf(value, value.length);  }  public int length() {          return value.length;  }  public boolean isEmpty() {          return value.length == 0;  }  public char charAt(int index) {          if ((index < 0) || (index >= value.length)) {              throw new StringIndexOutOfBoundsException(index);          }          return value[index];  }  public boolean equals(Object anObject) {          if (this == anObject) {              return true;          }          if (anObject instanceof String) {              String anotherString = (String) anObject;              int n = value.length;              if (n == anotherString.value.length) {                  char v1[] = value;                  char v2[] = anotherString.value;                  int i = 0;                  while (n-- != 0) {                      if (v1[i] != v2[i])                              return false;                      i++;                  }                  return true;              }          }          return false;   }  //其余方法省略。大家把如上列出的方法读懂，应该就完全理解了String类的核心了，而且通过String类的学习也掌握了数组的基本应用  } |

# 代码

|  |
| --- |
| package cn.bjsxt.string;  /\*\*   \* String：不可变字符序列！   \* 三个作业：   \* 1. 练习String类的常用方法   \* 2. 结合数组查看源码   \* 3. 提高：按照老师的方法将String类中相关方法的源码看一看。   \* @author dell   \*   \*/  public class TestString {  public static void main(String[] args) {      String str = new String("abcd");      String str2 = new String("abcd");      System.out.println(str2.equals(str));   //比较内容是否相等。      System.out.println(str2==str);       System.out.println(str.charAt(2));      String str3 = "def";      String str4 = "def";      System.out.println(str3.equals(str4));      System.out.println(str3==str4);      System.out.println(str3.indexOf('y'));      String s = str3.substring(0);      System.out.println(s);      String str5 = str3.replace('e', '\*');      System.out.println(str5);      String str6 = "abcde,rrtt,cccee";      String[] strArray = str6.split(",");      for(int i=0;i<strArray.length;i++){      System.out.println(strArray[i]);  }      String str7 = "  aa  bb  ";      String str77 = str7.trim();      System.out.println(str77.length());      System.out.println("Abc".equalsIgnoreCase("abc"));      System.out.println("Abcbd".indexOf('b'));      System.out.println("Abcbd".lastIndexOf('b'));      System.out.println("Abcbd".startsWith("Ab"));      System.out.println("Abcbd".endsWith("bd"));      System.out.println("Abcbd".toLowerCase());      System.out.println("Abcbd".toUpperCase());      System.out.println("##################");      String gh = new String("a");      for (int i = 0; i < 1000; i++) {      gh = gh + i;  }      System.out.println(gh);  }  } |

# 

# StringBuilder与StringBuffer

 一.   StringBuilder可变字符串，线程不太安全，但效率高，推荐使用。

    StringBuffer可变字符串，线程安全，但效率低，不推荐使用。

    String 是不可变字符串，根据实时情况使用，在循环当中不要使用，因为会产生好的没用的字符串。

二．   String,StringBuilder要学会查看api的良好习惯，通过api学会使用常见的方法，自己利用案例来理解每个方法的具体含义。这样你会更深刻理解每个方法的使用。

三．   如果看不懂api可以到我们官网去看三百集的第56与57集，理解一下字符串的扩容原理。方便理解后面容器的内容。

# 

# StringBuilder与StringBuffer区别与常见方法补充

    StringBuilder可变字符串，线程不太安全，但效率高，推荐使用。

    StringBuffer可变字符串，线程安全，但效率低，不推荐使用。

    String 是不可变字符串，根据实时情况使用，在循环当中不要使用，因为会产生好的没用的字符串。

# 通过实际代码来展示每个方法的用法

public class Test {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder("abcdefghijklmnopqrstuvwxyz");

sb.append(true).append("ABC");// append(参数)添加方法，返回值是this

System.out.println(sb);

sb.delete(3, 5);// 删除方法是[start end)这样的一个规律。

System.out.println(sb);

sb.replace(3, 5, "Hello");// 替换方法是从start 开始 end结束之间插入字符串。

System.out.println(sb);

System.out.println(sb.length());// 还有一些一看就明白的方法自己尝试一下。

}

}

# 面试题的回答

       StringBuilder 与StringBuffer的区别，StringBuilder与String的区别。

1）StringBuilder效率高，线程不安全，StringBuffer效率低，线程安全。

2）String是不可变字符串，StringBuilder是可变字符串。为什么有这样的差异，可以深入源码去解析，比如String类内的 priver final  char  value[] 等方法的原因。

3）如果是简单的声明一个字符串没有后续过多的操作，使用String,StringBuilder均可，若后续对字符穿做频繁的添加，删除操作,或者是在循环当中动态的改变字符穿的长度应该用StringBuilder。使用String会产生多余的字符串，占用内存空间。

# 

# 数组\_模拟ArrayList容器的底层实现\_JDK源码分析代码

package com.bjsxt.string;

import java.util.ArrayList;

public class MyArrayList {

private Object value[];

private int size;

public MyArrayList() {

// value = new Object[16];

this(2);

}

public MyArrayList(int size) {

value = new Object[size];

}

public void add(Object obj) {

value[size] = obj;

size++;

if (size >= value.length) {

// 装不下了，将要扩展容量了

int newCapacity = value.length \* 2;

Object[] newList = new Object[newCapacity];

// System.arraycopy(src, srcPos, dest, destPos, length);

for (int i = 0; i < value.length; i++) {

newList[i] = value[i];

}

value = newList;

}

}

public int size() {

return size;

}

public boolean isEmpty() {

return size == 0;

}

public int indexOf(Object obj) {

if (obj == null) {

return -1;

} else {

for (int i = 0; i < value.length; i++) {

if (obj == value[i]) {

return i;

}

}

return -1;

}

}

public int lastIndexOf(Object obj) {

if (obj == null) {

return -1;

} else {

for (int i = value.length - 1; i >= 0; i--) {

if (obj == value[i]) {

return i;

}

}

return -1;// return 作用：1，返回出去值，2，结束语句。

}

}

public void rangleCheck(int index) {

if (index < 0 || index > size - 1) {// [0,size-1]

try {

throw new Exception();

} catch (Exception e) {

e.printStackTrace();

}

}

}

public Object set(int index, Object object) {

rangleCheck(index);

Object old = value[index];

value[index] = object;

return old;

}

public Object get(int index) {

rangleCheck(index);

return value[index];

}

public static void main(String[] args) {

MyArrayList list = new MyArrayList();

list.add("aaa");

list.add(new Human("高淇"));

list.add("bbb");

list.add("bbb");

list.add("bbb");

list.add("bbb");

Human h = (Human) list.get(1);

System.out.println(h.getName());

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(list.get(0));

System.out.println(list.get(1));

System.out.println(list.get(2));

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(list.size);

}

}

class Human {

private String name;

public Human(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

# 

# 数组概述和特点

    数组的定义：数组是相同类型数据的有序集合. 数组描述的是相同类型的若干个数据,按照一定的先后次序排列组合而成。其中,每一个数据称作一个数组元素,每个数组元素可以通过一个下标来访问它们.

