## 第01章：Java开发入门

课程大纲

1. 计算机基本概念

什么是计算机?

计算机（Computer）全称：电子计算机，俗称电脑。是一种能够按照程序运行，自动、高速处理海量数据的现代化智能电子设备。由硬件和软件所组成，没有安装任何软件的计算机称为裸机。常见的形式有台式计算机、笔记本计算机、大型计算机等。

什么是软件?

计算机软件(Computer Software)是使用计算机过程中必不可少的东西，计算机软件可以使计算机按照事先预定好的顺序完成特定的功能。

一台计算机由：主板、CPU、内存、电源、显卡、硬盘等组成，这些是必须的，只要主机工作，这几样缺一不可。

计算机语言的发展：

1、机器语言

机器语言是直接用二进制代码指令表达的计算机语言，指令是用0和1组成的一串代码，它们有一定的位数，并分成若干段，

各段的编码表示不同的含义。

2、汇编语言

汇编语言是使用一些特殊的符号来代替机器语言的二进制码，计算机不能直接识别，需要用一种软件将汇编语言翻译成机器语言。

3、高级语言

使用普通英语进行编写源代码，通过编译器将源代码翻译成计算机直接识别的机器语言，之后再由计算机执行。

1. DOS 常用命令

1、dir

无参数：查看当前所在目录的文件和文件夹。

/s：查看当前目录以及其所有子目录的文件和文件夹。

/a：查看包括隐含文件的所有文件。

/ah：只显示出隐含文件。

/w：以紧凑方式（一行显示5个文件）显示文件和文件夹。

/p：以分页方式（显示一页之后会自动暂停）显示。

|more：前面那个符号是“\”上面的那个，叫做重定向符号，就是把一个

命令的结果输出为另外一个命令的参数。more也是一个命令，dir /w |more

得到的结果和dir /w /p的结果是一样的。

其他的参数大家可以用：dir/?查看。

2、cd

cd 目录名：进入特定的目录。如果看到有个目录显示为：abcdef ghi 就

输入：cdabcdef.ghi进入该目录。

cd\ 退回到根目录。

cd..退回到上一级目录。

3、md rd

md 目录名：建立特定的文件夹。 （dos下面习惯叫目录，win下面习惯叫文

件夹）

rd 目录名：删除特定的文件夹。

4、cls

清除屏幕。

5、copy

copy 路径\文件名 路径\文件名 ：把一个文件拷贝到另一个地方。

6、move

move 路径\文件名 路径\文件名 ：把一个文件移动（就是剪切+复制）到另

一个地方。

move 路径\文件名 路径\文件名 ：把一个文件移动（就是剪切+复制）到另

一个地方。

7、del

del 文件名：删除一个文件。

del \*.\*：删除当前文件夹下所有文件。

del不能删除文件夹。

8、deltree

删除文件夹和它下面的所有子文件夹还有文件，厉害啦。。。不要乱用。

9、type

type 文本文件名：显示出文本文件的内容。

12 ren

ren 旧文件名 新文件名：改文件名。

1. Java语言发展史

SUN公司于1995 年推出

1991 年Sun 公司的James Gosling 等人开始开发名称为Oak 的语言。希望用于控制嵌入在有线电视交换盒、PDA 等的微处理器

Internet 使Java 成为网上最流行的编程语言,Java 对Internet 的影响也意义深远

1994 年将Oak 语言更名为Java

Java 既安全、可移植，又可跨平台，而且人们发现它能够解决Internet 上的大型应用问题

1996 年获得1亿美金的投资

1998 年提出JDK1.2，更名为Java2

之后推出JDK1.3，JDK1.4

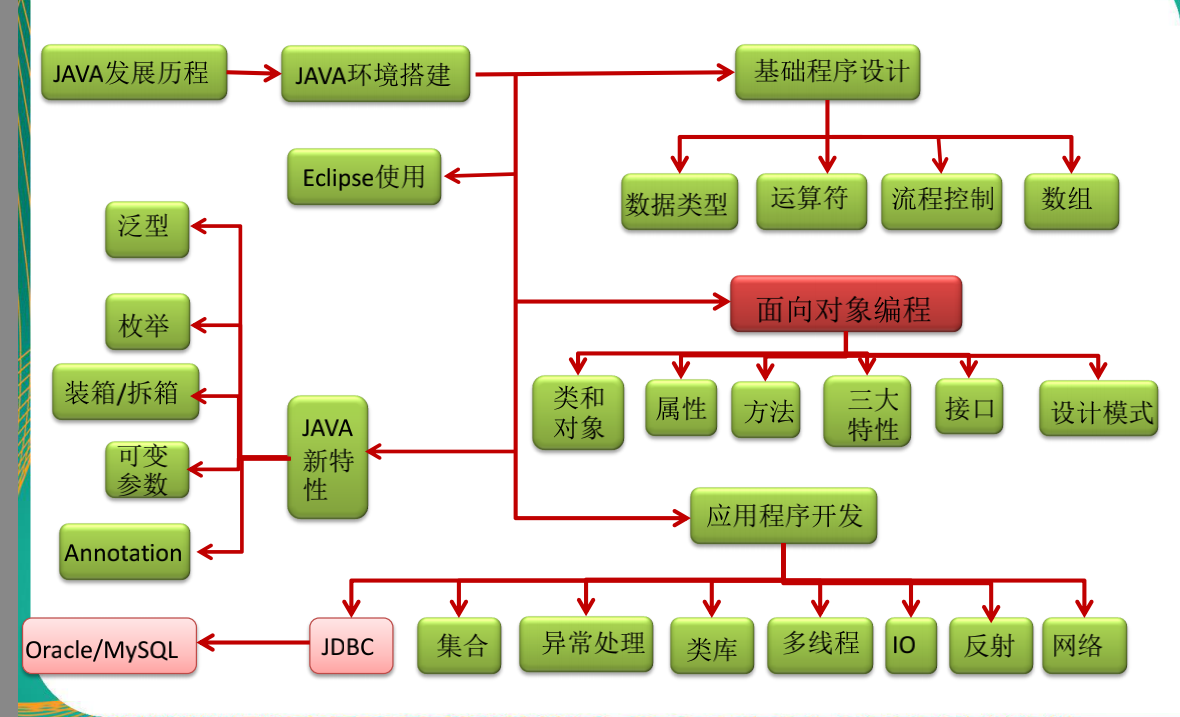
2005 年更名为JDK 5.0，是对以前版本最大改进

2007 年JDK6.0

2009年4月Oracle以74亿美元收购SUN

2011年7月由Oracle正式发布Java7

2014年3月18日发布Java8正式版

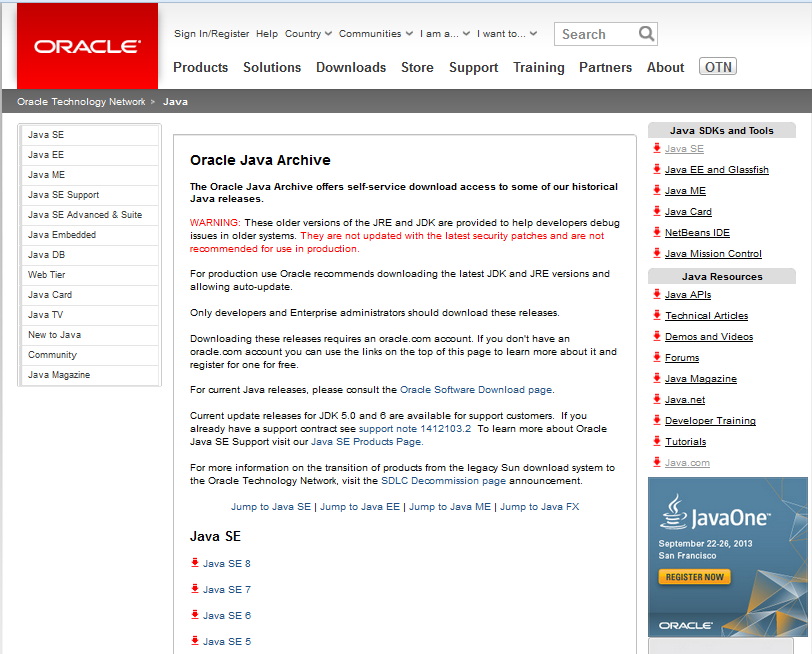


1. JDK下载

JDK（Java Development Kit）Java开发工具包

JRE（Java Runtime Environment）Java运行环境

下载地址：<http://www.oracle.com/technetwork/java/archive-139210.html>



1. Java体系与特点

ava SE：Java Platform，Standard Edition

标准版：各应用平台的基础，桌面开发和低端商务应用的解决方案。

Java EE：Java Platform，Enterprise Edition

企业版：以企业为环境而开发应用程序的解决方案

Java ME ：Java Platform, Micro Edition

微型版：致力于消费产品 和嵌入式设备的最佳解决方案



特点：

一种纯面向对象的编程语言。

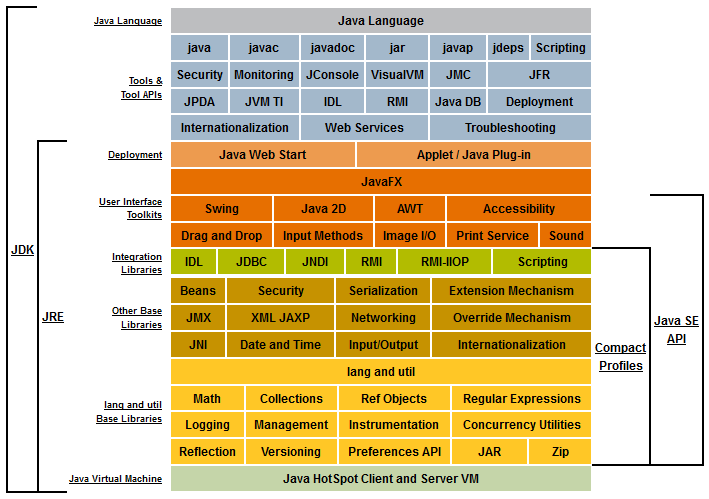
一种与平台无关（跨平台）的语言。

（它提供了在不同平台下运行的解释环境）

一种健壮的语言，吸收了C/C++语言的优点。

有较高的安全性。(自动回收垃圾，强制类型检查，取消指针)

1. JDK安装



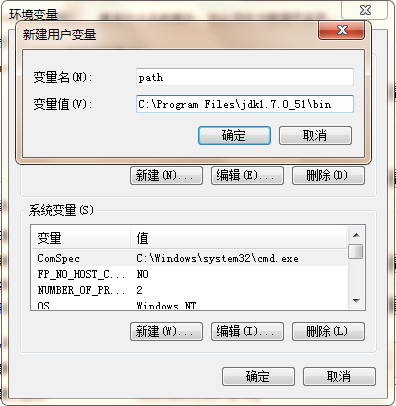
1. 环境变量配置

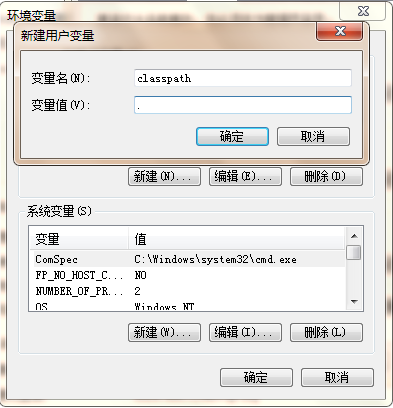
path：用于指定操作系统的可执行指令的路径

classpath：Java 虚拟机在运行某个类时会按classpath 指定的目录顺序去查找这个类

注：从JDK 5.0 开始默认就会到当前工作目录以及JDK 的lib 目录中查找

在桌面右击“我的电脑”－“属性”－“高级系统设置”－“环境变量”



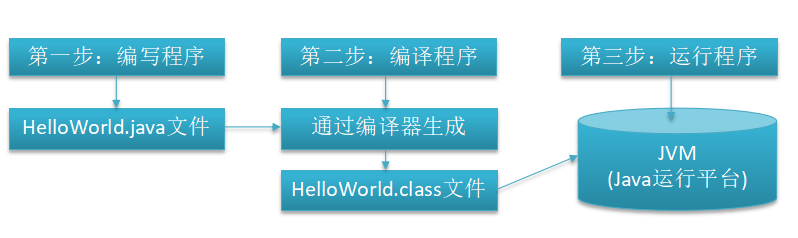


1. 程序开发步骤

程序（Program）是为实现特定目标或解决特定问题而用计算机语言编写的命令序列的集合。

程序一词来源于生活，指的是完成某些事情的一种既定方式和过程。

在日常生活中，可以将程序看成对一系列动作的执行过程的描述。



1. Hello World入门

代码文件内容：

public class HelloWorld{

public static void main(String[] args){

System.out.println("Hello World!");

}

}

Javac命令：编译程序

由于我们前面已经配置好了path环境变量，所以直接打开命令行程序（cmd），把目录切换到HelloWorld.java目录下，使用命令编译程序：

D:\>javac HelloWorld.java

Java命令：执行程序

编译程序完成后，在源文件目录上会自动生成一个HelloWorld.class文件，该文件即为java的字节码文件，此时，我们就可以使用java命令运行程序：

D:\>java HelloWorld

1. Hello World程序分析与注释

/\*\*

文档注释(使用javadoc生成文档)

\*/

public class HelloWorld{

public static void main(String[] args){

/\*.多行注释\*/

//单行注释

System.out.println("Hello World!");

}

}

代码格式：

java 代码的位置

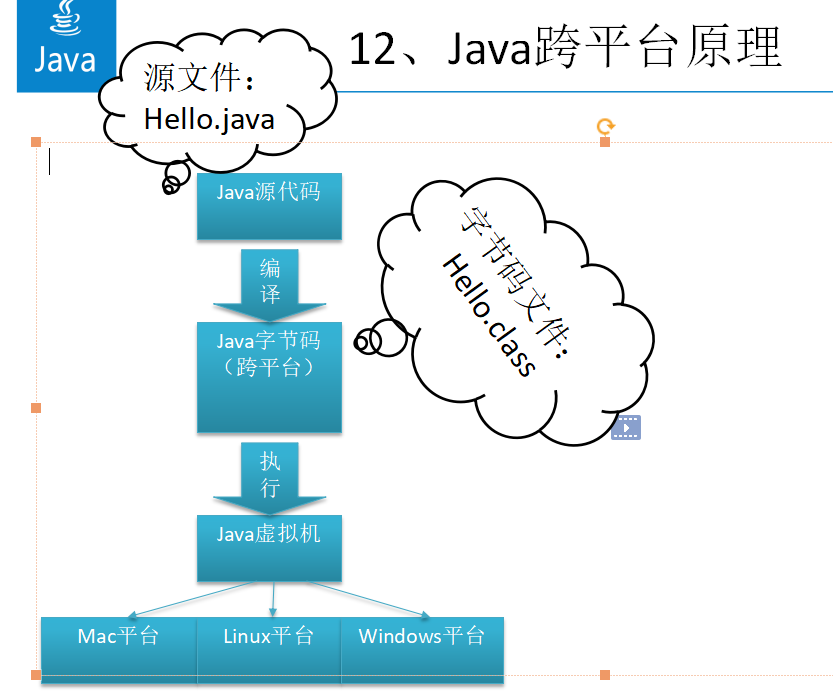
java 是严格区分大小写的

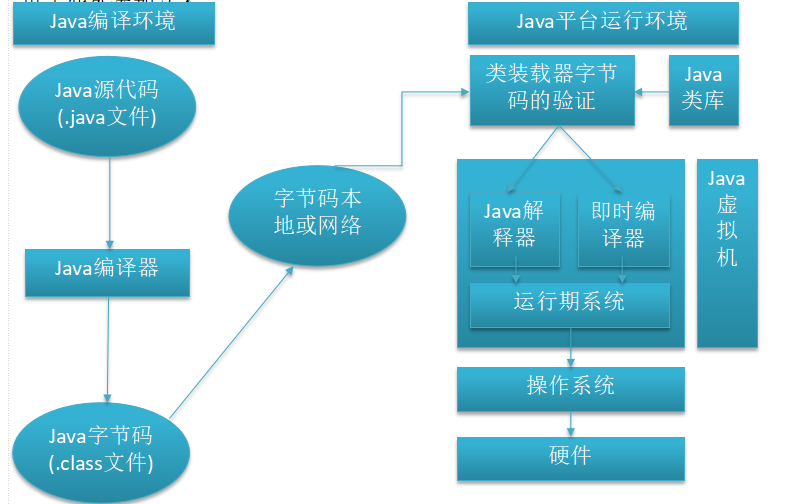
java 是一种自由格式的语言

代码分为结构定义语句和功能执行语句

功能执行语句的最后必须用分号结束

1. Java跨平台原理





对于不同的运行平台，有不同的JVM。

JVM 屏蔽了底层运行平台的差别，实现了“一次编译，随处运行”。

1. Java常用开发工具介绍

编辑器

记事本

Notepad++

EditPlus

UltraEdit （UE）

其它文本编辑器

集成开发环境

JBuilder

Eclipse （http://www.eclipse.org/）

NetBeans （http://www.netbeans.org/）

13、Java程序反编译介绍

Cavaj Java Decompiler 反编译器是一款图形界面的免费工具，可以重新从Java Class（类）文件中构建源代码。它支持Java applets，JAR和ZIP文件的反编译，从中还原出你所需要的源代码，之后你就可以通过Java编程工具中的Class浏览器查看，或进行相关操作。

Cavaj Java Decompiler是独立的Windows应用程序，即使没有安装Java，也可以正常使用。

下载地址：

http://cavaj-java-decompiler.en.softonic.com/

垃圾回收器（GC）

不再使用的内存空间应当进行回收——垃圾回收。在C/C++等语言中，由程序员负责回收无用内存。

垃圾回收器在Java 程序运行过程中自动启用，用于检查并释放那些可以被释放的存储空间，程序员无法精确控制和干预。

## 第02章：Java编程基础

课程大纲

1. 变量与标识符

数学名词：

变数或变量，是指没有固定的值，可以改变的数。变量以非数字的符号来表达，一般用拉丁字母。变量是常数的相反。变量的用处在于能一般化描述指令的方式。

计算机解释：

变量就是系统为程序分配的一块内存单元，用来存储各种类型的数据。根据所存储的数据类型的不同，有各种不同类型的变量。变量名代表这块内存中的数据 。

变量分类：

按所属的数据类型划分：

基本数据类型变量 引用数据类型变量

按被声明的位置划分：

局部变量：方法或语句块内部定义的变量

成员变量：方法外部、类的内部定义的变量

Java对包、类、方法、参数和变量等要素命名时使用的字符序列称为标识符。

Java标识符命名规则：

由字母、数字、下划线（\_）和美元符号（$）组成。

不能以数字开头。

区分大小。

长度无限制。

不能是Java中的关键字和保留关键字。

标识符命名习惯：驼峰命名法、见名知意。以单词或单词组合来命名

合法标识符：

vince、daoguo、user\_name、\_userName、$abc\_123

非法标识符：

2UserName、user#Name、Hello 威哥、class

1. 八种基本数据类型

字节(Byte)是：计算机文件大小的基本计算单位。  
1个字节(Byte) = 8个位（Bit)               
Bit意为“位”或“比特”，是计算机运算的基础； 二进制中的一位，是二进制最小信息单位.二进位可以用来表示一个简单的正/负的判断，有两种状态的开关(如电灯开关) .  
简单换算结果：

1个字节(Byte) = 8个位（Bit)

2个字节(Byte) = 16个位（Bit)

4个字节(Byte) = 32个位（Bit)

8个字节(Byte) = 64个位（Bit)

说明：  
在内存计算过程中，

字节少的数据运算速度更快;

在硬盘存储中，

字节少的数据类型也可充分存入更多的数据。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据类型 | 关键字 | 占用字节 | 取值范围 | 默认值 |
| 布尔型 | boolean | 不定值 | true,false | false |
| 字节型 | byte | 1 | -128~127 | 0 |
| 短整型 | short | 2 | -32768~32767 | 0 |
| 整型 | int | 4 | -2147483648~2147483647 | 0 |
| 长整型 | long | 8 | -9223372036854775808~9223372036854775807 | 0 |
| 字符型 | char | 2 | 0~65535 | ‘\u0000’ |
| 单精度浮点型 | float | 4 | 1.4E-45~3.4028235E38 | 0.0F |
| 双精度浮点型 | double | 8 | 4.9E-324~1.7976931348623157E308 | 0.0D |

1. 基本数据类型的声明

声明变量的格式：

（1）声明变量

类型 变量名;

（2）声明的同时给变量赋值

类型 变量名 ＝ 值;

（3）给已声明过的变量赋值

变量名 ＝ 值;

1. 进制与转换

计算机进制通常有：十进制、二进制、八进制、十六进制

十进制：

人类算数采用十进制，可能跟人类有十根手指有关。亚里士多德称人类普遍使用十进制，只不过是绝大多数人生来就有10根手指这样一个解剖学事实的结果。

10、150、1050…

二进制：

计算机中的数据都以二进制数字保存。二进制：逢二进一。即只有0、1两个值。

如：十进制的10在计算机内保存为二进制的1010

计算机中信息的存储单位

位（Bit）：表示一个二进制数码0或1，是计算机存储处理信息的最基本的单位。 字节（Byte）：一个字节由8个位组成。它表示作为一个完整处理单位的8个二进制数码。

八进制：

基数为八。

Java中八进制数据要以0开头。如：0123

八进制转换成二进制：只需将每个八进制数字替换为相对应的三个二进制位即可。

现在的计算机系统很少用八进制的了。

|  |  |
| --- | --- |
| 二进制 | 八进制 |
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

十六进制：

二进制表示法太冗长，所以在程序中一般喜欢用十六进制

十六进制：基数为十六，逢十六进一。它用abcdef表示从0-9之上的值

Java中十六进制数据要以0x或0X开头。如：0x23D

十六进制转换成二进制只需将每个十六进制数字替换为相对应的四个二进制位即可

|  |  |  |  |
| --- | --- | --- | --- |
| 二进制 | 十六进制 | 二进制 | 十六进制 |
| 0000 | 0 | 1001 | 9 |
| 0001 | 1 | 1010 | A |
| 0010 | 2 | 1011 | B |
| 0011 | 3 | 1100 | C |
| 0100 | 4 | 1101 | D |
| 0101 | 5 | 1110 | E |
| 0110 | 6 | 1111 | F |
| 0111 | 7 |  |  |
| 1000 | 8 |  |  |

补码：

事实上，计算机内的二进制数值是以补码形式表示的。

补码： 一个正数的补码和其原码的形式是相同的。

负数的补码是：将该数的绝对值的二进制形式，按位取反再加1。

由此可知，二进制补码数值的最高位(最左位)是符号位：该位为0，表示数值为正数；该位为1，表示数值为负数。

|  |  |
| --- | --- |
| **十进制** | **二进制** |
| 10 | 00000000  00000000 00000000 00001010 |
| -10 | 11111111  11111111  11111111  11110110 |

主要原因：使用补码，可以将符号位和其它位统一处理；

同时，减法也可按加法来处理。

另外，两个用补码表示的数相加时，

如果最高位（符号位）有进位，

则进位被舍弃。

二进制数转换成十进制数 ：

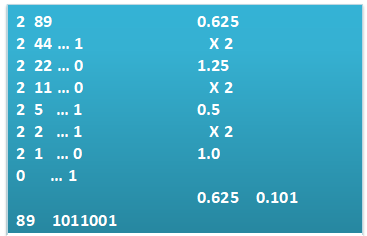
按权相加：把二进制数首先写成加权系数展开式，然后按十进制加法规则求和。

例：原码1011.01 = 1×23＋0×22＋1×21＋1×20＋0×2-1＋1×2-2  
 　　　　= 8＋0＋2＋1＋0＋0.25  
 　　　　=11.25

十进制转换成二进制

整数部分："除2取余，逆序排列"法

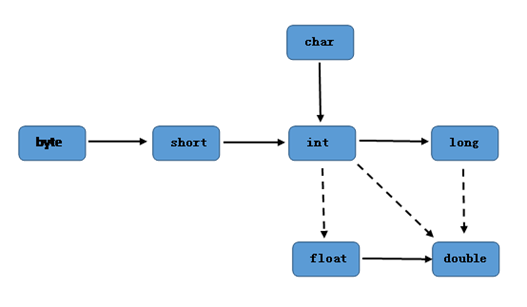
小数部分："乘2取整，顺序排列"法



1. 基本数据类型转换

自动类型转换：

容量小的类型自动转换成容量大的数据类型



byte,short,int，它们三者在计算时会转换成int类型

如果把int值转换为float值，或者long转换为double值，不需要强制转换，但可能丢失精度.

强制类型转换：

容量大的类型转换成容量小的数据类型时，要加上强制转换符。

long n = 100L;

int i = (int)n;

有可能造成精度降低或数据溢出，使用时要小心。

boolean 类型不能转换成任何其它数据类型。

1. 关键字

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | assert | boolean | break | byte |
| case | catch | char | class | const |
| continue | default | do | double | else |
| enum | extends | final | finally | float |
| for | goto | if | implements | import |
| instanceof | int | interface | long | native |
| new | package | private | protected | public |
| return | strictfp | short | static | super |
| switch | synchronized | this | throw | throws |
| transient | try | void | volatile | while |

保留关键字 如：goto 和 const

1. 转义字符

|  |  |
| --- | --- |
| \n | 换行(LF) ，将当前位置移到下一行开头 |
| \r | 回车(CR) ，将当前位置移到本行开头 |
| \t | 水平制表(HT) （跳到下一个TAB位置） |
| \\ | 代表一个反斜线字符''\' |
| \' | 代表一个单引号（撇号）字符 |
| \" | 代表一个双引号字符 |

1. 运算符与优先级

算术运算符：

表达式：由变量、常量运算符组成的式子。

++：如果是前缀：先对此变量加1，再执行其他的操作

如果是后缀：先执行其他的操作，再对此变量加1

--：同理

键盘输入：Scanner input=new Scanner(System.in);

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **描述** | **示例** | **结果** |
| + | 加法 | 5+5 | 10 |
| - | 减法 | 5-3 | 2 |
| \* | 乘法 | 2\*3 | 6 |
| / | 除法 | 10/3 | 3 |
| % | 取余（取模） | 10%3 | 1 |
| ++ | 自增（前，后） |  |  |
| -- | 自减（前，后） |  |  |

赋值运算符：

作用是将一个值赋给一个变量，运算顺序从右到左

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **描述** | **示例** | **结果** |
| = | 赋值 | a=10 | a=10 |
| += | 加等于 | a=1, a+=3 | a=4 |
| -= | 减等于 | a=1, a-=3 | a=-2 |
| \*= | 乘等于 | a=1, a\*=3 | a=3 |
| /= | 除等于 | a=10, a/=3 | a=3 |
| %= | 模等于 | a=10, a%=3 | a=1 |

关系运算符：

作用是比较两边的操作数，结果总是boolean型的。

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **描述** | **示例** | **结果** |
| == | 相等于 | a=1,a==10 | false |
| != | 不等于 | a=1, a!=3 | true |
| < | 小于 | a=1, a<3 | true |
| > | 大于 | a=1, a>3 | false |
| <= | 小于等于 | a=10, a<=3 | false |
| >= | 大于等于 | a=10, a>=3 | true |

逻辑运算符：

用于对boolean型结果的表达式进行运算，运算结果总是boolean型，后面结合条件结构讲解。

|  |  |  |  |
| --- | --- | --- | --- |
| **运算符** | **描述** | **示例** | **结果** |
| & | 与 | false & true | false |
| | | 或 | false & true | true |
| ^ | 异或 | true ^ false | true |
| ! | 非 | ! true | false |
| && | 短路与 | false && true | false |
| || | 短路或 | false || true | true |

在计算机中，非0即真，0为假

逻辑或：或1则1

逻辑与：全1才1

运算符练习：

1、通过键盘输入声明初始化一个圆柱体的高和底面半径,pai值是3.14,求圆柱体的体积, 并显示输出圆柱体的体积; π(r^2)h。

2、意定义一个五位整数，求各个位上的数的和。

3、根据天数（46）计算周数和剩余的天数，天数是通过键盘输入。

4、面试题：&和&&的区别

位运算符：

对两个操作数中的每一个二进制位都进行运算

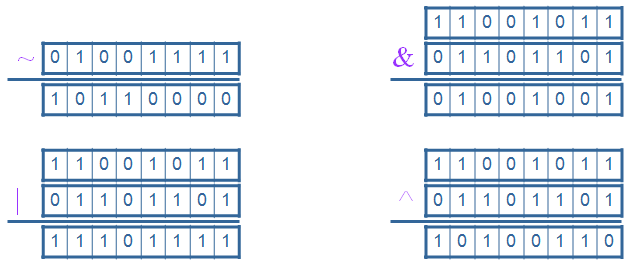
位运算符功能：

1、按位取反 ~

2、按位与 &

3、按位或 |

4、按位异或 ^



位移运算符：

左移："a<<b; "将二进制形式的a逐位左移b位，最低位空出的b位补0

带符号右移："a>>b; "将二进制形式的a逐位右移b位，最高位空出的b位补原来的符号位

无符号右移："a>>>b;"将二进制形式的a逐位右移b位，最高位空出的b位补0

位移运算符正数示例：

100 00000000 00000000 00000000 01100100

100<<3 00000000 00000000 00000011 00100000

100>>3 00000000 00000000 00000000 00001100

100>>>3 00000000 00000000 00000000 00001100

位移运算符负数示例：

-100 11111111 11111111 11111111 10011100

-100<<3 11111111 11111111 11111100 11100000

-100>>3 11111111 11111111 11111111 11110011

-100>>>3 00011111 11111111 11111111 11110011

面试题：以最快的速度计算8\*4结果？

00000000 00000000 00000000 00001000 8

10000000 00000000 00000000 00100000 32

使用位移的方法是最快的。

int result = 8<<2;

左移算法：位移数作为2的次幂与操作数相乘

10<<2=40

右移算法：操作数除以位移数的2次幂

通过键盘输入，初始化两个数将这两个数据交换位置后，输出显示交换后的数据。

三目运算符：

X ? Y : Z

X为boolean类型表达式，先计算x的值，若为true，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。

从键盘输入一个成绩，判断该分数是否及格。



1. 分支语句

if条件结构是根据条件判断之后再做处理

简单判断语句：

if(表达式){

执行语句;

}else{

执行语句;

}

如果...那么

执行过程：当程序执行到if时，首先会判断( )里面表达式的值，

如果表达式的值为真，那么就执行( )后{ }里的执行语句，

如果为假，那么就执行else后面{ }里的执行语句

& ,| ,^ ,&& ,|| , ! ,==

案例练习：

1、输入一个字符，判断是大写字母还是小写字母，输出提示信息。

2、写一个程序，判断某一年是否为闰年（被4整除但不能被100整除能被400整除)是闰年否则是平年

分支嵌套：

if(表达式1){

执行语句1;

}else if(表达式2){

执行语句2;

}else if(表达式3){

执行语句3;

}else{//可有可无

执行语句4;

}

执行过程:当程序执行到if时,首先会判断( )表达式1的值,如果为真,那么就执行{ }里面的执行语句1;然后这个if-else if结束,如果表达式1的值为假,那么就会继续向下执行,else if后面的表达式2,为真,就执行语句2,为假,就继续向下执行,直到为真,如果所有表达式的值都不为真,那么就会执行else后面{ }里的执行语句,然后真个if-else if结束。

案例练习：

1、给出一个百分制成绩，要求输出成绩等级‘A’、‘B’、‘C’、‘D’、‘E’。90分以上输出’A’，80～89分输出’B’，70～79分输出’C’，60～69分输出’D’，60分以下输出‘E’。

2、输入两个整数，打印出它们的大小关系，是小于、大于还是等于。

3、多条件嵌套分支，设计一个程序，根据天气情况选择外出或在家，外出是逛街还是逛公园，在家是看电影还是玩游戏。

语法：

switch(表达式){

case 取值1: 语句块1;

break;

case 取值n: 语句块n;

break;

default: 语句块n+1;

break;

}

switch与if嵌套

switch(表达式){

case 取值n: 语句块n;

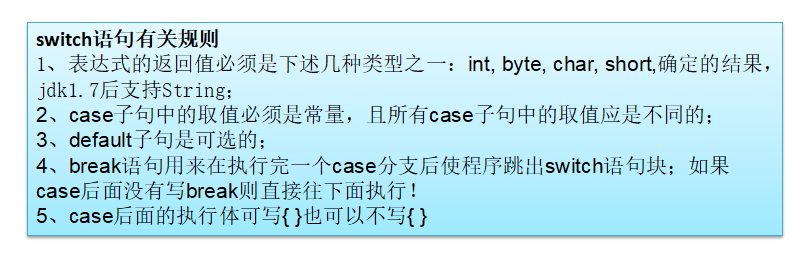
if(表达式){…}

break;

default: 语句块n+1;

break;

}



案例练习：

1、编写一程序实现如下功能：输入1、2、3、4、5、6、7（分别对应星期一至星期日）中的任何一个数。

2、输入一个月数，然后输出对应月份有多少天（不考虑闰年），将天数输出。

3、直接用JDK1.7的字符串条件判断，确定两个数是执行加、减、乘、除操作。

if vs switch

同：都是选择分支语句。

异：1、语法不同。

2、语句后表达式值的类型不同，if语句表达式值的类型是boolean，switch分支表达式值的类型是byte,short,char,int,枚举，String。

3、适用场景

if：变量的值在某个区间之内。

switch：变量的值是某个定值。

if vs 三目运算

三目运算符都可以使用if语句实现，反之不成立。

Java 7 switch支持String的实现原理:

这个新特性是在编译器这个层次上实现的

在编译的过程中，编译器会根据源代码的含义进行转换,不同的Java编译器可能采用不同的方式来转换

比如：如果switch语句中只包含一个case语句，那么就可以简单的将其转换成一个if语句。

如果包含一个case和一个default语句，就可以转换成if-else语句。而对于复杂的情况（多个case语句），

也可以转换成Java 7 之前的switch语句，只不过使用字符串的哈希值（整数）作为switch语句表达式的值

10、循环语句

循环语句分为：while，do while，for

语法：

while(条件表达式){

//语句块；

}

符合条件，循环继续执行；否则，循环退出

特点：先判断，再执行

1、计算1-100之间所有整数之和。5050

2、计算10的阶乘

3、打印100以内所有的奇数，偶数和3的倍数（不包括100）

语法：

do {

循环操作

}while ( 循环条件 );

先执行一遍循环操作，符合条件，循环继续执行；否则，循环退出

特点：先执行，再判断

1、逆序输出10~0

2、Java阶乘求和 1!+2!+...+10!