 数组的四个基本特点：

    1.    其长度是确定的。数组一旦被创建，它的大小就是不可以改变的。

    2.    其元素必须是相同类型,不允许出现混合类型。

    3.    数组中的元素可以是任何数据类型，包括基本类型和引用类型。

    4.    数组变量属引用类型，数组也可以看成是对象，数组中的每个元素相当于该对象的成员变量。数组本身就是对象，Java中对象是在堆中的，因此数组无论保存原始类型还是其他对象类型，数组对象本身是在堆中的。

# 创建数组和初始化

        声明（一维数组的声明方式有两种）

    type[]   arr\_name;（推荐使用这中方式）

    type   arr\_name[];

注：

    1.    声明的时候并没有实例化任何对象，只有在实例化数组对象时，JVM才分配空间，这时才与长度有关。因此，声明数组时不能指定其长度(数组中元素的个数)，例如： int a[5]; //非法

    2.    声明一个数组的时候并没有数组被真正的创建。

    3.    构造一个数组，必须指定长度。

构建数组：

创建基本类型一维数组示例：

|  |  |
| --- | --- |
| public class Test{           public static void main(String args[]){         int[] s = null;   // 1         s = new int[10];  //2         for ( int i=0; i<10; i++ ) {                        s[i] =2\*i+1;           System.out.println(s[i]);         }  //3             }      } |  |

 创建引用类型一维数组示例：

|  |  |
| --- | --- |
| class Man{  private int age;  private int id;  public Man(int id,int age) {  super();  this.age = age;   this.id = id;  }  }   public class AppMain {  public static void main(String[] args) {  Man[] mans;  mans = new Man[10];  //1  Man m1 = new Man(1,11);  Man m2 = new Man(2,22);  //2  mans[0]=m1;   mans[1]=m2;  //3  }  } |  |

     初始化

        静态初始化

除了用new关键字来产生数组以外,还可以直接在定义数组的同时就为数组元素分配空间并赋值。

int[] a = {1,2,3};

Man[] mans = {

new Man(1,1),

new Man(2,2)

};

    动态初始化

数组定义与为数组元素分配空间并赋值的操作分开进行。

int[] a = new int[2];

a[0]=1;

a[1]=2;

数组的默认初始化：

    数组是引用类型，它的元素相当于类的实例变量，因此数组一经分配空间，其中的每个元素也被按照实例变量同样的方式被隐式初始化。

int a[] = new int[2];  //0,0

boolean [] b = new boolean[2];  //false,false

String[] s = new String[2];     //null, null

数组的界限

1.    下标的合法区间：[0, length-1]

 1定义4特点3初始化方式！

# 数组常见操作

## 数组的拷贝

|  |
| --- |
| System类里也包含了一个static void arraycopy（object src，int srcpos，object dest， int destpos，int length）方法，该方法可以将src数组里的元素值赋给dest数组的元素，其中srcpos指定从src数组的第几个元素开始赋值，length参数指定将src数组的多少个元素赋给dest数组的元素。  String[] s = {"Mircosoft","IBM","Sun","Oracle","Apple"};      String[] sBak = new String[6];      System.arraycopy(s,0,sBak,0,s.length); |

n    java.util.Arrays

1.    打印数组

|  |
| --- |
| int[] a = {1,2};  System.out.println(a);   //[I@35ce36  System.out.println(Arrays.toString(a));  //[1, 2] |

## 数组排序

|  |
| --- |
| int[] a = {1,2,323,23,543,12,59};  System.out.println(Arrays.toString(a));  Arrays.sort(a);  System.out.println(Arrays.toString(a));  结果：  [1, 2, 323, 23, 543, 12, 59]  [1, 2, 12, 23, 59, 323, 543] |
| 如果数组元素是引用类型，那么需要怎么做呢？(今天先不讲！后面讲容器时再讲！)   public class Test{  public static void main(String[] args) {  Man[] msMans = {new Man(3,"a"),new Man(60,"b"),new Man(2,"c")};  Arrays.sort(msMans);  System.out.println(Arrays.toString(msMans));  }  }  class Man implements Comparable {  int age;  int id;  String name;  public Man(int age, String name) {  super();  this.age = age;  this.name = name;  }  public String toString(){  return this.name;  }    public int compareTo(Object o) {  Man man = (Man) o;  if(this.age<man.age){  return -1;  }  if(this.age>man.age){  return 1;  }  return 0;  }    } |

 2. 二分法查找

|  |
| --- |
| int[] a = {1,2,323,23,543,12,59};  Arrays.sort(a);   //使用二分法查找，必须先对数组进行排序  System.out.println("该元素的索引："+Arrays.binarySearch(a, 12)); |

 3. 复制

|  |
| --- |
| 一般用System. arraycopy |

 4. 填充

|  |
| --- |
| Arrays.fill(a, 2, 4, 100);  //将2到4索引的元素替换为100 |

5.    asList(暂时不讲，讲完容器后再说。事实上用的也不多)

|  |
| --- |
| int[] a = {1,2,323,23,543,12,59};  List<int[]> list = new ArrayList<int[]>();  list = Arrays.asList(a);   //将数组转成成list对象中的一个元素  System.out.println(list.get(0)[1]); |

## 多维数组

多维数组可以看成以数组为元素的数组。例如：

静态初始化：

|  |
| --- |
| int[][] a = {  {1,2,3},  {3,4},   {3,5,6,7}  };  System.out.println(a[2][3]); |

动态初始化：

|  |
| --- |
| int[][] a = new int[3][];  //a[0] = {1,2,5};   //错误，没有声明类型就初始化  a[0]=new int[]{1,2};  a[1]=new int[]{2,2};  a[2]=new int[]{2,2,3,4};  System.out.println(a[2][3]); |

 获取数组长度：

a.length获取的二维数组第一维数组的长度，a[0].length才是获取第二维第一个数组长度。

## **附录**

(面试前复习一下！！)

### 冒泡排序

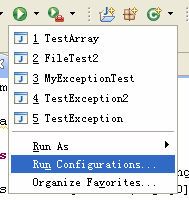
|  |
| --- |
| public class Test {  public static void main(String[] args) {      int[] values = { 3, 1, 6, 2, 9, 0, 7, 4, 5,8 };      sort(values);      System.out.println(Arrays.toString(values));  }  public static void sort(int[] values) {      int temp;      for (int i = 0; i < values.length; i++) {      for (int j = 0; j < values.length - 1- i ; j++) {      if (values[j] > values[j + 1]) {      temp = values[j];      values[j] = values[j + 1];      values[j + 1] = temp;  }  }  }  }  } |

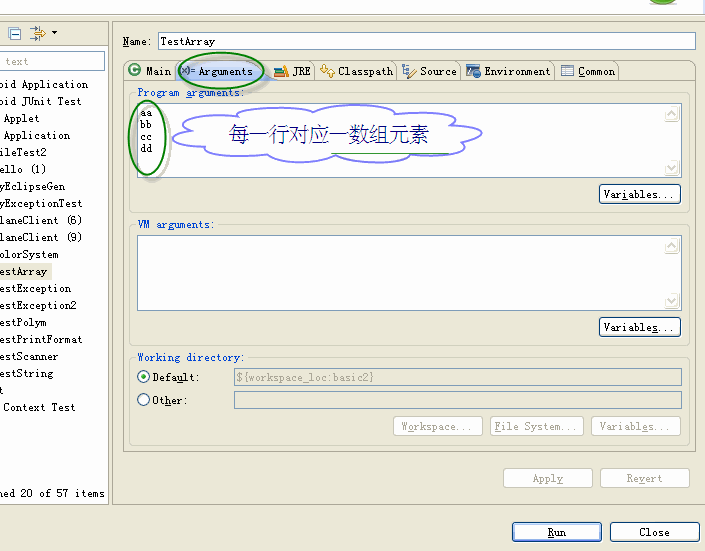
### 二分法查找

|  |
| --- |
| import java.util.Arrays;   public class TestBinarySearch {  public static void main(String[] args) {      System.out.println(args[1]);      int[] arr = {234,245,77,3,543,67,78,95,378,678,205,753,457,2903,340} ;      int searchWord = 6780;  //所要查找的数      int searchCount = 0;  //循环的次数      System.out.printf("普通循环查找%d的次数是%d",searchWord,genetalLoop(arr,searchWord));  System.out.printf("二分法查找%d的次数是%d",searchWord,binarySearch(arr,searchWord));  }  static int genetalLoop(int[] arr,int searchWord){  //普通的循环法，最少需要比较一次，比如查找1，最多需要比较15次，比如8721  int searchCount = 0;  for(int i=0;i<arr.length;i++){  searchCount++;  if (searchWord==arr[i])  break;  }  return searchCount;  }  //int[] arr = {234,245,77,3,543,67,78,95,378,678,205,753,457,2903,340};  static int binarySearch(int[] arr,int searchWord){  Arrays.sort(arr);  //先对传进来的数组进行排序  System.out.println("\n"+Arrays.toString(arr));  //二分法查找  int iIndex=0; //相当于指针的东西  int iStart=0;  int iEnd=arr.length-1;  int searchCount = 0;  for(int i=0;i<arr.length/2;i++) {  searchCount++;  iIndex = (iStart+iEnd)/2;  if(arr[iIndex]<searchWord){  System.out.println("aa");  iStart = iIndex;  }else if(arr[iIndex]>searchWord){  System.out.println("bb");  iEnd = iIndex;  }else{  break;  }  }  return searchCount;  }  } |

### 命令行参数的问题

|  |
| --- |
| public class Test {      public static void main(String[] args) {    for ( int i = 0; i < args.length; i++ ) {              System.out.println("args[" + i + "] = " + args[i]);  }         }     }  //运行程序Test13\_1.java  java Test  lisa   bily  “Mr  Brown”  //输出结果：  args[0] = lisa  args[1] = bily  args[2] = Mr Brown   eclipse里如何使用？ |





### 增强for循环

 (JDK5.0后增加的，一般用于读取数组或集合中所有的元素！)

|  |
| --- |
| String[] ss = {"aa","bbb","ccc","ddd"};  for (int i = 0; i < ss.length; i++) {  String temp = ss[i];  System.out.println(temp);  }   for (String temp : ss) {  System.out.println(temp);  } |

**上机作业**

1.    完成课堂上老师的代码

2.    利用二维数组实现：n阶矩阵的加法、乘法操作。(大家可以使用double类型，int容易产生溢出！)  难度较高

|  |
| --- |
| public class TestMatrix {  public static void main(String[] args){  int[][] a = {  {2,2,4,6},  {4,5,6,8},  {1,2,32,3},  {1,2,3,4}  } ;  int[][] b = {  {80,2,4,7},  {6,5,6,2},  {9,7,2,3},  {9,3,2,3}  } ;  mutiple(a,b);  }  static void mutiple(int[][] a,int[][] b){  int[][] c = new int[a.length][a.length];  for(int i=0;i<a.length;i++){  for(int j=0;j<a[i].length;j++){  for(int k=0;k<a[i].length;k++){  c[i][j] += a[i][k]\*b[k][j];  }  //    c[i][j] = a[i][0]\*b[0][j]+a[i][1]\*b[1][j]+a[i][2]\*b[2][j];  }  }  printArray(c);  }  static void add(int[][] a,int[][] b){  int[][] c = new int[a.length][a.length];  for(int i=0;i<a.length;i++){  for(int j=0;j<a[i].length;j++){  c[i][j] = a[i][j] +b[i][j] ;  }  }  printArray(c);  }  static void printArray(int[][] c){  for(int i=0;i<c.length;i++){  for(int j=0;j<c[i].length;j++){      //sddfdsfds  //System.out.printf((c[i][j]<10?" "+c[i][j]:c[i][j])+" ");  System.out.printf("%5d",c[i][j]);  }  System.out.println("");  }  }  } |

# 

# 常见的排序方法：

## 冒泡排序：

import java.util.Arrays;

public class Demo01 {

public static void main(String[] args) {

int[] values = { 3, 1, 6, 2, 9, 0, 7, 4, 5, 8 };

sort(values);

System.out.println(Arrays.toString(values));

}

public static void sort(int[] values) {

int temp;//声明一个第三变量用来存储中间的值

for (int i = 0; i < values.length; i++) {

//第一层循环是为了数据遍历开来

for (int j = 0; j < values.length - 1 - i; j++) {

//当i=0时候和下面的数据一一比较

if (values[j] > values[j + 1]) {

temp = values[j];

values[j] = values[j + 1];

values[j + 1] = temp;

}

}

}

}

}

[尝试一下](http://www.sxt.cn/searchsxt/tr?oper=l&&i=226)

## 二分法查找：

       条件

          1.首先要排序。

import java.util.Arrays;

public class Demo01 {

public static void main(String[] args) {

System.out.println(args[1]);

int[] arr = { 234, 245, 77, 3, 543, 67, 78, 95, 378, 678, 205, 753, 457, 2903, 340 };

int searchWord = 6780; // 所要查找的数

int searchCount = 0; // 循环的次数

System.out.printf("普通循环查找%d的次数是%d", searchWord, genetalLoop(arr, searchWord));

System.out.printf("二分法查找%d的次数是%d", searchWord, binarySearch(arr, searchWord));

}

static int genetalLoop(int[] arr, int searchWord) {

// 普通的循环法，最少需要比较一次，比如查找1，最多需要比较15次，比如8721

int searchCount = 0;

for (int i = 0; i < arr.length; i++) {

searchCount++;

if (searchWord == arr[i])

break;

}

return searchCount;

}

// int[] arr = {234,245,77,3,543,67,78,95,378,678,205,753,457,2903,340};

static int binarySearch(int[] arr, int searchWord) {

Arrays.sort(arr); // 先对传进来的数组进行排序

System.out.println("\n" + Arrays.toString(arr));

// 二分法查找

int iIndex = 0; // 相当于指针的东西

int iStart = 0;

int iEnd = arr.length - 1;

int searchCount = 0;

for (int i = 0; i < arr.length / 2; i++) {

searchCount++;

iIndex = (iStart + iEnd) / 2;

if (arr[iIndex] < searchWord) {

System.out.println("aa");

iStart = iIndex;

} else if (arr[iIndex] > searchWord) {

System.out.println("bb");

iEnd = iIndex;

} else {

break;

}

}

return searchCount;

}

}

[尝试一下](http://www.sxt.cn/searchsxt/tr?oper=l&&i=227)

## 增强for循环

 (JDK5.0后增加的，一般用于读取数组或集合中所有的元素！)

|  |
| --- |
| String[] ss = {"aa","bbb","ccc","ddd"};  for (int i = 0; i < ss.length; i++) {  String temp = ss[i];  System.out.println(temp);  }    for (String temp : ss) {  System.out.println(temp);  } |

# 

# 基本数据类型的包装类

## 包装类基本知识

     JAVA并不是纯面向对象的语言。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。但是我们在实际使用中经常需要将基本数据转化成对象，便于操作。比如：集合的操作中。例如使用Map对象要操作put()方法时，需要传入的参数是对象而不是基本数据类型。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)。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：

|  |  |
| --- | --- |
| 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。

## 包装类的用途

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（这些操作方法的作用是在基本类型数据、包装类对象、字符串之间提供转化！）。

所有的包装类(Wrapper Class)都有共同的方法,他们是:

/\*\*

 \* 测试Integer的用法，其他包装类类似

 \*/

public class Wraper {

public static void main(String[] args) {

// 基本类型转换成Integer对象

Integer int1 = new Integer(10);

Integer int2 = Integer.valueOf(20);