语法：

for(初始化参数;判断条件 ;更新循环变量){

循环体;

}

关键字：continue 表示跳过当次循环，继续下次循环。

For循环的六种写法：

1、标准写法

2、表达式1省略，但在外部声明

3、表达式2省略，死循环

4、表达式3省略，但是要声明在循环内部

5、表达式1,3省略，但是要声明在循环的外部和内部

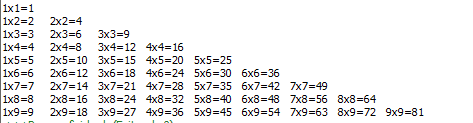
6、三个表达式都省略,死循环

1、打印输出0~200之间能被7整除但不能被4整除的所有整数；

2、输出10个斐波那契（Fibonacci）数列：1 1 2 3 5 8 13 21 34 55。（特点：第1,2两个数值为1，从第三个数开始，该数是其前面两个数之和）

多重循环：

多重循环指的是循环之间的相互嵌套，被嵌套的称为内循环，嵌套内循环的称为外循环，循环可以多重循环。





## 第03章：方法与数组

课程大纲

1. 方法的定义

为什么要有方法？

方法（又叫函数）就是一段特定功能的代码块。方法提高程序的复用性和可读性。

比如，有了方法，我们可以把要重复使用的一段代码提炼出来，然后在每个需要执行这段代码的地方去调用即可。

在一个打飞机的游戏中，飞机需要发射子弹，那么我们可以把发射子弹的这段代码写成方法，这样就可以避免每次重写相同的代码。

方法的格式

语法：

访问权限修饰符 [其他的修饰符 如static] 返回值类型 方法名(参数类型1 形参1,参数类型2 形参2,...){ //形参列表

//方法体

return 返回值;

}

方法的格式说明:

修饰符：目前就用 public static。后面我们再详细的讲解其他的修饰符。

返回值类型：就是功能结果的数据类型。

方法名：见名知意，首字母小写，遵守驼峰命名法。方便我们的调用。

1. 方法的形参与实参

参数：

实际参数：就是实际参与运算的。

形式参数；就是方法定义上的，用于接收实际参数的。

参数类型：就是参数的数据类型

参数名：就是变量名

方法体语句：就是完成功能的代码。

注意：

1、若当前方法中不要使用形参，那么形参列表中可以为空

2、实参和形参的类型要相互兼容，且：实参的取值范围要小于或者等于形参类型的取值范围。

在调用方法中，如果我们定义的方法有参数，就必须在调用方法的同时传入这个值，即给当前方法中的参数赋值，而这个传入的值我们称为实际参数，也就是实参

实参：传入的参数值

形参：接收实参传过来的值。

注意：实参名与形参名可以相同，也可以不同。

小结：形参就是一个变量，实参就是一个值，传参就是把一个值给一个形参赋值

1. 方法的返回值

return：结束方法的。

返回值：就是功能的结果，由return带给调用者。

注意：

1、若当前方法没有返回值类型，即返回值类型是void，那么当前方法中可以不写return

2、return即表示结束一个方法，也可以将返回值返回给调用当前方法的调用者

3、return返回值时一次只能返回一个值，不可以返回多个值

4、一个方法中可以有多个return，但被执行的只能有一个，所以需要判断

练习：

1、判断任意给定年份是否是闰年

2、根据传入的参数数量打印直角三角形

1. 方法的重载

方法重载：overloading method

在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义；

返回值不能作为重载的条件。

如：

public void method(int a){…}

public void method(char c){…}

1. Java 数组的定义

问题：一个班里有50名同学，如何在程序中输出所有同学的学号？

数组：一组能够存储相同数据类型值的变量的集合。

当我们有一组相同类型的数据需要存储，如果此时使用单个变量来存储，我们将要定义若干个变量名，这样将会

非常繁琐，并不利于维护。

1. 数组的赋值方式

四种：

一、使用默认的初始值来初始化数组中的每一个元素

语法：数组元素类型 [] 数组名 = new 数组元素类型[数组中元素的个数（数组的长度）];

如：int [] scores = new int[3];

二、先声明，然后再赋予默认的初始值。

语法：数组元素类型 [] 数组名;

数组名= new 数组元素类型[数组中元素的个数（数组的长度）];

如：int [] scores;

scores = new int[3];

三、先声明，然后再使用指定的值进行初始化。

语法：数组元素类型 [] 数组名 = new 数组元素类型[]{元素1，元素2,....};

如：int [] scores = new int[]{56,78,98};

四、将第三种写法可以简化为（使用数组常量值给数组进行赋值）

语法：数组元素类型 [] 数组名 = {元素1，元素2,....};

如：int [] scores = {56,78,98};

1. 数组的遍历

求数组的长度：

x.length

通过下标来访问数组中的元素。

下标：从0开始，到数组长度-1

如：int[] scores = {89,90};

访问最后一个元素值：int lastElement = scores[scores.length-1];

遍历：依次取出数组中的每一个元素。

遍历方式一：普通的for循环

语法：for(int i = 0; i< 数组的长度;i++){

//i:循环变量，同样：也是数组的下标（取值范围[0,数组长度)）

数组中元素的类型 变量 = 数组名[i];

}

1. 增强for循环与可变参数

JAVA1.5以后Java新增foreach循环和可变参数。

遍历方式二：使用增强for循环【foreach循环】

语法：for(数组中元素的类型 变量:数组名){

数组中元素的类型 临时变量 = 变量;

}

结合方法的定义，可以用可变参数来代替数组作为参数。

public static void print(int ... 变量名){

//可变参数在使用时作为数组使用。

}

1. 数组要注意的问题

使用数组要注意的问题：

空指针异常（NullPointerException）

数组越界异常（ArrayIndexOutOfBoundsException ）

数组内存结构分析：数组是引用类型，会存放在堆内存中。

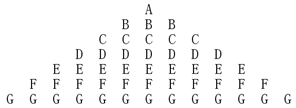


1. 数组示例

数组示例

1、猜数游戏：从键盘中任意输入一个数据，判断数列中是否包含此数。

2、打印正三角形。



多维数组示例：

1、一起来参加屌丝程序员大赛吧，有3个班级各3名学员参赛，

记录每个学员的成绩，并计算每个班的平均分。

1. 最大最小值算法

求最大值与最小值算法：

最大值：

在一个数列中找出最大的数

最小值：

在一个数列中找出最小的数

1. 冒泡排序算法

冒泡排序算法

冒泡排序算法的运作如下：（从后往前）

比较相邻的元素。如果第一个比第二个大，就交换他们两个。

对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。

针对所有的元素重复以上的步骤，除了最后一个。

持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

相同元素的前后顺序并没有改变，所以冒泡排序是一种稳定排序算法。

1. 选择排序算法

选择排序算法

每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最后，直到全部待排序的数据元素排完。 选择排序是不稳定的排序方法。

1. 直接插入排序算法

直接插入排序算法

（从后向前找到合适位置后插入）

基本思想：每步将一个待排序的记录，按其顺序码大小插入到前面已经排序的子序列的合适位置（从后向前找到

合适位置后），直到全部插入排序完为止。

1. 二分查找算法

二分法查找（折半查找）：前提是在已经排好序的数组中，通过将待查找的元素与中间索引值对应的元素进行比较，若大于中间索引值对应的元素，去右半部分查找，否则，去左半部分查找。依此类推。直到找到为止；找不到返回一个负数。

1. Arrays类

Arrays工具类：用来操作数组（比如排序和搜索）的各种方法

常用方法：

使用二分法查找

Arrays.binarySearch(int[] array, int value);

数组内容转成字符串的形式输出

Arrays.toString(int[] array);

数组排序

Arrays.sort(int[] array);

复制指定的数组

Arrays.copyOf(int[] array, int length);

Arrays.copyOf(int[] array, int from, int to);

System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

判断两个数组是否相等

Arrays.equels();

使用指定元素填充数组

Arrays.fill();

17、双色球模拟综合案例

一、双色球彩票玩法

玩法说明：

双色球投注区分为红球号码区和蓝球号码区，红球号码范围为01～33，蓝球号码范围为01～16。双色球每期从33个红球中开出6个号码，从16个蓝球中开出1个号码作为中奖号码，双色球玩法即是竞猜开奖号码的6个红球号码和1个蓝球号码，顺序不限。

二、案例分析：

1、如何产生蓝球和红球？

2、如何接收用户选号？

3、如何验证是否中奖？

4、公布本期中奖号码？

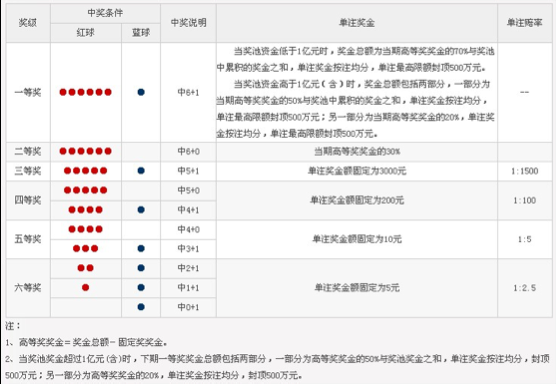
三、实现步骤：

1、整体实现思路

2、随机取值不重复算法（系统和用户）

3、判断是否中奖的逻辑

4、结果输出



## 第04章：面向对象（上）

1. 面向对象基本概念

一、什么是面向对象：

1、面向对象是一种编程思想。

2、面向对象是一种思考问题的思维方式。

二、建立面向对象思维方式：

1、先整体，再局部

2、先抽象，再具体

3、能做什么，再怎么做

三、如何学习面向对象：

1、掌握一门面向对象语言的语法

2、熟悉面向对象的设计原则

3、熟悉面向对象设计模式

1. 类与对象

什么是类？

1、类是：分类、类别。

2、通过分类，我们可以区别不同的事物种类，在日常生活当中，我们常常这样做。

3、所以，类是一组具有相同特性（属性）与行为（方法）的事物集合。

类与对象的关系

1、类表示一个共性的产物，是一个综合的特征，而对象，是一个个性的产物，是一个个体的特征。

2、类由属性和方法组成：

· 属性：就相当于一个个的特征

· 方法：就相当于人的一个个的行为，例如：说话、吃饭、唱歌、睡觉

1. 类和对象的定义格式

在Java中可以使用以下的语句定义一个类：

class 类名称{

属性名称;

返回值类型 方法名称(){}

}

对象的定义：

一个类要想真正的进行操作，则必须依靠对象，对象的定义格式如下：

类名称 对象名称 = new 类名称() ;

按照以上的格式就可以产生对象了。

如果要想访问类中的属性或方法（方法的定义），

则可以依靠以下的语法形式：

访问类中的属性：

对象.属性 ;

调用类中的方法：

对象.方法() ;

在java中对象声明有两种含义

声明对象：Horse horse= null; ;

// 表示声明了一个对象，但是此对象无法使用，horse没有具体的内存指向

实例化对象：horse= new Horse() ;

// 表示实例化了对象，可以使用

//通过对象调用方法：

horse.eat()

//匿名对象调用方法：

new Horse().eat()

1. 对象与内存分析

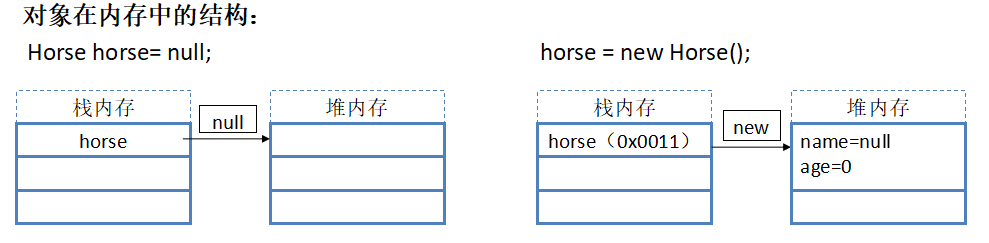
1、new 关键字表示创建一个对象

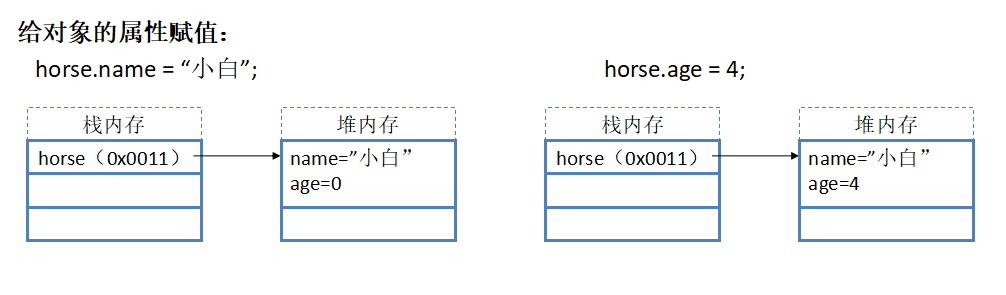
2、new 关键字表示实例化对象

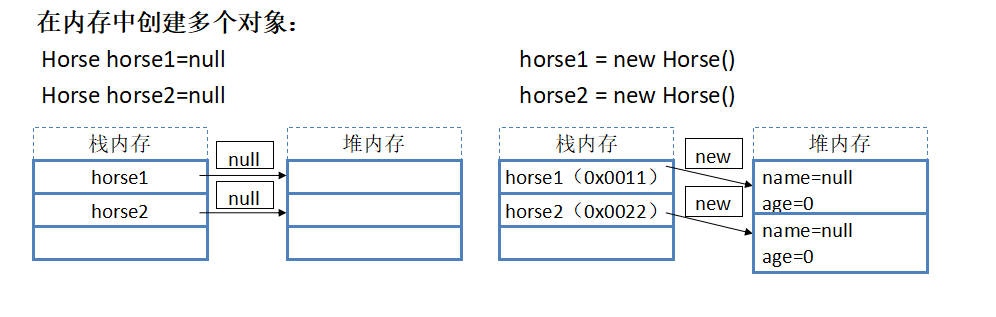
3、new 关键字表示申请内存空间

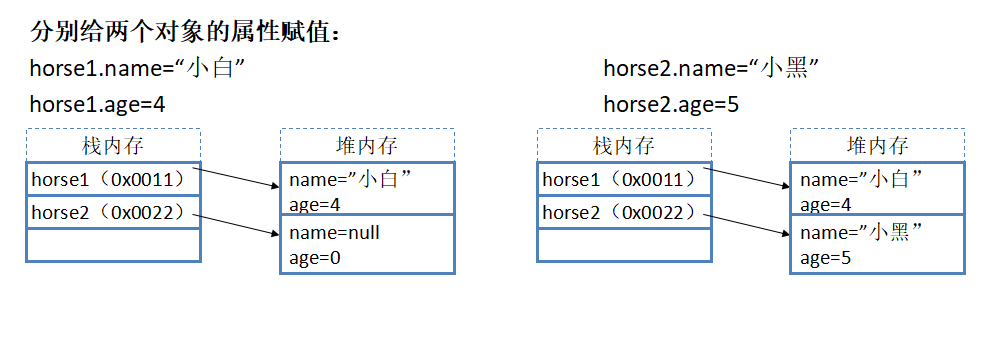
注意：如果使用一个没有申请内存空间的对象，会报空指针异常：

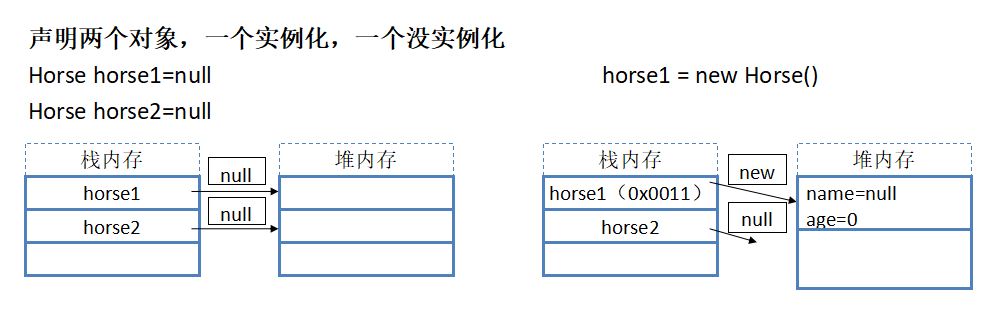
java.lang.NullPointerException

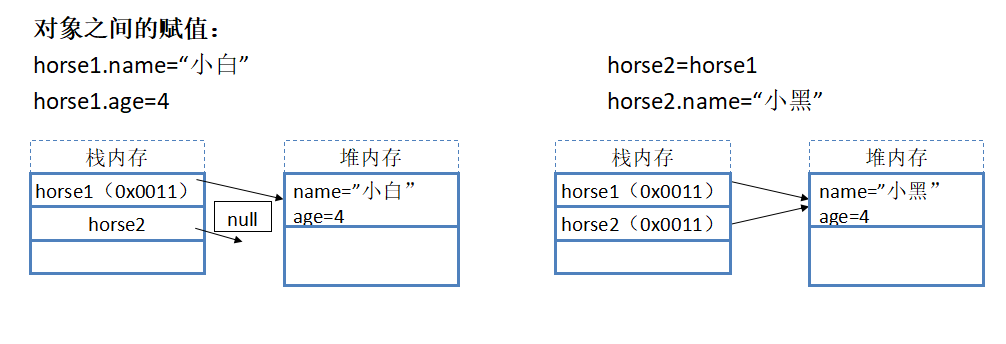


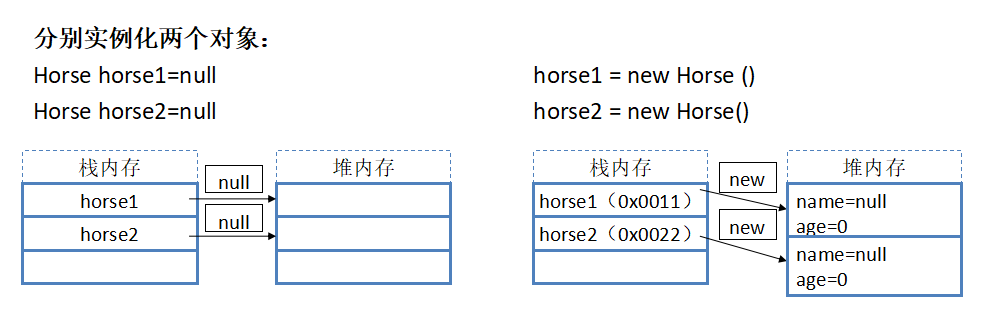


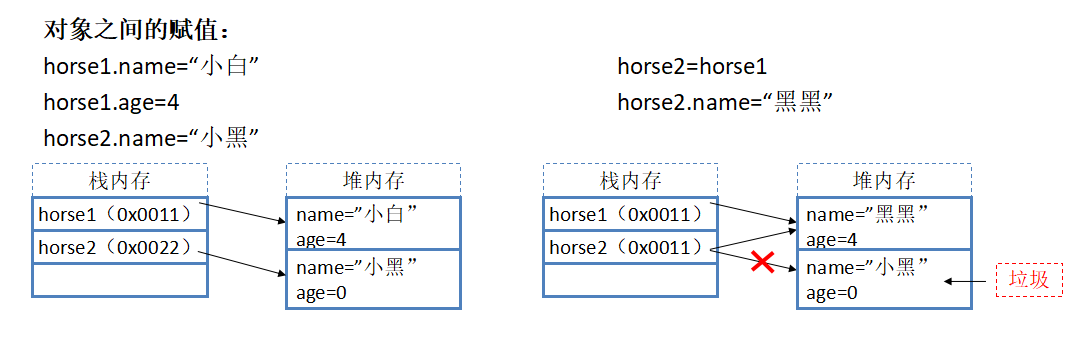












类与对象小结

（1）new关键字：表示向内存申请空间，也表示实例化一个对象，创建一个对象。

（2）一个对象在内存中的大小，由该对象的所有属性所占的内存大小的总和。引用类型变量在32位系统上占4个字节，在64位系统上占8个字节。加上而外的对象隐性数据所占的大小。

（3）相同的类型才可以赋值

（4）不同的引用，指向同一个对象，任何一个引用改变对象的值，其它引用都会反映出来。

（5）编程时要注意的问题，在确定不使用对象时，要尽早释放对象：引用=null

（6）当一个堆中的对象没有被任何引用变量所指向时，该对象会被JVM 的 GC 程序认为是垃圾对象，从而被回收。

1. 封装性

一、封装性的概念

1、封装性是面向对象思想的三大特征之一。

2、封装就是隐藏实现细节，仅对外提供访问接口。

封装有：

属性的封装、方法的封装、类的封装、组件的封装、模块化封装、系统级封装…

二、封装的好处

1、模块化

2、信息隐藏

3、代码重用

4、插件化易于调试

5、具有安全性

封装缺点：

会影响执行效率

封装之前：

class Person{

String name;

int age;

}

封装之后：

class Person{

//属性是成员变量

private String name;

private int age;

//参数及方法内定义的变量是局部变量

public void setName(String name){

this.name = name;}

public String getName(){

return name;}

}

成员变量和局部变量

1、在类中的位置不同

成员变量：在类中定义

局部变量：在方法中定义或者方法的参数

2、在内存中的位置不同

成员变量：在堆内存(成员变量属于对象，对象进堆内存)

局部变量：在栈内存(局部变量属于方法，方法进栈内存)

3、生命周期不同

成员变量：随着对象的创建而存在，随着对象的销毁而消失

局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

4、初始化值不同

成员变量：有默认初始化值，引用类型默认为null

局部变量：没有默认初始化值，必须定义，赋值，然后才能使用

注意：

局部变量名称可以和成员变量名称一样，在方法中使用的时候，采用的是就近原则。

6、构造方法

什么是构造方法

1、构造方法就是类构造对象时调用的方法，用于对象的初始化工 作

2、构造方法是实例化一个类的对象时，也就是new 的时候，最先调用的方法。

构造方法的定义：

构造方法是在类中定义的，构造方法的定义格式：方法名称与类名称相同，无返回值类型的声明。

对象的实例化语法：

Dog dog = new Dog(); //new Dog后面有个括号，带括号表示调用了方法，此时调用的方法就是构造方法了

构造方法重载：

无参构造方法：

public Dog(){}

带一个参数的构造方法：

public Dog(String name){

this.name = name;

}

带多个参数的构造方法：

public Dog(String name,int age){

this.name = name;

this.age = age;

}

构造方法小结

（1）构造方法名称与类名相同，没有返回值声明（包括 void）

（2）构造方法用于初始化数据（属性）

（3）每一个类中都会有一个默认的无参的构造方法

（4）如果类中有显示的构造方法，那么默认构造方法将无效

（5）如果有显示的构造方法，还想保留默认构造 方法，需要显示的写出来。

（6）构造方法可以有多个，但参数不一样，称为构造方法的重载

（7）在构造方法中调用另一个构造方法，使用this(...)，该句代码必须在第一句。

（8）构造方法之间的调用，必须要有出口。

（9）给对象初始化数据可以使用构造方法或setter方法，通常情况下，两者都会保留。

（10）一个好的编程习惯是要保留默认的构造方法。（为了方便一些框架代码使用反射来创建对象）

（11）private Dog(){}，构造方法私有化，当我们的需求是为了 保正该类只有一个对象时。

什么时候一个类只需要一个对象？比如，工具类（没有属性的类，只有行为）并且该工具对象被频繁使用。权衡只用一个对象与产生多个对象的内存使用，来确定该类是否要定义为只需要一个对象。

7、this关键字

在Java基础中，this关键字是一个最重要的概念。使用this关键字可以完成以下的操作：

· 调用类中的属性

· 调用类中的方法或构造方法

· 表示当前对象

8、值传递与引用传递

示例一：值传递

public class ValueDemo{

public static void main(String[] args){

int x = 10;

method(x) ;

System.out.println(“x=”+x) ;

}

public static void method(int mx){

mx = 20;

}

}

示例二：引用传递

public class RefDemo1{

public static void main(String[] args){

Duck d = new Duck();

method(d) ;

System.out.println(“Duck age = ”+d.age) ;

}

public static void method(Duck duck){

duck.age = 5;

}

}

class Duck{

int age = 2; //省略封装

}

示例三：String传递

public class RefDemo2{

public static void main(String[] args){

String name = “小飞”;

method(name) ;

System.out.println(“name=”+name) ;

}

public static void method(String sname){

sname = “小备”;

}

}

示例四：String传递

public class RefDemo3{

public static void main(String[] args){

Person p = new Person();

method(p) ;

System.out.println(“person name= ”+p.name) ;

}

public static void method(Person p ){

p.name = “备备”;

}

}

class Person{

String name = “飞飞”; //省略封装

}

9、对象的一对一关系

两个对象之间的一对一关系：

比如：

一个英雄（Hero）对一个兵器（Weapon）

代码如何表示？

10、static关键字

static关键字的作用：

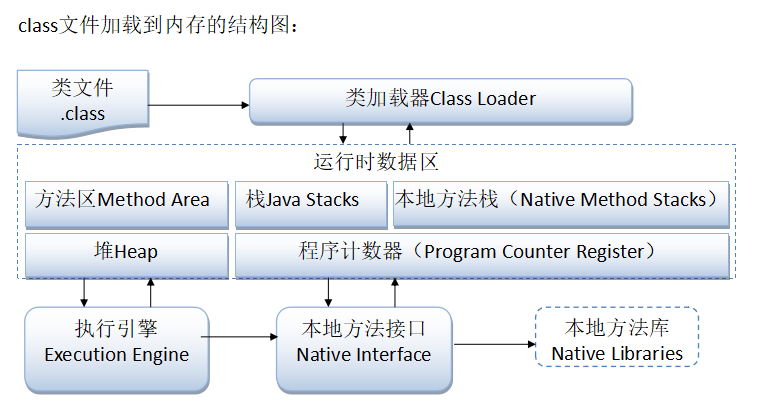
1、使用static关键字修饰一个属性

声明为static的变量实质上就是全局变量

2、使用static关键字修饰一个方法

通常，在一个类中定义一个方法为static，那就是说，无需本类的对象即可调用此方法

1. 使用static关键字修饰一个类（内部类）



声明为static的方法有以下几条限制：   
它们仅能调用其他的static 方法。   
它们只能访问static数据。   
它们不能以任何方式引用this 或super。

什么时候使用static?

所有对象共同的属性或方法，那么我们可以定义为静态的。

11、main方法分析

主方法：

public static void main(String[] args){

//代码块

}

public：公有的，最大的访问权限

static：静态的，无需创建对象

void:：表示没有返回值，无需向JVM返回结果

main：方法名，固定的方法名

String[] args：表示参数为字符串数组，可以在调用方法时传入参数

12、代码块

1、普通代码块

直接写在方法中的代码块就是普通代码块

示例：

public class Demo1{

public static void main(String []args){

{// 普通代码块

String info = “局部变量-1” ;

System.out.println(info) ;

}

String info = “局部变量-2” ;

System.out.println(info) ;

}

}

2、构造块是在类中定义的代码块

示例：

class Demo{

{ // 构造块

System.out.println("构造块") ;

}

public Demo(){

System.out.println("构造方法。") ;

}

}

3、在类中使用static声明的代码块称为静态代码块

示例：

class Demo{

{

System.out.println("构造块") ;

}

static{

System.out.println("静态代码块") ;

}

public Demo(){

System.out.println("构造方法。") ;

}

};

4、同步代码块（多线程中讲解）

13、单例设计模式

单例设计模式：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

1、构造方法私有化

2、声明一个本类对象

3、给外部提供一个静态方法获取对象实例

两种实现方式：

1、饿汉式

2、懒汉式

14、对象数组与管理

1、对象数组就是数组里的每个元素都是类的对象，赋值时先定义对象，然后将对象直接赋给数组。

示例：

Chicken[] cs= new Chicken[10];

使用对象数组实现多个Chicken的管理。

## 第05章：面向对象（下）

1、继承的基本概念

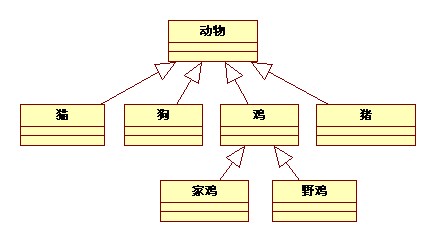
继承是从已有的类创建新类的过程。

1、继承是面向对象三大特征之一

2、被继承的类称为父类（超类），继承父类的类称为子类（派生类）

3、继承是指一个对象直接使用另一对象的属性和方法。

4、通过继承可以实现代码重用



语法：[访问权限] class 子类名 extends 父类名{

类体定义；

}

示例：

public class Dog{

private String name;

private String sex;

public void eat(){System.out.println(“吃饭”);}

}

public class HomeDog extends Dog{

//类的定义

}

public class HuskyDog extends Dog{

//类的定义

}

protected（受保护的访问权限修饰符，用于修饰属性和方法，使用protected修饰的属性和方法可以被子类继承）

2、继承的限制

继承的限制约定：

1、Java只能实现单继承，也就是一个类只能有一个父类

2、允许多层继承，即：一个子类可以有一个父类，一个父类还可以有其他的父类。

3、继承只能继承非私有的属性和方法。

4、构造方法不能被继承

继承小结

（1）继承是发生在多个类之间

（2）继承使用关键字extends

（3）JAVA只能单继承，允许多层继承

（4）被继承的类叫父类（超类），继承父类的类叫子类（派生类）

（5）在父类中的非私有属性和方法可以被子类继承

（6）protected（受保护的访问权限修饰符），修饰的属性或方法可以被子类继承

（7）构造方法不能被继承

（8）创建对象会调用构造方法，调用构造方法不一定就是创建对象

（9）实例化子类对象，会先调用父类的构造方法，如果父类中没有默认的构造方法，那么子类必须显示的通过super(...)来调用父类的带参构造方法，super也只能在子类构造方法中的第一句

继承的好处：

1、提高代码的复用性

2、提高代码的维护性

3、让类与类之间产生关系，是多态的前提

继承的缺点：

增强了类与类之间的耦合性

开发原则：高内聚，低耦合

3、子类的实例化过程

在子类进行实例化操作的时候，首先会先让其父类进行初始化操作。之后子类再自己进行实例化操作。

子类的实例化过程:

子类实例化时会先调用父类的构造方法

如果父类中没有默认的构造方法，在子类的构造方法中必须显示的调用父类的构造方法

结论：

构造方法只是用于初始化类中的字段以及执行一些初始化代码

调用构造方法并不代表会生成对象

4、方法的重写

方法重写(overriding method)

在Java中，子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想做一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖。

在子类和父类中，重写方法后，在调用时，以创建的对象类型为准，会调用谁的方法。

关于方法重写的一些特性：

1、发生在子父类中，方法重写的两个方法返回值、方法名、参数列表必须完全一致（子类重写父类的方法）

2、子类抛出的异常不能超过父类相应方法抛出的异常（子类异常不能大于父类异常）

3、子类方法的访问级别不能低于父类相应方法的访问级别(子类访问级别不能低于父类访问级别)

4、父类中的方法若使用private、static、final任意修饰符修饰，那么，不能被子类重写。

为什么要重写方法？或者方法重写的目的是什么？

若子类从父类中继承过来的方法，不能满足子类特有的需求时，子类就需要重写父类中相应的方法，方法的重写也是程序扩展的体现。

面试题 ：overloading与overriding的区别？

5、 super关键字

super可以完成以下的操作：

1、使用super调用父类中的属性，可以从父类实例处获得信息。

2、使用super调用父类中的方法，可以委托父类对象帮助完成某件事情。

3、使用super调用父类中的构造方法（super(实参)形式），必须在子类构造方法的第一条语句，调用父类中相应的构造方法，若不显示的写出来，默认调用父类的无参构造方法，比如：super();

this表示当前对象

使用super来调用父类的属性，方法，和构造方法

6、继承的应用示例

实现一个化妆品商城中的化妆品管理

1、定义一个化妆品类（Cosmetic）name,type,price

2、定义一个化妆品管理类（CosmeticManager）

（1）实现进货功能

（2）可以输出所有化妆品信息功能

3、使用继承实现一个可按单价排序输出所有化妆品的功能

4、使用继承实现一个只输出进口化妆品的功能

7、 final关键字

使用final关键字完成以下的操作：

1、使用final关键字声明一个常量

修饰属性或者修饰局部变量（最终变量），也称为常量。

2、使用final关键字声明一个方法

该方法为最终方法，且只能被子类继承，但是不能被子类重写。

3、使用final关键字声明一个类

该类就转变为最终类，没有子类的类，fianl修饰的类无法被继承。

4、在方法参数中使用final，在该方法内部不能修改参数的值（在内部类中详解）

final应用：

定义一个常量：

public static final int NUM = 10;

定义一个final方法（不常用）

定义一个final类：通常在常量类中使用

//常量类：在该类中只有常量，通常是应用程序中公共的常量或标记

public final class Constant{

public static final String SERVER\_ROOT\_URL = "http://www.baidu.com";

public static final String CACHE\_PATH = "data\_cache";

//....

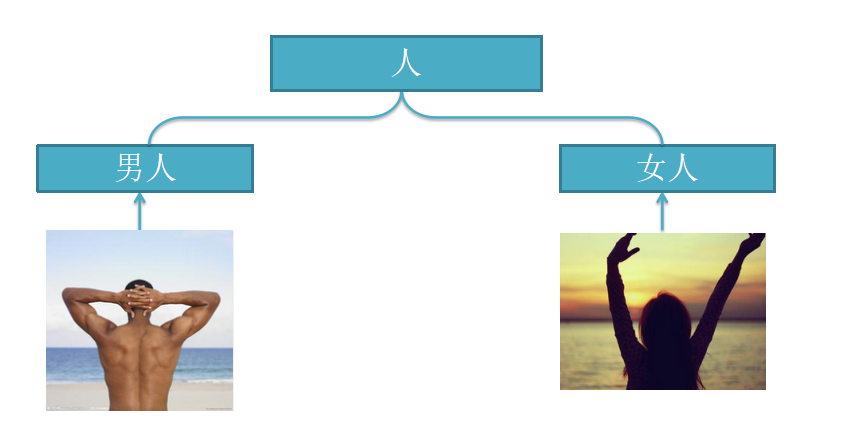
}

8、抽象类

抽象类的基本概念

（1）很多具有相同特征和行为的对象可以抽象为一个类；很多具有相同特征和行为的类可以抽象为一个抽象类。

（2）使用abstract关键字声明的类为抽象类。



定义一个抽象类

abstract class Animal{

public abstract void move();

}

abstract class Person extends Animal{

private String name;

//...

public abstract void eat();//抽象方法

}

//具体类

class Man extends Person{

public void eat(){

System.out.println("我是男人，我爱吃肉");

}

public void move(){

System.out.println("我爱跑步");

}

}

class Women extends Person{

public void eat(){

System.out.println("我是女人，我爱吃香蕉");

}

public void move(){

System.out.println("我喜欢逛街");

}

}

抽象类的规则：

1、抽象类可以没有抽象方法，有抽象方法的类必须是抽象类

2、非抽象类继承抽象类必须实现所有抽象方法

3、抽象类可以继承抽象类，可以不实现父类抽象方法。

4、抽象类可以有方法实现和属性

5、抽象类不能被实例化

6、抽象类不能声明为final

7、抽象类可以有构造方法

9、接口

接口的概念

1、接口是一组行为的规范、定义，没有实现（JDK1.8默认方法）

2、使用接口，可以让我们的程序更加利于变化

3、接口是面向对象编程体系中的思想精髓之一

4、面向对象设计法则：基于接口编程

接口的定义格式：

interface 接口名称{

全局常量 ;

抽象方法 ;

}

示列：

interface IEat{

//public abstract void eat();

void eat();

//public static final int NUM = 10;

int NUM = 10; }

interface ISleep extends IEat{

void sleep();

}

//实现接口的类

class Girl implements IEat,ISleep{

private String name;

public Girl(String name){

this.name = name;

}

public void eat(){

System.out.println("我是"+name+"的女票，我爱吃香蕉");

}

public void sleep(){

System.out.println("我爱睡觉");

}

}

接口的使用规则：

（1）定义一个接口，使用interface关键字

（2）在一个接口中，只能定义常量、抽象方法，JDK1.8后可以定义默认的实现方法

（3）接口可以继承多个接口：extends xxx,xxx

（4）一个具体类实现接口使用implements关键字

（5）一个类可以实现多个接口

（6）抽象类实现接口可以不实现接口的方法

（7）在接口中定义的方法没有声明 访问修饰符，默认为public

（8）接口不能有构造方法

（9）接口不能被实例化

面向对象设计原则：

（1）对修改关闭，对扩展开放

（2）面向接口编程

10、多态性

多态是面向对象三大特性之一

什么是多态性？

对象在运行过程中的多种形态。

多态性我们大概可以分为两类：

（1）方法的重载与重写

（2）对象的多态性

例如：

//用父类的引用指向子类对象（用大的类型去接受小的类型，向上转型、自动转换）

Chicken home = new HomeChicken();

结论：

在编程时针对抽象类型的编写代码，称为面向抽象编程（或面向接口编程）

父类通常都定义为抽象类、接口

对象的多态性：

对象多态性是从继承关系中的多个类而来，

向上转型：将子类实例转为父类引用

格式：父类 父类对象 = 子类实例 ； 自动转换

以基本数据类型操作为例：int i = ‘a' ;

（因为char的容量比int小，所以可以自动完成）

向下转型：将父类实例转为子类实例

格式：子类 子类对象 = （子类）父类实例 ；强制转换

以基本数据类型操作为例：char c = (char)97;

因为整型是4个字节比char 2个字节要大，所以需要强制完成

多态性小结：

1、方法的重载与重写就是方法的多态性表现

2、多个子类就是父类中的多种形态

3、父类引用可以指向子类对象，自动转换

4、子类对象指向父类引用需要强制转换（注意：类型不对会报异常）

5、在实际开发中尽量使用父类引用（更利于扩展）

11、 instanceof关键字

instanceof 是用于检查对象是否为指定的类型，通常在把父类引用强制转换为子类引用时要使用，以避免发生类型转换异常（ClassCastException）。

语法格式如下：

对象 instanceof 类 型 －－返回boolean类型值

示例：

if(homeChicken instanceof Chicken){

//...

}

该语句一般用于判断一个对象是否为某个类的实例，是返回true，否返回false

父类的设计法则

通过instanceof关键字，我们可以很方便 的检查对象的类型，但如果一个父类的子类过多，这样的判断还是显得很繁琐，那么如何去设计一个父类呢？

1、父类通常情况下都设计为抽象类或接口，其中优先考虑接口，如接口不能满足才考虑抽象类。

2、一个具体的类尽可能不去继承另一个具体类，这样的好处是无需检查对象是否为父类的对象。

12、抽象类应用—模板方法模式

模板方法模式（Templete Method）：定义一个操作中的算法的骨架，而将一些可变部分的实现延迟到子类中。模版方法模式使得子类可以不改变一个算法的结构即可重新定义该算法的某些特定的步骤。

13、接口应用—策略模式

策略模式（Strategy Pattern），定义了一系列的算法，将每一种算法封装起来并可以相互替换使用，策略模式让算法独立于使用它的客户应用而独立变化。

OO设计原则：

1、面向接口编程（面向抽象编程）

2、封装变化

3、多用组合，少用继承

14、 Object类

Object类 是类层次结构的根类。

每个类都使用 Object 作为超类。所有对象（包括数组）都实现这个类的方法

所有类都是Object类的子类。

public String toString()方法

返回该对象的字符串表示。

通常，toString 方法会返回一个“以文本方式表示”此对象的字符串。结果应是一个简明且易于读懂的信息表达式。建议所有子类都重写此方法。

public boolean equals(Object obj)

指示其他某个对象是否与此对象“相等”。 equals 方法在非空对象引用上实现相等关系：

自反性

对称性

传递性

一致性

protected void finalize()throws Throwable

当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。子类

重写 finalize 方法，以配置系统资源或执行其他清除。

public final Class<?> getClass()

返回此 Object 的运行时类。

其它方法在后续章节中详解。

15、简单工厂模式

简单工厂模式是由一个工厂对象决定创建出哪一种产品类的实例。简单工厂模式是工厂模式家族中最简单实用的模式。

16、静态代理模式

代理模式（Proxy）：为其他对象提供一种代理以控制对这个对象的访问。

代理模式说白了就是“真实对象”的代表，在访问对象时引入一定程度的间接性，因为这种间接性可以附加多种用途。

17、适配器模式

适配器模式（ Adapter ）：将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

OO设计原则：

1、面向接口编程（面向抽象编程）

2、封装变化

3、多用组合，少用继承

4、对修改关闭，对扩展开放

18、内部类

内部类就是在一个类的内部定义的类。

成员内部类格式如下：

class Outer {

class Inner{}

}

编译上述代码会产生两个文件：

Outer.class和Outer$Inner.class。

在外部创建内部类对象

内部类除了可以在外部类中产生实例化对象，也可以在外部类的外部来实例化。

那么，根据内部类生成的\*.class文件：Outer$Inner.class

“$” 符号在程序运行时将替换成“.”