// Integer对象转换成int

int a = int1.intValue();

// 字符串转化成Integer对象

Integer int3 = Integer.parseInt("334");

Integer int4 = new Integer("999");

// Integer对象转化成字符串

String str1 = int3.toString();

// 一些常见的int类型相关的常量。

System.out.println("int能表示的最大整数：" + Integer.MAX\_VALUE);

System.out.println("int能表示的最小整数：" + Integer.SIZE);

// 数字转字符串非常简单

String str = 234 + "";

System.out.println(str);

}

}

# 

# 自动装箱和拆箱？autoboxing，unboxing

   就是将基本类型和包装类进行自动的互相转换。

    JDK5.0后，将自动装箱/拆箱引入java中。  
自动装箱的过程：每当需要一种类型的对象时，这种基本类型就自动地封装到与它相同类型的包装中。  
自动拆箱的过程：每当需要一个值时，被装箱对象中的值就被自动地提取出来，没必要再去调用intValue()和doubleValue()方法。

自动装箱与拆箱的功能事实上是编译器来帮您的忙，编译器在编译时期依您所编写的语法，决定是否进行装箱或拆箱动作。例如：

Integer i = 100;

相当于编译器自动为您作以下的语法编译：

Integer i = new Integer(100);

所以自动装箱与拆箱的功能是所谓的“编译器蜜糖”(Compiler Sugar)，虽然使用这个功能很方便，但在程序运行阶段您得了解Java的语义。例如下面的程序是可以通过编译的：

Integer i = null;

int j = i;

这样的语法在编译时期是合法的，但是在运行时期会有错误，因为这种写法相当于：

Integer i = null;

int j = i.intValue();

null表示i没有参考至任何的对象实体，它可以合法地指定给对象参考名称。由于实际上i并没有参考至任何的对象，所以也就不可能操作intValue()方法，这样上面的写法在运行时会出现NullPointerException错误。

/\*\*

 \* 测试自动装箱和拆箱

 \* 结论：虽然很方便，但是如果不熟悉特殊情况，可能会出错！

 \*/

static void testBoxing(){

Integer b = 23;    //自动装箱

int a = new Integer(20);    //自动拆箱

//自动装箱和拆箱，实际上是编译器替我们完成了代码的自动编译，比如：Integer b = 23, 其实运行时执行的仍然是：Integer b = new Integer(23);

//下面的问题我们需要注意：

// Integer  c = null;

// int d = c;    //此处其实就是：c.intValue(),因此抛空指针异常。

//自动装箱拆箱时，对于-128-127之间的值,编译器仍然会把它当做基本类型处理。

Integer h = 100; Integer u = 100;

Integer h2 = 200; Integer u2 = 200;

if(h==u){

System.out.println("100等于");

}

if(h2==u2){

System.out.println("200等于");

}

}

自动装箱和自动拆箱的案例代码：

public class AutoBox {

public static void main(String[] args) {

//Integer i= new Integer(10);

Integer i1=100;//jdk5.0之后，编译器帮我们改进代码相当于Integer i1=new Integer(100);

Integer i2=2000;

int c=i2;//自动拆箱，编译器帮我们改进：i2.intValue();

System.out.println(c);

Integer d=1234;

Integer d2=1234;

System.out.println(d==d2);

System.out.println(d.equals(d2));

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Integer d3=-129;//[-128,127]之间的数，仍然当做基本数据类型来处理。

Integer d4=-129;

System.out.println(d3==d4);

System.out.println(d3.equals(d4));

}

}

# 

# 时间处理相关类

## Date时间类(java.util.Date)

    在标准Java类库中包含一个Date类。它的对象表示一个特定的瞬间，精确到毫秒。

[Date](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html)()分配一个Date对象，并初始化此对象为当前的日期和时间精确到毫秒）。

[Date](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html)(long date) 分配 Date 对象并初始化此对象，以表示自从标准基准时间（称为“历元（epoch）”，即 1970 年 1 月 1 日 00:00:00 GMT）以来的指定毫秒数。

    Boolean [after](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html)([Date](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html) when) 测试此日期是否在指定日期之后。

    boolean [before](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html)([Date](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html) when)  测试此日期是否在指定日期之前。

        Boolean [equals](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html)([Object](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/lang/Object.html) obj) 比较两个日期的相等性。

        Long [getTime](http://www.sxt.cn/html_zh_CN_1.5/html/zh_CN/api/java/util/Date.html)()返回自 1970 年 1 月 1 日 00:00:00 GMT 以来此 Date 对象表示的毫秒数。

        String [toString](mk:@MSITStore:F:\book\J2SE\J2SE_5.0_API.chm::/java/util/Date.html)() 把此 Date 对象转换为以下形式的 String：

   dow mon dd hh:mm:ss zzz yyyy 其中： dow 是一周中的某一天 (Sun, Mon, Tue, Wed, Thu, Fri, Sat)。

public class TestDate {

public static void main(String[] args) {

// 要去看看Date类的源码，可以深入理解Date类。

Date d = new Date();// 得到了当前时间的毫秒数。

long l1 = System.currentTimeMillis();

System.out.println(l1);

Date d2 = new Date(1000);

System.out.println(d2.toString());

System.out.println(d2.toGMTString());// 不建议使用。

d2.setTime(24324324);

System.out.println(d2.getTime());

System.out.println(d.getTime() < d2.getTime());

Date date1 = new Date();

System.out.println(date1.toString());

long i = date1.getTime();

Date date2 = new Date(i - 1000);

Date date3 = new Date(i + 1000);

System.out.println(date1.after(date2));

System.out.println(date1.before(date2));

System.out.println(date1.equals(date2));

System.out.println(date1.after(date3));

System.out.println(date1.before(date3));

System.out.println(date1.equals(date3));

System.out.println(

new Date(1000L \* 60 \* 60 \* 24 \* 365 \* 39L).toString()); // why?

}

}

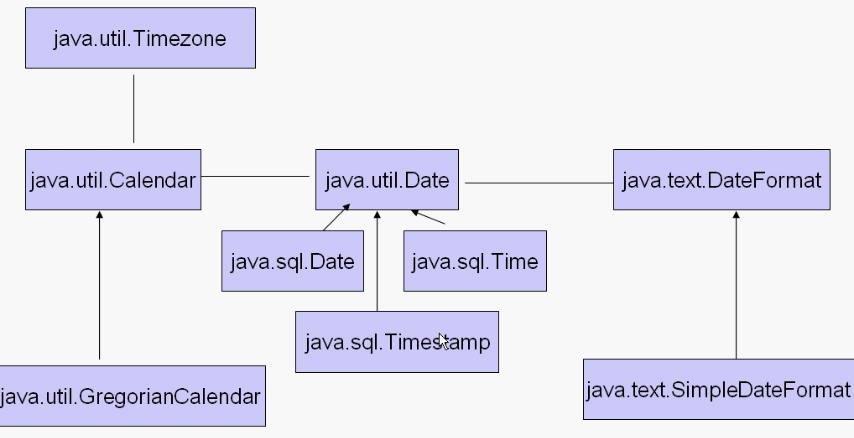
[尝试一下](http://www.sxt.cn/searchsxt/tr?oper=l&&i=230)

     查看API文档大家可以看到很多方法过时了，JDK1.1之前的Date包含了：日期操作、字符串转化成时间对象，时间对象。  1.1之后，日期操作使用：Canlendar类来。 字符串转化：DateFormat。

# 

# DateFormat和SimpleDateFormat类

作用：把时间对象转化成指定格式的字符串。反之，把指定格式的字符串转化成时间对象。



   DateFormat是一个抽象类。一般使用SimpleDateFormat类。

示例：

public class TestDate {

public static void main(String[] args) throws ParseException {

// new出SimpleDateFormat对象

SimpleDateFormat s1 = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

SimpleDateFormat s2 = new SimpleDateFormat("yyyy-MM-dd");

// 将时间对象转换成字符串

String daytime = s1.format(new Date());

System.out.println(s2.format(new Date()));

System.out.println(new SimpleDateFormat("hh:mm:ss").format(new Date()));

// 将符合指定格式的字符串转成成时间对象.字符串格式需要和指定格式一致。

String time = "2007-10-7";

Date date = s2.parse(time);

System.out.println("date1: " + date);

time = "2007-10-7 20:15:30";

date = s1.parse(time);

System.out.println("date2: " + date);

}

}

[尝试一下](http://www.sxt.cn/searchsxt/tr?oper=l&&i=231)

格式化字符的含义：

|  |  |  |  |
| --- | --- | --- | --- |
| 字母 | 日期或时间元素 | 表示 | 示例 |
| G | Era 标志符 | [Text](http://www.sxt.cn/searchsxt/#text) | AD |
| y | 年 | [Year](http://www.sxt.cn/searchsxt/#year) | 1996; 96 |
| M | 年中的月份 | [Month](http://www.sxt.cn/searchsxt/#month) | July; Jul; 07 |
| w | 年中的周数 | [Number](http://www.sxt.cn/searchsxt/#number) | 27 |
| W | 月份中的周数 | [Number](http://www.sxt.cn/searchsxt/#number) | 2 |
| D | 年中的天数 | [Number](http://www.sxt.cn/searchsxt/#number) | 189 |
| d | 月份中的天数 | [Number](http://www.sxt.cn/searchsxt/#number) | 10 |
| F | 月份中的星期 | [Number](http://www.sxt.cn/searchsxt/#number) | 2 |
| E | 星期中的天数 | [Text](http://www.sxt.cn/searchsxt/#text) | Tuesday; Tue |
| a | Am/pm 标记 | [Text](http://www.sxt.cn/searchsxt/#text) | PM |
| H | 一天中的小时数（0-23） | [Number](http://www.sxt.cn/searchsxt/#number) | 0 |
| k | 一天中的小时数（1-24） | [Number](http://www.sxt.cn/searchsxt/#number) | 24 |
| K | am/pm 中的小时数（0-11） | [Number](http://www.sxt.cn/searchsxt/#number) | 0 |
| h | am/pm 中的小时数（1-12） | [Number](http://www.sxt.cn/searchsxt/#number) | 12 |
| m | 小时中的分钟数 | [Number](http://www.sxt.cn/searchsxt/#number) | 30 |
| s | 分钟中的秒数 | [Number](http://www.sxt.cn/searchsxt/#number) | 55 |
| S | 毫秒数 | [Number](http://www.sxt.cn/searchsxt/#number) | 978 |
| z | 时区 | [General time zone](http://www.sxt.cn/searchsxt/#timezone) | Pacific Standard Time; PST; GMT-08:00 |
| Z | 时区 | [RFC 822 time zone](http://www.sxt.cn/searchsxt/#rfc822timezone) | -0800 |

# 

# Calendar 日历类

Calendar 类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY\_OF\_MONTH、HOUR 等 [日历字段](http://www.sxt.cn/searchsxt/#fields)之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。瞬间可用毫秒值来表示，它是距历元（即格林威治标准时间 1970 年 1 月 1 日的 00:00:00.000，格里高利历）的偏移量。

GregorianCalendar 是 Calendar 的一个具体子类，提供了世界上大多数国家/地区使用的标准日历系统。

注意月份的表示，一月是0，二月是1，以此类推，是12月是11。因为大多数人习惯于使用单词而不是使用数字来表示月份，这样程序也许更易读，父类Calendar使用常量来表示月份：JANUARY, FEBRUARY,等等。

package com.bjsxt.string;

import java.util.Calendar;

import java.util.Date;

import java.util.GregorianCalendar;

public class CalendarTest {

public static void main(String[] args) {

Calendar calendar = new GregorianCalendar(2015, 11, 9, 11, 9, 50);

Calendar calendar2 = new GregorianCalendar();

// 设定日期

calendar2.set(Calendar.YEAR, 2015);

calendar2.set(Calendar.MONTH, 11);

calendar2.set(Calendar.DATE, 9);

calendar2.set(Calendar.HOUR\_OF\_DAY, 11);

calendar2.set(calendar.MINUTE, 45);

calendar2.set(calendar.SECOND, 34);

// 得到相关的日期元素。

int year = calendar2.get(Calendar.YEAR);

int month = calendar2.get(Calendar.MONTH);

int Date = calendar2.get(Calendar.DATE);

// 日：Calendar.Date和Calendar.DAY\_OF\_MONTH同义。

int hour = calendar2.get(Calendar.HOUR);

int Minute = calendar2.get(Calendar.MINUTE);

int ss = calendar2.get(Calendar.SECOND);

int week = calendar2.get(Calendar.DAY\_OF\_WEEK);

// 星期几 这里：1-7.周日是1，周一是2，，，，，周六是7

System.out.printf("%d年%d月%d日,星期%d\n", year, month, Date, week);

// 日期计算

GregorianCalendar calendar3 = new GregorianCalendar(2015, 11, 9, 14, 55, 50);

;

calendar3.add(Calendar.MONTH, -7);// 月份减7

calendar3.add(Calendar.DATE, 7);// 增加7天

printCalendar(calendar3);

// 日历对象和时间对象转化。

Date d = calendar3.getTime();

long millSecond = calendar3.getTimeInMillis();

Date date2 = new Date();

GregorianCalendar calendar4 = new GregorianCalendar();

calendar4.setTime(date2);

long g = System.currentTimeMillis();

printCalendar(calendar4);

}

public static void printCalendar(Calendar calendar) {

int year = calendar.get(Calendar.YEAR);

int month = calendar.get(Calendar.MONTH) + 1;

int day = calendar.get(Calendar.DAY\_OF\_MONTH);

int day2 = calendar.get(Calendar.DATE);

// 日：Calendar.Date和Calendar.DAY\_OF\_MONTH

int date = calendar.get(Calendar.DAY\_OF\_WEEK) - 1;// 星期几

String week = "" + ((date == 0) ? "日" : date);

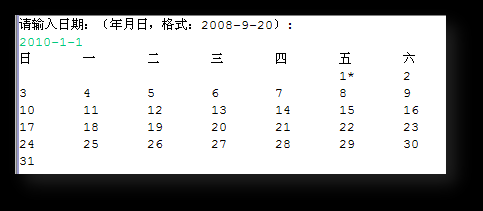
System.out.printf("%d年%d月%d日，星期%s\n", year, month, day, week);