所以内部类的访问：通过“外部类.内部类”的形式表示。

Outer out = new Outer() ;// 产生外部类实例

Outer.Inner in = null; // 声明内部类对象

in = out.new Inner() ; // 实例化内部类对象

方法内部类

内部类可以作为一个类的成员外，还可以把类放在方法内定义。

注意：

1、方法内部类只能在定义该内部类的方法内实例化，不可以在此方法外对其实例化。

1. 方法内部类对象不能使用该内部类所在方法的非final局部变量。

格式如下：

class Outer {

public void doSomething(){

class Inner{

public void seeOuter(){}

}

}

}

静态内部类

在一个类内部定义一个静态内部类：

静态的含义是该内部类可以像其他静态成员一样，没有外部类对象时，也能够访问它。静态嵌套类仅能访问外部类的静态成员和方法。

class Outer{

static class Inner{}

}

class Test {

public static void main(String[] args){

Outer.Inner n = new Outer.Inner();

}

}

匿名内部类

匿名内部类就是没有名字的内部类。

匿名内部类的三种情况：

（1）继承式的匿名内部类

（2）接口式的匿名内部类

（3）参数式的匿名内部类

在使用匿名内部类时，要记住以下几个原则：

（1）不能有构造方法，只能有一个实例。

（2）不能定义任何静态成员、静态方法。

（3）不能是public,protected,private,static。

（4）一定是在new的后面，用其隐含实现一个接口或继承一个类。

（5）匿名内部类为局部的，所以局部内部类的所有限制都对其生效。

问题：局部内部类访问局部变量必须用final修饰，为什么?

当调用这个方法时，局部变量如果没有用final修饰，他的生命周期和方法的生命周期是一样的，当方法被调用时会入栈，方法结束后即弹栈，这个局部变量也会消失，那么如果局部内部类对象还没有马上消失想用这个局部变量，显然已无法使用了，如果用final修饰会在类加载的时候进入常量池，即使方法弹栈，常量池的常量还在，也就可以继续使用了。

注意：在jdk1.8中取消了在局部内部类中使用的变量必须显示的使用final修饰，编译器默认会为这个变量加上final

内部类的作用

每个内部类都能独立地继承自一个（接口的）实现，所以无论外部类是否已经继承了某个（接口的）实现，对于内部类都没有影响。如果没有内部类提供的可以继承多个具体的或抽象的类的能力，一些设计与编程问题就很难解决。从这个角度看，内部类使得多重继承的解决方案变得完整。接口解决了部分问题，而内部类有效地实现了“多重继承”。

成员

成员内部类

静态内部类

局部

方法内部类

匿名内部类

依赖外部类对象的：成员内部类，方法内部类，匿名内部类

静态内部类不依赖外部类的对象。所以，我们在项目中优先考虑选择静态内部类（不会产生内存泄露）

我们项目开发中如何选择？

19、数据结构之链表

链表(Linked list）一种常见的基础数据结构，是一种线性表，但是并不会按线性的顺序存储数据，而是在每一个节点里存到是下一个节点的指针(Pointer）。

在链表数据结构中，我们需要使用到递归算法。

递归算法是一种直接或者间接地调用自身算法的过程。在计算机编写程序中，递归算法对解决一大类问题是十分有效的，它往往使算法的描述简洁而且易于理解。

20、基本数据类型包装类

在Java中有一个设计的原则“一切皆对象”，Java中的基本数据类型就完全不符合这种设计思想，因为八种基本数据类型并不是引用数据类型，所以Java中为了解决这样的问题，JDK1.5以后引入了八种基本数据类型的包装类。

八种包装类分为两大类型：

· Number：Integer、Short、Long、Double、Float、Byte都是Number的子 类表示是一个数字。

· Object：Character、Boolean都是Object的直接子类。

|  |  |
| --- | --- |
| 基本数据类型 | 包装类 |
| int | Integer |
| char | Character |
| float | Float |
| double | Double |
| boolean | Boolean |
| byte | Byte |
| short | Short |
| long | Long |

装箱及拆箱操作

将一个基本数据类型转换为包装类，那么这样的操作称为装箱操作。将一个包装类转换为一个基本数据类型，这样的操作称为拆箱操作。

|  |  |
| --- | --- |
| 方法 | 描述 |
| byteValue() | Byte  byte |
| doubleValue() | Double  double |
| floatValue() | Float  float |
| intValue() | Integer  int |
| longValue() | Long  long |
| shortValue() | Short  short |

转型操作

在包装类中，可以将一个字符串变为指定的基本数据类型，一般在输入数据时会使用较多。

在Integer类中将String变为int型数据：public static int parseInt(String s)

在Float类中将String变为float型数据：public static float parseFloat(String s)

注意：转型操作时，字符串必须由数字组成，否则会出现错误哦

享元模式（Flyweight Pattern）它使用共享对象，用来尽可能减少内存使用量以及分享资讯给尽可能多的相似对象；它适合用于当大量对象只是重复因而导致无法令人接受的使用大量内存。通常对象中的部分状态是可以分享。常见做法是把它们放在外部数据结构，当需要使用时再将它们传递给享元。

运用共享技术有效的支持大量细粒度的对象。

21、包与访问修饰符

包用于对多个java源文件的管理，就像我们的文件目录一样。

定义一个包：

package com.vince;

该语句只能出现在代码中的第一句。

访问修饰符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **访问修饰符** | **同一个类** | **同包** | **不同包子类** | **不同包非子类** |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ |  |
| 默认 | √ | √ |  |  |
| private | √ |  |  |  |

OO原则：

（1）开闭原则

一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。

（2）合成/聚合复用原则

新对象的某些功能在已创建好的对象里已实现，那么尽量用已有对象提供的功能，使之成为新对象的一部分，而不要再重新创建。

（3）依赖倒置原则

高层模块不应该依赖低层模块，二者都应该依赖其抽象；抽象不应该依赖细节；细节应该依赖抽象。

（4）接口隔离原则

客户端不应该依赖它不需要的接口；一个类对另一个类的依赖应该建立在最小的接口上。

（5）迪米特法则

一个对象应该对其他对象保持最少的了解

（6）里氏替换原则

所有引用基类的地方必须能透明地使用其子类的对象

（7）单一职责原则

不要存在多于一个导致类变更的原因，即一个类只负责一项职责。

## 第06章：Eclipse与异常处理

1、Eclipse开发环境使用入门

2、异常处理

3、自定义异常

4、受检与非受检异常

5、Eclipse之debug技术

## 第07章：常用类库API

1、字符串操作

String类

1、String可以表示一个字符串。

2、String类实际是使用字符数组存储的。

String类的两种赋值方式：

（1）一种称为直接赋值：

String name = “小白”

（2）通过关键字new调用String的构造方法赋值

String name = new String(“小白”)

String类的两种赋值分析：

1、字符串常量池

2、String name = new String(“小白”)，在内存中的分析

3、String name = “小白”，在内存中的分析

String类编译期与运行期分析：

1、编译期和运行期

2、代码示例：4种情况分析

String类字符与字符串操作方法：

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public char charAt(int index) | 普通 | 根据下标找到指定的字符 |
| 2 | public char[] toCharArray() | 普通 | 以字符数组的形式返回全部的字符串内容 |
| 3 | public String(char[] value) | 构造 | 将全部的字符数组变为字符串 |
| 4 | public String(char[] value,int offset,int count) | 构造 | 将指定范围的字符数组变为字符串 |

String类字节与字符串操作方法：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 1 | public byte[] getBytes() | 普通 | 将字符串变为字节数组 |
| 2 | public String(byte[] bytes) | 构造 | 将字节数组变为字符串 |
| 3 | public String(byte[] bytes,int offset,int length) | 构造 | 将指定范围的字节数组变为字符串 |
| 4 | public String(byte[] bytes, String charsetName) | 构造 | 通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String |

String类判断是否以指定内容开头或结尾：

|  |  |  |  |
| --- | --- | --- | --- |
| No | 方法名称 | 类型 | 描述 |
| 1 | public boolean startsWith(String prefix) | 普通 | 从第一个位置开始判断是否以指定的内容开头 |
| 2 | public boolean startsWith(String prefix,int toffset) | 普通 | 从指定的位置开始判断是否以指定的内容开头 |
| 3 | public boolean endsWith(String suffix) | 普通 | 判断是否以指定的内容结尾 |

String类替换操作：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 1 | public String replace(char oldChar,char newChar) | 普通 | 替换指定字符 |
| 2 | public String replace(CharSequence target, CharSequence replacement) | 普通 | 替换指定字符串 |
| 3 | public String replaceAll(String regex,String replacement) | 普通 | 替换指定的字符串 |
| 4 | public String replaceFirst(String regex,String replacement) | 普通 | 替换第一个满足条件的字符串 |

String类字符串截取操作：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 1 | public String substring(int beginIndex) | 普通 | 从指定位置开始一直截取到末尾 |
| 2 | public String substring(int beginIndex,int endIndex) | 普通 | 截取指定范围的字符串 |

String类字符串拆分操作：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 1 | public String[] split(String regex) | 普通 | 按照指定的字符串拆分 |
| 2 | public String[] split(String regex,int limit) | 普通 | 拆分字符串，并指定拆分的个数 |

String类字符串查找操作：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 1 | public boolean contains(String s) | 普通 | 返回一个字符串是否存在 |
| 2 | public int indexOf(int ch) | 普通 | 从头查找指定的字符是否存在，char  int，如果存在则返回位置，如果不存在则返回“-1”。 |
| 3 | public int indexOf(int ch,int fromIndex) | 普通 | 从指定位置查找指定的字符是否存在，char  int，如果存在则返回位置，如果不存在则返回“-1”。 |
| 4 | public int indexOf(String str) | 普通 | 从头查找指定的字符串是否存在，如果存在则返回位置，如果不存在则返回“-1”。 |
| 5 | public int indexOf(String str,int fromIndex) | 普通 | 从指定位置查找字符串是否存在，如果存在则返回位置，如果不存在则返回“-1”。 |

String类字符串查找操作：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 6 | public int lastIndexOf(int ch) | 普通 | 从字符串的最后向前查找，指定的字符是否存在，如果存在则返回位置，如果不存在则返回“-1”。 |
| 7 | public int lastIndexOf(int ch,int fromIndex) | 普通 | 从字符串的指定的末尾向前查找，指定的字符是否存在，如果存在则返回位置，如果不存在则返回“-1”。 |
| 8 | public int lastIndexOf(String str) | 普通 | 从字符串的最后向前查找，指定的字符串是否存在，如果存在则返回位置，如果不存在则返回“-1”。 |
| 9 | public int lastIndexOf(String str,int fromIndex) | 普通 | 从字符串的指定的末尾向前查找，指定的字符串是否存在，如果存在则返回位置，如果不存在则返回“-1”。 |

String类其它操作方法：

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 方法名称 | 类型 | 描述 |
| 1 | public boolean isEmpty() | 普通 | 判断是否为空，指的是内容为空“""” |
| 2 | public int length() | 普通 | 取得字符串的长度 |
| 3 | public String toLowerCase() | 普通 | 转小写 |
| 4 | public String toUpperCase() | 普通 | 转大写 |
| 5 | public String trim() | 普通 | 去掉开头和结尾的空格，中间的空格不去 |
| 6 | public String concat(String str) | 普通 | 字符串连接操作 |

StringBuffer类

在实际开发当中，我们经常会使用到字符串连接的操作，如果用String来操作，则使用“+”号完成字符串的连接操作。

使用String连接字符串，代码性能会非常低，因为String的内容不可改变。

解决这个问题的方法是使用StringBuffer。

StringBuffer类源码分析

StringBuffer常用操作方法

|  |  |
| --- | --- |
| **方法名称** | **描述** |
| public StringBuffer() | 构造一个空的StringBuffer对象 |
| public StringBuffer(String str) | 将指定的String变为StringBuffer的内容 |
| public StringBuffer(CharSequence seq) | 接收CharSequence接口的实例 |
| public StringBuffer append(数据类型 b) | 提供了很多的append()方法，用于进行字符串连接 |
| public StringBuffer delete(int start,int end) | 删除指定位置的内容 |
| public int indexOf(String str) | 字符串的查询功能 |
| public StringBuffer insert(int offset,数据类型 b) | 在指定位置上增加一个内容 |
| public StringBuffer replace(int start,int end,String str) | 将指定范围的内容替换成其他内容 |
| public String substring(int start,int end) | 截取指定范围的字符串 |
| public String substring(int start) | 字符串截取 |
| public StringBuffer reverse() | 字符串反转 |

StringBuffer的兄弟StringBuilder：

一个可变的字符序列。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。

JDK1.5以后，字符串相加原理分析

2、程序国际化

1、对国际化程序的理解

Internationalization：国际化程序可以这样理解：

同一套程序代码可以在各个语言环境下进行使用。

各个语言环境下，只是语言显示的不同，那么具体的程序操作本身都是一样的，那么国际化程序完成的就是这样的一个功能。



2、Locale类

Locale 对象表示了特定的地理、政治和文化地区。需要 Locale 来执行其任务的操作称为语言环境敏感的操作，它使用 Locale 为用户量身定制信息。例如，显示一个数值就是语言环境敏感的操作，应该根据用户的国家、地区或文化的风俗/传统来格式化该数值。

使用此类中的构造方法来创建 Locale：

Locale(String language)

Locale(String language, String country)

通过静态方法创建Locale：

getDefault()

3、ResourceBundle类

国际化的实现核心在于显示的语言上，通常的做法是将其定义成若干个属性文件（文件后缀是\*.properties），属性文件中的格式采用“key=value”的格式进行操作。

ResourceBundle类表示的是一个资源文件的读取操作，所有的资源文件需要使用ResourceBundle进行读取，读取的时候不需要加上文件的后缀。

getBundle(String baseName)

getBundle(String baseName,Locale locale)

getString(String key)

4、处理动态文本

前面的示例读取的内容都是固定的，如果现在假设要想打印这样的信息“欢迎你，XXX！”，具体的名字不是固定的，那么就要使用动态文本进行程序的处理。

进行动态的文本处理，必须使用java.text.MessageFormat类完成。这个类是java.text.Format的子类。

3、Math与Random类

1、Math类

Math 类包含用于执行基本数学运算的方法，如初等指数、对数、平方根和三角函数。

使用Math类可以有两种方式：

(1)、直接使用（Math所在的包java.lang为默认引入的包）

(2)、使用 import static java.lang.Math.abs; 静态导入

|  |  |
| --- | --- |
| static double PI | 比任何其他值都更接近 *pi*的 double 值 |
| **abs**(double a) | 返回 double 值的绝对值 |
| **random**() | 返回带正号的 double 值，该值大于等于 0.0 且小于 1.0 |
| **round**(double a) | 返回最接近参数并等于某一整数的 double 值 |
| **sqrt**(double a) | 返回正确舍入的 double 值的正平方根 |

2、Random类

Random：此类的实例用于生成伪随机数流

|  |  |
| --- | --- |
| nextLong() | 返回下一个伪随机数的long 值 |
| nextBoolean() | 返回下一个伪随机数boolean 值 |
| nextDouble() | 返回下一个伪随机数，在 0.0 和 1.0 之间的 double 值 |
| nextFloat() | 返回下一个伪随机数，在 0.0 和 1.0 之间的 float 值 |
| nextInt() | 返回下一个伪随机数，int 值 |
| nextInt(int n) | 返回一个伪随机数，在 0（包括）和指定值分布的 int 值 |

4、日期操作类

1、Date类

类 Date 表示特定的瞬间，精确到毫秒，也就是程序运行时的当前时间。

Date date = new Date(); // 实例化Date对象，表示当前时间

2、Calendar类

Calendar，日历类，使用此类可以将时间精确到毫秒显示。

//两种实例化方式

Calendar c = Calendar.getInstance();

Calendar c = new GregorianCalendar();

3、 DateFormat类及子类SimpleDateFormat

5、对象比较器

对两个或多个数据项进行比较，以确定它们是否相等，或确定它们之间的大小关系及排列顺序称为比较。

前面我学习过Arrays.sort方法可实现对象的排序操作：

public static void sort(Object[] a)

（1）Comparable接口：

此接口强行对实现它的每个类的对象进行整体排序。这种排序被称为类的自然排序，类的 compareTo 方法被称为它的自然比较方法。

（2）Comparator接口：

Comparable接口是要求自定义类去实现，按照OO原则：对修改关闭，对扩展开放。

那么如果这个类已经定义好了，不想再去修改它，那如何实现比较呢？

Comparator接口：强行对某个对象collection进行整体排序的比较。

6、对象的克隆

将一个对象复制一份，称为对象的克隆技术。

在Object类中存在一个clone()方法：

protected Object clone() throws CloneNotSupportedException

如果某个类的对象要想被克隆，则对象所在的类必须实现Cloneable接口。此接口没有定义任何方法，是一个标记接口。

7、System与 Runtime类

System类代表系统，系统级的很多属性和控制方法都放置在该类的内部。该类位于java.lang包。

1、成员变量

System类内部包含in、out和err三个成员变量，分别代表标准输入流(键盘输入)，标准输出流(显示器)和标准错误输出流。

2、成员方法

System类中提供了一些系统级的操作方法

（1）public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

该方法的作用是数组拷贝，也就是将一个数组中的内容复制到另外一个数组中的指定位置，由于该方法是native方法，所以性能上比使用循环高效。

（2）public static long currentTimeMillis()

该方法的作用是返回当前的计算机时间，时间的表达格式为当前计算机时间和GMT时间(格林威治时间)1970年1月1号0时0分0秒所差的毫秒数。

（3）public static void exit(int status)

该方法的作用是退出程序。其中status的值为0代表正常退出，非零代表异常退出。使用该方法可以在图形界面编程中实现程序的退出功能等。

（4）public static void gc()

该方法的作用是请求系统进行垃圾回收。至于系统是否立刻回收，则取决于系统中垃圾回收算法的实现以及系统执行时的情况。

（5）public static String getProperty(String key)

该方法的作用是获得系统中属性名为key的属性对应的值。

java.version Java 运行时环境版本

java.home Java 安装目录

os.name 操作系统的名称

os.version 操作系统的版本

user.name 用户的账户名称

user.home 用户的主目录

user.dir 用户的当前工作目录

Runtime类：每个 Java 应用程序都有一个 Runtime 类实例，使应用程序能够与其运行的环境相连接。

//获取Java运行时相关的运行时对象

Runtime rt = Runtime.getRuntime();

System.out.println("处理器数量：" + rt.availableProcessors()+" 个");

System.out.println("Jvm总内存数 ："+ rt.totalMemory()+" byte");

System.out.println("Jvm空闲内存数： "+ rt.freeMemory()+" byte");

System.out.println("Jvm可用最大内存数： "+ rt.maxMemory()+" byte");

//在单独的进程中执行指定的字符串命令。

rt.exec("notepad");

8、数字处理工具类

BigInteger：可以让超过Integer范围内的数据进行运算。

构造方法：public BigInteger(String val)

常用方法：

public BigInteger add(BigInteger val)

public BigInteger subtract(BigInteger val)

public BigInteger multiply(BigInteger val)

public BigInteger divide(BigInteger val)

public BigInteger[] divideAndRemainder(BigInteger val)

BigDecimal：由于在运算的时候，float类型和double很容易丢失精度，为了能精确的表示、计算浮点数，Java提供了BigDecimal，不可变的、任意精度的有符号十进制数。

构造方法：public BigDecimal(String val)

常用方法：

public BigDecimal add(BigDecimal augend)

public BigDecimal subtract(BigDecimal subtrahend)

public BigDecimal multiply(BigDecimal multiplicand)

public BigDecimal divide(BigDecimal divisor)

DecimalFormat：Java 提供 DecimalFormat类，帮你用最快的速度将数字格式化为你需要的样子。

例如，取2位小数。

示例：

double pi=3.1415927;　//圆周率

//取一位整数，结果：3

System.out.println(new DecimalFormat("0").format(pi));

//取一位整数和两位小数，结果3.14

System.out.println(new DecimalFormat("0.00").format(pi));

//取两位整数和三位小数，整数不足部分以0填补，结果：03.142

System.out.println(new DecimalFormat("00.000").format(pi));

//取所有整数部分，结果：3

System.out.println(new DecimalFormat("#").format(pi));

//以百分比方式计数，并取两位小数，结果：314.16%

System.out.println(new DecimalFormat("#.##%").format(pi));

9、MD5工具类

MD5的全称是Message-Digest Algorithm 5（信息-摘要算法）

//确定计算方法

MessageDigest md5=MessageDigest.getInstance("MD5");

//JDK1.8新增Base64

String newstr = Base64.getEncoder().encodeToString(md5.digest(str.getBytes("utf-8")));

//1.8之前使用sun.misc.BASE64Encoder(此类没有访问权限，在rt.jar中添加访问权限：sun/misc/\*)

BASE64Encoder base64 = new BASE64Encoder();

base64.encode(md5.digest(str.getBytes("utf-8")));

10、数据结构之二叉树实现

树是一种重要的非线性数据结构，直观地看，它是数据元素（在树中称为结点）按分支关系组织起来的结构。二叉树（Binary Tree）是每个节点最多有两个子树的有序树。通常子树被称作“左子树” 和“右子树”。

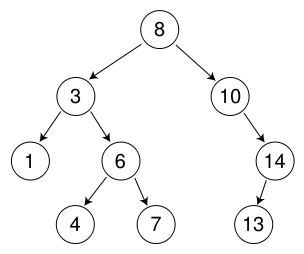
二叉树算法的排序规则：

1、选择第一个元素作为根节点

2、之后如果元素大于根节点放在右子树，如果元素小于根节点，则放在左子树

3、最后按照中序遍历的方式进行输出，则可以得到排序的结果（左根右）

8、3、10、1、6、14、4、7、13



11、JDK1.8新特性

1、Lambda表达式

Lambda表达式（也称为闭包）是整个Java 8发行版中最受期待的在Java语言层面上的改变，Lambda允许把函数作为一个方法的参数（函数作为参数传递进方法中），或者把代码看成数据。Lambda表达式用于简化JAVA中接口式的匿名内部类。被称为函数式接口的概念。函数式接口就是一个具有一个方法的普通接口。像这样的接口，可以被隐式转换为lambda表达式。

语法：

(参数1,参数2...) -> { ... }

（1）没有参数时使用Lambda表达式

（2）带参数时使用Lambda表达式

（3）代码块中只一句代码时使用Lambda表达式

（4）代码块中有多句代码时使用Lambda表达式

（5）有返回值的代码块

（6）参数中使用final关键字

2、接口中的默认方法和静态方法

interface A{

public default void print(){}

public static void method(){}

}

默认方法与静态方法并不影响函数式接口的契约，可以任意使用

## 第08章：文件与IO

1、File类

1、File类的基本概念

File类：表示文件和目录路径名的抽象表示形式。

File类可以实现文件的创建、删除、重命名、得到路径、创建时间等等，是唯一与文件本身有关的操作类。

2、File类的操作方法

|  |  |
| --- | --- |
| public static final String separator | 表示路径分隔符“\” |
| public File(String pathname) | 构造File类实例，要传入路径 |
| public boolean createNewFile() | 创建新文件 |
| public boolean delete() | 删除文件 |
| public boolean isDirectory() | 判断给定的路径是否是文件夹 |

|  |  |
| --- | --- |
| public boolean isFile() | 判断给定的路径是否是文件 |
| public String[] list() | 列出文件夹中的文件 |
| public File[] listFiles() | 列出文件夹中的所有文件 |
| public boolean mkdir() | 创建新的文件夹 |
| public boolean renameTo(File dest) | 为文件重命名 |
| public long length() | 返回文件大小 |
| String getPath() | 路径名字符串。 |

2、字节流

1、IO流概述

IO流：输入输出流（Input/Output）

流是一组有顺序的，有起点和终点的字节集合，是对数据传输的总称或抽象。即数据在两设备间的传输称为流

流的本质是数据传输，根据数据传输特性将流抽象为各种类，方便更直观的进行数据操作。

IO流的分类

根据处理数据类型的不同分为：字符流和字节流

根据数据流向不同分为：输入流和输出流

2、字节输出流

OutputStream类定义

public abstract class OutputStream extends Object implements Closeable, Flushable

此抽象类是表示输出字节流的所有类的超类。输出流接受输出字节并将这些字节发送到InputStream 类某个接收器要向文件中输出，使用FileOutputStream类

3、字节输入流

定义：

public abstract class InputStream extends Object implements Closeable

此抽象类是表示字节输入流的所有类的超类。

FileInputStream 从文件系统中的某个文件中获得输入字节。

3、字符流

Writer

写入字符流的抽象类。子类必须实现的方法仅有 write(char[], int, int)、flush() 和 close()。但是，多数子类将重写此处定义的一些方法，以提供更高的效率和/或其他功能。

与OutputStream一样，对文件的操作使用：FileWriter类完成。

Reader  
用于读取字符流的抽象类。子类必须实现的方法只有 read(char[], int, int) 和 close()。但是，多数子类将重写此处定义的一些方法，以提供更高的效率和/或其他功能。

使用FileReader类进行实例化操作。

字节流与字符流的区别

在所有的流操作里。字节永远是最基础的。任何基于字节的操作都是正确的。无论你是文本文件还是二进制的文件。 如果确认流里面只有可打印的字符，包括英文的和各种国家的文字，也包括中文，那么可以考虑用字符流。 由于编码不同，多字节的字符可能占用多个字节。比如GBK的汉字就占用2个字节，而UTF-8的汉字就占用3个字节。 所以，字符流是根据指定的编码，将1个或多个字节转化为java里面的unicode的字符，然后进行操作。 字符操作一般使用Writer,Reader等， 字节操作一般都是InputStream, OutputStream 以及各种包装类，比如BufferedInputStream和BufferedOutputStream等。   
  
总结：如果你确认你要处理的流是可打印的字符，那么使用字符流会看上去简单一点。如果不确认，那么用字节流总是不会错的。

指定一个盘符下的文件，把该文件复制到指定的目录下。

4、字节字符转换流

转换流，可以将一个字节流转换为字符流，也可以将一个字符流转换为字节流。

OutputStreamWriter：可以将输出的字符流转换为字节流的输出形式

InputStreamReader：将输入的字节流转换为字符流输入形式

5、缓冲流

首先要明确一个概念：

对文件或其它目标频繁的读写操作，效率低，性能差。

使用缓冲流的好处是，能够更高效的读写信息，原理是将数据先缓冲起来，然后一起写入或者读取出来。

BufferedInputStream： 为另一个输入流添加一些功能，在创建 BufferedInputStream 时，会创建一个内部缓冲区数组，用于缓冲数据。

BufferedOutputStream：通过设置这种输出流，应用程序就可以将各个字节写入底层输出流中，而不必针对每次字节写入调用底层系统。

BufferedReader：从字符输入流中读取文本，缓冲各个字符，从而实现字符、数组和行的高效读取。

BufferedWriter：将文本写入字符输出流，缓冲各个字符，从而提供单个字符、数组和字符串的高效写入。

6、打印流

打印流的主要功能是用于输出，在整个IO包中打印流分为两种类型：

字节打印流：PrintStream

字符打印流：PrintWriter

打印流可以很方便的进行输出

7、对象流

对象流的两个类：

ObjectOutputStream 将 Java 对象的基本数据类型和图形写入 OutputStream

ObjectInputStream 对以前使用 ObjectOutputStream 写入的基本数据和对象进行反序列化。

序列化一组对象：

在序列化操作中，同时序列化多个对象时，反序列化也必须按顺序操作，如果想要序列化一组对象该如何操作呢？

序列化一组对象可采用：对象数组的形式，因为对象数组可以向Object进行转型操作。

transient关键字：

如果用transient声明一个实例变量，当对象存储时，它的值不需要维持。

8、字节数组流

ByteArrayInputStream

包含一个内部缓冲区，该缓冲区包含从流中读取的字节。内部计数器跟踪 read 方法要提供的下一个字节。 关闭 ByteArrayInputStream 无效。此类中的方法在关闭此流后仍可被调用，而不会产生任何 IOException。

ByteArrayOutputStream

此类实现了一个输出流，其中的数据被写入一个 byte 数组。缓冲区会随着数据的不断写入而自动增长。可使用 toByteArray() 和 toString() 获取数据。 关闭 ByteArrayOutputStream 无效。此类中的方法在关闭此流后仍可被调用，而不会产生任何 IOException。

9、数据流

DataInputStream：

数据输入流允许应用程序以与机器无关方式从底层输入流中读取基本 Java 数据类型。应用程序可以使用数据输出流写入稍后由数据输入流读取的数据。 DataInputStream 对于多线程访问不一定是安全的。 线程安全是可选的，它由此类方法的使用者负责。

DataOutputStream：

数据输出流允许应用程序以适当方式将基本 Java 数据类型写入输出流中。然后，应用程序可以使用数据输入流将数据读入。

案例：

实现文件分割合并。

10、字符串流、管道流、合并流

一、合并流：

SequenceInputStream 表示其他输入流的逻辑串联。它从输入流的有序集合开始，并从第一个输入流开始读取，直到到达文件末尾，接着从第二个输入流读取，依次类推，直到到达包含的最后一个输入流的文件末尾为止。

二、字符串流

1、StringReader

其源为一个字符串的字符流。

2、 StringWriter

一个字符流，可以用其回收在字符串缓冲区中的输出来构造字符串。 关闭 StringWriter 无效。此类中的方法在关闭该流后仍可被调用，而不会产生任何 IOException。

三、管道流

管道输入流应该连接到管道输出流；管道输入流提供要写入管道输出流的所有数据字节。通常，数据由某个线程从 PipedInputStream 对象读取，并由其他线程将其写入到相应的 PipedOutputStream。不建议对这两个对象尝试使用单个线程，因为这样可能死锁线程。管道输入流包含一个缓冲区，可在缓冲区限定的范围内将读操作和写操作分离开。如果向连接管道输出流提供数据字节的线程不再存在，则认为该管道已损坏。

11、RandomAccessFile

RandomAccessFile是IO包的类，从Object直接继承而来。 只可以对文件进行操作，可以对文件进行读取和写入。  
当模式为r是，当文件不存在时会报异常，当模式为rw时，当文件不存在时，会自己动创建文件，当文件已经存在时 不会对原有文件进行覆盖。  
RandomAccessFile有强大的文件读写功能，其内部是大型 byte[]，可以通过seek(),getFilePointer()等方法操作的指针，方便对数据进行写入与读取。还可以对基本数据类型进行直接的读和写操作。

RandomAccessFile的绝大多数功能，已经被JDK 1.4的nio的“内存映射文件(memory-mapped files)”给取代了，你该考虑一下是不是用“内存映射文件”来代替RandomAccessFile了。

12、Properties文件操作

Properties（Java.util.Properties），主要用于读取Java的配置文件，各种语言都有自己所支持的配置文件，配置文件中很多变量是经常改变的，这样做也是为了方便用户，让用户能够脱离程序本身去修改相关的变量设置。

它提供了几个主要的方法：

1、getProperty ( String key)，用指定的键在此属性列表中搜索属性。也就是通过参数 key ，得到 key 所对应的 value。

2、 load ( InputStream inStream)，从输入流中读取属性列表（键和元素对）。通过对指定的文件（比如说上面的 test.properties 文件）进行装载来获取该文件中的所有键 - 值对。以供 getProperty ( String key) 来搜索。

3、setProperty ( String key, String value) ，调用 Hashtable 的方法 put 。他通过调用基类的put方法来设置 键 - 值对。

4、store ( OutputStream out, String comments)，以适合使用 load 方法加载到 Properties 表中的格式，将此 Properties 表中的属性列表（键和元素对）写入输出流。与 load 方法相反，该方法将键 - 值对写入到指定的文件中去。

5、 clear ()，清除所有装载的 键 - 值对。该方法在基类中提供。

13、文件压缩与解压缩

java中实现zip的压缩与解压缩

ZipOutputStream

实现文件的压缩

ZipOutputStream(OutputStream out)

创建新的 ZIP 输出流。

void putNextEntry(ZipEntry e)

开始写入新的 ZIP 文件条目并将流定位到条目数据的开始处。

ZipEntry(String name) //test/mm.jpg /test/a.txt

使用指定名称创建新的 ZIP 条目。

ZipInputStream

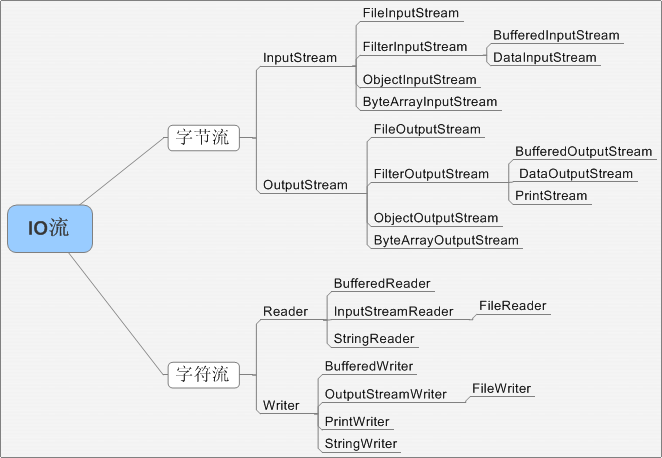
实现文件的解压

ZipInputStream(InputStream in)

创建新的 ZIP 输入流。

ZipEntry getNextEntry()

读取下一个 ZIP 文件条目并将流定位到该条目数据的开始处。



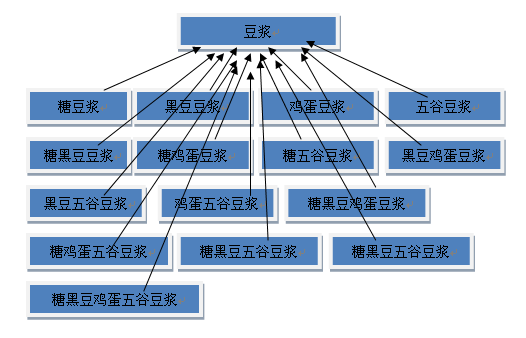
14、装饰者模式

1、继承带来的问题

话说威哥当年创业开店－－卖豆浆！因豆浆纯，分店几乎开遍全县城所有村，由于发展的实在是太快了，所以急于实现一套由计算机管理的自动化记账系统。

主料：豆浆

配料：糖 黑豆 五谷 鸡蛋…



装饰者模式，看看她能给我们带来什么惊喜吧！

意图：

动态地给一个对象添加一些额外的职责。就增加功能来说，Decorator模式相比生成子类更为灵活。该模式以对客户端透明的方式扩展对象的功能。

适用环境

在不影响其他对象的情况下，以动态、透明的方式给单个对象添加职责。

处理那些可以撤消的职责。

当不能采用生成子类的方法进行扩充时。一种情况是，可能有大量独立的扩展，为支持每一种组合将产生大量的子类，使得子类数目呈爆炸性增长。另一种情况可能是因为类定义被隐藏，或类定义不能用于生成子类。

类图：

Component（被装饰对象基类）

定义对象的接口，可以给这些对象动态增加职责；

ConcreteComponent（具体被装饰对象）

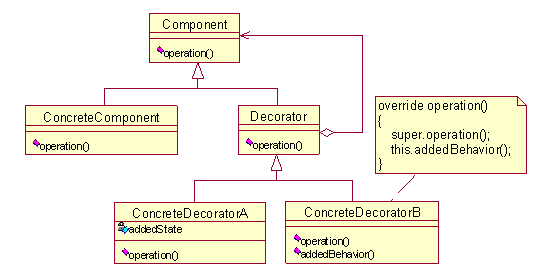
定义具体的对象，Decorator可以给它增加额外的职责；

Decorator（装饰者抽象类）

维护指向Component实例的引用，定义与Component一致的接口；

ConcreteDecorator（具体装饰者）

具体的装饰对象，给内部持有的具体被装饰对象增加具体的职责；



涉及角色

抽象组件：定义一个抽象接口，来规范准备附加功能的类。

具体组件：将要被附加功能的类，实现抽象构件角色接口。

抽象装饰者：持有对具体构件角色的引用并定义与抽象构件角色一致的接口。

具体装饰：实现抽象装饰者角色，负责为具体构件添加额外功能。

代码实现：

Drink.java 被装饰者对象的接口

SoyaBeanMilk.java 具体的被装饰者对象

EggDecorator.java 具体装饰者对象

SugarDecorator.java 具体装饰者对象

BlackBeanDecorator.java 具体装饰者对象

Decorator.java 装饰者基类

Test.java 测试

装饰者模式小结：

OO原则：动态地将责任附加到对象上。想要扩展功能，装饰者提供有别于继承的另一种选择。

 要点：

1、继承属于扩展形式之一，但不见得是达到弹性设计的最佳方案。

2、在我们的设计中，应该允许行为可以被扩展，而不须修改现有的代码。

3、组合和委托可用于在运行时动态地加上新的行为。

4、除了继承，装饰者模式也可以让我们扩展行为。

5、装饰者模式意味着一群装饰者类，这些类用来包装具体组件。

6、装饰者类反映出被装饰的组件类型（实际上，他们具有相同的类型，

都经过接口或继承实现）。

7、装饰者可以在被装饰者的行为前面与/或后面加上自己的行为，

甚至将被装饰者的行为整个取代掉，而达到特定的目的。

8、你可以有无数个装饰者包装一个组件。

9、装饰者一般对组建的客户是透明的，

除非客户程序依赖于组件的具体类型。

15、常见字符编码

在程序中如果没有处理好字符的编码，就有可能出现乱码问题。下面我们一起为大家介绍在开发中常见的编码有

哪些。

在计算机世界里，任何的文字都是以指定的编码方式存在的。

常见编码有：ISO8859-1、GBK/GB2312、unicode、UTF。

iso8859-1：

编码属于单字节编码，最多只能表示0——255的字符范围，主要在英文上应用。

GBK/GB2312：

中文的国际编码，专门用来表示汉字，是双字节编码

unicode：

java中就是使用此编码方式，也是最标准的一种编码，是使用16进制表示的编码。但此编码不兼容iso8859-1编码。

UTF：

由于unicode不支持iso8859-1编码，而且容易占用更多的空间，而且对于英文母也需要使用两个字节编码，这样使

用unicode不便于传输和储存，因此产生了utf编码，utf编码兼容了iso8859-1编码，也可以用来表示所有语言字符，

不过utf是不定长编码，每个字符的长度从1－6个字节不等，一般在中文网页中使用此编码，因为这样可以节省空

间。

造成乱码的根本原因：

1、程序使用的编码与本机的编码不统一

2、在网络中，客户端与服务端编码不统一（WEB开发中出现的乱码情况）

16、New IO

1、为什么要使用 NIO?