}

}

 编写程序，利用GregorianCalendar类，打印当前月份的日历，样式如下： 可视化日历的编写

今天的日期是 2008-02-23 ,如下为今日所在月份的日历



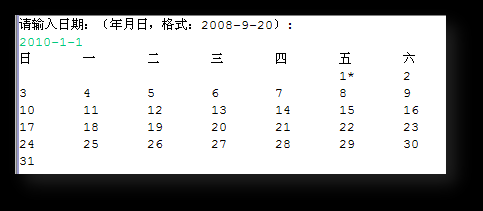
|  |
| --- |
| import java.text.ParseException;  import java.text.SimpleDateFormat;  import java.util.Calendar;  import java.util.Date;  import java.util.GregorianCalendar;  import java.util.Scanner;    public class AppMain {  public static void main(String[] args) throws ParseException {  System.out.println("请输入日期（格式为：2010-3-3）：");  Scanner  scanner = new Scanner(System.in);  String  dateString =scanner.nextLine();    //2010-3-1  //将输入的字符串转化成日期类  System.out.println("您刚刚输入的日期是："+dateString);  String[]  str = dateString.split("-");  int year = Integer.parseInt(str[0]);  int month = new Integer(str[1]);  int day = new Integer(str[2]);  Calendar c = new GregorianCalendar(year,month-1, day);   //Month:0-11 ,1-12    //大家自己补充另一种方式：将字符串通过SImpleDateFormat转化成Date对象，再将Date对象转化成日期类  //  SimpleDateFormat sdfDateFormat  = new SimpleDateFormat("yyyy-MM-dd");  //  Date date = sdfDateFormat.parse(dateString);  //  Calendar c = new GregorianCalendar();  //  c.setTime(date);  //  int day = c.get(Calendar.DATE);    c.set(Calendar.DATE, 1);  int  dow = c.get(Calendar.DAY\_OF\_WEEK);  //week:1-7 日一二三四五六  System.out.println("日\t一\t二\t三\t四\t五\t六");  for(int i=0;i<dow-1;i++){  System.out.print("\t");  }  int maxDate = c.getActualMaximum(Calendar.DATE);  //  System.out.println("maxDate:"+maxDate);  for(int i=1;i<=maxDate;i++){  StringBuilder  sBuilder = new StringBuilder();  if(c.get(Calendar.DATE)==day){  sBuilder.append(c.get(Calendar.DATE)+"\*\t");  }else{  sBuilder.append(c.get(Calendar.DATE)+"\t");  }  System.out.print(sBuilder);  //   System.out.print(c.get(Calendar.DATE)+((c.get(Calendar.DATE)==day)?"\*":"")+"\t");  if(c.get(Calendar.DAY\_OF\_WEEK)==Calendar.SATURDAY){  System.out.print("\n");  }  c.add(Calendar.DATE, 1);  }  }   static void printCalendar(Calendar calendar){  int year = calendar.get(Calendar.YEAR);  int month = calendar.get(Calendar.MONTH)+1;  int day = calendar.get(Calendar.DAY\_OF\_MONTH);  int day2 = calendar.get(Calendar.DATE); //日：Calendar.DATE和Calendar.DAY\_OF\_MONTH同义  int date = calendar.get(Calendar.DAY\_OF\_WEEK)-1; //星期几  String week = ""+(  (date==0)?"日":date   );  System.out.printf("-----%d年%d月%d日,星期%s\n",year,month,day,week);  }  } |

# 

# 可视化日历的编写

编写程序，利用GregorianCalendar类，打印当前月份的日历，样式如下：

今天的日期是 2008-02-23 ,如下为今日所在月份的日历



package com.bjsxt.string;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

import java.util.GregorianCalendar;

import java.util.Scanner;

/\*\*

 \* 思路总结：

 \*  1.第一步先通过具体行动把骨架建起来。

 \*  2.根据题意,利用日历知识解决实际问题，三个知识点

 \*  a)本月的第一天是星期几，确定了前面的空格的个数。int w=Calendar.get(Calendar.Date)

 \*

 \*  b)本月的最后一天是30还是31，还是28，决定的循环的临界值。 c)输入的字符串通过new

 \*  simpleDateFormat("yyyy-MM-dd").parse(str)变成日期对象date。

有Calendar c =new GregorianCalendar();c.setTime(date)方法把日期对象变成日历。

 \*/

public class CalendarTest2 {

public static void main(String[] args) throws ParseException {

System.out.println("请输入日期，例如：2015-11-15");

Scanner scanner = new Scanner(System.in);

String temp = scanner.nextLine();

DateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

Date date = sdf.parse(temp);

Calendar c = new GregorianCalendar();

c.setTime(date);

c.add(Calendar.DATE, 1);// 月份的第一天。

// System.out.println(c.get(Calendar.DAY\_OF\_WEEK));

int maxDate = c.getActualMaximum(Calendar.DATE);

int day = c.get(Calendar.DATE);

// int week = c.get(Calendar.DAY\_OF\_WEEK);

// System.out.println(week);

/\*\*

 \* 1.首先判断当前日期是星期几？ 2.判断当前月是几月？

 \*

 \*/

System.out.println("日\t一\t二\t三\t四\t五\t六");

for (int i = 0; i < c.get(Calendar.DAY\_OF\_WEEK) - 2; i++) {// 根据月初是星期几，判断前面空格的数量。

System.out.print("\t");

}

for (int i = 1; i <= maxDate; i++) {

System.out.print(i + "\t");

int w = c.get(Calendar.DAY\_OF\_WEEK);

if (w == Calendar.SATURDAY) {

System.out.println();

}

c.add(Calendar.DATE, 1);

}

}

}

# 

# File类

## File类的基本用法

1 java.io.File类：文件和目录路径名的抽象表示形式。

        File类的常见构造方法：

   public File(String pathname)

  以pathname为路径创建File对象，如果pathname是相对路径，则默认的当前路径在系统属性user.dir中存储。

2 File的静态属性String separator存储了当前系统的路径分隔符。

3 通过File对象可以访问文件的属性。  
    public boolean canRead()     public boolean canWrite()  
    public boolean exists()      public boolean isDirectory()  
    public boolean isFile()      public boolean isHidden()  
    public long lastModified()   public long length()  
    public String getName()      public String getPath()

4 通过File对象创建空文件或目录（在该对象所指的文件或目录不存在的情况下）。

    public boolean createNewFile()throws IOException

    public boolean delete()

public boolean mkdir()， mkdirs()

5常见构造器，方法的演示案例：

package com.bjsxt.file;

import java.io.File;

import java.io.IOException;

public class TestFile {

/\*\*

 \* File文件类 1.代表文件 2.代表目录

 \*

 \*

 \*/

public static void main(String[] args) {

File f = new File("d:/src3/TestObject.java");

File f2 = new File("d:/src3");

File f3 = new File(f2, "TestFile.java");

File f4 = new File(f2, "TestFile666.java");

File f5 = new File("d:/src3/aa/bb/cc/dd");

//f5.mkdirs();

f5.delete();

try {

f4.createNewFile();

System.out.println("文件创建成功！");

} catch (IOException e) {

e.printStackTrace();

}

if (f.isFile()) {

System.out.println("是一个文件！");

}

if (f2.isDirectory()) {

System.out.println("是一个目录！");

}

if (f3.isFile()) {

System.out.println("是一个文件奥");

}

}

}

# 

# 树状结构展现文件结构

    编写一个程序，在命令行中以树状结构展现特定的文件夹及其子文件(夹)，利用File类和递归知识，实现树状结构的展示，代码如下：

package com.bjsxt.file;

import java.io.File;

import java.io.IOException;

public class TestFile {

/\*\*

 \* 1.File文件类 的常见构造器，常见方法的使用

 \* 2.递归：

 \*      概念：在java代码中出现，出现本身调用自己的现象，我们称之为递归。

 \*  组成：

 \*  递归体。

 \*  递归头。

 \*  注意事项：

 \*  递归若没有递归头，递归将是死递归了。

 \*

 \*/

public static void main(String[] args) {

File f = new File("d:/src3");

printFile(f, 0);

}

static void printFile(File file, int level) {

for (int i = 0; i < level; i++) {

System.out.print("-");

}

System.out.println(file.getName());

if (file.isDirectory()) {

File[] files = file.listFiles();

for (File temp : files) {

printFile(temp, level + 1);

}

}

}

}

# 枚举

³ 枚举类型：

² 只能够取特定值中的一个

² 使用enum关键字

² 所有的枚举类型隐性地继承自 java.lang.Enum。（枚举实质上还是类！而每个被枚举的成员实质就是一个枚举类型的实例，他们默认都是public static final的。可以直接通过枚举类型名直接使用它们。）

² 强烈建议当你需要定义一组常量时，使用枚举类型

² 尽量不要使用枚举的高级特性，事实上高级特性都可以使用普通类来实现，没有必要引入复杂性！

package com.bjsxt.Enum;

public class TestEnum {

public static void main(String[] args) {

Week ww = Week.FRIDAY;// 定义一个枚举类型对象

Week ww2 = Week.values()[2];// 返回每局的第三个元素！！

System.out.println(ww2);

String temp = testEnum(Week.SUNDAY);

System.out.println(temp);

}

static String testEnum(Week w) {

switch (w) {

case SUNDAY:

return "日";

case MONDAY:

return "一";

case TUESDAY:

return "二";

case WEDNESDAY:

return "三";

case THURSDAY:

return "四";

case FRIDAY:

return "五";

case SATURDAY:

return "六";

}

return "";

}

}

enum Week {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

}

# 

# 导引问题

        实际工作中，遇到的情况不可能是非常完美的。比如：你写的某个模块，用户输入不一定符合你的要求、你的程序要打开某个文件，这个文件可能不存在或者文件格式不对，你要读取数据库的数据，数据可能是空的等。我们的程序再跑着，内存或硬盘可能满了。等等。

软件程序在运行过程中，非常可能遇到刚刚提到的这些异常问题，我们叫异常，英文是：Exception，意思是例外。这些，例外情况，或者叫异常，怎么让我们写的程序做出合理的处理。而不至于程序崩溃。

 常见的错误：

    1.    用户输入错误

    2.    设备错误

硬件问题，比如打印机关掉、服务器问题

    ·3.    物理限制

磁盘满了。

    4.    代码限制

数组下标越界等

设计良好的程序应该在异常发生时提供处理这些错误的方法，使得程序不会因为异常的发生而终断或产生不可预见的结果。

如果没有异常处理机制，那么：

|  |
| --- |
| //将d:/a.txt复制到e:/a.txt    if("d:/a.txt"这个文件存在){  if(e盘的空间大于a.txt文件长度){  if(文件复制一半IO流断掉){  停止copy，输出：IO流出问题！  }else{  copyFile("d:/a.txt","e:/a.txt");  }  }else{  输出：e盘空间不够存放a.txt！  }  }else{  输出：a.txt不存在！  } |

两个坏处：

1．    逻辑代码和错误处理代码放一起！

2．    程序员本身需要考虑的例外情况较复杂，对程序员本身要求较高！

异常机制就是当程序出现错误，程序如何安全退出的机制。

# 异常（Exception）的概念

JAVA如何处理异常？

第一个异常示例和解析：

|  |
| --- |
| public static void main(String[] args) {          int i = 1/0;          gggggg  }  Exception in thread "main" java.lang.ArithmeticException: / by zero  at chuji.BubbleSort.main(BubbleSort.java:11) |
|  |

JAVA是采用面向对象的方式来处理异常的。处理过程：

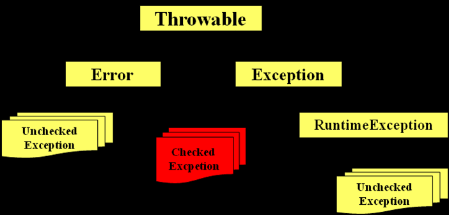
1.    抛出异常：在执行一个方法时，如果发生异常，则这个方法生成代表该异常的一个对象，停止当前执行路径，并把异常对象提交给JRE。

2.    捕获异常：JRE得到该异常后，寻找相应的代码来处理该异常。JRE在方法的调用栈中查找，从生成异常的方法开始回溯，直到找到相应的异常处理代码为止。

# 

# 异常分类

JDK 中定义了很多异常类，这些类对应了各种各样可能出现的异常事件，所有异常对象都是派生于Throwable类的一个实例。如果内置的异常类不能够满足需要，还可以创建自己的异常类。



## Error

    Error类层次描述了Java运行时系统内部错误和资源耗尽错误。这类错误是我们无法控制的，同时也是非常罕见的错误。所以在编程中，不去处理这类错误。

Error表明系统JVM已经处于不可恢复的崩溃状态中。我们不需要管他。

打开JDK的：java.lang.error，查看他的所有子类。

 Error和Exception的区别：

    我开着车走在路上，一头猪冲在路中间，我刹车。这叫一个异常。

我开着车在路上，发动机坏了，我停车，这叫错误。系统处于不可恢复的崩溃状态。发动机什么时候坏？我们普通司机能管吗？不能。发动机什么时候坏是汽车厂发动机制造商的事。

## Exception

所有异常类的父类，其子类对应了各种各样可能出现的异常事件。

n    Runtime Exception（unchecked Exception）

    出现RuntimeException就一定是你的问题，可以不捕获，因为小心点这些异常是可以避免的。 派生于RuntimeException的异常。是一类特殊的异常，如被 0 除、数组下标超范围等，其产生比较频繁，处理麻烦，如果显式的声明或捕获将会对程序可读性和运行效率影响很大。因此由系统自动检测并将它们交给缺省的异常处理程序（用户可不必对其处理）。这类异常通常是由编程错误导致的，因为只有小心点，这些异常都是可以避免的，所以在编写程序时，并不要求必须使用异常处理机制来处理这类异常,所有这类异常都继承自java.lang.RuntimeException。

注意：RuntimeException这个名字很容易让人产生错误影响。事实上，我们这里讨论的所有Error、Exception都是运行时发生的。

常见的有：

1.    ArithmeticException

如试图除以0

|  |
| --- |
| if(b!=0){  int i = 1/b;  }else{  System.out.println("不能用0做除数！！");  } |

 2. NullPointerException

当程序访问一个空对象的成员变量或方法，访问一个空数组的成员时发生。怎么处理？

|  |
| --- |
| //  TestException te = null;  TestException te = new TestException();  if(te!=null){  te.test1(2);  } |

 3. ClassCastException

|  |
| --- |
| void test2(Object obj){  if(obj instanceof Man){  Man man = (Man) obj;  }  } |

 4. ArrayIndexOutOfBoundsException

访问的元素下表超过数组长度

|  |
| --- |
| int[] a = {1,2,3};  int idx = 3;  if(idx<=a.length-1){  System.out.println(a[idx]);  } |

 5. NumberFormatException

数字格式异常！

|  |
| --- |
| String str = "1234abcf";  Integer i = new Integer(str);  //  Integer i2 = Integer.parseInt(str); |

 n Checked Exception:

    所有不是Runtime Exception的异常，统称为Checked Exception，又被称为“已检查异常”。 这类异常的产生不是程序本身的问题，通常由外界因素造成的。为了预防这些异常产生时，造成程序的中断或得到不正确的结果，Java要求编写可能产生这类异常的程序代码时，一定要去做异常的处理。