NIO是JDK1.4加入的新包，NIO 的创建目的是为了让 Java 程序员可以实现高速 I/O 而无需编写自定义的本机代码。NIO 将最耗时的 I/O 操作（即填充和提取缓冲区）转移回操作系统，因而可以极大地提高速度。

流与块的比较

原来的 I/O 库(在 java.io.\*中) 与 NIO 最重要的区别是数据打包和传输的方式，原来的 I/O 以流的方式处理数据，而 NIO 以块的方式处理数据。

面向流 的 I/O 系统一次一个字节地处理数据。一个输入流产生一个字节的数据，一个输出流消费一个字节的数据。不利的一面是，面向流的 I/O 通常相当慢。

一个 面向块 的 I/O 系统以块的形式处理数据。每一个操作都在一步中产生或者消费一个数据块。按块处理数据比按(流式的)字节处理数据要快得多。但是面向块的 I/O 缺少一些面向流的 I/O 所具有的优雅性和简单性。

缓冲区

在 NIO 库中，所有数据都是用缓冲区处理的。在读取数据时，它是直接读到缓冲区中的。在写入数据时，它是写入到缓冲区中的。任何时候访问 NIO 中的数据，您都是将它放到缓冲区中。

缓冲区实质上是一个数组。通常它是一个字节数组，但是也可以使用其他种类的数组。但是一个缓冲区不 仅仅 是一个数组。缓冲区提供了对数据的结构化访问，而且还可以跟踪系统的读/写进程。

缓冲区类型

最常用的缓冲区类型是 ByteBuffer。一个 ByteBuffer 可以在其底层字节数组上进行 get/set 操作（即字节的获取和设置）。ByteBuffer 不是 NIO 中唯一的缓冲区类型。事实上，对于每一种基本 Java 类型都有一种缓冲区类型：

ByteBuffer

CharBuffer

ShortBuffer

IntBuffer

LongBuffer

FloatBuffer

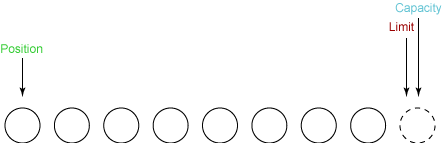
DoubleBuffer

缓冲区内部细节：

状态变量

可以用三个值指定缓冲区在任意时刻的状态：

position limit capacity



通道： Channel

Channel 是一个对象，可以通过它读取和写入数据。拿 NIO 与原来的 I/O 做个比较，通道就像是流。

正如前面提到的，所有数据都通过 Buffer 对象来处理。您永远不会将字节直接写入通道中，相反，您是将数据写入包含一个或者多个字节的缓冲区。同样，您不会直接从通道中读取字节，而是将数据从通道读入缓冲区，再从缓冲区获取这个字节。

使用通道读写文件示例

JDK1.7引入了新的IO操作类， java.nio.file包下，Java NIO Path接口和Files类

Path接口

1、Path表示的是一个目录名序列，其后还可以跟着一个文件名，路径中第一个部件是根部件时就是绝对路径，例如 / 或 C:\ ，而允许访问的根部件取决于文件系统；

2、以根部件开始的路径是绝对路径，否则就是相对路径；

3、静态的Paths.get方法接受一个或多个字符串，字符串之间自动使用默认文件系统的路径分隔符连接起来（Unix是 /，Windows是 \ ），这就解决了跨平台的问题，接着解析连接起来的结果，如果不是合法路径就抛出InvalidPathException异常，否则就返回一个Path对象；

Files工具类

1、读写文件

static path write(Path path, byte[] bytes, OpenOption... options) 写入文件

static byte[] readAllBytes(Path path) 读取文件中的所有字节。

2、复制、剪切、删除

static path copy(Path source, Path target, CopyOption... options)

static path move(Path source, Path target, CopyOption... options)

static void delete(Path path) //如果path不存在文件将抛出异常，此时调用下面的比较好

static boolean deleteIfExists(Path path)

3、创建文件和目录

//创建新目录，除了最后一个部件，其他必须是已存在的

Files.createDirectory(path);

//创建路径中的中间目录，能创建不存在的中间部件

Files.createDirectories(path);

//创建一个空文件，检查文件存在，如果已存在则抛出异常而检查文件存在是原子性的，

//因此在此过程中无法执行文件创建操作

Files.createFile(path);

//添加前/后缀创建临时文件或临时目录

Path newPath = Files.createTempFile(dir, prefix, suffix);

Path newPath = Files.createTempDirectory(dir, prefix);

## 第09章：集合

1、集合框架概述

1、集合框架的作用

在实际开发中，我们经常会对一组相同类型的数据进行统一管理操作。到目前为止，我们可以使用数组结构，链表结构，二叉树结构来实现。

数组的最大问题在于数组中的元素个数是固定的，要实现动态数组，必竟还是比较麻烦，自己实现链表或二叉树结构来管理对象更是不方便。

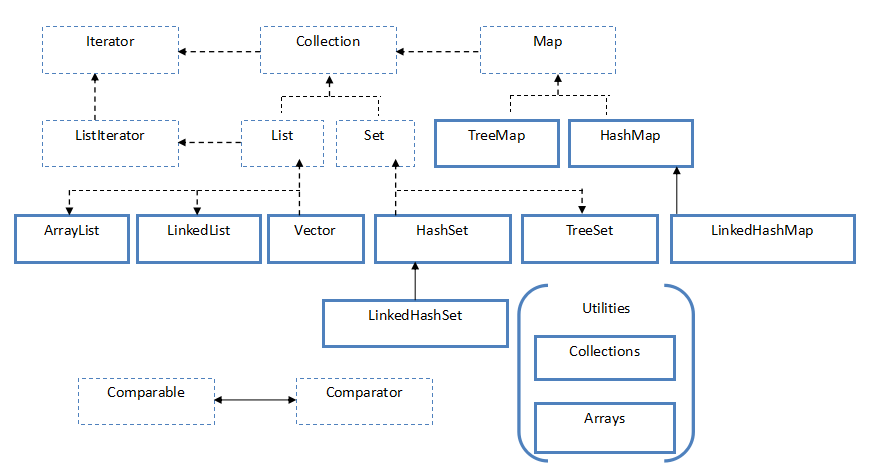
在JDK1.2版本后，JAVA完整的提供了类集合的概念，封装了一组强大的、非常方便的集合框架API，让我们在开发中大大的提高了效率。

集合中分为三大接口：

Collection、Map、Iterator

集合框架的接口和类在java.util包中

1. 集合框架结构图



3、Collection接口

Collection 层次结构 中的根接口。Collection 表示一组对象，这些对象也称为 collection 的元素。一些 collection 允许有重复的元素，而另一些则不允许。一些 collection 是有序的，而另一些则是无序的。JDK 不提供此接口的任何直接 实现：它提供更具体的子接口（如 Set 和 List）实现。此接口通常用来传递 collection，并在需要最大普遍性的地方操作这些 collection。

接口的定义：

public interface Collection<E>

extends Iterable<E>

2、集合框架List接口

1、 List接口

public interface List<E> extends Collection<E>

有序的 collection（也称为序列）。此接口的用户可以对列表中每个元素的插入位置进行精确地控制。用户可以根据元素的整数索引（在列表中的位置）访问元素，并搜索列表中的元素。

2、ArrayList

public class ArrayList<E> extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, Serializable

List 接口的大小可变数组的实现。实现了所有可选列表操作，并允许包括 null 在内的所有元素。除了实现 List 接口外，此类还提供一些方法来操作内部用来存储列表的数组的大小。

3、Vector

public class Vector<E> extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, Serializable

Vector 类可以实现可增长的对象数组。与数组一样，它包含可以使用整数索引进行访问的组件。但是，Vector 的大小可以根据需要增大或缩小，以适应创建 Vector 后进行添加或移除项的操作。

4、LinkedList

public class LinkedList<E> extends AbstractSequentialList<E>

implements List<E>, Deque<E>, Cloneable, Serializable

List 接口的链接列表实现。实现所有可选的列表操作，并且允许所有元素（包括 null）。除了实现 List 接口外，LinkedList 类还为在列表的开头及结尾 get、remove 和 insert 元素提供了统一的命名方法。

3、集合框架Set接口

1、Set接口

public interface Set<E> extends Collection<E>

一个不包含重复元素的 collection。更确切地讲，set 不包含满足 e1.equals(e2) 的元素对 e1 和 e2，并且最多包含一个 null 元素。正如其名称所暗示的，此接口模仿了数学上的 set 抽象。

2、HashSet

public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable

类实现 Set 接口，由哈希表（实际上是一个 HashMap 实例）支持。它不保证 set 的迭代顺序；特别是它不保证该顺序恒久不变。此类允许使用 null 元素。

hashCode深入分析

hashcode() 方法，在object类中定义如下：

public native int hashCode();

hashCode是本地方法，它的实现是根据本地机器相关，当然我们可以在自己写的类中覆盖hashcode()方法，比如String、Integer、Double。。。。等等这些类都是覆盖了hashcode()方法的。

在java的集合中，判断两个对象是否相等的规则是：

（1）判断两个对象的hashCode是否相等

如果不相等，认为两个对象也不相等，结束

如果相等，转入2

（2）判断两个对象用equals运算是否相等

如果不相等，认为两个对象也不相等

如果相等，认为两个对象相等

（equals()是判断两个对象是否相等的关键）

3、TreeSet

public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable

基于 TreeMap 的 NavigableSet 实现。使用元素的自然顺序对元素进行排序，或者根据创建 set 时提供的 Comparator 进行排序，具体取决于使用的构造方法。

4、LinkedHashSet

public class LinkedHashSet<E> extends HashSet<E> implements Set<E>, Cloneable, Serializable

具有可预知迭代顺序的 Set 接口的哈希表和链接列表实现。此实现与 HashSet 的不同之外在于，后者维护着一个运行于所有条目的双重链接列表。此链接列表定义了迭代顺序，即按照将元素插入到 set 中的顺序（插入顺序）进行迭代。注意，插入顺序不 受在 set 中重新插入的 元素的影响。（如果在 s.contains(e) 返回 true 后立即调用 s.add(e)，则元素 e 会被重新插入到 set s 中。）

4、集合框架Iterator接口

1、集合输出

前面我们已经学习了集合的基本操作，很多情况下，我们需要把集合的内容进行输出，也就是遍历集合。

遍历集合的方式有以下几种：

1、 Iterator

2、 ListIterator

3、 Enumeration

4、 foreach

其中Iterator的使用率最高，在JDK1.5后新增的foreach也被大量使用。

2、Iterator

public interface Iterator<E>

对 collection 进行迭代的迭代器。迭代器取代了 Java Collections Framework 中的 Enumeration。

|  |  |
| --- | --- |
| boolean hasNext() | 如果仍有元素可以迭代，则返回 true。 |
| E next() | 返回迭代的下一个元素。 |
| void remove() | 从迭代器指向的 collection 中移除迭代器返回的最后一个元素。 |

3、ListIterator

public interface ListIterator<E>

extends Iterator<E>

系列表迭代器，允许程序员按任一方向遍历列表、迭代期间修改列表，并获得迭代器在列表中的当前位置。

|  |  |
| --- | --- |
| void add(E e) | 增加元素 |
| boolean hasPrevious() | 判断是否有前一个元素 |
| E previous() | 取出前一个元素 |
| void set(E e) | 修改元素的内容 |
| int previousIndex() | 前一个索引位置 |
| int nextIndex() | 下一个索引位置 |

4、Enumeration

public interface Enumeration<E>

实现 Enumeration 接口的对象，它生成一系列元素，一次生成一个。连续调用 nextElement 方法将返回一系列的连续元素。

注：此接口的功能与 Iterator 接口的功能是重复的。此外，Iterator 接口添加了一个可选的移除操作，并使用较短的方法名。新的实现应该优先考虑使用 Iterator 接口而不是 Enumeration 接口。

|  |  |
| --- | --- |
| boolean hasMoreElements() | 判断是否有下一个元素 |
| E nextElement() | 取出当前元素 |

5、foreach

在前面的知识讲解中，我们使用foreach来输出数组的内容，那么也可以输出集合中的内容。在使用foreach输出的时候一定要注意的是，创建集合时要指定操作泛型的类型。

List<Integer> numbers = new ArrayList<>();

JDK1.8新特性：

//no.1

numbers.forEach((Integer integer) -> {System.out.println(integer);});

//no.2

numbers.forEach(integer -> {System.out.println(integer);});

//no.3

numbers.forEach(integer -> System.out.println(integer));

//no.4

numbers.forEach(System.out::println);

//no.5

numbers.forEach(new MyConsumer());

5、JDK1.8新特性之Stream

Consumer<T>接口 消费者接口

Function<T,R> 接口 表示接受一个参数并产生结果的函数。

Supplier<T>接口 代表结果供应商。

Predicate<T>接口 断言接口

什么是Stream？

Stream是元素的集合，这点让Stream看起来用些类似Iterator；

可以支持顺序和并行的对原Stream进行汇聚的操作；

我们可以把Stream当成一个高级版本的Iterator。原始版本的Iterator，用户只能一个一个的遍历元素并对其执行某些操作；高级版本的Stream，用户只要给出需要对其包含的元素执行什么操作，比如“过滤掉长度大于10的字符串”、“获取每个字符串的首字母”等，具体这些操作如何应用到每个元素上，就给Stream就好了！

Stream的常见操作有：

|  |  |  |
| --- | --- | --- |
| <R> Stream<R> | map(Function<? super T,? extends R> mapper) | 返回由给定函数应用于此流的元素的结果组成的流。 |
| <R> Stream<R> | flatMap(Function<? super T,? extends Stream<? extends R>> mapper) | 返回由通过将提供的映射函数应用于每个元素而产生的映射流的内容来替换该流的每个元素的结果的流。 |
| Stream<T> | filter(Predicate<? super T> predicate) | 返回由与此给定谓词匹配的此流的元素组成的流。 |
| void | forEach(Consumer<? super T> action) | 对此流的每个元素执行操作。 |
| Optional<T> | findFirst() | 返回描述此流的第一个元素的Optional如果流为空，则返回一个空的Optional 。 |
| Optional<T> | reduce(BinaryOperator<T> accumulator) | 使用 associative累积函数对此流的元素执行 reduction ，并返回描述减小值的 Optional （如果有）。 |
| Stream<T> | peek(Consumer<? super T> action) | 返回由该流的元素组成的流，另外在从生成的流中消耗元素时对每个元素执行提供的操作。 |
| Stream<T> | sorted() | 返回由此流的元素组成的流，根据自然顺序排序。 |

6、集合框架Map接口

1、Map接口

public interface Map<K,V>

将键映射到值的对象，一个映射不能包含重复的键；每个键最多只能映射到一个值。

|  |  |
| --- | --- |
| void clear() | 清空Map集合中的内容 |
| boolean containsKey(Object key) | 判断集合中是否存在指定的key |
| boolean containsValue(Object value) | 判断集合中是否存在指定的value |
| Set<Map.Entry<K,V>> entrySet() | 将Map接口变为Set集合 |
| V get(Object key) | 根据key找到其对应的value |

|  |  |
| --- | --- |
| boolean isEmpty() | 判断是否为空 |
| Set<K> keySet() | 将全部的key变为Set集合 |
| Collection<V> values() | 将全部的value变为Collection集合 |
| V put(K key,V value) | 向集合中增加内容 |
| void putAll(Map<? extends K,? extends V> m) | 增加一组集合 |
| V remove(Object key) | 根据key删除内容 |

2、HashMap

public class HashMap<K,V> extends AbstractMap<K,V>

implements Map<K,V>, Cloneable, Serializable

基于哈希表的 Map 接口的实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null 键。（除了非同步和允许使用 null 之外，HashMap 类与 Hashtable 大致相同。）此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

3、Hashtable

public class Hashtable<K,V> extends Dictionary<K,V>

implements Map<K,V>, Cloneable, Serializable

此类实现一个哈希表，该哈希表将键映射到相应的值。任何非 null 对象都可以用作键或值。

为了成功地在哈希表中存储和获取对象，用作键的对象必须实现 hashCode 方法和 equals 方法。

4、LinkedHashMap

public class LinkedHashMap<K,V>

extends HashMap<K,V> implements Map<K,V>

Map 接口的哈希表和链接列表实现，具有可预知的迭代顺序。此实现与 HashMap 的不同之处在于，后者维护着一个运行于所有条目的双重链接列表。

5、TreeMap

public class TreeMap<K,V> extends AbstractMap<K,V>

implements NavigableMap<K,V>, Cloneable, Serializable

基于红黑树（Red-Black tree）的 NavigableMap 实现。该映射根据其键的自然顺序进行排序，或者根据创建映射时提供的 Comparator 进行排序，具体取决于使用的构造方法。

java1.8 Map接口的新方法：

在JDK8中Map接口提供了一些新的便利的方法。因为在本文中我所提到的所有Map方法都是以默认值方法的方式实现的，所以现有的Map接口的实现可以直接拥有这些在默认值方法中定义的默认行为，而不需要新增一行代码。

getOrDefault(Object, V)

putIfAbsent(K,V)

remove(Object key, Object value)

replace(K,V)

replace(K,V,V)

compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)

computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)

merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)

7、Collections工具类

Collections类

Collections工具类提供了大量针对Collection/Map的操作，总体可分为四类，都为静态（static）方法：

1. 排序操作（主要针对List接口相关）

reverse(List list)：反转指定List集合中元素的顺序

shuffle(List list)：对List中的元素进行随机排序（洗牌）

sort(List list)：对List里的元素根据自然升序排序

sort(List list, Comparator c)：自定义比较器进行排序

swap(List list, int i, int j)：将指定List集合中i处元素和j出元素进行交换

rotate(List list, int distance)：将所有元素向右移位指定长度，如果distance等于size那么结果不变

2. 查找和替换（主要针对Collection接口相关）

binarySearch(List list, Object key)：使用二分搜索法，以获得指定对象在List中的索引，前提是集合已经排序

max(Collection coll)：返回最大元素

max(Collection coll, Comparator comp)：根据自定义比较器，返回最大元素

min(Collection coll)：返回最小元素

min(Collection coll, Comparator comp)：根据自定义比较器，返回最小元素

fill(List list, Object obj)：使用指定对象填充

frequency(Collection Object o)：返回指定集合中指定对象出现的次数

replaceAll(List list, Object old, Object new)：替换

3. 同步控制

Collections工具类中提供了多个synchronizedXxx方法，该方法返回指定集合对象对应的同步对象，从而解决多线程并发访问集合时线程的安全问题。HashSet、ArrayList、HashMap都是线程不安全的，如果需要考虑同步，则使用这些方法。这些方法主要有：synchronizedSet、synchronizedSortedSet、synchronizedList、synchronizedMap、synchronizedSortedMap。

特别需要指出的是，在使用迭代方法遍历集合时需要手工同步返回的集合。

4. 设置不可变集合

Collections有三类方法可返回一个不可变集合：

emptyXxx()：返回一个空的不可变的集合对象

singletonXxx()：返回一个只包含指定对象的，不可变的集合对象。

unmodifiableXxx()：返回指定集合对象的不可变视图

5. 其它

disjoint(Collection<?> c1, Collection<?> c2) - 如果两个指定 collection 中没有相同的元素，则返回 true。

addAll(Collection<? super T> c, T... a) - 一种方便的方式，将所有指定元素添加到指定 collection 中。

Comparator<T> reverseOrder(Comparator<T> cmp) - 返回一个比较器，它强行反转指定比较器的顺序。如果指定比较器为 null，则此方法等同于 reverseOrder()（换句话说，它返回一个比较器，该比较器将强行反转实现 Comparable 接口那些对象 collection 上的自然顺序）。

Optional容器类（JDK1.8）

这是一个可以为null的容器对象。如果值存在则isPresent()方法会返回true，调用get()方法会返回该对象。

of：

为非null的值创建一个Optional。

ofNullable：

为指定的值创建一个Optional，如果指定的值为null，则返回一个空的Optional。

isPresent：

如果值存在返回true，否则返回false。

get：

如果Optional有值则将其返回，否则抛出NoSuchElementException。

ifPresent：

如果Optional实例有值则为其调用consumer，否则不做处理

orElse：

如果有值则将其返回，否则返回指定的其它值。

orElseGet：

orElseGet与orElse方法类似，区别在于得到的默认值。orElse方法将传入的字符串作为默认值，orElseGet方法可以接受Supplier接口的实现用来生成默认值。

orElseThrow：

如果有值则将其返回，否则抛出supplier接口创建的异常。

map：

如果有值，则对其执行调用mapping函数得到返回值。如果返回值不为null，则创建包含mapping返回值的Optional作为map方法返回值，否则返回空Optional。

flatMap：

如果有值，为其执行mapping函数返回Optional类型返回值，否则返回空Optional。flatMap与map（Funtion）方法类似，区别在于flatMap中的mapper返回值必须是Optional。调用结束时，flatMap不会对结果用Optional封装。

filter：

如果有值并且满足断言条件返回包含该值的Optional，否则返回空Optional。

8、Queue、Deque接口

队列（Queue）是一种特殊的线性表，是一种先进先出（FIFO）的数据结构。它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。

LinkedList是Queue接口的实现类

boolean add(E e) ：将指定的元素插入此队列（如果立即可行且不会违反容量限制），在成功时返回 true，如果当前没有可用的空间，则抛出 IllegalStateException。

E element() ：获取，但是不移除此队列的头。

boolean offer(E e) ：将指定的元素插入此队列（如果立即可行且不会违反容量限制），当使用有容量限制的队列时，此方法通常要优于 add(E)，后者可能无法插入元素，而只是抛出一个异常。

E peek() ：获取但不移除此队列的头；如果此队列为空，则返回 null。

E poll() ：获取并移除此队列的头，如果此队列为空，则返回 null。

E remove() ：获取并移除此队列的头。

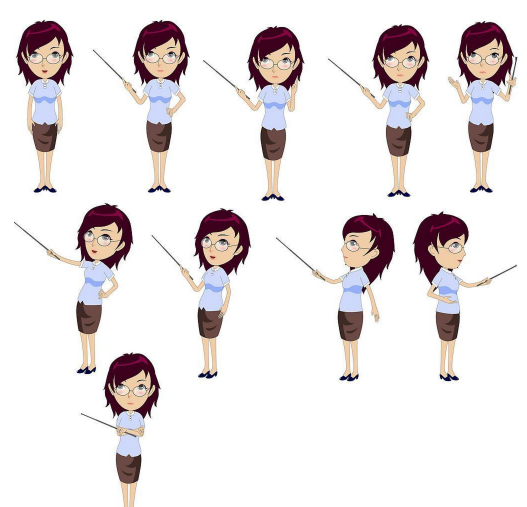
Deque：一个线性 collection，支持在两端插入和移除元素。

此接口既支持有容量限制的双端队列，也支持没有固定大小限制的双端队列。

接口定义在双端队列两端访问元素的方法。提供插入、移除和检查元素的方法。

9、对象一对多与多对多关系

在学习完集合框 架后，我们可以使用集合来表示实际开发中对象的一对多关系和多对多关系。





10、迭代器设计模式

提供一个方法按顺序遍历一个集合内的元素，而又不需要暴露该对象的内部表示。

应用场景

1、访问一个聚合的对象，而不需要暴露对象的内部表示

2、支持对聚合对象的多种遍历

3、对遍历不同的对象，提供统一的接口。

迭代器模式的角色构成

(1)迭代器角色（Iterator）:定义遍历元素所需要的方法，一般来说会有这么三个方法：取得下一个元素的方法next()，判断是否遍历结束的方法hasNext()），移出当前对象的方法remove(),

(2)具体迭代器角色（Concrete Iterator）：实现迭代器接口中定义的方法，完成集合的迭代。

(3)容器角色(Aggregate): 一般是一个接口，提供一个iterator()方法，例如java中的Collection接口，List接口，Set接口等

(4)具体容器角色（ConcreteAggregate）：就是抽象容器的具体实现类，比如List接口的有序列表实现ArrayList，List接口的链表实现LinkedList，Set接口的哈希列表的实现HashSet等。

11、guava对集合的支持

版本下载：https://repo1.maven.org/maven2/com/google/guava/guava/

Guava工程包含了若干被Google的 Java项目广泛依赖 的核心库，例如：集合 [collections] 、缓存 [caching] 、原生类型支持 [primitives support] 、并发库 [concurrency libraries] 、通用注解 [common annotations] 、字符串处理 [string processing] 、I/O 等等。 所有这些工具每天都在被Google的工程师应用在产品服务中。

Guava对JDK集合的扩展，这是Guava最成熟和为人所知的部分。

1、 不可变集合：用不变的集合进行防御性编程和性能提升。

2 、新集合类型： multisets, multimaps, tables,等

3、 强大的集合工具类：提供java.util.Collections中没有的集合工具

4 、扩展工具类：让实现和扩展集合类变得更容易，比如创建Collection的装饰器，或实现迭代器

1、只读设置

2、函数式编程:过滤器

3、函数式编程:转换

4、组合式函数编程

5、加入约束：非空、长度验证

6、集合操作：交集、差集、并集

7、Multiset：无序可重复

8、Multimap key可以重复

9、BiMap：双向Map(bidirectional Map) 键与值不能重复

10、双键的Map -->Table --->rowKey+columnKye+value

## 第10章：多线程与并发

1、进程与线程

1、什么是进程

程序是指令和数据的有序集合，其本身没有任何运行的含义，是一个静态的概念。而进程是程序在处理机上的一次执行过程，它是一个动态的概念。

进程是一个具有一定独立功能的程序，一个实体，每一个进程都有它自己的地址空间。

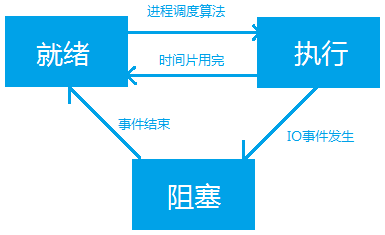
2、进程的状态

进程执行时的间断性，决定了进程可能具有多种状态。事实上，运行中的进程具有以下三种基本状态。

1）就绪状态（Ready）

2）运行状态（Running）

3）阻塞状态（Blocked）



3、线程

线程实际上是在进程基础之上的进一步划分，一个进程启动之后，里面的若干程序又可以划分成若干个线程。

线程：是进程中的一个执行路径，共享一个内存空间，线程之间可以自由切换，并发执行，一个进程最少有一个线程（单线程程序）

一个程序可以同时执行多个任务，来提高效率。

例如：

（1）、同时下载多个电影

（2）、同时与多人聊天

并行：就是两个任务同时运行(多个CPU)

并发：是指两个任务同时请求运行，而处理器一次只能接受一个任务，就会把两个任务安排轮流执行，由于CPU时间片运行时间较短,就会感觉两个任务在同时执行

2、线程的基本使用

线程实现的两种方式

在Java中如果要想实现多线程的操作，有两种实现方法：

（1）一种是继承Thread类

class MyThread extends Thread{

public void run(){

//逻辑处理

}

}

MyThread mt = new MyThread();

mt.start();

（2）另外一种是实现Runnable接口

class MyRunnable implements Runnable{

public void run(){

//逻辑处理

}

}

MyRunnable mr = new MyRunnable();

Thread t = new Thread(mr);

t.start();

3、线程休眠

public static void sleep(long millis)

throws InterruptedException使当前正在执行的线程以指定的毫秒数暂停（暂时停止执行），释放CPU的时间片，具体取决于系统定时器和调度程序的精度和准确性。 线程不会丢失任何显示器的所有权。

参数

millis - 以毫秒为单位的睡眠时间长度

异常

IllegalArgumentException - 如果 millis值为负数

InterruptedException - 如果任何线程中断当前线程。 当抛出此异常时，当前线程的中断状态将被清除。

public static void sleep(long millis,int nanos)throws InterruptedException 毫秒，纳秒

static Thread currentThread() 返回对当前正在执行的线程对象的引用。

4、join与中断线程

public final void join() throws InterruptedException

等待这个线程死亡。

调用此方法的行为方式与调用完全相同

join (0)

异常 InterruptedException - 如果任何线程中断当前线程。 当抛出此异常时，当前线程的中断状态将被清除。

public void interrupt()

中断这个线程。

除非当前线程中断自身，这是始终允许的

public static boolean interrupted()

测试当前线程是否中断。 该方法可以清除线程的中断状态 。 换句话说，如果这个方法被连续调用两次，那么第二个调用将返回false（除非当前线程再次中断，在第一个调用已经清除其中断状态之后，在第二个调用之前已经检查过）。

忽略线程中断，因为线程在中断时不存在将被该方法返回false所反映。

自定义标记中断线程

5、守护线程与yield

public final void setDaemon(boolean on)

将此线程标记为daemon线程或用户线程。 当运行的唯一线程都是守护进程线程时，Java虚拟机将退出。

public final boolean isDaemon()

测试这个线程是否是守护线程。

public static void yield()

暂停当前正在执行的线程对象，并执行其他线程。（了解）

6、其它方法与优先级

long getId()   
          返回该线程的标识符。

String getName()   
          返回该线程的名称。

void setName(String name)   
          改变线程名称，使之与参数 name 相同。

boolean isAlive()   
          测试线程是否处于活动状态。

void setPriority(int newPriority)   
          更改线程的优先级。

static int MAX\_PRIORITY   
          线程可以具有的最高优先级。

static int MIN\_PRIORITY   
          线程可以具有的最低优先级。

static int NORM\_PRIORITY   
          分配给线程的默认优先级。

7、线程同步

1、多线程共享数据

在多线程的操作中，多个线程有可能同时处理同一个资源，这就是多线程中的共享数据。

2、线程同步

解决数据共享问题，必须使用同步，所谓同步就是指 多个线程在同一个时间段内只能有一个线程执行指定代码，其他线程要等待此线程完成之后才可以继续执行。

线程进行同步，有以下三种方法：

（1）同步代码块

synchronized(要同步的对象){

要同步的操作 ;

}

（2）同步方法

public synchronized void method(){

要同步的操作 ;

}

1. Lock（ReentrantLock）

3、同步准则

当编写 synchronized 块时，有几个简单的准则可以遵循，这些准则在避免死锁和性能危险的风险方面大有帮助：

（1）使代码块保持简短。把不随线程变化的预处理和后处理移出synchronized 块。

（2）不要阻塞。如InputStream.read()。

（3）在持有锁的时候，不要对其它对象调用方法。

8、死锁

过多的同步有可能出现死锁，死锁的操作一般是在程序运行的时候才有可能出现。

多线程中要进行资源的共享，就需要同步，但同步过多，就可能造成死锁。

9、生产者与消费者应用案例

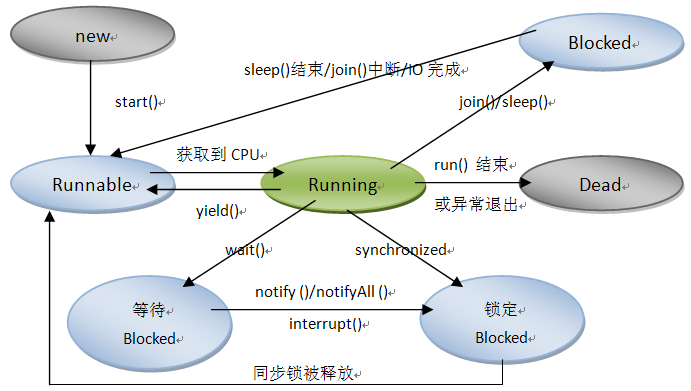
多线程的开发中有一个最经典的操作案例，就是生产者-消费者，生产者不断生产产品，消费者不断取走产品。

例如：饭店里的有一个厨师和一个服务员，这个服务员必须等待厨师准备好膳食。当厨师准备好时，他会通知服务员，之后服务员上菜，然后返回继续等待。这是一个任务协作的示例，厨师代表生产者，而服务员代表消费者。



10、线程生命周期

线程生命周期图：



11、线程池

线程池是预先创建线程的一种技术。线程池在还没有任务到来之前，创建一定数量的线程，放入空闲队列中，然后对这些资源进行复用。减少频繁的创建和销毁对象。

jdk1.5版本以上提供了现成的线程池。

Java里面线程池的顶级接口是Executor，是一个执行线程的工具。

线程池接口是ExecutorService。

java.util.concurrent 包：并发编程中很常用的实用工具类

Executor 接口：

执行已提交的 Runnable 任务的对象。

ExecutorService 接口：

Executor 提供了管理终止的方法，以及可为跟踪一个或多个异步任务执行状况而生成 Future 的方法。

Executors 类：

此包中所定义的Executor、ExecutorService等的工厂和实用方法。

在Executors类里面提供了一些静态工厂，生成一些常用的线程池。

newSingleThreadExecutor：

创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。

线程池的大小一旦达到最大值就会保持不变，

如果某个线程因为执行异常而结束，

那么线程池会补充一个新线程。

newCachedThreadPool：

创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。

newScheduledThreadPool：

创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。

## 第11章：网络编程

1、网络编程基本概念

1、什么是计算机网络

把分布在不同地理区域的计算机与专门的外部设备用通信线路互连成一个规模大、功能强的网络系统，从而使众多的计算机可以方便地互相传递信息，共享硬件、软件、数据信息等资源。

2、计算机网络的主要功能

资源共享

信息传输与集中处理

均衡负荷与分布处理

综合信息服务(www/综合业务数字网络ISDN)等

3、网络通信协议

要使计算机连成的网络能够互通信息，需要对数据传输速率、传输代码、代码结构、传输控制步骤、出错控制等制定一组标准，这一组共同遵守的通信标准就是网络通信协议，不同的计算机之间必须使用相同的通讯协议才能进行通信。

网络通信接口

为了使两个结点之间能进行对话，必须在它们之间建立通信工具(即接口)，使彼此之间能进行信息交换。接口包括两部分：

（1）硬件装置：实现结点之间的信息传送

（2）软件装置：规定双方进行通信的约定协议

4、TCP/IP

TCP/IP：传输控制协议/因特网互联协议，又叫网络通讯协议，这个协议是Internet最基本的协议、Internet国际互联网络的基础，简单地说，就是由网络层的IP协议和传输层的TCP协议组成的。

IP地址：网络中每台计算机的一个标识号，本地IP：127.0.0.1 localhost

端口号(PORT)：端口号的范围：0~65535之间，0~1023之间的端口数是用于一些知名的网络服务和应用



5、程序开发结构

网络编程主要是指完成C/S程序的开发，程序的开发结构有两种：

· C/S（客户端/服务器）

开发两套程序，两套程序需要同时维护，例如：QQ。CS程序一般比较稳定

· B/S（浏览器/服务器）

开发一套程序，客户端使用浏览器进行访问，例如：各个论坛。BS程序一般稳定性较差，而且安全性较差。

但是，C/S的程序开发在实际的Java应用中毕竟很少了，而且整个java基本上都是以B/S为主。

C/S程序主要可以完成以下两种程序的开发：

· TCP：（Transmission Control Protocol）传输控制协议，采用三方握手的方式，保证准确的连接操作。

· UDP：（User Datagram Protocol）数据报协议，发送数据报，例如：手机短信或者是QQ消息。

 TCP、UDP的数据帧格式简单图例：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **协议类型** | **源IP** | **目标IP** | **源端口** | **目标端口** | **帧序号** | **帧数据** |

其中协议类型用于区分TCP、UDP

2、网络编程TCP协议

1、TCP程序概述

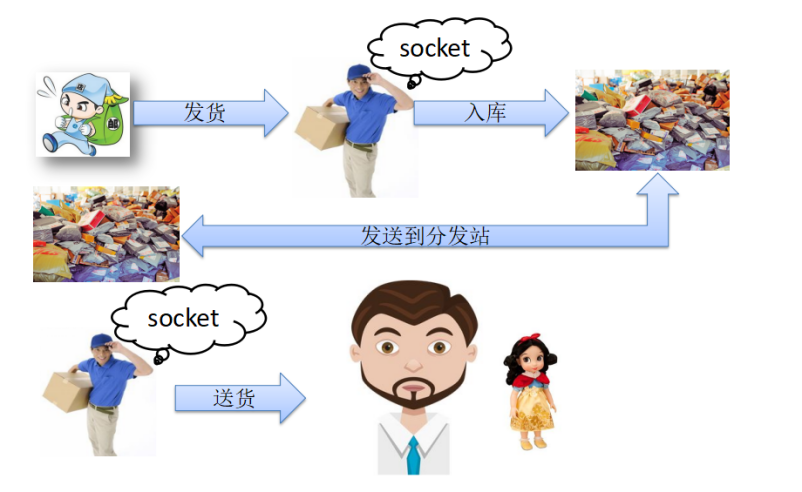
TCP是一个可靠的协议，面向连接的协议。

实现TCP程序，需要编写服务器端和客户端，Java API为我们提供了java.net 包，为实现网络应用程序提供类。

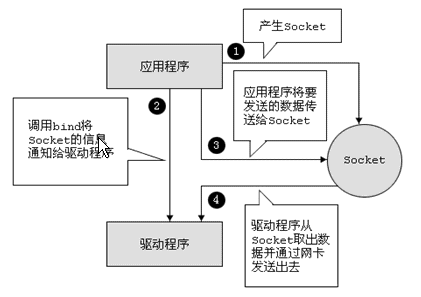
ServerSocket ：此类实现服务器套接字。

Socket ：此类实现客户端套接字（也可以就叫“套接字”）。

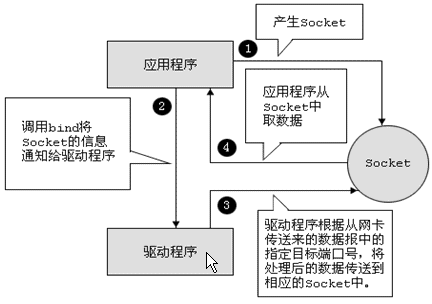
Socket是网络驱动层提供给应用程序编程的接口和一种机制。



1. 数据发送过程



1. 数据接收过程



4、实现服务器端与客户端程序

服务器端：

public class ServerSocket extends Object

此类实现服务器套接字。服务器套接字等待请求通过网络传入。它基于该请求执行某些操作，然后可能向请求者返回结果。

ServerSocket(int port)   
          创建绑定到特定端口的服务器套接字。

 void setSoTimeout(int timeout)   
          通过指定超时值启用/禁用 SO\_TIMEOUT，

以毫秒为单位。

 InetAddress getInetAddress()   
          返回此服务器套接字的本地地址。

 Socket accept()   
          侦听并接受到此套接字的连接。

客户端：

public class Socket extends Object

此类实现客户端套接字（也可以就叫“套接字”）。套接字是两台机器间通信的端点。

Socket(String host, int port)   
          创建一个流套接字并将其连接到指定主机上的指定端口号。

InputStream getInputStream()   
          返回此套接字的输入流。

OutputStream getOutputStream()   
          返回此套接字的输出流。

 void setSoTimeout(int timeout)   
          启用/禁用带有指定超时值的

SO\_TIMEOUT，以毫秒为单位。

3、TCP实现ECHO程序

 Echo，意为应答，程序的功能是客户端向服务器发送一个字符串，服务器不做任何处理，直接把字符串返回给客户端，Echo程序是最为基本的客户/服务器程序。

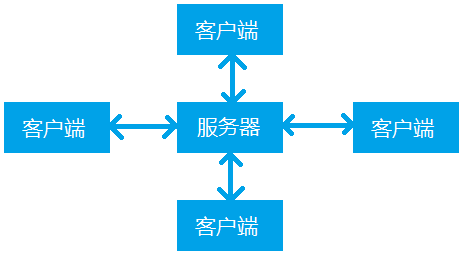
4、服务器与多客户端通信

目前为止我们编写的程序中，服务器只能同时处理一个客户端连接，要想服务器同时支持多个客户端的连接，就必须加入多线程的处理机制，将每一个连接的客户端都创建一个新的线程对象。

5、多客户端之间的通信

服务器可以与多个客户端实现通信了，那我们真正的目的是要实现多个客户端之间的通信，使用TCP协议实现的方案是：

客户端的数据包通过服务器中转，发送到另一个客户端，如下图所示：



6、网络编程UDP协议

1、UDP协议概述

UDP是User Datagram Protocol的简称，是一种无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的，每个被传输的数据报必须限定在64KB之内。

主要使用以下的两个类：

DatagramPacket：此类表示数据报包。

DatagramSocket：此类表示用来发送和接收数据报包的套接字

2、UDP服务器与客户端程序

服务器端：

String info = “….”

// 将信息封装成数据报

byte[] bytes = info.getBytes();

DatagramPacket dp = new DatagramPacket(bytes, 0, bytes

.length(), InetAddress.getByName("localhost"), 5000);// 客户端在5000端口监听

DatagramSocket server = new DatagramSocket(3000);// 服务器的端口

server.send(dp);// 发送数据报

server.close();

客户端：

byte b[] = new byte[1024];// 接收内容

DatagramPacket dp = new DatagramPacket(b, b.length);// 接收内容

// 客户端在5000端口等待

DatagramSocket client = new DatagramSocket(5000);

client.receive(dp); // 接收内容

System.out.println(new String(dp.getData(), 0, dp.getLength()));

client.close();

7、URL

1、URL概述

URL(uniform resource location )类 URL 代表一个统一资源定位符，它是指向互联网“资源”的指针。

抽象类 URLConnection 是所有类的超类，它代表应用程序和 URL 之间的通信链接。

8、MINA框架

1、什么是MINA？ 一句话就是：一个简洁易用的基于 TCP/IP 通信的 JAVA框架。

2、下载地址：http://mina.apache.org/downloads-mina.html

3、一个简单的网络程序需要的最少jar包：mina-core-2.0.16.jar、slf4j-api-1.7.21.jar

4、开发一个 Mina 应用，简单的说，就是创建连接，设定过滤规则，编写自己的消息处理器

5、示例：

//创建一个非阻塞的Server端Socket,用NIO

SocketAcceptor acceptor = new NioSocketAcceptor(); //创建接收数据的过滤器

DefaultIoFilterChainBuilder chain = acceptor.getFilterChain();

//设定这个过滤器将一行一行(/r/n)的读取数据

chain.addLast("myChin", new ProtocolCodecFilter(new TextLineCodecFactory()));

//设定服务器端的消息处理器:一个SampleMinaServerHandler对象

acceptor.setHandler(new SampleMinaServerHandler());

int bindPort = 9999;

//绑定端口,启动服务器

try {acceptor.bind(new InetSocketAddress(bindPort)); } catch (IOException e) {e.printStackTrace();}

System.out.println("Mina Server is Listing on:= " + bindPort);

public class SampleMinaServerHandler extends IoHandlerAdapter{

public void sessionOpened(IoSession session) throws Exception {

super.sessionOpened(session);

System.out.println("incomming client : "+session.getRemoteAddress());

}

public void sessionClosed(IoSession session) throws Exception {

super.sessionClosed(session);

System.out.println("one Clinet Disconnect !");

}

public void messageReceived(IoSession session, Object message) throws Exception {

//我们己设定了服务器解析消息的规则是一行一行读取,这里就可转为String:

String s=(String)message;

System.out.println("收到客户机发来的消息: "+s);

session.write("echo:"+s);

}

}

1、使用telnet测试：telnet localhost 9999

2、编写客户端：

NioSocketConnector connector = new NioSocketConnector(); // 创建接收数据的过滤器

DefaultIoFilterChainBuilder chain = connector.getFilterChain();

//设定这个过滤器将一行一行(/r/n)的读取数据

chain.addLast("myChin", new ProtocolCodecFilter(new TextLineCodecFactory()));

//设定服务器端的消息处理器:一个 SamplMinaServerHandler 对象

connector.setHandler(new SampleMinaClientHandler());

connector.setConnectTimeout(30);// Set connect timeout.

//连接到服务器:

ConnectFuture cf = connector.connect(new InetSocketAddress("localhost", 9999));

// Wait for the connection attempt to be finished.

cf.awaitUninterruptibly();

//发送消息

//cf.getSession().getCloseFuture().awaitUninterruptibly();

//connector.dispose();

public class SampleMinaClientHandler extends IoHandlerAdapter {

//当一个客端端连结进入时

public void sessionOpened(IoSession session) throws Exception {

System.out.println("incomming client :"+session.getRemoteAddress()); session.write("我来啦........");

}

//当一个客户端关闭时

public void sessionClosed(IoSession session) {

System.out.println("one Clinet Disconnect !"); }

//当客户端发送的消息到达时:

public void messageReceived(IoSession session, Object message)throws Exception {

//我们己设定了服务器解析消息的规则是一行一行读取,这里就可转为 String:

String s=(String)message;

System.out.println("服务器发来的收到消息: "+s);

//测试将消息回送给客户端

session.write(s);

} }

使用 Mina 直接传送对象

1、public class Userinfo implements java.io.Serializable

2、 服务器，客户端都设定以对象为单位

//设定这个过滤器将以对象为单位读取数据

ProtocolCodecFilter filter= new ProtocolCodecFilter(new ObjectSerializationCodecFactory());

chain.addLast("objectFilter",filter);

3、接收对象

public void messageReceived(IoSession session, Object message) throws Exception {

//我们己设定了服务器解析消息的规则一个Userinfo对象为单位传输:

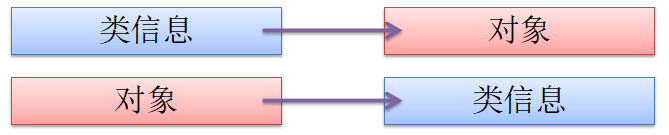
Userinfo us=(Userinfo)message;

}

## 第12章：反射与内省

1、什么是反射

反射： Reflection



2、Class类

Class类是一切的反射根源。

Class类表示什么？

很多的人——可以定义一个Person类（有年龄，性别，姓名等）

很多的车——可以定义一个Car类（有发动机，颜色，车轮等）

很多的类——Class类（类名，构造方法，属性，方法）

得到Class类的对象有三种方式：

第一种形式：Object类中的getClass()方法

第二种形式：类.class

第三种形式：通过Class类的forName方法

使用Class类进行对象的实例化操作

调用无参构造进行实例化

public T newInstance() throws InstantiationException,IllegalAccessException

调用有参构造进行实例化

public Constructor<?>[] getConstructors() throws SecurityException

3、通过Class类取得类信息

取得类所在的包

public Package getPackage() //得到一个类所在的包

public String getName() //得到名字

取得一个类中的全部方法

public Method[] getMethods()

public int getModifiers() //Modifier.toString(mod); // 还原修饰符

public Class<?> getReturnType()

public Class<?>[] getParameterTypes()

public Class<?>[] getExceptionTypes()

public static String toString(int mod)

取得一个类中的全部属性

public Field[] getFields()

public Field[] getDeclaredFields()

public Class<?> getType()

public int getModifiers()

public String getName()

4、通过Class类调用属性或方法

调用类中的方法

调用类中的方法，传入实例化对象，以及具体的参数内容

public Object invoke(Object obj,Object... args)

直接调用属性

取得属性

public Object get(Object obj)

//设置属性，等同于使用“=”完成操作

public void set(Object obj,Object value)

//让属性对外部可见

 public void setAccessible(boolean flag)

5、动态代理

所谓动态代理，即通过代理类：Proxy的代理，接口和实现类之间可以不直接发生联系，而可以在运行期（Runtime）实现动态关联。

java动态代理主要是使用java.lang.reflect包中的两个类。

InvocationHandler类

public Object invoke(Object obj,Method method,Object[] obs)

其中第一个参数 obj 指的是代理类，method是被代理的方法，obs是指被代理的方法的参数组。此方法由代理类来实现。

Proxy类

protected Proxy(InvocationHandler h);

static Class getProxyClass(ClassLoader loader,Class[] interfaces);

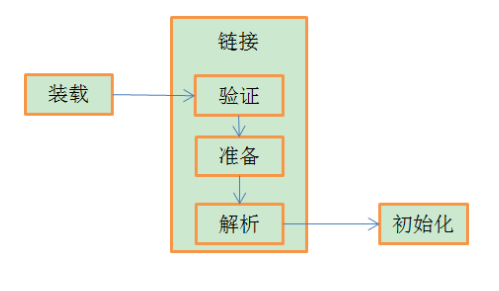
static Object newProxyInstance(ClassLoader loader,Class[]interfaces,InvocationHandler h);

动态代理其实是在运行时生成class，所以，我们必须提供一组interface，然后告诉他class已经实现了这些interface，而且在生成Proxy的时候，必须给他提供一个handler，让他来接管实际的工作。

类加载器原理分析与实现

1、类的加载过程

JVM将类加载过程分为三个步骤：装载（Load），链接（Link）和初始化(Initialize)链接又分为三个步骤，如下图所示：



1) 装载：查找并加载类的二进制数据；

2)链接：

验证：确保被加载类的正确性；

准备：为类的静态变量分配内存，并将其初始化为默认值；

解析：把类中的符号引用转换为直接引用；

3)初始化：为类的静态变量赋予正确的初始值；

2、类的初始化，类什么时候才被初始化：

1）创建类的实例，也就是new一个对象

2）访问某个类或接口的静态变量，或者对该静态变量赋值

3）调用类的静态方法

4）反射（Class.forName("com.vince.Dog")）

5）初始化一个类的子类（会首先初始化子类的父类）

6）JVM启动时标明的启动类，即文件名和类名相同的那个类

3、类的加载：

指的是将类的.class文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后在堆区创建一个这个类的Java.lang.Class对象，用来封装类在方法区类的对象。

6、JavaBean的概念

什么是 JavaBean？

Bean理解为组件意思， JavaBean就是Java组件，在广泛的理解就是一个类，对于组件来说，关键在于要具有“能够被IDE构建工具侦测其属性和事件”的能力，通常在Java中。

一个JavaBean要具备这样的命名规则：

1、对于一个名称为xxx的属性，通常你要写两个方法：getXxx()和setXxx()。任何浏览这些方法的工具，都会把get或set后面的第一个字母自动转换为小写。

2、对于布尔型属性，可以使用以上get和set的方式，不过也可以把get替换成is。

3、Bean的普通方法不必遵循以上的命名规则，不过它们必须是public的。

4、对于事件，要使用Swing中处理监听器的方式。如addWindowListener，removeWindowListener

BeanUtils工具类：http://apache.org/

7、内省基本概念

内省(Introspector)是Java 语言对 Bean 类属性、事件的一种缺省处理方法。例如类 A 中有属性 name, 那我们可以通过 getName,setName 来得到其值或者设置新的值。

通过 getName/setName 来访问 name 属性，这就是默认的规则。

Java 中提供了一套 API 用来访问某个属性的 getter/setter 方法，通过这些 API 可以使你不需要了解这个规则，这些 API 存放于包 java.beans 中，一般的做法是通过类 Introspector 的 getBeanInfo方法 来获取某个对象的 BeanInfo 信息，然后通过 BeanInfo 来获取属性的描述器(PropertyDescriptor)，通过这个属性描述器就可以获取某个属性对应的 getter/setter 方法，然后我们就可以通过反射机制来调用这些方法。

8、 Introspector 相关API

1、Introspector类：

Introspector 类为通过工具学习有关受目标 Java Bean 支持的属性、事件和方法的知识提供了一个标准方法。

static BeanInfo getBeanInfo(Class<?> beanClass)   
          在 Java Bean 上进行内省，了解其所有属性、公开的方法和事件。

2、BeanInfo类：

该类实现此 BeanInfo 接口并提供有关其 bean 的方法、属性、事件等显式信息。

MethodDescriptor[] getMethodDescriptors()   
          获得 beans MethodDescriptor。

PropertyDescriptor[] getPropertyDescriptors()   
          获得 beans PropertyDescriptor。

Properties 属性文件工具类的使用

3、PropertyDescriptor 类：

PropertyDescriptor 描述 Java Bean 通过一对存储器方法导出的一个属性。

 Method getReadMethod()   
          获得应该用于读取属性值的方法。

Method getWriteMethod()   
          获得应该用于写入属性值的方法。

4、MethodDescriptor 类：

MethodDescriptor 描述了一种特殊方法，

即 Java Bean 支持从其他组件对其进行外部访问。

 Method getMethod()   
          获得此 MethodDescriptor 封装的方法。

初探：理解可配置的AOP框架

补充知识：

1、AOP的概念：Aspect Oriented Programming（面向切面编程）

2、可配置 AOP框架实现

AOP使用场景

AOP用来封装横切关注点，具体可以在下面的场景中使用:

权限

缓存

错误处理

调试

记录跟踪

持久化

同步

事务

等等。

9、单例模式优化

1、使用同步保正线程安全 synchronized

2、使用volatile关键字

volatile提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据。如果没有volatile关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致的现象。

3、防止反射调用私有构造方法

4、让单例类序例化安全

## 第13章：泛型

1、为什么需要泛型

List list = new ArrayList();

list.add(“vince”);

list.add(10);

list.add(new Person());

list.add(obj);

List中可以添加任意类型，因为参数是Object，这样一个我们在遍历这个集合时：

for(int i=0;i<list.size();i++){

//此处需要判断元素的类型，才能执行不同的操作

}

以上操作主要存在两个问题：

1.当我们将一个对象放入集合中，集合不会记住此对象的类型，当再次从集合中取出此对象时，改对象的编译类型变成了Object类型，但其运行时类型依然为其本身类型。

2.因此，//取出集合元素时需要人为的强制类型转化到具体的目标类型，且很容易出现“java.lang.ClassCastException”异常。

那么有没有什么办法可以使集合能够记住集合内元素各类型，且能够达到只要编译时不出现问题，运行时就不会出现“java.lang.ClassCastException”异常呢？答案就是使用泛型。

2、什么是泛型

JDK 1.5之后出现了新的技术 —— 泛型（Generic），此技术的最大特点是类中的属性的类型可以由外部决定。泛型，即“参数化类型”。一提到参数，最熟悉的就是定义方法时有形参，然后调用此方法时传递实参。那么参数化类型怎么理解呢？顾名思义，就是将类型由原来的具体的类型参数化，类似于方法中的变量参数，此时类型也定义成参数形式（可以称之为类型形参），然后在使用/调用时传入具体的类型（类型实参）。

List<String> list = new ArrayList<String>();//... = new ArrayList<>(); //... = new ArrayList();

list.add("vince");

list.add("bin");

//list.add(100); // 提示编译错误

for (int i = 0; i < list.size(); i++) {

String name = list.get(i); // 2

System.out.println("name:" + name);

}

3、自定义泛型接口、泛型类

泛型类和方法定义：

class Node<T>{

private T data;

public Node(){}

public Node(T data){

this.data = data;

}

public T getData() {

return data;

}

public void setData(T data) {

this.data = data;

}

}

interface Shopping<T>{public void shopping(T name);}

在泛型接口、泛型类和泛型方法的定义过程中，我们常见的如T、E、K、V等形式的参数常用于表示泛型形参，由于接收来自外部使用时候传入的类型实参，从编码的角度也称为参数化类型了。

泛型只是作用于代码编译阶段，在编译过程中，对于正确检验泛型结果后，会将泛型的相关信息擦除，也就是说，成功编译过后的class文件中是不包含任何泛型信息的。泛型信息不会进入到运行时阶段。

4、通配符

问题：

Node<Number> c1 = new Node<Number>();

Node<Integer> c2 = new Node<Integer>() ;

c1 = c2; // 此时无法转换

public static void getData(Node<Number> data) {

System.out.println("data :" + data.getData());

}

此时，我们可以使用通配符来解决

“?”表示的是可以接收任意的泛型类型，但是只是接收输出，并不能修改。

public static void getData(Node<?> data) {

System.out.println("data :" + data.getData());

}

提示：

使用通配符可以引用其他各种参数化类型，通配符定义的变量主要用作引用，可以调用与参数无关的方法，不能调用与参数有关的方法。

泛型上限就指一个的操作泛型最大的操作父类，例如，现在最大的上限设置成“Number”类型，那么此时，所能够接收到的类型只能是Number及其子类（Integer）。

泛型的上限通过以下的语法完成：

? extends 类

public static void getUpperNumberData(Node<? extends Number> data){ //只能是Number类及其子类

System.out.println("data :" + data.getData());

}

泛型的下限指的是只能设置其具体的类或者父类。设置的语法如下：

? super 类

5、泛型方法

泛型除了在类中定义之外，还可以在方法上定义，而且在方法上使用泛型，此方法所在的类不一定是泛型的操作类。

定义一个方法，实现任意类型数组中两个位置值的调换

public static <T> T[] func(T[] array,int i,int t){

T temp = array[i];

array[i] = array[t];

array[t] = temp;

return array;

}

6、泛型的嵌套使用

在使用集合Map的时候，我们可以这样遍历：

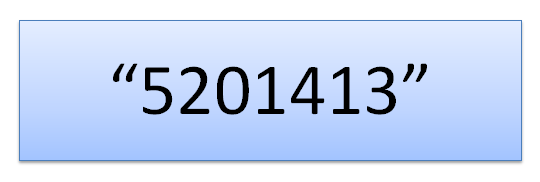
Set<Entry<Integer, String>> entrys = map.entrySet();

## 第14章：正则表达式

1、认识正则

通过一个程序来简单了解一下正则有那些用处：

问题：判断一个字符串是否由数字组成



2、正则表达式

正则表达式（Regular Expression）

正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。

则表达式通常被用来检索、替换那些符合某个模式的文本

java.util.regex包中提供以下两个类对正则表达式的支持：

Matcher 类

通过解释 Pattern 对 character sequence 执行匹配操作的引擎。

Pattern 类

正则表达式的编译表示形式。

3、Pattern类

public final class Pattern extends Objectimplements Serializable

正则表达式的编译表示形式。指定为字符串的正则表达式必须首先被编译为此类的实例。

典型的调用顺序是

Pattern p = Pattern.compile("a\*b");

Matcher m = p.matcher("aaaaab");

boolean b = m.matches();

4、Matcher类

public final class Matcher extends Object implements MatchResult

Matcher类的主要功能是用于进行正则的匹配，通过Pattern类中定义完的正则，再使用Matcher类进行验证或者替换。

常用方法：

boolean matches()   
          尝试将整个区域与模式匹配。

 String replaceAll(String replacement)   
          替换模式与给定替换字符串相匹配的

输入序列的每个子序列。

 String replaceFirst(String replacement)   
          替换模式与给定替换字符串匹配的

输入序列的第一个子序列。

5、String类对正则的支持

在JDK 1.4之后加入了正则，随后又更新了String的操作类，因为在使用正则中，所有的内容通过字符串表示的比较多。在String类中有以下的方法可以完成对正则的支持：

boolean matches(String regex)   
         告知此字符串是否匹配给定的正则表达式。

String replaceAll(String regex, String replacement)   
         使用给定的 replacement 替换此字符串

所有匹配给定的正则表达式的子字符串。

String replaceFirst(String regex, String replacement)   
         使用给定的 replacement 替换此字符串

匹配给定的正则表达式的

第一个子字符串。

String[] split(String regex)   
         根据给定正则表达式的

匹配拆分此字符串。

6、示例

1、验证电话号码（如：010-38389438 ）

2、验证手机号码

3、验证用户名，只能是字母开头的数字、字母或下划线的组合

4、验证IP地址（如：192.168.1.1）

5、验证网址（如：http://www.baidu.com）

6、验证年龄（100以内）

7、验证金额（可以有小数位）

## 第15章：枚举

1、认识枚举

枚举就是要让某个类型的变量的取值只能为若干个固定值中的一个，否则编译器就会报错，枚举可以让编译器在编译时就可以控制源程序赋给的非法值，使用普通变量的方式在开发阶段无法实现这一目标。

在JDK 1.5之后，使用关键字enum定义一种新的类型，称为枚举类型。

2、Enum类与enum关键字

使用enum关键字定义的枚举类，实际上就相当于定义了一个类，此类继承了Enum类而已。

Enum类中定义了如下的方法：

|  |  |
| --- | --- |
| protected Enum(String name,int ordinal) | 此构造方法不能被外部直接调用，只能被其子类访问，此构造方法为自动调用。 |
| public final String name() | 枚举的名字 |
| public final int ordinal() | 枚举的序号 |

3、集合对枚举的支持

在JDK 1.5之后，对于Set和Map接口而言又增加了两个新的子类：EnumSet、EnumMap两个类。

4、带构造方法的枚举

public enum Color {

RED(10),BLUE(10);

private Color() {}

private int color;

private Color(int color){

this.color = color;

}

}

5、让枚举实现接口

interface Info {

public String getColor();

}

public enum Color implements Info {

RED {

public String getColor() {return "红色";}

},

GREEN {

public String getColor() {return "绿色";}

},

BLUE {

public String getColor() {return "蓝色";}

};

}

6、在枚举中定义抽象方法

public enum Color{

RED {

public String getColor() {return "红色";}

},

GREEN {

public String getColor() {return "绿色";}

},

BLUE {public String getColor() {return "蓝色";}

};

public abstract String getColor();

}

## 第16章：注解

1、认识Annotation

JDK1.5开始，Java增加了对元数据（即类的组成单元数据）的支持，也就是（Annotation）注解，他是代码里做的特殊标记，这些标记可以在编译，类加载，运行时在不改变原有逻辑的情况下，被读取，并执行相应的处理，通过使用Annotation，程序员可以在源文件中嵌入一些补充的信息。代码分析工具，开发工具和部署工具可以通过这些补充信息进行验证或者进行部署。Annotation类似于修饰符一样被使用，可以用于包，类，构造方法，方法，成员变量，参数，局部变量的声明。

 注意：

Annotation是一个接口

java.lang.Annotation接口.

2、系统定义的Annotation

在JDK 1.5之后，在系统中提供了三个Annotation，分别是：@Override、@Deprecated、@SuppressWarnings。

@Override

表示当前的方法定义将覆盖超类中的方法。如果你不小心拼写错误，或者方法签名对不上被覆盖的方法，编译器就会发出错误提示。

@Deprecated

表示的是一个类或方法已经不再建议继续使用了，标记为已过时。

@SuppressWarnings

表示关闭不当的编译器警告信息。

@SuppressWarnings(“unchecked”)//未检查的转化，如集合没有指定类型

@SuppressWarnings(“unused”) //未使用的变量

@SuppressWarnings(“resource”) //有泛型未指定类型

@SuppressWarnings(“path”) //在类路径，原文件路径中有不存在的路径

@SuppressWarnings("deprecation") //使用了某些不赞成使用的类和方法

@SuppressWarnings("fallthrough") //switch语句执行到底没有break关键字

@SuppressWarnings("serial")//某类实现Serializable 但是没有定义serialVersionUID 这个需要但是不必须的字段

@SuppressWarnings("rawtypes") //没有传递带有泛型的参数

@SuppressWarnings("all") //全部类型的警告

3、自定义Annotation

注解应用需要三个步骤：

（1）编写注解

（2）在类上应用注解

（3）对应用了注解的类进行反射操作的类

自定义Annotation的语法如下：

访问控制权限 @interface Annotation名称{}

例如：

public @interface MyAnnotation {}

在Annotation中定义变量

public @interface MyAnnotation {

public String name();

public String info();

}

定义变量后，在调用此Annotation时必须设置变量值。

@MyAnnotation(name = “vince", info = “hello")

public class Demo {

}

通过default指定变量默认值，

有了默认值在使用时可以不设值

public @interface MyAnnotation {

public String name() default “vince";

public String info() default “hello";

}

定义一个变量的数组，接收一组参数

public @interface MyAnnotation {

public String[] name();

}

使用时指定数组值

@MyAnnotation(name = { “jack", “vince" })

public class Demo {

}

使用枚举限制变量取值范围

public enum Color {

RED, GREEN, BLUE

}

public @interface MyAnnotation {

public Color color();

}

4、Retention和RetentionPolicy

Annotation要想决定其作用的范围，通过@Retention指定，而Retention指定的范围由RetentiontPolicy决定，RetentionPolicy指定了三种范围：

|  |  |
| --- | --- |
| 范围 | 描述 |
| public static final RetentionPolicy SOURCE | 在java源程序中存在 |
| public static final RetentionPolicy CLASS | 在java生成的class中存在 |
| public static final RetentionPolicy RUNTIME | 在java运行的时候存在 |

示列：

@Retention(value = RetentionPolicy.RUNTIME)

public @interface MyAnnotation {

public String name();

}

5、反射与Annotation

一个Annotation真正起作用，必须结合反射机制，在反射中提供了以下的操作方法：java.lang.reflect.AccessibleObject

|  |  |
| --- | --- |
| 方法名称 | 描述 |
| public boolean isAnnotationPresent(Class<? extends Annotation> annotationClass) | 判断是否是指定的Annotation |
| public Annotation[] getAnnotations() | 得到全部的Annotation |

示例：

Class<?> cls = Class.forName(“com.vince.annotation.Test");

Method met = cls.getMethod(“setName"); // 找到setName()方法

if (met.isAnnotationPresent(MyAnnotation.class)) {

MyAnnotation my = (MyAnnotation) met

.getAnnotation(MyAnnotation.class);

String name = my.name();

String info = my.info();

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

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

}

6、@Documented注解

此注解表示的是文档化，可以在生成doc文档的时候添加注解。

@Documented

@Retention(value = RetentionPolicy.RUNTIME)

public @interface MyAnnotation {

public String name();

public String info();

}

可以增加一些DOC注释。

/\*\*

\* 这个方法是从Object类中覆写而来的

\*/

@MyAnnotation(name = “vince", info = “teacher")

public String toString() {

return "hello";

}

7、@Target注解

@Target注解表示的是一个Annotation的使用范围，例如：之前定义的MyAnnotation可以在任意的位置上使用。

|  |  |
| --- | --- |
| 范围 | 描述 |
| public static final ElementType TYPE | 只能在类或接口或枚举上使用 |
| public static final ElementType FIELD | 在成员变量使用 |
| public static final ElementType METHOD | 在方法中使用 |
| public static final ElementType PARAMETER | 在参数上使用 |
| public static final ElementType CONSTRUCTOR | 在构造中使用 |
| public static final ElementType LOCAL\_VARIABLE | 局部变量上使用 |
| public static final ElementType ANNOTATION\_TYPE | 只能在Annotation中使用 |
| public static final ElementType PACKAGE | 只能在包中使用 |

8、@Inherited注解

@Inherited表示一个Annotation是否允许被其子类继承下来。

示例

@Inherited

@Target(value = ElementType.TYPE)

@Retention(value = RetentionPolicy.RUNTIME)

public @interface MyAnnotation {

public String name();

public String info();

}

使用时允许被其子类所继承。

## 第17章：XML与JSON

1、什么是XML

XML（Extensible Markup Language 可扩展标记语言），XML是一个以文本来描述数据的文档。

一个XML文档示例：

<?xml version="1.0" encoding="UTF-8"?>

<people>

<person personid="E01">

<name>Tony</name>

<address>10 Downing Street, London, UK</address>

<tel>(061) 98765</tel>

<fax>(061) 98765</fax>

<email>tony@everywhere.com</email>

</person>

<person personid="E02">

<name>Bill</name>

<address>White House, USA</address>

<tel>(001) 6400 98765</tel>

<fax>(001) 6400 98765</fax>

<email>bill@everywhere.com</email>

</person>

</people>

2、XML的用途

XML技术的用途：

（1）充当显示数据（以XML充当显示层）

（2）存储数据（存储层）的功能

（3）以XML描述数据，并在联系服务器与系统的其余部分之间传递。(传输数据的一样格式)

从某种角度讲，XML是数据封装和消息传递技术。

3、SAX解析XML

SAX是Simple API for XML的缩写

SAX 是读取和操作 XML 数据更快速、更轻量的方法。SAX 允许您在读取文档时处理它，从而不必等待整个文档被存储之后才采取操作。它不涉及 DOM 所必需的开销和概念跳跃。 SAX API是一个基于事件的API ，适用于处理数据流，即随着数据的流动而依次处理数据。SAX API 在其解析您的文档时发生一定事件的时候会通知您。在您对其响应时，您不作保存的数据将会被抛弃。

SAX API中主要有四种处理事件的接口，它们分别是ContentHandler，DTDHandler， EntityResolver 和 ErrorHandler 。实际上只要继承DefaultHandler 类就可以，DefaultHandler实现了这四个事件处理器接口，然后提供了每个抽象方法的默认实现。

// 创建SAX解析器工厂对象

SAXParserFactory spf = SAXParserFactory.newInstance();

// 使用解析器工厂创建解析器实例

SAXParser saxParser = spf.newSAXParser();

// 创建SAX解析器要使用的事件侦听器对象

PersonHandler handler =

new PersonHandler();

// 开始解析文件

saxParser.parse(

new File(fileName), handler);

4、DOM解析XML

JAVA 解析 XML 通常有两种方式，DOM 和 SAX

DOM：Document Object Model（文档对象模型）

DOM的特性：

定义一组 Java 接口，基于对象，与语言和平台无关将 XML 文档表示为树，在内存中解析和存储 XML 文档，允许随机访问文档的不同部分。

DOM解析XML

DOM的优点，由于树在内存中是持久的，因此可以修改后更新。它还可以在任何时候在树中上下导航，API使用起来也较简单。

解析步骤：

DocumentBuilderFactory builder = DocumentBuilderFactory.newInstance();

DocumentBuilder db = builder.newDocumentBuilder();

db.parse("person.xml");

NodeList node\_person = doc.getElementsByTagName("person");

5、JDOM解析XML

JDOM是两位著名的 Java 开发人员兼作者，Brett Mclaughlin 和 Jason Hunter 的创作成果， 2000 年初在类似于Apache协议的许可下，JDOM作为一个开放源代码项目正式开始研发了。

JDOM 简化了与 XML 的交互并且比使用 DOM 实现更快，JDOM 与 DOM 主要有两方面不同。首先，JDOM 仅使用具体类而不使用接口。这在某些方面简化了 API，但是也限制了灵活性。第二，API 大量使用了 Collections 类，简化了那些已经熟悉这些类的 Java 开发者的使用。

下载地址：

http://www.jdom.org/downloads/index.html

解析步骤：

（1）SAXBuilder sax = new SAXBuilder();

（2）Document doc = sax.build(….);

（3）Element el = doc.getRootElement();  
（4）List list = el.getChildren();

（5）遍历内容

6、DOM4J解析XML

dom4j是一个非常非常优秀的Java XML API，具有性能优异、功能强大和极端易用使用的特点，同时它也是一个开放源代码的软件，可以在SourceForge上找到它。在对主流的Java XML API进行的性能、功能和易用性的评测，dom4j无论在那个方面都是非常出色的。如今你可以看到越来越多的Java软件都在使用dom4j来读写XML，特别值得一提的是连Sun的JAXM也在用dom4j。这是必须使用的jar包， Hibernate用它来读写配置文件。

下载地址：

https://dom4j.github.io/

解析步骤：

（1）SAXReader sax = new SAXReader();

（2）Document doc = sax.read(Thread.currentThread().getContextClassLoader()

.getResourceAsStream("person.xml"));

（3）Element root = doc.getRootElement();

（4）Iterator iterator = root.elementIterator();

（5）遍历迭代器

7、通过对象生成XML文件

根据对象生成XML文档.