1.    要打开的文件不存在

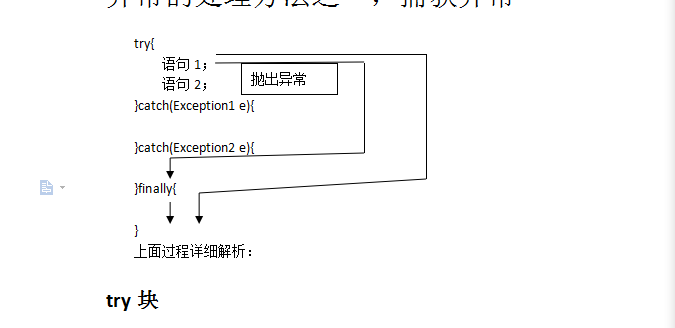
编译器将检查是否为所有已检查异常提供异常处理。

  这一类异常，我们必须捕获进行处理。

 Java语言将派生于RuntimeException类或Error类的所有异常称为“未检查异常”。

# 

# 异常的处理办法之一，捕获异常



上面过程详细解析：

## try块

     try语句指定了一段代码，该段代码就是一次捕获并处理的范围。在执行过程中，当任意一条语句产生异常时，就会跳过该段中后面的代码。代码中可能会产生并抛出一种或几种类型的异常对象，它后面的catch语句要分别对这些异常做相应的处理

一个try语句必须带有至少一个catch语句块或一个finally语句块 。。

 注：当异常处理的代码执行结束以后，是不会回到try语句去执行尚未执行的代码。

## catch

  a)    每个try语句块可以伴随一个或多个catch语句，用于处理可能产生的不同类型的异常对象。

  b)    常用方法：

   toString ( )方法，显示异常的类名和产生异常的原因

  getMessage( )  方法，只显示产生异常的原因，但不显示类名。

  printStackTrace( )  方法，用来跟踪异常事件发生时堆栈的内容。

 这些方法均继承自Throwable类

  c) Catch捕获异常时的捕获顺序：

i.    如果异常类之间有继承关系，在顺序安排上需注意。越是顶层的类，越放在下面。再不然就直接把多余的catch省略掉。

## finally

   有些语句，不管是否发生了异常，都必须要执行，那么就可以把这样的语句放到finally语句块中。

通常在finally中关闭程序块已打开的资源，比如：文件流、释放数据库连接等。

典型代码（先不要敲！！）：

package com.bjsxt.Exception;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class TestException {

public static void main(String[] args) {

FileReader reader = null;

try {

reader = new FileReader("d:/a.txt");

char temp = (char) reader.read();

System.out.println("读出的内容：" + temp);

} catch (FileNotFoundException e) {

System.out.println("文件没有找到！！");

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

System.out.println("文件读取错误！");

} finally {

System.out.println(" 不管有没有异常，我肯定会被执行！");

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

## try, catch,finally ,return 执行顺序

package com.bjsxt.Exception;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class TestException {

public static void main(String[] args) {

String str = new TestException().openFile();

System.out.println(str);

}

String openFile() {

try {

System.out.println("aaa");

FileInputStream fis = new FileInputStream("d:/a.txt");

int a = fis.read();

System.out.println("bbb");

return "step1";

} catch (FileNotFoundException e) {

System.out.println("catching !!!!!");

e.printStackTrace();

return "step2";// 先确定返回值，并不会直接结束运行。

} catch (IOException e) {

e.printStackTrace();

return "step3";

} finally {

System.out.println("finally !!!!!");

// return "fff";不要在finally中使用return.

}

}

}

执行顺序：

1．    执行try，catch ， 给返回值赋值

2．    执行finally

3．    return

# 

# 异常的处理办法之二，声明异常：throws子句

    当Checked Exception产生时，不一定立刻处理它，可以再把异常Throws出去。

    在方法中使用try-chatch-finally由这个方法处理异常。在一些情况下，当前方法并不需要处理发生的异常，而是向上传递给调用它的方法处理。

如果一个方法中可能产生某种异常，但是并不能确定如何处理这种异常，则应根据异常规范在方法的首部声明该方法可能抛出的异常。

如果一个方法抛出多个已检查异常，就必须在方法的首部列出所有的异常，之间以逗号隔开。

典型代码（先不敲！）：

package com.bjsxt.Exception;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class FileTest {

public static void main(String[] args) {

try {

readFile("d:/a.txt");

} catch (FileNotFoundException e) {

System.out.println("所需要的文件不存在！");

e.printStackTrace();

} catch (IOException e) {

System.out.println("文件读写出错误！");

e.printStackTrace();

}

}

public static void readFile(String fileName) throws IOException {

FileReader in = new FileReader(fileName);

try {

int tem = 0;

tem = in.read();

while (tem != -1) {

System.out.println((char) tem);

tem = in.read();

}

} finally {

in.close();

}

}

}

方法重写中声明异常原则

    子类声明的异常范围不能超过父类声明的范围。包含如下意思：

       1.    父类没有声明异常，子类也不能；

      2.    不可抛出原有方法抛出异常类的父类或上层类

      3.    抛出的异常类型的数目不可以比原有的方法抛出的还多（不是指个数）

代码示例：

package com.bjsxt.Exception;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.text.ParseException;

public class TestException1 {

public static void main(String[] args) {

}

}

class A {

public void method() throws IOException, ParseException {

}

}

class B extends A {

public void method() throws FileNotFoundException {

}

}

class C extends A {

public void method() {

}

}

class D extends A {

public void method() throws IOException {

}

}

class E extends A {

public void method() throws IOException, FileNotFoundException {

}

}

class F extends A {

public void method() throws IOException, ArithmeticException {

}

}

class G extends A {

public void method() throws IOException, ParseException {

}

}

# 异常的处理办法之三，手动抛出异常，throw子句

    Java异常类对象除在程序执行过程中出现异常时由系统自动生成并抛出，也可根据需要手工创建并抛出。

      在捕获一个异常前，必须有一段代码先生成异常对象并把它抛出。这个过程我们可以手工做，也可以由JRE来实现，但是他们调用的都是throw子句。

      对于一个已经存在的异常类，抛出该类异常对象过程如下：

      找到一个合适的异常类。

   创建一个该类的对象。

   将对象抛出

|  |
| --- |
| File f = new File("c:/tt.txt");  if (!f.exists())  {  try {  throw new FileNotFoundException("File can't be found!");  } catch (FileNotFoundException e) {  e.printStackTrace();  }  } |

# 

# 自定义异常

    在程序中，可能会遇到任何标准异常类都没有充分的描述清楚的问题，这种情况下可以创建自己的异常类

 从Exception类或者它的子类派生一个子类即可

    习惯上，定义的类应该包含2个构造器：一个是默认的构造器，另一个是带有详细信息的构造器

|  |
| --- |
| class  IllegalAgeException   extends   Exception {  public IllegalAgeException() {  }  public IllegalAgeException(String  msg) {  super(msg);  }  }    class Person {  private String name;  private int age;  public void setName(String name) {  this.name = name;  }  public void setAge(int age) {  if (age < 0)  throw new IllegalAgeException("人的年龄不应该为负数");  this.age = age;  }  public String toString() {  return "name is " + name + " and age is " + age;  }  }   public class MyExceptionTest {  public static void main(String[] args) {  Person p = new Person();  try {  p.setName("Lincoln");  p.setAge(-1);  } catch (IllegalAgeException e) {  e.printStackTrace();  System.exit(-1);  }  System.out.println(p);  }  } |

 使用异常机制建议

   1.    要避免使用异常处理代替错误处理，这样会降低程序的清晰性，并且效率低下。

  2.    处理异常不可以代替简单测试--- 只在异常情况下使用异常机制

  3.    不要进行小粒度的异常处理---应该将整个任务包装在一个Try语句块中

  4.    异常往往在高层处理(先了解！后面做项目会说！)

# 总结

³    一个图

³    五个关键字（try, catch, finally,  throws,    throw）

³    先逮小的，再逮大的

³    异常和重写的关系

³    自定义异常