使用Java提供的java.beans.XMLEncoder和java.beans.XMLDecoder类。

这是JDK 1.4以后才出现的类

步骤：

（1）实例化XML编码器

XMLEncoder xmlEncoder = new XMLEncoder(new BufferedOutputStream(new FileOutputStream(new File(“a.xml”))));

（2）输出对象

（3）关闭

8、各种解析方法比较

JDOM 和 DOM 在性能测试时表现不佳，在测试 10M 文档时内存溢出。

SAX表现较好，这要依赖于它特定的解析方式。一个 SAX 检测即将到来的XML流，但并没有载入到内存（当然当XML流被读入时，会有部分文档暂时隐藏在内存中）。   
  
DOM4J是这场测试的获胜者，目前许多开源项目中大量采用 DOM4J，例如大名鼎鼎的 Hibernate 也用 DOM4J 来读取 XML 配置文件。

xstream 实现XML的转换

9、JSON

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。 JSON 官方:http://www.json.org

JSON 数据格式的特点

JSON 建构于两种结构：

1、 “名称/值”对的集合

2.、 值的有序列表（数组）

JSON 表示名称 / 值对的方式 ：{ "firstName": "vince", "lastName":"ma", "email": "finally\_m@foxmail.com" }

表示数组

{ "user": [ { "firstName": "vince", "lastName":"ma", "email": "finally\_m@foxmail.com" },

{ "firstName": "lin", "lastName":"jacks", "email": “jacks@qq.com”}]

}

10、GSON组件的使用

GSON是Google开发的Java API，用于转换Java对象和Json对象。

下载地址：http://www.mvnrepository.com/artifact/com.google.code.gson/gson

解析JSON：JsonReader reader = new JsonReader(new StringReader(jsonData));

生成JSON：

private String createJSON(ArrayList<User> users) {

JSONObject jsonObject = new JSONObject();JSONArray array = new JSONArray();

int size = users.size();

try {for (int i = 0; i < size; i++) {

User user = users.get(i); JSONObject object = new JSONObject();

object.put("name", user.name);object.put("age", user.age);array.put(object);}

jsonObject.put("users",array);

return jsonObject.toString();

} catch (JSONException e) {e.printStackTrace();}

return null;

}

static class User {String name; int age;//…}

使用 Gson 直接把 JSON 数据转换成 Java 对象

public Student parserJSON2(String data){

Gson gson = new Gson();

Student s = gson.fromJson(data, Student.class);

return s;

}

使用 Gson 直接把 JSON 数组转换成 Java 对象

public List<Student> parserJSON3(String data){

Type type = new TypeToken<ArrayList<Student>>(){}.getType();

Gson gson = new Gson();

List<Student> list = gson.fromJson(data, type);

return list;

}

11、XML与JSON的比较

从以下6点比较JSON与XML：

1、JSON和XML的数据可读性基本相同

2、JSON和XML同样拥有丰富的解析手段

3、JSON相对于XML来讲，数据的体积小

4、JSON与JavaScript的交互更加方便

5、JSON对数据的描述性比XML较差

6、JSON的速度要远远快于XML

适合的场景：

（1）数据传输：JSON要比XML更有优势

（2）存储数据：XML描述性更强

（3）XML通常用做配置文件（WEB课程中会有详细介绍）

## 第18章：GUI事件处理

1、GUI组件介绍

GUI编程（Graphic User Interface，图形用户接口）

GUI的各种元素，如：容器、按钮、文本框等

1、Frame类

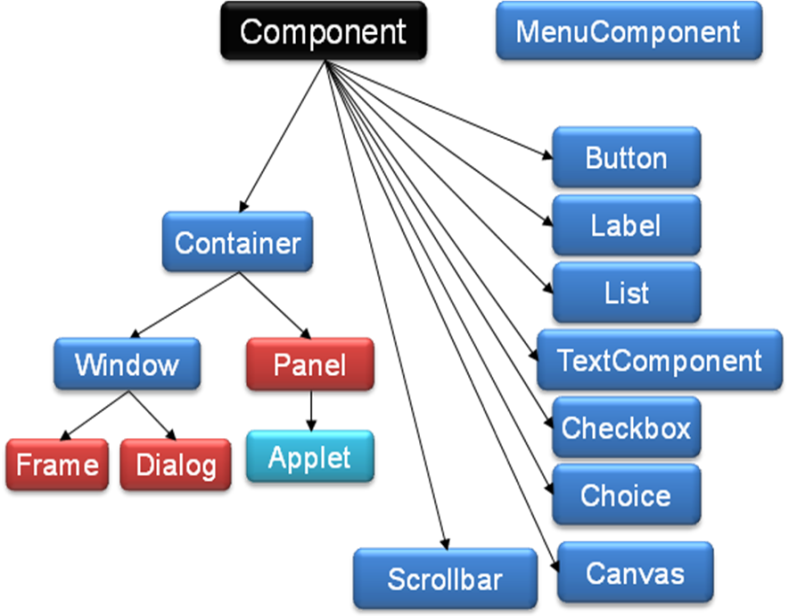
2、Button类

3、Panel类

5、Toolkit类

5、布局管理器

6、基本组件



2、事件处理

事件处理：

事件(Event)：用户对组件的一个操作，称之为一个事件

事件源(Event source) ：产生事件的对象

事件处理方法(Event handler) ： 能够接收、解析和处理事件类对象、实现和用户交互的方法 ，事件监听器。

为简化编程，针对大多数事件监听器接口定义了相应的实现类----事件适配器类，在适配器类中，实现了相应监听器接口中所有的方法，但不做任何事情。

3、观察者模式

1、观察者模式原理

观察者模式定义：简单地说，观察者模式定义了一个一对多的依赖关系，让一个或多个观察者对象监察一个主题对象。这样一个主题对象在状态上的变化能够通知所有的依赖于此对象的那些观察者对象，使这些观察者对象能够自动更新。

2、观察者模式实现

Subject（被观察的对象接口）

规定ConcreteSubject的统一接口；

每个Subject可以有多个Observer；

ConcreteSubject（具体被观察对象）

维护对所有具体观察者的引用的列表；

状态发生变化时会发送通知给所有注册的观察者。

Observer（观察者接口）

规定ConcreteObserver的统一接口；

定义了一个update()方法，

在被观察对象状态改变时会被调用。

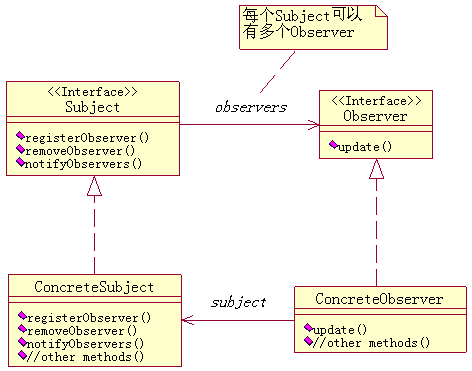
ConcreteObserver（具体观察者）

维护一个对ConcreteSubject的引用；

特定状态与ConcreteSubject同步；

实现Observer接口，通过update()方法

接收ConcreteSubject的通知。



3、观察者模式作用

观察者模式在被观察者和观察者之间建立一个抽象的耦合。被观察者角色所知道的只是一个具体观察者列表。

 由于被观察者和观察者没有紧密地耦合在一起，因此它们可以属于不同的抽象化层次。如果被观察者和观察者都被扔到一起，那么这个对象必然跨越抽象化和具体化层次。

 观察者模式支持广播通讯。被观察者会向所有的登记过的观察者发出通知。

## 第19章：项目实战

1、需求分析

2、环境准备

3、项目框架搭建

4、功能实现

5、项目总结

## Java常用专业英语词汇

abstract (关键字) 抽象 ['.bstr.kt]

access vt.访问,存取 ['.kses]‘(n.入口,使用权)

algorithm n.算法 ['.lg.riem]

annotation [java]代码注释 [.n.u'tei..n]

anonymous adj.匿名的[.'n.nim.s]‘

(反义：directly adv.直接地,立即[di'rektli, dai'rektli])

apply v.应用,适用 [.'plai]

application n.应用,应用程序 [,.pli'kei..n]‘ (application crash 程序崩溃)

arbitrary a.任意的 ['ɑ:bitr.ri]

argument n.参数;争论,论据 ['ɑ:gjum.nt]‘(缩写args)

assert (关键字) 断言 [.'s.:t] ‘ (java 1.4 之后成为关键字)

associate n.关联(同伴,伙伴) [.'s.u.ieit]

attribute n.属性(品质,特征) [.'tribju:t]

boolean (关键字) 逻辑的, 布尔型

call n.v.调用; 呼叫; [k.:l]

circumstance n.事件(环境,状况) ['s.:k.mst.ns]

crash n.崩溃,破碎 [kr..]

cohesion 内聚,黏聚,结合 [k.u'hi:..n]

a class is designed with a single, well-focoused purpose.

应该不止这点

command n. 命令,指令 [k.'mɑ:nd](指挥, 控制)

(command-line 命令行)

Comments [java]文本注释 ['k.ments]

compile [java] v.编译 [k.m'pail]‘

Compilation n.编辑[,k.mpi'lei..n]

const (保留字)

constant n. 常量, 常数, 恒量 ['k.nst.nt]

continue (关键字)

coupling 耦合,联结 ['k.pli.]

making sure that classes know about other classes only through their APIs.

declare [java]声明 [di'kl..]

default (关键字) 默认值 ; 缺省值 [di'f.:lt]

delimiter 定义符; 定界符

Encapsulation [java]封装 (hiding implementation details)

Exception [java]例外; 异常 [ik'sep..n]

entry n.登录项, 输入项, 条目['entri]

enum (关键字) 枚举

execute vt.执行 ['eksikju:t]

exhibit v.显示, 陈列 [ig'zibit]

exist 存在, 发生 [ig'zist] ‘(SQL关键字exists)

extends (关键字) 继承、扩展 [ik'stend]

false (关键字)

final (关键字) finally (关键字)

fragments 段落; 代码块 ['fr.gm.nt]

FrameWork [java]结构,框架 ['freimw.:k]

Generic [java]泛型 [d.i'nerik]

goto (保留字) 跳转

heap n.堆 [hi:p]

implements (关键字) 实现 ['implim.nt]

import (关键字) 引入(进口,输入)

Info n.信息

(information [,inf.'mei..n] )

Inheritance [java]继承 [in'herit.ns] (遗传,遗产)

initialize 预置初始化 [i'ni..laiz]

instanceof (关键字) 运算符，用于引用变量，以检查这个对象

是否是某种类型。返回 boolean 值。

interface (关键字) 接口 ['int.feis]

invoke vt.调用 [in'v.uk]‘ (invocation [,inv.u'kei..n])

Iterator [java]迭代器, 迭代程序

legal 合法的 ['li:g.l]

log n.日志,记录 [l.g]

native (关键字) ['neitiv]

nested [java]嵌套的 ['nestid] ‘如：内部类(nested classes)

Object [java]对象 ['.bd.ekt]

Overload [java]方法的重载(不同参数列表的同名方法) [,.uv.'l.ud]

Override [java]方法的覆盖(覆盖父类的方法) [,.uv.'raid]

polymiorphism [java]多态 (polymorphism 多形性[,p.li'm.:fizm])

allowing a single object to be seen as having many types.

principle n.原则,原理,主义 ['prinsipl]

priority n. 优先级 [prai'.riti]

process n. 程序, 进程 ['pr.ses]

protected (关键字) 受保护的,私有的 [pr.'tektid]

provide v.规定(供应,准备,预防)[pr.'vaid]

refer to v.引用 [ri'f.:][tu:]

reference n. 参考(引用,涉及)['ref.r.ns]‘

–>reference variable 参量, 参考变量,引用变量

Reflection [java]反射 [ri'flek..n]

script n.手写体,小型程序 [skript]

serialized vt.序列化,串行化 ['si.ri.laiz]‘

(serializable adj.)(deserialize反序列化,反串行化)

Socket [java]网络套接字['s.kit]

stack n.堆栈 [st.k] (对应heap 堆)

statement 程序语句; 语句 ['steitm.nt]‘ n. 陈述,指令

subclass n.子类 ['s.bklɑ:s]‘ (supertype 父类)

switch (关键字) 选择语句。 n.开关,道岔 [swɪtʃ]

synchronized (关键字) 同步(锁) ['si.kr.naiz]

Thread [java]线程 [θred]

throw (关键字) throws (关键字) [θr.u] 抛出(异常)

transient (关键字) 瞬变;临时的['tr.nzi.nt]‘(可序列化)

valid 正确的,有效的 ['v.lid]

variable n.变量 a.可变的['v..ri.bl]

volatile (关键字) 不稳定的['v.l.tail]

while (关键字) 循环语句。当…的时候 [hwail]

java常用单词以及它的用法

lang包：

字符串类的方法：

character类：

isLetter():判断是不是字母

isDigit():判断是不是数字

isWhiteSpace():判断是不是空格

isUpperCase():判断是不是大写

isLowerCase():判断是不是小写

String类：

equals():比较对象中值是否相等

length():返回字符串长度

CompareTo():比较相同索引位置上字符的 ASCII

StartsWith()：判断是否以 …开头

endsWith():判断是否以…结尾

toUpperCase():将所有小写转为大写

toLowerCase():将所有大些转为小写

trim():截掉空格

indexOf():搜索某个字符在字符串中是否存在

LastIndexOf():从后往前搜索第一次遇到字符的索引

CharAt：根据下标找字符

subString：截取

concat：连结

StringBuffer类：

append():追加

setcharAt():将下表对应的字符更改为某个字符

insert():插入字符

reverse():倒置，反置

replace():替换

toString():返回当前对象的字符串表现形式

Math类:

static：静态

Class类：

getClass():获得类的描述信息

getSuperClass():获得父类的描述信息

forName():java与

sql建立连接时使用

util包：

Calendar类：日历类

Calendar.getInstance();获得该类的实例

Date：日期

Date.gettime();获得毫秒数

get():获得时间组成部分

add():加上时间组成部分

set():修改时间组成部分

DAY\_OF\_YEAR 这一年的第几天

DAY\_OF\_MONTH 这一月的第几天

DAY\_OF\_WEEK 这一周的第几天

Random 类:

Rand.nextInt 随机数

框架集合：

ArrayList： 查询数据比较多时用该类

size():测试容量

get():获得集合框架中的数据

add():向集合框架中添加数据

set():修改集合框架中的数据

isEmpty():判断集合框架是否为空

indexof():搜索内容

remove():删除

clear():清空

LinkedList ：当插入修改数据情况多时用该类

removeFirst():删除第一个元素

removeLast():删除最后一个元素

addFirst():添加第一个元素

addLast():添加最后一个元素

Vector类

addElement():添加

removeElement():删除

Hashmap类

put():添加方法

get():查询数据

remove():删除数据

IO包

length():测试文件容量

exists():判断文件是否存在

CreateNewFile():创建新文件

delete():删除原文件

getAbsolutePath():获得文件的绝对路径

FileOutputStream():输出流

FileInputStream():输入流

字符流：

FileReader：读文件

FileWriter:写文件

BufferedReader:缓冲读取流

readline():读取一行内容

BufferedReader 字符串读取的缓冲流

write():写

newLine():换行

System.in的read方法，可以从控制台接受输入：

InputStreamReader() 配合 BufferedReader()使用

SQL包

Java与 SQL server 2005 连接

六个步骤:

//建立数据源

//加载驱动程序

Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

//建立连接

Connection con=java.sql.DriverManager.getConnection(“jdbc:\\jdbc:myODBC”,”sa”,”");

//创建语句对象

Statement sta=con.createStatement();

//执行

SQL语句

String strsql=”Update teacher set address=’地名’ where name=’姓名’”;

int count= sta.executeUpdate(strsql);

System.out.println (“更新了”+count+”行记录”);

//关闭连接

sta.close();

con.close();

捕捉异常

catch(ClassNotFoundException cnfe) {

cnfe.printStackTrace();

}

catch(SQLException sqle) {

sqle.printStackTrace();

}

预编译语句对象：preparedStatement

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

path [ pɑ:θ ] 路径

class [klɑ:s] 类

classpath [klɑ:s'pɑ:θ ] 类路径

public ['p ʌblik] 公共的,公用的

private ['praivit] 私有的,私人的

static ['stæ tik] 静的;静态的;静止的

void [vɔid] 空的,没有返回值的

main [mein] 主要的,重要的

System ['sistəm] 系统,方法

out [aut] 往外,出现,出外

print [print ] 打印

Demo [ 'deməu ] 演示

version [ 'və:ʃən] 版本

author [ 'ɔ:θə ] 作者

byte [bait] 字节

short [ʃɔ:t] 短整型

int [int] 整型

long [lɔ:ŋ] 长整形

char [tʃɑ:] 字符型

String [striŋ] 字符串类型

float [fləut] 单精度浮点类型

double ['dʌbl] 双精度浮点型,双倍

type [taip] 类型

boolean ['bu:li:ən] 布尔类型真假二值

true [tru:] 真，正确的(成立的)

false [fɔ:ls] 假，错误的(不成立的)

binary ['bainəri] 二进制

octal ['ɔktəl] 八进制

hex [heks] 十六进制

count [kaunt] 计数；计算；(计数器思想,上课注意听)

sum [ s ʌm] 总数,总计

salary [ 'sæləri ] 薪水

Random ['ræn dəm] 随机的

Scanner ['skænə] 接收输入

import [im' pɔ:t] 导入,输入

if [if] 如果

else [els] 否则

switch [switʃ] 判断语句

case [keis] 实例,情况

break [breik] 退出

continue [kən 'tinju] 跳出…继续

return [ri tə:n] 返回

default [di'fɔ:lt]或者的意思(在软件配置中还有默认的意思)

while [wail] 当……的时候

loop [lu:p] 循环

Max [ ma: k s] 最大的

Min [ mi n] 最小的

Function [ 'fʌŋ k ʃən ] 功能,函数

Method [ 'me θə d ] 方法

result [ ri'zʌlt ]结果

Compare [ kəm' pεə ] 比较

temp [ tem p ] 临时

Array [ ə'rei ] 数组

null [nʌl ] 空,无效的

pointer ['p ɔi n tə] 指针

Exception [ ik 'sep ʃən] 异常

Error [ erə ] 错误

Thread [ θred ] 线程,线状物

Lang [læŋ] language['læŋɡ wi dʒ] 语言的缩写

index ['indeks] 角标,索引,指针

Outof [ a:ut ɔf] 在…之外

bound [baund] 范围

length [ leŋθ] 长度

sort [sɔ:t] 将…排序 //Arrays.sort(arr)

select [si'lekt] 选择 //selectSort

bubble ['bʌbl] 冒泡,气泡 //bullbeSort

change [ tʃeindʒ ] 改变，交换

swap [ swɔp ] 交换

nullpointerException 空指针异常

ArrayIndexOutOfBoundsException 数组角标越界异常

half [ hɑ:f ] 一半的,半个的

search [ sə: tʃ] 搜索,查找

mid [ mid] 中间的

number [ 'nʌmbə ] 数字,号码

util [ ju'til]是utility[ ju'tiliti ]功能,功效的缩写

Transition [ træn'ziʃən] 转换通常会用trans为其简写形式

Object [ 'ɔb dʒi kt ] 对象, 物体

Class member [ 'membə ] 类成员

Class method [ 'meθəd] ] 类方法

Class variable [ 'vεə riə ble ] 类变量

Constructor [ kən'strʌktə ] 构造方法,构造器

package [ 'pækidʒ] 包

Car [ kɑ: ] 汽车,小轿车

Color [ 'kʌlə] 颜色

Red [ red ] 红色

Blue [ blu: ] 蓝色

Black [ blæ k] 黑色

White [ hwait ] 白色

run [ run] 跑, 运行

Person [ 'pə:sən ] 人

People [ 'pi:pl ] 人,人们

Age [ eidʒ] 年龄

Name [ neim ] 名字,名称,名誉,名人

Speak [ spi:k] 说话,讲话,

set [ set] 设置

reset [r i set] 重新设置

get [get ] 获取,获得

code [ kəud ] 代码,代号,编码

country [ 'kʌn tri ] 国家,国土

University [ju:ni'və:siti] 大学

College [ 'kɔlidʒ] 大学

Private [ 'praivit ] 私有的,私人的

show [ ʃəu] 显示,展示

sleep [ sli:p ] 睡, 睡觉, 睡眠 1秒=1000毫秒

single [ 'siŋgl] 单一的,单独的,单身的,单程的

instance [ 'instəns ]实例,情况

tool [ tu:l] 工具

Data [ 'deitə ] 资料

develop [ di'veləp ] 开发

extends [ ik'stends] 继承,延伸,推广

Phone [ fəun ] 电话

Call [ kɔ:l] 呼叫,给…打电话

Study [ 'stʌdi] 学习,研究

work [ wə:k ] 工作,操作

super ['sju: pə] 父类的,超级的,,超类的

final ['fai nl] 最终的

finally

finalized

abstract ['æbstrækt] 抽象的

interface [ 'intəfeis ] 接口

implements ['impliments] 实现

mouse [maus ] 鼠标, 老鼠

bonus [ 'bəunəs ] 奖金

inner [ 'inə ] 内部的

protect [prəu'tekt] 保护，受保护的，防卫

equals ['i:kwəls]判断两个字符串是否相等

memory ['meməri] 记忆，内存，存储器

OutOfMemoryError 在内存之外的错误：内存溢出（专业说法）

heap [hi:p] 堆

space [speis] 空间

Java heap space Java堆内存

Throw [θrəu] 抛出,投掷

Throws [θrəuz] throw的复数形式

finally ['fainəli] 最后,最终

Message ['mesidʒ] 消息,信息,通知

try [trai] 尝试,试图

catch [kætʃ] 捕获,捕捉,抓住

close [kləuz] 关闭,结束

open ['əupən] 打开,公开

add [æd] 添加,增加

prelect [pri'lekt ] 讲课,讲演

directory [ di'rektəri ] 目录

throwable [ 'θrəuə'ei bl ] 直译是可抛出的,在Java中指异常类.

ClassCastException (“类型错误”),类型转换异常

Runtime [run'taim] 运行时间

RuntimeException 运行时异常

Current ['kʌrənt] 当前的,现在的

Arithmetic [ə'riθmətik ] 算数

synchronized [ 'sɪŋkrə,naɪzd] 同步的

Runnable [ 'rʌnəbl] 可捕捉的(在java中它是一个接口)

Inter rupted [,intə'rʌptid] 中断的,被阻止的

Ticket ['tikit] 票

Bank [bæŋk] 银行

tank 坦克

flag [flæɡ] 标记, 旗帜

dead [ded] 死亡的,无生命的

lock [ lɔk] 锁

unlock

Resource [ ri'sɔ:s] 资源

Notify ['nəutifai ] 唤醒, 告知

Input ['input] 输入

Output [ 'autput ] 输出

bounded ['baundid] 有界限的

Buffer ['bʌfə] 缓冲区

boundedbuffer 有界缓冲区

Signal ['siɡ nəl] 信号,标志

Condition [kən'diʃən] 条件

Producer [prə'du:sə] 生产者

Consumer [ kən'sju:mə ] 消费者

substring [ 'sʌb striŋ] 子字符串, 部分字符串,子链

Trim [trim] 修剪, 修整

Append [ə'pend] 添加

Add [ æd] 增加

Insert [in'sə:t] 插入, 比如把一段字符串 “cd” 插入到 “abef”

Delete [di'li:t] 删除

remove

replace [ri'pleis] 代替,取代,更换

update [ ʌp'deit] 更新

Create [ kri'eit ] 创造,创作

Builder ['bildə] Java中为 “缓冲器”, 直译为建筑者

StringBuffer 字符串缓冲区(同步的)

StringBuilder 字符串缓冲区(不同步的)

char [tʃɑ:] 字符型

character ['kærəktə] 字符

int [int] 整型

integer ['intidʒə] 整数类型

parse [pɑ:z] 解析, 转换成…

parseInt 转换为int类型

Retain [ri'tein] 保持

reverse [ ri'və:s ] 反转,翻转

collection [kə'lekʃən] 集合,聚集,采集

Remove [ri'mu:v] 移除,删除

clear [kliə] 清空,清除,清楚的.

Iterator [itə'rei tə] 迭代器

list [list] 列表,清单(集合之一)

ArrayList ['əri list] 数组列表(最常用的集合,重点掌握)

HasPrevious ['pri:vjəs] 在…之前

HasNext

Split [split] 切割

Element ['elimənt] 元素

NoSuchElementException

linked [ liŋkt] 连接的

contains [kən'teinz] 包含

Set [ set] 集合

Comparator ['kəmpə reitə] 比较仪

style [stail] 风格,类型

math [mæθ] 数学,算数

ceil() [ si:l] 返回大于参数的最小整数。

floor() [flɔ:] 返回小于参数的最大整数。

round() [raund] 返回四舍五入的整数。

pow(a,b) [ p a u]a的b次方。

execute ['eksikju:t] 执行

process ['prəuses] 处理,加工

Notepad ['nəutpæd] 笔记本

destroy [di'strɔi] 消毁, 破坏,消灭

Separator ['sepə reitə] 分隔符,分离器

Generic [dʒi'nerik] 类的,一般的

vector ['vektə] 也是一个集合(早期的一个集合现在几乎不用了)

value ['vælju:] 值,价值

Calendar ['kæ lən dɚ] 日历,日历表,历法

Day [dei] 一天, 每天

Week [wi:k] 周,星期

Month [ mʌnθ] 月

Year [jiə] 年

format ['fɔ:mæt] 格式,样式

ParseException 解析异常

stack [stæk] 堆

trace [treis] 痕迹

printStackTrace

by [bai] 通过,按照

order ['ɔ:də] 顺序,规则

byOrder

line [lain] 一行

readLine() 读取一行

stream [stri:m] 流 (生活中常见的流有: 水流,电流,数据流)

flush [flʌʃ]清空缓冲区数据,然后缓冲区的数据就去了它们该去的地方

filter [filtə]过滤器

Absolute ['æbsəlju:t] 绝对的

hidden ['hidən] 隐藏的

Suffix [sə'fiks] 后缀的

level ['levəl] 标准,水平的

store [stɔ:] 储存

accept [ək'sept] 接受,承认,同意

sequence ['si:kwəns] 序列

Enumeration [i,nju:mə'reiʃən] 枚举,列举

merge [mə:dʒ] 合并

access ['ækses] 使用

Un supported [,ʌnsə'pɔ:tid] 不支持的

Unsupported Encoding Exception

compare [kəm'pεə] 比较

comparable ['kɔm pə rəbl] 可以比较的

field ['fi:ld] 字段

button ['bʌtən] 按钮

frame [freim] 框架,结构

action ['ækʃən] 行动,动作

event [i'vent] 事件,

visible ['vizəbl] 看的见的,可见的

menu ['menju:] 菜单

area ['εə r iə] 区域,地区

item ['aitəm] 条目,项目

Scroll [sk r əul] 卷轴,滚动

pane [pein] 窗格

ScrollPane 滚动窗口；滚动面板

Content ['kɔntent] 内容

Context

choose [tʃu:z] 选择

dialog ['daiəlɔg] 对话,会话

Relative [ 'relətiv ] 相对的

Init [ i,nit ] 初始化

Initialization [ i,ni ʃəl'zeiʃən] 初始化

perform [pə'fɔ:m] 执行,运转

Port [ pɔ:t ] (计算机的)端口

Viewport [ 'vju:pɔ:t ] 视口

pressed [prest] 紧迫的, 加压的

display [,dis'plei] 显示; 表现

dispose [dis'pəuz] 处理,安排

Operator ['ɔpəreitə] 操作员

operation [,ɔpə'reiʃən] 操作(动词)

host [həust] 主机,主人

address [ə'dres] 地址

HostAddress

receive [ri'si:v] 接收

send [send] 发送

client ['klaiənt] 客户

Server ['sə:və] 服务器

log [lɔ'gin] 注册

register ['redʒistə] 登录

upload [ 'ʌpləud] 上传

Download [daun'ləud] 下载

Task [tɑ:sk] 任务

Browse [brauz] 浏览

Browser [ brawə:] 浏览器

connection [kə'nekʃən] 连接

layout ['leiaut] 布局

Terminate [ 'tə:mineit ] 结束

Instantiation [in,stænʃi'eiʃən] 实例化

Instance

declared [di'klεəd] 公然的

Access [ ək'ses ] 进入,接近,入口,通道

Accessible [ək'sesəbl] 可进入的

invoke [in'vəuk] 调用

board [bɔ:d] 木板,甲板

MainBoard

card [kɑ:d] 卡片

Network card

match [mætʃ] 匹配

matcher ['mætʃə] 匹配器

pattern ['pætən] 模式,图案

Regex [ ri'dʒeks ] 正则表达式

title ['taitl] 标题

untitled [,ʌn'taitld] 无标题的

Charset [ tʃɑ: set ] 字符集,编码集

Encoding

Rows [rəus ] 多行,

Cols [ kɔls] 列数

Top [ tɔp] 顶部,顶端

center ['sentə] 中心, 中央

target ['tɑ:ɡit] 目标

document ['dɔkjumənt] 文件

table ['teibl] 表格,桌子

span [spæn] 跨度,范围

User ['juzə] 用户

password ['pɑ:swə:d ] 密码

Radio [ 'reidiəu ] 单选框

Checkbox [ 'tʃekbɔks ] 复选框

image ['imidʒ] 图像,影像

none [nʌn] 没有

option ['ɔpʃən] 选项

Form [ fɔ;m] 结构,形式

Height [hait] 高度

Width [wiθ] 宽度

border ['bɔ:də] 边框宽度

alert [ə'lə:t] 警告,弹出

Onclick [ ɔn'klik] 鼠标单击

reset [ 'ri:set] 重置,清零

submit [səb'mit] 提交

Stylesheet [stail' ʃ:t] 样式表

Background ['bækɡraʊnd] 背景

decoration [,dekə'reiʃən] 装饰,装潢

Hover ['hɔvə, 'hʌvə] 鼠标移动到链接上

Active ['æktiv] 选定的链接

Visit ed ['vi zi tid] 已访问的链接

Link [liŋk] 未访问的链接

Letter ['le tə]

Focus ['fəukəs] 焦点,中心

blank [blæŋk] 空白的

Bottom ['bɔtəm] 底部,末端

Botton

Pad ding [ pæd iŋ ]

Solid ['sɔlid] 实线边框

Dashed [dæʃt] 在MAC平台上IE4+与WINDOWS和UNIX平台上IE5.5+为虚线。否则为实线

Groove [ɡru:v] 凹槽

Dotted ['dɔtid] 在MAC平台上IE4+与WINDOWS和UNIX平台上IE5.5+为点线。否则为实线

Position [pə'zɪʃən] 位置

Var [vɑ:] 在Script中用来定义变量类型的单词

undefined [,ʌndi'faind] 未定义的,不明确的

global ['ɡləubəl] 全局的, 总体的

Generate [ dʒnəreit] 生成

Modified [ 'mɔdi faid] 改进的,改良的

Eclipse中用到的单词

application

perspective [pə'spektiv]

location [ ləu'keiʃən] 位置(文件在电脑中的位置)

encoding [in'kəudiŋ] 编码

Properties [ 'prɔpətis] 属性,性能

File [ 'fail] 文件

Folder ['fəʊldə] 文件夹

View [ vju: ] 视图, 视线,看

Show view 显示视图

Refresh [ri'freʃ] 更新,刷新

Edit ['edit] 编辑

Project [prədʒekt] 项目,工程,计划

Preference ['prefə rəns] 个人喜好/在汉化版中是”属性”

rename [,ri:'neim] 改名,重新命名 (F2快捷键)

template ['templit] 模板

console [kən'səul] 控制台

font [fɔnt] 字体

syntax ['sintæks] 语法

bracket ['brækit] 括号

keyword ['ki:wə:d] 关键字

excluding [ik'sklu:diŋ] 将…排除在外

Java基础常见英语词汇(共70个)

OO: object-oriented ,面向对象

OOP: object-oriented programming,面向对象编程

JDK:Java development kit, java开发工具包

JVM:java virtual machine ,java虚拟机

Compile:编绎

Run:运行

Class:类

Object:对象

System:系统

out:输出

print:打印

line:行

variable:变量

type:类型

operation:操作,运算

array:数组

parameter:参数

method:方法

function:函数

member-variable:成员变量

member-function:成员函数

get:得到

set:设置

public:公有的

private:私有的

protected:受保护的

default:默认

access:访问

package:包

import:导入

static:静态的

void:无(返回类型)

extends:继承

parent class:父类

base class:基类

super class:超类

child class:子类

derived class:派生类

override:重写,覆盖

overload:重载

final:最终的,不能改变的

abstract:抽象

interface:接口

implements:实现

exception:异常

Runtime:运行时

ArithmeticException:算术异常

ArrayIndexOutOfBoundsException:数组下标越界异常

NullPointerException:空引用异常

ClassNotFoundException:类没有发现异常

NumberFormatException:数字格式异常(字符串不能转化为数字)

Try:尝试

Catch:捕捉

Finally:最后

Throw:抛出

Throws: (投掷)表示强制异常处理

Throwable:(可抛出的)表示所有异常类的祖先类

Lang:language,语言

Util:工具

Display:显示

Random:随机

Collection:集合

ArrayList:(数组列表)表示动态数组

HashMap: 散列表,哈希表

Swing:轻巧的

Awt:abstract window toolkit:抽象窗口工具包

Frame:窗体

Size:尺寸

Title:标题

Add:添加

Panel:面板

Layout:布局

Scroll:滚动

Vertical:垂直

Horizonatal:水平

Label:标签

TextField:文本框

TextArea:文本域

Button:按钮

Checkbox:复选框

Radiobutton:单选按钮

Combobox:复选框

Event:事件

Mouse:鼠标

Key:键

Focus:焦点

Listener:监听

Border:边界

Flow:流

Grid:网格

MenuBar:菜单栏

Menu:菜单

MenuItem:菜单项

PopupMenu:弹出菜单

Dialog:对话框

Message:消息

Icon:图标

Tree:树

Node:节点

Jdbc:java database connectivity,java数据库连接

DriverManager:驱动管理器

Connection:连接

Statement:表示执行对象

Preparedstatement:表示预执行对象

Resultset:结果集

Next:下一个

Close:关闭

executeQuery:执行查询

Jbuilder中常用英文(共33个)

File:文件

New:新建

New Project:新建项目

New Class: 新建类

New File:新建文件

Open project:打开项目

Open file:打开文件

Reopen:重新打开

Close projects:关闭项目

Close all except…:除了..全部关闭

Rename:重命名

Exit:退出

View:视图

Panes:面板组

Project:项目

Content:内容

Structure:结构

Message:消息

Source:源文件

Bean:豆子

Properties:属性

Make:编绎

Build:编绎

Rebuild:重编绎

Refresh:刷新

Project properties:项目属性

Default project properties:默认的项目属性

Run:运行

Debug:调试

Tools:工具

Preferences:参数配置

Configure:配置

Libraries:库

JSP中常用英文

URL: Universal Resource Location:统一资源定位符

IE: Internet Explorer 因特网浏览器

JSP:java server page.java服务器页面

Model:模型

View:视图

C:controller:控制器

Tomcat:一种jsp的web服务器

WebModule:web模块

Servlet:小服务程序

Request:请求

Response:响应

Init: initialize,初始化

Service:服务

Destroy:销毁

Startup:启动

Mapping:映射

pattern:模式

Getparameter:获取参数

Session:会话

Application:应用程序

Context:上下文

redirect:重定向

dispatch:分发

forward:转交

setattribute:设置属性

getattribute:获取属性

page:页面

contentType:内容类型

charset:字符集

include:包含

tag:标签

taglib:标签库

EL:expression language,表达式语言

Scope:作用域

Empty:空

JSTL:java standard tag library,java标准标签库

TLD:taglib description,标签库描述符

Core:核心

Test:测试

Foreach:表示循环

Var:variable,变量

Status:状态

Items:项目集合

Fmt:format,格式化

Filter:过滤器

Data Structures 基本数据结构

Dictionaries 字典

Priority Queues 堆

Graph Data Structures 图

Set Data Structures 集合

Kd-Trees 线段树

Numerical Problems 数值问题

Solving Linear Equations 线性方程组

Bandwidth Reduction 带宽压缩

Matrix Multiplication 矩阵乘法

Determinants and Permanents 行列式

Constrained and Unconstrained Optimization 最值问题

Linear Programming 线性规划

Random Number Generation 随机数生成

Factoring and Primality Testing 因子分解/质数判定

Arbitrary Precision Arithmetic 高精度计算

Knapsack Problem 背包问题

Discrete Fourier Transform 离散Fourier变换

Combinatorial Problems 组合问题

Sorting 排序

Searching 查找

Median and Selection 中位数

Generating Permutations 排列生成

Generating Subsets 子集生成

Generating Partitions 划分生成

Generating Graphs 图的生成

Calendrical Calculations 日期

Job Scheduling 工程安排

Satisfiability 可满足性

Graph Problems — polynomial 图论-多项式算法

Connected Components 连通分支

Topological Sorting 拓扑排序

Minimum Spanning Tree 最小生成树

Shortest Path 最短路径

Transitive Closure and Reduction 传递闭包

Matching 匹配

Eulerian Cycle / Chinese Postman Euler回路/中国邮路

Edge and Vertex Connectivity 割边/割点

Network Flow 网络流

Drawing Graphs Nicely 图的描绘

Drawing Trees 树的描绘

Planarity Detection and Embedding 平面性检测和嵌入

Graph Problems — hard 图论-NP问题

Clique 最大团

Independent Set 独立集

Vertex Cover 点覆盖

Traveling Salesman Problem 旅行商问题

Hamiltonian Cycle Hamilton回路

Graph Partition 图的划分

Vertex Coloring 点染色

Edge Coloring 边染色

Graph Isomorphism 同构

Steiner Tree Steiner树

Feedback Edge/Vertex Set 最大无环子图

Computational Geometry 计算几何

Convex Hull 凸包

Triangulation 三角剖分

Voronoi Diagrams Voronoi图

Nearest Neighbor Search 最近点对查询

Range Search 范围查询

Point Location 位置查询

Intersection Detection 碰撞测试

Bin Packing 装箱问题

Medial-Axis Transformation 中轴变换

Polygon Partitioning 多边形分割

Simplifying Polygons 多边形化简

Shape Similarity 相似多边形

Motion Planning 运动规划

Maintaining Line Arrangements 平面分割

Minkowski Sum Minkowski和

Set and String Problems 集合与串的问题

Set Cover 集合覆盖

Set Packing 集合配置

String Matching 模式匹配

Approximate String Matching 模糊匹配

Text Compression 压缩

Cryptography 密码

Finite State Machine Minimization 有穷自动机简化

Longest Common Substring 最长公共子串

Shortest Common Superstring 最短公共父串

DP——Dynamic Programming——动态规划

recursion ——递归

)

其它

第一章：

JDK(Java Development Kit) java开发工具包

JVM(Java Virtual Machine) java虚拟机

Javac 编译命令

java 解释命令

Javadoc 生成java文档命令

classpath 类路径

Version 版本

author 作者

public 公共的

class 类

static 静态的

void 没有返回值

String 字符串类

System 系统类

out 输出

print 同行打印

println 换行打印

JIT(just-in-time) 及时处理

第二章：

byte 字节

char 字符

boolean 布尔

short 短整型

int 整形

long 长整形

float 浮点类型

double 双精度

if 如果

else 否则

switch 多路分支

case 与常值匹配

break 终止

default 默认

while 当到循环

do 直到循环

for 已知次数循环

continue结束本次循环进行下次跌代

length 获取数组元素个数

第三章：

OOP object oriented programming 面向对象编程

Object 对象

Class 类

Class member 类成员

Class method 类方法

Class variable 类变量

Constructor 构造方法

Package 包

Import package 导入包

第四章：

Extends 继承

Base class 基类

Super class 超类

Overloaded method 重载方法

Overridden method 重写方法

Public 公有

Private 私有

Protected 保护

Static 静态

Abstract 抽象

Interface 接口

Implements interface 实现接口

第五章：

Exception 意外，异常

RuntimeExcepiton 运行时异常

ArithmeticException 算术异常

IllegalArgumentException 非法数据异常

ArrayIndexOutOfBoundsException 数组索引越界异常

NullPointerException 空指针异常

ClassNotFoundException 类无法加载异常（类不能找到）

NumberFormatException 字符串到float类型转换异常（数字格式异常）

IOException 输入输出异常

FileNotFoundException 找不到文件异常

EOFException 文件结束异常

InterruptedException （线程）中断异常

try 尝试

catch 捕捉

finally 最后

throw 投、掷、抛

throws 投、掷、抛

print Stack Trace() 打印堆栈信息

get Message（）获得错误消息

get Cause（）获得异常原因

method 方法

able 能够

instance 实例

check 检查

第六章：

byte（字节）

char（字符）

int（整型）

long（长整型）

float（浮点型）

double（双精度）

boolean（布尔）

short（短整型）

Byte （字节类）

Character （字符类）

Integer（整型类）

Long （长整型类）

Float（浮点型类）

Double （双精度类）

Boolean（布尔类）

Short （短整型类）

Digit （数字）

Letter （字母）

Lower (小写)

Upper (大写)

Space (空格)

Identifier (标识符)

Start (开始)

String (字符串)

length （值）

equals (等于)

Ignore （忽略）

compare （比较）

sub （提取）

concat （连接）

replace （替换）

trim （整理）

Buffer (缓冲器)

reverse (颠倒)

delete （删除）

append （添加）

Interrupted （中断的）

第七章：

Date 日期，日子

After 后来，后面

Before 在前，以前

Equals 相等，均等

toString 转换为字符串

SetTime 设置时间

Display 显示，展示

Calendar 日历

Add 添加，增加

GetInstance 获得实例

getTime 获得时间

Clear 扫除，清除

Clone 克隆，复制

Util 工具，龙套

Components 成分，组成

Month 月份

Year 年，年岁

Hour 小时，钟头

Minute 分钟

Second 秒

Random 随意，任意

Next Int 下一个整数

Gaussian 高斯

ArrayList 对列

LinkedList 链表

Hash 无用信息，杂乱信号

Map 地图

Vector 向量，矢量

Size 大小

Collection 收集

Shuffle 混乱，洗牌

RemoveFirst 移动至开头

RemoveLast 移动至最后

lastElement 最后的元素

Capacity 容量，生产量

Contains 包含，容纳

Copy 副本，拷贝

Search 搜索，查询

InsertElementAt 插入元素在某一位置

第八章：

io->in out 输入/输出

File 文件

import 导入

exists 存在

isFile 是文件

isDirectory 是目录

getName 获取名字

getPath 获取路径

getAbsolutePath 获取绝对路径

lastModified 最后修改日期

length 长度

InputStream 输入流

OutputStream 输出流

Unicode 统一的字符编码标准, 采用双字节对字符进行编码

Information 信息

FileInputStream 文件输入流

FileOutputStream文件输出流

IOException 输入输出异常

fileobject 文件对象

available 可获取的

read 读取

write 写

BufferedReader 缓冲区读取

FileReader 文本文件读取

BufferedWriter 缓冲区输出

FileWriter 文本文件写出

flush 清空

close 关闭

DataInputStream 二进制文件读取

DataOutputStream二进制文件写出

EOF 最后

encoding 编码

Remote 远程

release 释放

第九章：

JBuider Java 集成开发环境（IDE）

Enterprise 企业版

Developer 开发版

Foundation 基础版

Messages 消息格

Structure 结构窗格

Project 工程

Files 文件

Source 源代码

Design 设计

History 历史

Doc 文档

File 文件

Edit 编辑

Search 查找

Refactor 要素

View 视图

Run 运行

Tools 工具

Window 窗口

Help 帮助

Vector 矢量

addElement 添加内容

Project Winzard 工程向导

Step 步骤

Title 标题

Description 描述

Copyright 版权

Company 公司

Aptech Limited Aptech有限公司

author 作者

Back 后退

Finish 完成

version 版本

Debug 调试

New 新建

ErrorInsight 调试

第十章：

JFrame 窗口框架

JPanel 面板

JScrollPane 滚动面板

title 标题

Dimension 尺寸

Component 组件

Swing JAVA轻量级组件

getContentPane 得到内容面板

LayoutManager 布局管理器

setVerticalScrollBarPolicy 设置垂直滚动条策略

AWT（Abstract Window Toolkit）抽象窗口工具包

GUI （Graphical User Interface）图形用户界面

VERTICAL\_SCROLLEARAS\_NEEDED 当内容大大面板出现滚动条

VERTICAL\_SOROLLEARAS\_ALWAYS 显示滚动条

VERTICAL\_SOROLLEARAS\_NEVER 不显示滚动条

JLabel 标签

Icon 图标

image 图象

LEFT 左对齐

RIGHT 右对齐

JTextField 单行文本

getColumns 得到列数

setLayout 设置布局

BorderLayout 边框布局

CENTER 居中对齐

JTextArea 多行文本

setFont 设置字体

setHorizontalAlignment 设置文本水平对齐方式

setDefaultCloseOperation 设置默认的关闭操作

add 增加

JButton 按钮

JCheckBox 复选框

JRadioButton单选按钮

addItem 增加列表项

getItemAt 得到位置的列表项

getItemCount 得到列表项个数

setRolloverIcon 当鼠标经过的图标

setSelectedIcon 当选择按钮的图标

getSelectedItem 得到选择的列表项

getSelectedIndex 得到选择的索引

ActionListener 按钮监听

ActionEvent 按钮事件

actionPerformed 按钮单击方法

(编程词汇)

A.

abstract 抽象的

abstract base class (ABC)抽象基类

abstract class 抽象类

abstraction 抽象、抽象物、抽象性

access 存取、访问

access level访问级别

access function 访问函数

account 账户

action 动作

activate 激活

active 活动的

actual parameter 实参

adapter 适配器

add-in 插件

address 地址

address space 地址空间

address-of operator 取地址操作符

ADL (argument-dependent lookup)

ADO(ActiveX Data Object)ActiveX数据对象

advanced 高级的

aggregation 聚合、聚集

algorithm 算法

alias 别名

align 排列、对齐

allocate 分配、配置

allocator分配器、配置器

angle bracket 尖括号

annotation 注解、评注

API (Application Programming Interface) 应用(程序)编程接口

app domain (application domain)应用域

application 应用、应用程序

application framework 应用程序框架

appearance 外观

append 附加

architecture 架构、体系结构

archive file 归档文件、存档文件

argument引数(传给函式的值)。参见parameter

array 数组

arrow operator 箭头操作符

ASP(Active Server Page)活动服务器页面

ASP.NET worker process ASP.NET工作者进程

assembly 装配件、配件

assembly language 汇编语言

assembly manifest 装配件清单

assert(ion) 断言

assign 赋值

assignment 赋值、分配

assignment operator 赋值操作符

associated 相关的、相关联的

associative container 关联式容器(对应sequential container)

asynchronous 异步的

atomic 原子的

atomic operation 原子操作

attribute 特性、属性

authentication service 验证服务

authorization 授权

audio 音频

A.I. 人工智能

B.

B2B integration B2B整合、B2B集成(business-to-business integration)

background 背景、后台(进程)

backward compatible 向后兼容、向下兼容

backup 备份

backup device备份设备

backup file 备份文件

bandwidth 带宽

base class 基类

base type 基类型

batch 批处理

BCL (base class library)基类库

binary 二进制

binary search 二分查找

binary tree 二叉树

binary function 双参函数

binary large object二进制大对象

binary operator 二元操作符

binding 绑定

bit 位

bitmap 位图

bitwise 按位…

bitwise copy 为单元进行复制；位元逐一复制,按位拷

bitwise operation 按位运算

block 块、区块、语句块

bookkeeping 簿记

boolean 布林值(真假值，true或false)

border 边框

bounds checking 边界检查

boxing 装箱、装箱转换

brace (curly brace) 大括号、花括号

bracket (square brakcet) 中括号、方括号

breakpoint 断点

browser applications 浏览器应用(程序)

browser-accessible application 可经由浏览器访问的应用程序

build 编连(专指编译和连接

built-in 内建、内置

bus 总线

business 业务、商务(看场合)

business Logic 业务逻辑

business rules 业务规则

buttons 按钮

bug 臭虫

by/through 通过

byte 位元组(由8 bits组成)

C.

cache 高速缓存

calendar 日历

call 调用

callback 回调

call-level interface (CLI)调用级接口(CLI)

call operator 调用操作符

candidate key 候选键 (for database)

cascading delete 级联删除 (for database)

cascading update 级联更新 (for database)

casting 转型、造型转换

catalog 目录

chain 链(function calls)

character 字符

character format 字符格式

character set 字符集

CHECK constraints CHECK约束 (for database)

checkpoint 检查点 (for database)

check box 复选框

check button 复选按钮

child class 子类

CIL (common intermediate language)通用中间语言、通用中介语言

class 类

class declaration 类声明

class definition 类定义

class derivation list 类继承列表

class factory 类厂

class hierarchy 类层次结构

class library 类库

class loader 类装载器

class template 类模板

class template partial specializations 类模板部分特化

class template specializations 类模板特化

classification 分类

clause 子句

client application 客户端应用程序

client cursor 客户端游标 (for database)

code page 代码页

cleanup 清理、清除

CLI (Common Language Infrastructure) 通用语言基础设施

client 客户、客户端

client area 客户区

client-server 客户机/服务器、客户端/服务器

clipboard 剪贴板

clone 克隆

CLS (common language specification) 通用语言规范

code access security 代码访问安全

COFF (Common Object File Format) 通用对象文件格式

collection 集合

COM (Component Object Model) 组件对象模型

combo box 组合框

command line 命令行

comment 注释

commit 提交 (for database)

communication 通讯

compatible 兼容

compile time 编译期、编译时

compiler 编译器

component组件

composite index 复合索引、组合索引 (for database)

composite key 复合键、组合键 (for database)

composition 复合、组合

concept 概念

concrete具体的

concrete class 具体类

concurrency 并发、并发机制

constraint 约束 (for database)

configuration 配置、组态

connection 连接 (for database)

connection pooling 连接池

console 控制台

constant 常量

construct 构件、成分、概念、构造（for language）

constructor (ctor) 构造函数、构造器

container 容器

containment包容

context 环境、上下文

control 控件

cookie (不译)

copy 拷贝

CORBA 通用对象请求中介架构(Common Object Request Broker Architecture)

cover 覆盖、涵盖

create/creation 创建、生成

crosstab query 交叉表查询 (for database)

CRTP (curiously recurring template pattern)

CTS (common type system)通用类型系统

cube 多维数据集 (for database)

cursor 光标

cursor 游标 (for database)

custom 定制、自定义

D.

data 数据

data connection 数据连接 (for database)

Data Control Language (DCL) 数据控制语言(DCL) (for database)

Data Definition Language (DDL) 数据定义语言(DDL) (for database)

data dictionary 数据字典 (for database)

data dictionary view 数据字典视图 (for database)

data file 数据文件 (for database)

data integrity 数据完整性 (for database)

data manipulation language (DML)数据操作语言(DML) (for database)

data mart 数据集市 (for database)

data pump 数据抽取 (for database)

data scrubbing 数据清理 (for database)

data source 数据源 (for database)

Data source name (DSN) 数据源名称(DSN) (for database)

data warehouse 数据仓库 (for database)

dataset 数据集 (for database)

database 数据库 (for database)

database catalog 数据库目录 (for database)

database diagram 数据关系图 (for database)

database file 数据库文件 (for database)

database object 数据库对象 (for database)

database owner 数据库所有者 (for database)

database project 数据库工程 (for database)

database role 数据库角色 (for database)

database schema 数据库模式、数据库架构 (for database)

database script 数据库脚本 (for database)

data-bound 数据绑定 (for database)

data-aware control数据感知控件 (for database)

data member 数据成员、成员变量

dataset 数据集 (for database)

data source 数据源 (for database)

data structure数据结构

data table 数据表 (for database)

datagram 数据报文

DBMS (database management system)数据库管理系统 (for database)

DCOM (distributed COM)分布式COM

dead lock 死锁 (for database)

deallocate 归还

debug 调试

debugger 调试器

decay 退化

decision support 决策支持

declaration 声明

declarative referential integrity (DRI)声明引用完整性(DRI) (for database)

deduction 推导

DEFAULT constraint默认约束 (for database)

default database 默认数据库 (for database)

default instance 默认实例 (for database)

default result set 默认结果集 (for database)

default 缺省、默认值

defer 推迟

definition 定义

delegate 委托

delegation 委托

dependent name

deploy 部署

dereference 解引用

dereference operator (提领)运算子

derived class 派生类

design by contract 契约式设计

design pattern 设计模式

destroy 销毁

destructor(dtor)析构函数、析构器

device 设备

DHTML (dynamic HyperText Markup Language)动态超文本标记语言

dialog 对话框

digest 摘要

digital 数字的

DIME (Direct Internet Message Encapsulation)直接Internet消息封装

directive (编译)指示符

directory 目录

dirty pages脏页 (for database)

dirty read 脏读 (for database)

disassembler 反汇编器

DISCO (Discovery of Web Services)Web Services的查找

disk 盘

dispatch 调度、分派、派发（我喜欢”调度”）

DISPID (Dispatch Identifier)分派标识符

distributed computing 分布式计算

distributed query 分布式查询 (for database)

DNA (Distributed interNet Application) 分布式网间应用程序

document 文档

DOM (Document Object Model)文档对象模型

dot operator (圆)点操作符

driver 驱动(程序)

DTD (document type definition) 文档类型定义

double-byte character set (DBCS)双字节字符集(DBCS)

dump 转储

dump file 转储文件

dynamic cursor 动态游标 (for database)

dynamic filter 动态筛选 (for database)

dynamic locking 动态锁定 (for database)

dynamic recovery 动态恢复 (for database)

dynamic snapshot 动态快照 (for database)

dynamic SQL statements 动态SQL语句 (for database)

dynamic assembly 动态装配件、动态配件

dynamic binding 动态绑定

E.

EAI (enterprise application integration)企业应用程序集成(整合)

EBCO (empty base class optimization) 空基类优化（机制）

e-business 电子商务

EDI (Dlectronic Data Interchange)电子数据交换

efficiency 效率

efficient 高效

end-to-end authentication 端对端身份验证

end user 最终用户

engine 引擎

entity 实体

encapsulation 封装

enclosing class 外围类别(与巢状类别 nested class有关)

enum (enumeration) 枚举

enumerators 枚举成员、枚举器

equal 相等

equality 相等性

equality operator 等号操作符

error log 错误日志 (for database)

escape code 转义码

escape character 转义符、转义字符

exclusive lock 排它锁 (for database)

explicit transaction 显式事务 (for database)

evaluate 评估

event 事件

event driven 事件驱动的

event handler 事件处理器

evidence 证据

exception 异常

exception declaration 异常声明

exception handling 异常处理、异常处理机制

exception-safe 异常安全的

exception specification 异常规范

exit 退出

explicit 显式

explicit specialization 显式特化

export 导出

expression 表达式

facility 设施、设备

fat client 胖客户端

feature 特性、特征

fetch 提取

field 字段(java)

field 字段 (for database)

field length 字段长度 (for database)

file 文件

filter 筛选 (for database)

finalization 终结

firewall 防火墙

finalizer 终结器

firmware 固件

flag 标记

flash memory 闪存

flush 刷新

font 字体

foreign key (FK) 外键(FK) (for database)

form 窗体

formal parameter 形参

forward declaration 前置声明

forward-only 只向前的

forward-only cursor 只向前游标 (for database)

fragmentation 碎片 (for database)

framework 框架

full specialization 完全特化

function 函数

function call operator (即operator ()) 函数调用操作符

function object 函数对象

function overloaded resolution函数重载决议

functionality 功能

function template函数模板

functor 仿函数

GAC (global assembly cache) 全局装配件缓存、全局配件缓存

GC (Garbage collection) 垃圾回收(机制)、垃圾收集(机制)

game 游戏

generate 生成

generic 泛化的、一般化的、通用的

generic algorithm通用算法

genericity 泛型

getter (相对于 setter)取值函数

global 全局的

global object 全局对象

global scope resolution operator 全局范围解析操作符

grant 授权 (for database)

granularity 粒度

group 组、群

group box 分组框

GUI 图形界面

GUID (Globally Unique Identifier) 全球唯一标识符

hand shaking 握手

handle 句柄

handler 处理器

hard-coded 硬编码的

hard-copy 截屏图

hard disk 硬盘

hardware 硬件

hash table 散列表、哈希表

header file头文件

heap 堆

help file 帮助文件

hierarchy 层次结构、继承体系

hierarchical data 阶层式数据、层次式数据

hook 钩子

Host (application)宿主(应用程序)

hot key 热键

hyperlink 超链接

HTML (HyperText Markup Language) 超文本标记语言

HTTP pipeline HTTP管道

HTTP (HyperText Transfer Protocol) 超文本传输协议

I.

icon 图标

IDE (Integrated Development Environment)集成开发环境

IDL (Interface Definition Language) 接口定义语言

identifier 标识符

idle time 空闲时间

if and only if当且仅当

IL (Intermediate Language) 中间语言、中介语言

image 图象

IME 输入法

immediate base 直接基类

immediate derived 直接派生类

immediate updating 即时更新 (for database)

implicit transaction隐式事务 (for database)

incremental update 增量更新 (for database)

index 索引 (for database)

implement 实现

implementation 实现、实现品

implicit 隐式

import 导入

increment operator 增加操作符

infinite loop 无限循环

infinite recursive 无限递归

information 信息

infrastructure 基础设施

inheritance 继承、继承机制

inline 内联

inline expansion 内联展开

initialization 初始化

initialization list 初始化列表、初始值列表

initialize 初始化

inner join 内联接 (for database)

in-place active 现场激活

instance 实例

instantiated 具现化、实体化(常应用于template)

instantiation 具现体、具现化实体(常应用于template)

integrate 集成、整合

integrity 完整性、一致性

integrity constraint完整性约束 (for database)

interprocess communication (IPC)进程间通讯(IPC)

interacts 交互

interface 接口

for GUI 界面

interoperability 互操作性、互操作能力

interpreter 解释器

introspection 自省

invariants 不变性

invoke 调用

isolation level 隔离级别 (for database)

iterate 迭代

iterative 反复的、迭代的

iterator 迭代器

iteration 迭代(回圈每次轮回称为一个iteration)

item 项、条款、项目

J.

JIT compilation JIT编译即时编译

K.

key 键 (for database)

key column 键列 (for database)

L.

laser 激光

late binding 迟绑定

left outer join 左向外联接 (for database)

level 阶、层例

high level 高阶、高层

library 库

lifetime 生命期、寿命

link 连接、链接

linkage 连接、链接

linker 连接器、链接器

literal constant 字面常数

list 列表、表、链表

list box 列表框

livelock 活锁 (for database)

load 装载、加载

load balancing 负载平衡

loader 装载器、载入器

local 局部的

local object 局部对象

lock 锁

log 日志

login 登录

login security mode登录安全模式 (for database)

lookup table 查找表 (for database)

loop 循环

loose coupling 松散耦合

lvalue 左值

M.

machine code 机器码、机器代码

macro 宏

maintain 维护

managed code 受控代码、托管代码

Managed Extensions 受控扩充件、托管扩展

managed object 受控对象、托管对象

mangled name

manifest 清单

manipulator 操纵器(iostream预先定义的一种东西)

many-to-many relationship 多对多关系 (for database)

many-to-one relationship 多对一关系 (for database)

marshal 列集

member 成员

member access operator 成员取用运算子(有dot和arrow两种)

member function 成员函数

member initialization list成员初始值列表

memberwise 以member为单元…、members 逐一…

memberwise copy

memory 内存

memory leak 内存泄漏

menu 菜单

message 消息

message based 基于消息的

message loop 消息环

message queuing消息队列

metadata 元数据

metaprogramming元编程

method 方法

micro 微

middleware 中间件

middle tier 中间层

modeling 建模

modeling language 建模语言

modifier 修饰字、修饰符

modem 调制解调器

module 模块

most derived class最底层的派生类

mouse 鼠标

mutable 可变的

mutex 互斥元、互斥体

multidimensional OLAP (MOLAP) 多维OLAP(MOLAP) (for database)

multithreaded server application 多线程服务器应用程序

multiuser 多用户

multi-tasking 多任务

multi-thread 多线程

multicast delegate 组播委托、多点委托

N.

named parameter 命名参数

named pipe 命名管道

namespace 名字空间、命名空间

native 原生的、本地的

native code 本地码、本机码

Native Image Generator (NGEN)本地映像生成器

nested class 嵌套类

nested query 嵌套查询 (for database)

nested table 嵌套表 (for database)

network 网络

network card 网卡

nondependent name

O.

object 对象

object based 基于对象的

object file 目标文件

object model 对象模型

object oriented 面向对象的

object pooling 对象池化

ODBC data source ODBC数据源 (for database)

ODBC driver ODBC驱动程序 (for database)

ODR (one-definition rule)

OLE Automation objects OLE自动化对象 (for database)

OLE Automation server OLE自动化服务器 (for database)

OLE DB consumer OLE DB使用者 (for database)

OLE DB for OLAP 用于OLAP的OLE DB (for database)

OLE DB provider OLE DB提供者 (for database)

one-to-many relationship 一对多关系 (for database)

one-to-one relationship 一对一关系 (for database)

online analytical processing (OLAP) 联机分析处理(OLAP) (for database)

online redo log 联机重做日志 (for database)

online transaction processing (OLTP) 联机事务处理(OLTP) (for database)

Open Data Services (ODS) 开放式数据服务(ODS) (for database)

Open Database Connectivity (ODBC) 开放式数据库连接(ODBC) (for database)

operand 操作数

operating system (OS) 操作系统

operation 操作

operator 操作符、运算符

option 选项

optimizer 优化器

outer join 外联接 (for database)

overflow 上限溢位(相对于underflow)

overhead 额外开销

overload 重载

overload resolution 重载决议

overloaded function 重载的函数

overloaded operator 被重载的操作符

override 覆写、重载、重新定义

P.

package 包

packaging 打包

palette 调色板

parallel 并行

parameter 参数、形式参数、形参

parameter list 参数列表

parameterize 参数化

parent class 父类

parentheses 圆括弧、圆括号

parse 解析

parser 解析器

part 零件、部件

partial specialization 局部特化

pass by address 传址(函式引数的传递方式)(非正式用语)

pass by reference 传地址、按引用传递

pass by value 按值传递

pattern 模式

PDA (personal digital assistant)个人数字助理

PE (Portable Executable) file 可移植可执行文件

performance 性能

persistence 持久性

PInvoke (platform invoke service) 平台调用服务

pixel 像素

placement delete

placement new

placeholder 占位符

platform 平台

POD (plain old data (type))

POI (point of instantiation)

pointer 指针

poll 轮询

pooling 池化

polymorphism 多态

pop up 弹出式

port 端口

postfix 后缀

precedence 优先序(通常用于运算子的优先执行次序)

prefix 前缀

preprocessor 预处理器

primary key (PK)主键(PK) (for database)

primary table 主表 (for database)

primary template原始模板

primitive type 原始类型

print 打印

printer 打印机

procedure 过程

procedural 过程式的、过程化的

process 进程

profile 评测

profiler 效能(性能)评测器

program 程序

programmer 程序员

programming编程、程序设计

progress bar 进度指示器

project 项目、工程

property 属性

protocol 协议

pseudo code伪码

Q.

qualified 经过资格修饰(例如加上scope运算子)

qualified name

qualifier 修饰符

quality 质量

queue 队列

R.

race condition 竞争条件（多线程环境常用语）

radian 弧度

radio button 单选按钮

raise 引发(常用来表示发出一个exception)

random number 随机数

range 范围、区间

rank 等级

raw 未经处理的

readOnly只读

record 记录 (for database)

recordset 记录集 (for database

recursive 递归

re-direction 重定向

refactoring 重构

refer 引用、参考

reference 引用、参考

reference counting引用计数

referential integrity (RI)引用完整性(RI) (for database)

register 寄存器

reflection 反射

refresh data 刷新数据 (for database)

regular expression 正则表达式

relational database 关系数据库

remote 远程

remote request 远程请求

represent 表述，表现

resolve 解析、决议

resolution 解析过程

result set 结果集 (for database)

retrieve data 检索数据

return 返回

return type 返回类型

return value 返回值

right outer join 右向外联接 (for database)

revoke 撤销

robust 健壮

robustness 健壮性

roll back 回滚 (for database)

roll forward 前滚 (for database)

routine 例程

row 行 (for database)

row lock 行锁 (for database)

rowset 行集 (for database)

RPC (remote procedure call)RPC(远程过程调用)

runtime 执行期、运行期、执行时、运行时

rvalue 右值

S.

save 保存

savepoint 保存点 (for database)

SAX (Simple API for XML)

scalable 可伸缩的、可扩展的

schedule 调度

scheduler 调度程序

schema 模式、纲目结构

scroll bar滚动条

scope 作用域、生存空间

scope operator 生存空间操作符

scope resolution operator 生存空间解析操作符

screen 屏幕

SDK (Software Development Kit)软件开发包

sealed class 密封类

search 查找

semantics 语义

semaphore 信号量

sequential container序列式容器

server 服务器、服务端

serial 串行

serialization/serialize 序列化

server cursor服务端游标、服务器游标 (for database)

session 会话 (for database)

setter 设值函数

shared lock 共享锁 (for database)

sibling 同级

side effect 副作用

signature 签名

single-threaded 单线程

slider滑块

slot 槽

smart pointer 智能指针

SMTP (Simple Mail Transfer Protocol) 简单邮件传输协议

snapshot 截屏图

snapshot 快照 (for database)

specialization 特化

specification 规范、规格

splitter 切分窗口

SOAP (simple object access protocol) 简单对象访问协议

software 软件

source code 源码、源代码

SQL (Structured Query Language) 结构化查询语言 (for database)

stack 栈、堆栈

stack unwinding 叠辗转开解(此词用于exception主题)

standard library 标准库

standard template library 标准模板库

stateless 无状态的

statement 语句、声明

static cursor 静态游标 (for database)

static SQL statements 静态SQL语句 (for database)

stored procedure 存储过程 (for database)

status bar 状态条

stream 流

string 字符串

stub 存根

subobject子对象

subquery 子查询 (for database)

subroutine 子例程

subscript operator 下标操作符

subset 子集

subtype 子类型

support 支持

suspend 挂起

symbol 记号

syntax 语法

system databases 系统数据库 (for database)

system tables 系统表 (for database)

T.

table 表 (for database)

table lock 表锁 (for database)

table-level constraint 表级约束 (for database)

tape backup 磁带备份 (for database)

target 标的,目标

task switch 工作切换

TCP (Transport Control Protocol) 传输控制协议

template 模板

template-id

template argument deduction 模板参数推导

template explicit specialization 模板显式特化

template parameter 模板参数

template template parameter

temporary object 临时对象

temporary table 临时表 (for database)

text 文本

text file 文本文件

thin client 瘦客户端

third-party 第三方

thread 线程

thread-safe 线程安全的

throw 抛出、引发(常指发出一个exception)

token 符号、标记、令牌（看场合）

trace 跟踪

transaction 事务 (for database)

transaction log 事务日志 (for database)

transaction rollback 事务回滚 (for database)

transactional replication 事务复制 (for database)

translation unit 翻译单元

traverse 遍历

trigger 触发器 (for database)

two-phase commit 两阶段提交 (for database)

tuple

two-phase lookup 两阶段查找

type 类型

U.

UDDI(Universary Description, Discovery and Integration)统一描述、查询与集成

UML (unified modeling language)统一建模语言

unary function 单参函数

unary operator 一元操作符

unboxing 拆箱、拆箱转换

underflow 下限溢位(相对于overflow)

Union query 联合查询 (for database)

UNIQUE constraints UNIQUE约束 (for database)

unique index 唯一索引 (for database)

unmanaged code 非受控代码、非托管代码

unmarshal 散集

unqualified 未经限定的、未经修饰的

URI (Uniform Resource identifier) 统一资源标识符

URL (Uniform Resource Locator) 统一资源定位器

user 用户

user interface 用户界面

V.

value types 值类型

variable 变量

vector 向量(一种容器，有点类似array)

viable 可行的

video 视频

view 视图

VEE (Virtual Execution Engine)虚拟执行引擎

vendor 厂商

view 视图 (for database)

virtual function 虚函数

virtual machine 虚拟机

virtual memory 虚拟内存

vowel 元音字母

W.

Web Services web服务

WHERE clause WHERE子句 (for database)

wildcard characters 通配符字符 (for database)

wildcard search 通配符搜索 (for database)

window 窗口

window function 窗口函数

window procedure 窗口过程

Windows authentication Windows身份验证

wizard 向导

word 单词

word processor 字处理器

wrapper 包装、包装器

write enable 写启用 (for database)

write-ahead log 预写日志 (for database)

write-only 只写

WSDL (Web Service Description Language)Web Service描述语言

X.

XML Message Interface (XMI) XML消息接口

XML (eXtensible Markup Language) 可扩展标记语言

XSD (XML Schema Definition) XML模式定义语言

XSL (eXtensible Stylesheet Language) 可扩展样式表语言

XSLT (eXtensible Stylesheet Language Transformation)可扩展样式表语言转换

xxx based 基于xxx的

xxx oriented 面向xxx

)

A.

argument 参量

abstract 抽象

ascent 提升

already 已经

AWT(Abstract Window Toolkit)抽象窗口工具

API(Application Programming Interface)应用程序接口

B.

byte 字节

Boolean 布尔

banana香蕉

base 基础

buffer缓冲器

button 按钮

break 中断

body 身体

C.

color 颜色

class 类

count 计数

client 客户

code 代码

calculation 计算

cell 单元

circle圆

capital首都

catch捕获

check 检查

container容器

component 组件

command 命令

cube立方,三次方

char(=character)字符

cancel取消

case 情况

choice选择

click单击

center 中心

compile编译

clone克隆，复制

continue 继续

create建立

D.

draw 绘图

data数据

demo 示例

DLL(Dynamic Link Library)动态链接库

document 文档

descent 下降

division 分裂，除法

define定义，说明

display显示

E.

error 错误

extends 扩展

executed 执行

event 事件

enter 输入，回车键

exception 异常

except 除外

employee 雇员

environment 环境

east 东方

equal 相等

echo 重复

F.

false 假的

float 单精度型

fruit 水果

file 文件

find 发现

found 发现

field 域，属性

final 终结的

friend 朋友

fill 填充

focus 焦点

font 字体

factorial 阶乘

G.

graphic 图像

grid 方格

GUI图形化用户接口

get 得到

H.

host 主机

height 高度

I.

init(=initialize)初始化

input 输入

implement 实现

instance 实例

io(=input/output)输出输入

interrupted 中断

int(=integer)整型

item元素

interface 接口

inner 内部的

import 导入

index 索引

image 图像

J.

Java 爪哇

JDK(Java Development Kit) Java开发工具

JSP(Java Server Page) Java服务页

JVM(Java Virtual Machine) Java虚拟机

K.

Kit 工具

L.

language 语言

loop 循环

long 长整型

label 标签

layout 布局

list 列表

listener 收听者

M.

move 移动

menu 菜单

mode 模式

method 方法

metric 米的，公尺

motion 运动

manager 经理

main 主要的

msg(=message) 消息

N.

new 新的

number 数字

north 北方

null 空的

native 本地的

O.

override 重写

overload 重载

orange 橘子

output 输出

object 对象

out 外部的

oval 椭圆

P.

public 公共的

protected 保护的

private 私有的

property 属性

point 点

price 价格

problem 问题

package 打包，包裹

print 打印

path 路径

program 程序

prompt 提示

parse 分析

press 按，压

panel 面板

paint 画

R.

return 返回

radius 半径

round 环绕

release 释放

rect(=rectangle)长方形

radio 无线电

resolve 解析

S.

short 短整型

south 南方的

string 字符串

static 静态的

system 系统

seed 种子

seasonal 季节的

set 设置

super 超级

square 平方，二次方

sub 替代的

screen 屏幕

sound声音

state 状态

salary 薪水

sleep 睡觉

size 大小，尺寸

start 开始

sort 排序

status 状态

synchronize 同步发生

switch 开关

stream 流

symbol 符号

T.

true 真的

title 标题

type 类型

temp(=temporary)暂时的

throw 扔

thread 线程

temperate 温度

tool 工具

try 试图

U.

undefined 未定义

UI(User Interface) 用户接口

update 更新

URL(Uniform Resource Locator) 统一资源定位器

V.

volatile 挥发性

visible 不可见的

virtual 虚拟的

variable 变量

value 数值

void 无返回值的

volume 列

viewer 观察者

vector 矢量

constructor 建构式

declaration 宣告式

definition 定义式

destructor 解构式

expression 算式（运算式）

function 函式

pattern 范式、模式、样式

program 程式

signature 标记式（签名式/署名式）

assembly （装）配件

component 组件

construct 构件

control 控件

event 事件

hardware 硬件

object 物件

part 零件、部件

singleton 单件

software 软件

work 工件、机件

adapter 配接器

allocator 配置器

compiler 编译器

container 容器

iterator 迭代器

linker 连结器

listener 监听器

interpreter 直译器

translator 转译器/翻译器

class 类别

type 型别

generalized 泛化

specialized 特化

overloaded 多载化（重载）

polymorphism 多态

genericity 泛型

process 行程/进程（大陆用语）

thread 绪程/线程（大陆用语）

programming 编程

●英中繁简编程术语对照

define 定义预定义

abstract 抽象的抽象的

abstraction 抽象体、抽象物、抽象性抽象体、抽象物、抽象性

access 存取、取用存取、访问

access level 存取级别访问级别

access function 存取函式访问函数

activate 活化激活

active 作用中的

adapter 配接器适配器

address 位址地址

address space 位址空间，定址空间

address-of operator 取址运算子取地址操作符

aggregation 聚合

algorithm 演算法算法

allocate 配置分配

allocator （空间）配置器分配器

application 应用程式应用、应用程序

application framework 应用程式框架、应用框架应用程序框架

architecture 架构、系统架构体系结构

argument 引数（传给函式的值）。参见 parameter 参数、实质参数、实参、自变量

array 阵列数组

arrow operator arrow（箭头）运算子箭头操作符

assembly 装配件

assembly language 组合语言汇编语言

assert(ion) 断言

assign 指派、指定、设值、赋值赋值

assignment 指派、指定赋值、分配

assignment operator 指派（赋值）运算子 = 赋值操作符

associated 相应的、相关的相关的、关联、相应的

associative container 关联式容器（对应 sequential container）关联式容器

atomic 不可分割的原子的

attribute属性属性、特性

audio 音讯音频

A.I. 人工智慧人工智能

background背景背景（用於图形着色）

後台（用於行程）

backward compatible 回溯相容向下兼容

bandwidth 频宽带宽

base class 基础类别基类

base type 基础型别 (等同於 base class)

batch 批次（意思是整批作业）批处理

benefit 利益收益

best viable function 最佳可行函式最佳可行函式

（从 viable functions 中挑出的最佳吻合者）

binary search 二分搜寻法二分查找

binary tree 二元树二叉树

binary function 二元函式双叁函数

binary operator 二元运算子二元操作符

binding 系结绑定

bit 位元位

bitwise 以 bit 为单元逐一

bitwise copy 以 bit 为单元进行复制；位元逐一复制位拷贝

block 区块,区段块、区块、语句块

boolean 布林值（真假值，true 或 false）布尔值

border 边框、框线边框

brace(curly brace) 大括弧、大括号花括弧、花括号

bracket(square brakcet) 中括弧、中括号方括弧、方括号

breakpoint 中断点断点

build 建造、构筑、建置（MS 用语）

build-in 内建内置

bus 汇流排总线

business 商务,业务业务

buttons按钮按钮

byte 位元组（由 8 bits 组成）字节

cache 快取高速缓存

call 呼叫、叫用调用

callback 回呼回调

call operator call（函式呼叫）运算子 () 调用操作符

（同 function call operator）

candidate function 候选函式候选函数

（在函式多载决议程序中出现的候选函式）

chain 串链（例 chain of function calls）链

character字符

check box 核取方块 (i.e. check button) 复选框

checked exception 可控式异常(Java)

check button 方钮 (i.e. check box) 复选按钮

child class 子类别（或称为derived class, subtype）子类

class 类别类

class body 类别本体

class declaration 类别宣告、类别宣告式类声明

class definition 类别定义、类别定义式类定义

class derivation list 类别衍化列类继承列表

class head 类别表头

class hierarchy 类别继承体系, 类别阶层类层次体系

class library 类别程式库、类别库类库

class template 类别模板、类别范本类模板

class template partial specializations

类别模板偏特化类模板部分特化

class template specializations

类别模板特化类模板特化

cleanup 清理、善後清理、清除

client 客端、客户端、客户客户

client-server 主从架构客户/服务器

clipboard 剪贴簿剪贴板

cursor游标

record记录类型

rollback回滚

declare声明

begin开始

end结束

sqrt开平方根

open打开

fetch提取

close关闭

function函数

exception异常

replace更换

procedure过程

trigger触发器

commit提交

object Relation Mapping 对象关系映射

vector列表

load装载

connection链接

Driver驱动

Manager管理

DriverManager驱动管理

localhost本地主机

Resultset结果集

register注册

constraint约束

view视图

index索引

procedure存储过程

setsavepoint设置保存点

previous之前的

first首个

last最后

next之后的

absolute绝对的

properties性能

Instance实例

newInstance创建实例

system系统

user用户名

password密码

Dimensions 维度

Component type 多维数组的每一行（可以理解成二维数组的第一个元素是一维数组，这个一维数组的类型就是component type）

Invoke 调用

Specified 指定的

Belong to 属于

Automatically自动地

Loaded 被加载

Loader加载器

Cast 转换

Associated with 和什么关联

Compare…against 和…比较

Declaration order声明序列

Modifier 修饰语，修饰语

Underlying下一层的

Dynamic动态的

a widening conversion一个扩展转换

anonymous class匿名类

canonical按照规定的，权威的

assertion status 声明状态或者使用状态

be assigned to 指定给….

Fully qualified name 完全限定的名字

The Instance of the class represented by this Class object 由这个Class对象表示这个类的实例.

Create a new Instance of the class represented by this Class object 生成一个Class对象所代表的这个类的新的实例

The class is instantiated as if by a new expression with an empty argument list 这个类被实例化好像是通过一个表达式并且是一个空的参数列表

Returns a Constructor object that reflects the specified public constructor of the class representedby this Class object返回一个Constructor 对象，这个对象会反射出指定的public的构造方法由这个类所代表的类的public构造方法。

Returns a Method object that reflects the specified public member method of the class or interface represented by this Class object返回一个Method对象，这个Method会反射出指定的类或者接口由这个Class对象所代表的类或者接口当中指定的public方法。

The accessibleObject class is the base class for Field,Method and Constructor objects,It provides the ability to flag a reflected object as suppressing default Java language access control checks when it is used这个AccessibleObject类是Filed,Method,Constructor的父类，它提供了这种能力去给一个反射的对象去压制默认的java语言访问控制检查。

Set the accessible flag for this object to the indicated boolean value. A value of true indicates that the reflected object should suppress Java language access checking when it is used. A value of false indicates that the reflected object should enforce Java language access checks.用这个表示布尔值的值设置这个对象的访问的标志，如果为真值，这个反射的对象应当压制java语言的访问检查当它被使用的时候，如果为假，这个反射对象应当强制java的访问检查。

Java.lang.Class<T>– à Type getGenericSuperclass()

Returns the Type representing the direct superclass of the entity (class, interface, primitive type or void) represented by this Class .

org.w3c.dom- à getDocumentElement()

This is a convenience attribute that allows direct access to the child node that is the document element ofthe document.这是一个方便的特性，即允许直接访问文档的元素子节点

## Java关键字及其作用

一、 总览:

访问控制

private protected public

类,方法和变量修饰符

abstract class extends final implements interface native new

static strictfp synchronized transient volatile

程序控制

break continue return do while if else for instanceof switch

case default

异常处理

try cathc throw throws

包相关

import package

基本类型

boolean byte char double float int long short null true false

变量引用

super this void

保留字

goto const

二、 详细解释

1. 访问控制

1) private 私有的

private 关键字是访问控制修饰符，可以应用于类、方法或字段（在类中声明的变量）。 只能在声明 private（内部）类、方法或字段的类中引用这些类、方法或字段。在类的外部或者对于子类而言，它们是不可见的。 所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

2) protected 受保护的

protected 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符。可以在声明 protected 类、方法或字段的类、同一个包中的其他任何类以及任何子类（无论子类是在哪个包中声明的）中引用这些类、方法或字段。所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

3) public 公共的

public 关键字是可以应用于类、方法或字段（在类中声明的变量）的访问控制修饰符。 可能只会在其他任何类或包中引用 public 类、方法或字段。所有类成员的默认访问范围都是 package 访问，也就是说，除非存在特定的访问控制修饰符，否则，可以从同一个包中的任何类访问类成员。

2. 类、方法和变量修饰符

1) abstract 声明抽象

abstract关键字可以修改类或方法。abstract类可以扩展（增加子类），但不能直接实例化。abstract方法不在声明它的类中实现，但必须在某个子类中重写。采用 abstract方法的类本来就是抽象类，并且必须声明为abstract。

2) class类

class 关键字用来声明新的 Java 类，该类是相关变量和/或方法的集合。类是面向对象的程序设计方法的基本构造单位。类通常代表某种实际实体，如几何形状或人。类是对象的模板。每个对象都是类的一个实例。要使用类，通常使用 new 操作符将类的对象实例化，然后调用类的方法来访问类的功能。

3) extends 继承、扩展

extends 关键字用在 class 或 interface 声明中，用于指示所声明的类或接口是其名称后跟有 extends 关键字的类或接口的子类。子类继承父类的所有 public 和 protected 变量和方法。 子类可以重写父类的任何非 final 方法。一个类只能扩展一个其他类。

4) final 最终、不可改变

final 关键字可以应用于类，以指示不能扩展该类（不能有子类）。final 关键字可以应用于方法，以指示在子类中不能重写此方法。一个类不能同时是 abstract 又是 final。abstract 意味着必须扩展类，final 意味着不能扩展类。一个方法不能同时是 abstract 又是 final。abstract 意味着必须重写方法，final 意味着不能重写方法。

5) implements实现

implements 关键字在 class 声明中使用，以指示所声明的类提供了在 implements 关键字后面的名称所指定的接口中所声明的所有方法的实现。类必须提供在接口中所声明的所有方法的实现。一个类可以实现多个接口。

6) interface 接口

interface 关键字用来声明新的 Java 接口，接口是方法的集合。

接口是 Java 语言的一项强大功能。任何类都可声明它实现一个或多个接口，这意味着它实现了在这些接口中所定义的所有方法。

实现了接口的任何类都必须提供在该接口中的所有方法的实现。一个类可以实现多个接口。

7) native 本地

native 关键字可以应用于方法，以指示该方法是用 Java 以外的语言实现的。

8) new 新,创建

new 关键字用于创建类的新实例。

new 关键字后面的参数必须是类名，并且类名的后面必须是一组构造方法参数（必须带括号）。

参数集合必须与类的构造方法的签名匹配。

= 左侧的变量的类型必须与要实例化的类或接口具有赋值兼容关系。

9) static 静态

static 关键字可以应用于内部类（在另一个类中定义的类）、方法或字段（类的成员变量）。

通常，static 关键字意味着应用它的实体在声明该实体的类的任何特定实例外部可用。

static（内部）类可以被其他类实例化和引用（即使它是顶级类）。在上面的示例中，另一个类中的代码可以实例化 MyStaticClass 类，方法是用包含它的类名来限定其名称，如 MyClass.MyStaticClass。

static 字段（类的成员变量）在类的所有实例中只存在一次。

可以从类的外部调用 static 方法，而不用首先实例化该类。这样的引用始终包括类名作为方法调用的限定符。

模式：public final static <type> varName = <value>; 通常用于声明可以在类的外部使用的类常量。在引用这样的类常量时需要用类名加以限定。在上面的示例中，另一个类可以用 MyClass.MAX\_OBJECTS 形式来引用 MAX\_OBJECTS 常量。

10) strictfp 严格,精准

strictfp的意思是FP-strict，也就是说精确浮点的意思。在Java虚拟机进行浮点运算时，如果没有指定strictfp关键字时，Java的编译器以及运行环境在对浮点运算的表达式是采取一种近似于我行我素的行为来完成这些操作，以致于得到的结果往往无法令人满意。而一旦使用了strictfp来声明一个类、接口或者方法时，那么所声明的范围内Java的编译器以及运行环境会完全依照浮点规范IEEE-754来执行。因此如果想让浮点运算更加精确，而且不会因为不同的硬件平台所执行的结果不一致的话，那就请用关键字strictfp。

可以将一个类、接口以及方法声明为strictfp，但是不允许对接口中的方法以及构造函数声明strictfp关键字

11) synchronized线程、同步

synchronized 关键字可以应用于方法或语句块，并为一次只应由一个线程执行的关键代码段提供保护。

synchronized 关键字可防止代码的关键代码段一次被多个线程执行。

如果应用于静态方法，那么，当该方法一次由一个线程执行时，整个类将被锁定。

如果应用于实例方法，那么，当该方法一次由一个线程访问时，该实例将被锁定。

如果应用于对象或数组，当关联的代码块一次由一个线程执行时，对象或数组将被锁定。

12) transient 短暂

transient 关键字可以应用于类的成员变量，以便指出该成员变量不应在包含它的类实例已序列化时被序列化。

当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

Java的serialization提供了一种持久化对象实例的机制。当持久化对象时，可能有一个特殊的对象数据成员，我们不想用serialization机制来保存它。为了在一个特定对象的一个域上关闭serialization，可以在这个域前加上关键字transient。

transient是Java语言的关键字，用来表示一个域不是该对象串行化的一部分。当一个对象被串行化的时候，transient型变量的值不包括在串行化的表示中，然而非transient型的变量是被包括进去的。

13) volatile 易失

volatile 关键字用于表示可以被多个线程异步修改的成员变量。

注意：volatile 关键字在许多 Java 虚拟机中都没有实现。 volatile 的目标用途是为了确保所有线程所看到的指定变量的值都是相同的。

Java 语言中的 volatile 变量可以被看作是一种 “程度较轻的 synchronized”；与 synchronized 块相比，volatile 变量所需的编码较少，并且运行时开销也较少，但是它所能实现的功能也仅是 synchronized 的一部分。

3. 程序控制语句

1) break 跳出，中断

break 关键字用于提前退出 for、while 或 do 循环，或者在 switch 语句中用来结束 case 块。

break 总是退出最深层的 while、for、do 或 switch 语句。

2) continue 继续

continue 关键字用来跳转到 for、while 或 do 循环的下一个迭代。

continue 总是跳到最深层 while、for 或 do 语句的下一个迭代。

3) return 返回

return 关键字会导致方法返回到调用它的方法，从而传递与返回方法的返回类型匹配的值。

如果方法具有非 void 的返回类型，return 语句必须具有相同或兼容类型的参数。

返回值两侧的括号是可选的。

4) do 运行

do 关键字用于指定一个在每次迭代结束时检查其条件的循环。

do 循环体至少执行一次。

条件表达式后面必须有分号。

5) while 循环

while 关键字用于指定一个只要条件为真就会重复的循环。

6) if 如果

if 关键字指示有条件地执行代码块。条件的计算结果必须是布尔值。

if 语句可以有可选的 else 子句，该子句包含条件为 false 时将执行的代码。

包含 boolean 操作数的表达式只能包含 boolean 操作数。

7) else 否则

else 关键字总是在 if-else 语句中与 if 关键字结合使用。else 子句是可选的，如果 if 条件为 false，则执行该子句。

8) for 循环

for 关键字用于指定一个在每次迭代结束前检查其条件的循环。

for 语句的形式为 for(initialize; condition; increment)

控件流进入 for 语句时，将执行一次 initialize 语句。

每次执行循环体之前将计算 condition 的结果。如果 condition 为 true，则执行循环体。

每次执行循环体之后，在计算下一个迭代的 condition 之前，将执行 increment 语句。

9) instanceof 实例

instanceof 关键字用来确定对象所属的类。

10) switch 观察

switch 语句用于基于某个表达式选择执行多个代码块中的某一个。

switch 条件的计算结果必须等于 byte、char、short 或 int。

case 块没有隐式结束点。break 语句通常在每个 case 块末尾使用，用于退出 switch 语句。

如果没有 break 语句，执行流将进入所有后面的 case 和/或 default 块。

11) case 返回观察里的结果

case 用来标记 switch 语句中的每个分支。

case 块没有隐式结束点。break 语句通常在每个 case 块末尾使用，用于退出 switch 语句。

如果没有 break 语句，执行流将进入所有后面的 case 和/或 default 块。

12) default 默认

default 关键字用来标记 switch 语句中的默认分支。

default 块没有隐式结束点。break 语句通常在每个 case 或 default 块的末尾使用，以便在完成块时退出 switch 语句。

如果没有 default 语句，其参数与任何 case 块都不匹配的 switch 语句将不执行任何操作。

4. 错误处理

1) try 捕获异常

try 关键字用于包含可能引发异常的语句块。

每个 try 块都必须至少有一个 catch 或 finally 子句。

如果某个特定异常类未被任何 catch 子句处理，该异常将沿着调用栈递归地传播到下一个封闭 try 块。如果任何封闭 try 块都未捕获到异常，Java 解释器将退出，并显示错误消息和堆栈跟踪信息。

2) catch 处理异常

catch 关键字用来在 try-catch 或 try-catch-finally 语句中定义异常处理块。

开始和结束标记 { 和 } 是 catch 子句语法的一部分，即使该子句只包含一个语句，也不能省略这两个标记。

每个 try 块都必须至少有一个 catch 或 finally 子句。

如果某个特定异常类未被任何 catch 子句处理，该异常将沿着调用栈递归地传播到下一个封闭 try 块。如果任何封闭 try 块都未捕获到异常，Java 解释器将退出，并显示错误消息和堆栈跟踪信息。

3) throw 抛出一个异常对象

throw 关键字用于引发异常。

throw 语句将 java.lang.Throwable 作为参数。Throwable 在调用栈中向上传播，直到被适当的 catch 块捕获。

引发非 RuntimeException 异常的任何方法还必须在方法声明中使用 throws 修饰符来声明它引发的异常。

4) throws 声明一个异常可能被抛出

throws 关键字可以应用于方法，以便指出方法引发了特定类型的异常。

throws 关键字将逗号分隔的 java.lang.Throwables 列表作为参数。

引发非 RuntimeException 异常的任何方法还必须在方法声明中使用 throws 修饰符来声明它引发的异常。

要在 try-catch 块中包含带 throws 子句的方法的调用，必须提供该方法的调用者。

5. 包相关

1) import 引入

import 关键字使一个包中的一个或所有类在当前 Java 源文件中可见。可以不使用完全限定的类名来引用导入的类。

当多个包包含同名的类时，许多 Java 程序员只使用特定的 import 语句（没有“\*”）来避免不确定性。

2) package 包

package 关键字指定在 Java 源文件中声明的类所驻留的 Java 包。

package 语句（如果出现）必须是 Java 源文件中的第一个非注释性文本。

例:java.lang.Object。

如果 Java 源文件不包含 package 语句，在该文件中定义的类将位于“默认包”中。请注意，不能从非默认包中的类引用默认包中的类。

6. 基本类型

1) boolean 布尔型

boolean 是 Java 原始类型。boolean 变量的值可以是 true 或 false。

boolean 变量只能以 true 或 false 作为值。boolean 不能与数字类型相互转换。

包含 boolean 操作数的表达式只能包含 boolean 操作数。

Boolean 类是 boolean 原始类型的包装对象类。

2) byte 字节型

byte 是 Java 原始类型。byte 可存储在 [-128, 127] 范围以内的整数值。

Byte 类是 byte 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

3) char 字符型

char 是 Java 原始类型。char 变量可以存储一个 Unicode 字符。

可以使用下列 char 常量：\b - 空格, \f - 换页, \n - 换行, \r - 回车, \t - 水平制表符, \' - 单引号, \" - 双引号, \\ - 反斜杠, \xxx - 采用 xxx 编码的 Latin-1 字符。\x 和 \xx 均为合法形式，但可能引起混淆。 \uxxxx - 采用十六进制编码 xxxx 的 Unicode 字符。

Character 类包含一些可用来处理 char 变量的 static 方法，这些方法包括 isDigit()、isLetter()、isWhitespace() 和 toUpperCase()。

char 值没有符号。

4) double 双精度

double 是 Java 原始类型。double 变量可以存储双精度浮点值。

由于浮点数据类型是实际数值的近似值，因此，一般不要对浮点数值进行是否相等的比较。

Java 浮点数值可代表无穷大和 NaN（非数值）。Double 包装对象类用来定义常量 MIN\_VALUE、MAX\_VALUE、NEGATIVE\_INFINITY、POSITIVE\_INFINITY 和 NaN。

5) float 浮点

float 是 Java 原始类型。float 变量可以存储单精度浮点值。

使用此关键字时应遵循下列规则：

Java 中的浮点文字始终默认为双精度。要指定单精度文字值，应在数值后加上 f 或 F，如 0.01f。

由于浮点数据类型是实际数值的近似值，因此，一般不要对浮点数值进行是否相等的比较。

Java 浮点数值可代表无穷大和 NaN（非数值）。Float 包装对象类用来定义常量 MIN\_VALUE、MAX\_VALUE、NEGATIVE\_INFINITY、POSITIVE\_INFINITY 和 NaN。

6) int 整型

int 是 Java 原始类型。int 变量可以存储 32 位的整数值。

Integer 类是 int 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

7) long 长整型

long 是 Java 原始类型。long 变量可以存储 64 位的带符号整数。

Long 类是 long 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

8) short 短整型

short 是 Java 原始类型。short 变量可以存储 16 位带符号的整数。

Short 类是 short 原始类型的包装对象类。它定义代表此类型的值的范围的 MIN\_VALUE 和 MAX\_VALUE 常量。

Java 中的所有整数值都是 32 位的 int 值，除非值后面有 l 或 L（如 235L），这表示该值应解释为 long。

9) null 空

null 是 Java 的保留字，表示无值。

将 null 赋给非原始变量相当于释放该变量先前所引用的对象。

不能将 null 赋给原始类型（byte、short、int、long、char、float、double、boolean）变量。

10) true 真

true 关键字表示 boolean 变量的两个合法值中的一个。

11) false 假

false 关键字代表 boolean 变量的两个合法值之一。

7. 变量引用

1) super 父类,超类

super 关键字用于引用使用该关键字的类的超类。

作为独立语句出现的 super 表示调用超类的构造方法。

super.<methodName>() 表示调用超类的方法。只有在如下情况中才需要采用这种用法：要调用在该类中被重写的方法，以便指定应当调用在超类中的该方法。

2) this 本类

this 关键字用于引用当前实例。

当引用可能不明确时，可以使用 this 关键字来引用当前的实例。

3) void 无返回值

void 关键字表示 null 类型。

void 可以用作方法的返回类型，以指示该方法不返回值。

8. 保留字

正确识别java语言的关键字（keyword）和保留字（reserved word）是十分重要的。Java的关键字对java的编译器有特殊的意义，他们用来表示一种数据类型，或者表示程序的结构等。保留字是为java预留的关键字，他们虽然现在没有作为关键字，但在以后的升级版本中有可能作为关键字。

识别java语言的关键字，不要和其他语言如c/c++的关键字混淆。

const和goto是java的保留字。 所有的关键字都是小写

1) goto 跳转

goto 保留关键字，但无任何作用。结构化程序设计完全不需要 goto 语句即可完成各种流程，而 goto 语句的使用往往会使程序的可读性降低，所以 Java 不允许 goto 跳转。

2) const 静态

const 保留字，是一个类型修饰符，使用const声明的对象不能更新。与final某些类似。

3) native 本地

　Java不是完美的，Java的不足除了体现在运行速度上要比传统的C++慢许多之外，Java无法直接访问到操作系统底层（如系统硬件等)，为此Java使用native方法来扩展Java程序的功能。

　　可以将native方法比作Java程序同Ｃ程序的接口，其实现步骤：

　　１、在Java中声明native()方法，然后编译；

　　２、用javah产生一个.h文件；

　　３、写一个.cpp文件实现native导出方法，其中需要包含第二步产生的.h文件（注意其中又包含了JDK带的jni.h文件）；

　　４、将第三步的.cpp文件编译成动态链接库文件；

　　５、在Java中用System.loadLibrary()方法加载第四步产生的动态链接库文件，这个native()方法就可以在Java中被访问了。

## Java 常用API的运用，效率及技巧

1.     [Java面向对象基本概念](#_1.   Java面向对象基本概念)

2.     [System](#_2．System)

3.     [String, StringBuffer](#_3．String, StringBuffer)

4.     [数值，字符，布尔对象与简单类型的操作](#_4．数值，字符，布尔对象与简单类型的操作)

5.     [Class, ClassLoader](#_5．Class, ClassLoader)

6.     [Java IO系统](#_6. Java IO系统)

7.     [Java集合类](#_7. Java集合类)

8.     [ResourceBundle, Properties](#_8．ResourceBundle, Properties)

9.     [Exceptions](#_9. Exceptions)

10.   [JDBC类库](#_10. JDBC类库)

11.   [常用设计模式](#_11. 常用设计模式)

### 1.   Java面向对象基本概念

Java基本上是面向对象的程序设计语言， 除了一些简单类型(primitive)的变量以外，一切都是对象， 程序是对象的组合， 每个对象都有自己的空间， 并且每个对象都有一种类型， 同一类所有对象都能接受相同的消息。 下面只对Java中对象的结构作简单的说明：

         类（class）:  class是定义类的关键字，  类中包含类变量， 方法， 内部类， 内部接口等。由class可以生成类的实例， 即一个个对象。 如果一个类的成员被定义成static的，则这个成员不专属于任何对象， 而是属于这个类， 所有的对象共享这个成员。

         抽象类(abstract class): 抽象类不能直接生成一个实例， 抽象类中必需有方法是abstract的，抽象类的意思就是它实现了一部分的方法， 而定义为abstract的方法则需要在它的字类中去实现。

         接口(interface): 接口可以理解为纯抽象的类， 它的每个方法都是未实现的， 它可以有成员变量， 但必须是static的。  一个类如果从这个接口继承（implements）则它必须实现这个接口的所有方法。

继承类用关键字：extends，继承接口用关键字：implements。 一个类只能从一个类继承下来， 但可以从多个接口继承（类似于C++的多重继承）。 字类可以覆盖父类的方法(method)， 但不能覆盖父类的成员变量(field)。 如果父类的方法为final或static的则不能被覆盖。类的初始化顺序是， 如果有父类， 则先初始化父类的field，然后执行父类的构造函数， 如果子类没有显式的去调父类的构造函数则缺省的会去调父类的无参数构造函数。 然后是子类的field与构造函数的初始化。

public interface SuperInterface {

             public staitc String SOME\_FLAG = “1”;

             public void someMethod();

}

public Class SuperClass {

             { System.out.println(“init SuperClass field”);}

             public SuperClass() {System.out.println(“init SuperClass Constructor”); }

             public void runMethod() { System.out.println(“run SuperClass runMethod()”); }

}

public Class SubClass extends SuperClass implements SuperInterface {

             { System.out.println(“init SubClass field”); }

       public SubClass() {System.out.println(“init SubClass Constructor”); }

             public void someMethod() {System.out.println(“run SubClass someMethod()”); }

             public void runMethod() {System.out.println(“run SubClass runMethod()”); }

}

有以下test代码：

public class Test {

public void main(String[] args) {

       SubClass sub = new SubClass();

       sub. runMethod();

}

}

则会输出：

init SuperClass field

init SuperClass Constructor

init SubClass field

init SubClass Constructor

run SubClass runMethod()

以下章节所讲述到的常用的Java API就是一些Java自带的一些Class或Interface的用法。

### 2．System

System类位于package java.lang下面， 凡是此package下面的类我们可以直接引用无需先import进来， 因为JVM缺省就load了这下面的所有class。

System包含了一些我们常用的方法与成员变量。 System不能被实例化， 所有的方法都可以直接引用。 主要作用大致有：

         输入输出流：   
(PrintStream) System.out （标准终端输出流），   
(PrintStream) System.err（标准错误输出流），   
(InputStream) System.in（标准输入流）。   
我们还可以重定向这些流， 比如将所有的System.out的输出全部重定向至一文件中去。   
System.setOut(PrintStream) 标准输出重定向  
System.setErr(PrintStream) 标准错误输出重定向  
System.setIn(InputStream) 标准输入重定向

         取当前时间：  
System.currentTimeMillis() 所取到的时间是从1970/01/01以来1/1000秒计算的long型值。这个值可以转换至Date或Timestamp值。 它一般还可以用来计算程序执行的时间。例：  
long beginTime = System. currentTimeMillis();  
…  
…  
System.out.println(“run time = ” + (System. currentTimeMillis() – beginTime));

         数组拷贝：  
System.arraycopy([Object](http://hyper.sunjapan.com.cn/DocumentLanguageJavaJdk1.3%20docAPIjavalangObject.html) src, int src\_position, [Object](http://hyper.sunjapan.com.cn/DocumentLanguageJavaJdk1.3%20docAPIjavalangObject.html) dst, int dst\_position, int length)  
src： 源数组。  
src\_position： 源数组拷贝的起始位置。  
dst： 目标数组  
dst\_position： 拷贝至目标数组的起始位置  
length： 拷贝元素的长度  
利用System.arraycopy进行数组的拷贝效率是最高的， 一般情况下我们自己很少直接用到这个方法，但在集合类的内部中都大量使用了这个方法。   
例：  
int[] array1 = {1, 2, 3, 4, 5};  
int[] array2 = {4, 5, 6, 7, 8};  
int array3 = new int[8];  
System.arraycopy(array1, 0, array3, 0, 5);  
System.arraycopy(array2, 2, array3, 5, 3);  
此时array3 = {1, 2, 3, 4, 5, 6, 7, 8}  
这比用for循环来进行赋值效率要高。

         存取系统的Properties：  
System.getProperties()：取得当前所有的Properties， Properties将在后面的集合一节进行详细的论述。  
System.setProperties(Properties props)：设置系统的Properties。  
System.getProperty(String key)： 根据一个键值来取得一个Property。  
System.setProperty(String key, String value)： 设置系统的一个Property。  
JVM起动的时候将会有一些缺省的Properties值， 例如：  
java.version Java运行环境版本  
java.home Java主目录 installation directory  
java.class.path Java 的class path  
java.ext.dirs Java的扩展目录路径  
file.separator 文件分隔符("/" on UNIX)  
path.separator 路径分隔符(":" on UNIX)  
line.separator 行分隔符 ("\n" on UNIX)  
user.name 用户名  
user.home 用户主目录  
user.dir 用户当前工作目录  
更详细的信息请参照Java API。 另外在起动一个java程序的时候可以通过-D来设置系统的Property， 比如 java –Dejb.file=ejb\_Test PrintTest 在PrintTest里面就可以通过System.getProperty(“ejb.file”)来取得值ejb\_Test。

         其它  
System. loadLibrary(String libname)： 加载native的动态库。 可以用C写JNI的库， 然后在java中通过native方法来调用。  
System.setSecurityManager(SecurityManager s)  
System.getSecurityManager()： 设置与取得系统的security class。

### 3．String, StringBuffer

**3.1 基本用法**

String可以说是我们最常用的一个类， 熟练掌握它的一些基本用法是很有用的。

String是由一组字符组成的字符串， 下标由0开始。 一旦有必要改变原来的内容， 每个String方法都有返回了一个新的String对象。

         char charAt(int index) 返回指定位置的字符。

         int compareTo(Object o)   
int compareTo(String anotherString)   
与另外一个对象进行比较。

         int compareToIgnoreCase(String str)  与另一个String进行比较， 不区分大小写

         String concat(String str) 连接两字符串， 可以直接用+， 因为Java给String覆盖了+

         static String copyValueOf(char[] data)   
static String copyValueOf(char[] data, int offset, int count)   
将data数组转换至String

         boolean endsWith(String suffix) 测试此String是否以suffix结尾。   
boolean startsWith(String prefix) 测试此String是否以prefix开头。

         boolean equals(Object anObject)   
boolean equalsIgnoreCase(String anotherString)   
比较两字符串的值。 不相等则返回false

         byte[] getBytes() 根据缺省的字符编码将String转换成字节数组。  
byte[] getBytes(String enc) 根据指定的编码将String转换万字节数组。

         void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 拷贝字符至一数组中

         int indexOf(int ch) 从字串的起始位置查找字符ch第一次出现的位置  
int indexOf(int ch, int fromIndex) 从指定的fromIndex位置向后查找第一次出现ch的位置，  
int indexOf(String str)   
int indexOf(String str, int fromIndex)   
如果不存在ch或str都返回-1

         int lastIndexOf(int ch) 从字串的最终位置往前查找第一次出现ch的位置  
int lastIndexOf(int ch, int fromIndex)  从指定的位置往前查找第一次出现ch的位置，  
int lastIndexOf(String str)   
int lastIndexOf(String str, int fromIndex)   
如果不存在则返回-1

         int length() 该字符串的字符长度（一个全角的汉字长度为1）

         String replace(char oldChar, char newChar) 将字符oldChar全部替换为newChar， 返回一个新的字符串。

         String substring(int beginIndex) 返回从beginIndex开始的字符串子集  
String substring(int beginIndex, int endIndex) 返回从beginIndex至endIndex结束的字符串的子集。 其中endIndex – beginIndex等于子集的字符串长度

         char[] toCharArray() 返回该字符串的内部字符数组

         String toLowerCase() 转换至小写字母的字符串  
String toLowerCase(Locale locale)   
String toUpperCase() 转换至大写字母的字符串  
String toUpperCase(Locale locale)

         String toString() 覆盖了Object的toString方法， 返回本身。

         String trim() 将字符串两边的半角空白字符去掉， 如果需要去掉全角的空白字符得要自己写。

         static String valueOf(primitive p) 将其它的简单类型的值转换为一个String

StingBuffer是一个可变的字符串，它可以被更改。同时StringBuffer是Thread safe的， 你可以放心的使用， 常用的方法如下：

         StringBuffer append(param)  在StringBuffer对象之后追加param(可以为所有的简单类型和Object) 返回追加后的StringBuffer， 与原来的对象是同一份。

         char charAt(int index) 返回指定位置index的字符。

         StringBuffer delete(int start, int end) 删除指定区域start~end的字符。

         StringBuffer deleteCharAt(int index) 删除指定位置index的字符。

         void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin) 同String的getChars方法

         StringBuffer insert(int offset, boolean b) 在指定位置offset插入param(为所有的简单类型与Object)

         int length() 同String的length()

         StringBuffer replace(int start, int end, String str) 将指定区域start~end的字符串替换为str

         StringBuffer reverse() 反转字符的顺序

         void setCharAt(int index, char ch) 设置字符ch至index位置。

         String substring(int start)

         String substring(int start, int end) 同String的subString

         String toString() 返回一个String

大家可能已经注意到很多方法都返回了一个StringBuffer对象， 但返回的这个对象与String的方法返回的String不一样， 返回的StringBuffer对象与被操作的StringBuffer对象是同一份， 而String的方法返回的String则肯定是重新生成的一个String。

**3.2性能对比**

因为String被设计成一种安全的字符串， 避免了C/C++中的尴尬。因此在内部操作的时候会频繁的进行对象的交换， 因此它的效率不如StringBuffer。 如果需要频繁的进行字符串的增删操作的话最好用StringBuffer。 比如拼SQL文， 写共函。 另： 编绎器对String的+操作进行了一定的优化。

x = "a" + 4 + "c"

会被编绎成

x = new StringBuffer().append("a").append(4).append("c").toString()

但：

x = “a”;

x = x + 4;

x = x + “c”;

则不会被优化。 可以看出如果在一个表达式里面进行String的多次+操作会被优化， 而多个表达式的+操作不会被优化。

**3.3技巧**

1.       在Servlet2.3与JSP1.1以前画面post到后台的数据是通过ISO88591格式进行编码的， 则当遇到全角日文字的时候, 在后台request得到的数据可能就是乱码, 这个时候就得自己进行编码转换, 通过String.getBytes(String enc)方法得到一个字节流, 然后通过String(byte[] bytes, String enc)这个构造函数得到一个用新的编码生成的字符串. 例如将ISO88591的字符串转换成Shift\_JIS的字符串, 方法如下:

public static String convertString(String str) {

       if (str == null) { return null; }

       try {

              byte[] buf = str.getBytes("ISO8859\_1");

              return new String(buf, "Shift\_JIS");

       } catch (Exception ex) {ex.printStackTrace();return null; }

}

不过在最新的Servlet2.3与Jsp1.2中可以通过request.setCharacterEncoding来进行设置取值的码制, 不需要自己再做转换。

2．因为Java在计算String的长度是以字符为单位的， 因此一个全角与半角的字符长度是一样的， 但是DB中往往是根据字节来计算长度的， 因此我们在做Check的时候得要判断String的字节长， 可以用以下的方法：

public static String length(String str) {

       if (str == null) { return 0; }

       return str.getBytes().length;

}

### 4．数值，字符，布尔对象与简单类型的操作

简单的对照表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| Object | | Primitive | 范围 |
| Number | Long | long | -9223372036854775808 to 9223372036854775807 |
| Integer | int | -2147483648 to 2147483647 |
| Short | short | -32768 to 32767 |
| Byte | byte | -128 to 127 |
| Double | double |  |
| Float | float |  |
| Character | | char | '\u0000' to '\uffff' |
| Boolean | | boolean | false and true |

与C等其它语言不同的是数值的范围不随平台的改变而改变， 这就保证了平台之间的统一性，提高了可移植性。

**Number:** Number本身是个抽象类， 不能直接使用， 所有直接从Number继承下来的子类都有以下几种方法：

         byte byteValue() 返回字节值

         double doubleValue() 返回double值

         float floatValue() 返回float值

         int intValue() 返回float值

         long longValue() 返回long值

         short shortValue() 返回short值

在需要通过Object来取得简单数据类型的值的时候就得用到以上的方法， 不过我不推荐不同类型之间的取值， 比如Long型的Object不要直接去调用intValue()，精度可能会丢失。

如果想通过String来得到一个数值类型的简单类型值， 一般在每个Number的类里面都有一个parseXXX(String)的静态方法， 如下：

         byte Byte.parseByte(String s)

         double Double.parseDouble(String s)

         float Float.parseFloat(String s)

         int Integer.parseInt(String s)

         long Long.parseLong(String s)

         short Short.parseShort(String s)

如果想直接从String得到一个Number型的Object，则每个Number类里面都有valueOf(String s) 这个静态方法。如：

         Byte Byte.valueOf(String s)

         Double Double.valueOf(String s)

         Float Float.valueOf(String s)

         Integer Integer.valueOf(String s)

         Long Long.valueOf(String s)

         Short Short.valueOf(String s)

一般的在构造一个Number的时候都可以通过一个String来完成， 比如：

Long longObject = new Long(“1234567890”);

等价于

Long longObject = Long.valueOf(“1234567890”);

因为每个Number的子类都实现了Object的toString()方法， 所以， 如果想得到一个String型的数值， 直接调用XXX.toString()就可以了。 如果想得到一个简单类型的String， 方法很多总结如下：

         首先生成对应的Number Object类型， 然后调用toString()

         调用Number子类 .toString(type t) 其中t就是简单类型的数据。

         调用String.valueOf(type t) （推荐使用这种方法）

大家可以看出， 往往一种结果可以用多种方法实现， 总的原则就是深度最少优先。比如由一个String得到一个简单类型的值可以有以下两种方法：

Integer.parseInt(“12345”);

或

(new Integer(s)).intValue(“12345”);

当然应该使用第一种方法。

**Character:**Character对应着char类型， Character类里面有很多静态的方法来对char进行判断操作， 详细的操作请参照JDK API。 Java对字符的判断操作基本都是以Unicode进行的， 比如Character.isDigit(char ch)这个方法， 不光半角的0-9符合要求， 全角的日文0-9也是符合要求的。

**Boolean:** Boolean对应着boolean类型， boolean只有true和false两个值， 不能与其它数值类型互换， 可以通过字符串”true”以及”false”来得到Object的Boolean， 也可以通过简单类型的boolean得到Boolean， 常用方法如下：

         Boolean(boolean value) 通过简单类型的boolean构造Boolean

         Boolean(String s) 通过String(“true”, “false”)构造Boolean

         boolean booleanValue() 由Object的Boolean得到简单类型的boolean值

         boolean equals(Object obj) 覆盖了Object的.equals方法， Object值比较

         static Boolean valueOf(String s) 功能与构造函数Boolean(String s)一样

### 5．Class, ClassLoader

Java是一种介于解释与编绎之间的语言， Java代码首先编绎成字节码， 在运行的时候再翻译成机器码。 这样在运行的时候我们就可以通过Java提供的反射方法(reflect)来得到一个Object的Class的额外信息， 灵活性很大，可以简化很多操作。

**Class:**任何一个Object都能通过getClass()这个方法得到它在运行期间的Class。 得到这个Class之后可做的事情就多了， 比如动态得到它的构造函数， 成员变量， 方法等等。 还可以再生成一份新的实例， 下面只给出几个我们常用的方法， 更详细的用法参照Java API

         Class Class.forName(String className) throws ClassNotFoundException： 这是个静态方法， 通过一个Class的全称来得到这个Class。

         String getName() 取得这个Class的全称， 包括package名。

         Object newInstance() 得到一个实例， 调用缺省的构造函数。

例如我们有一个类： com.some.util.MyClass  如果得到它的一个实例呢？ 可能有以下两种方法：

MyClass myClass = new MyClass()， 直接通过操作符new生成；

或者：

MyClass myClass =  (MyClass) Class.forName(“com.some.util.MyClass”).newInstance();

也许有人就会怀疑第二种方法实际意义， 能够直接new出来干嘛绕弯。 但实际上它的用处却很大， 举个例子： 用过struts的人都知道， 在action-config.xml当中定义了一系列的formBean与actionBean， 当然每个form与action都具有同类型， 这样在一个request过来的时候我可以动态的生成一个form与action的实例进行具体的操作， 但在编码的时候我并不知道具体是何种的form与action， 我只调用它们父类的方法。 你如果要用第一种方法的话， 你得在编码的时候通过一个标志来判断每一次request需要具体生成的form与action， 代码的灵活性大大降低。 总的来说在面向接口的编程当中经常使用这种方法， 比如不同数据库厂家的JDBC Driver都是从标准的JDBC接口继承下去的， 我们在写程序的时候用不着管最终是何种的Driver， 只有在运行的时候确定。 还有XML的Parser也是， 我们使用的只是标准的接口， 最后到底是谁来实现它的， 我们用不着去管。

**ClassLoader:**ClassLoader是一个抽象类，一般的系统有一个缺省的ClassLoader用来装载Class， 用ClassLoader.getSystemClassLoader()可以得到。不过有时候为了安全或有其它的特殊需要我们可以自定义自己的ClassLoader来进行loader一些我们需要的Class， 比如有的产品它用了自己的ClassLoader可以指定Class只从它指定的特定的JAR文件里面来loader，如果你想通过覆盖ClassPath方法来想让它用你的Class是行不通的。 有兴趣的可以参照Java API 的更详细的用法说

### 6. Java IO系统

**JDK1.0输入流**

|  |
| --- |
| InputStream |

|  |
| --- |
| ByteArrayInputStream |

|  |
| --- |
| FileInputStream |

|  |
| --- |
| FilterInputStream |

|  |
| --- |
| ObjectInputStream |

|  |
| --- |
| PipedInputStream |

|  |
| --- |
| SequenceInputStream |
|  |

|  |
| --- |
| BufferedInputStream |

|  |
| --- |
| DataInputStream |

|  |
| --- |
| PushbackInputStream |

|  |
| --- |
| ~~LineNumberInputStream~~ |

**JDK1.1输入流**

|  |
| --- |
| Reader |

|  |
| --- |
| BufferedReader |

|  |
| --- |
| FilterReader |

|  |
| --- |
| PipedReader |

|  |
| --- |
| StringReader |

|  |
| --- |
| InputStreamReader |

|  |
| --- |
| CharArrayReader |

|  |
| --- |
| FileReader |

|  |
| --- |
| PushbackReader |

|  |
| --- |
| LineNumberReader |

**JDK1.0输出流**

|  |
| --- |
| OutputStream |

|  |
| --- |
| BufferedOutputStream |

|  |
| --- |
| DataOutputStream |

|  |
| --- |
| PrintStream |

|  |
| --- |
| ByteArrayOutputStream |

|  |
| --- |
| FileOutputStream |

|  |
| --- |
| FilterOutputStream |

|  |
| --- |
| ObjectOutputStream |

|  |
| --- |
| PipedOutputStream |

**JDK1.1输出流**

|  |
| --- |
| Writer |

|  |
| --- |
| BufferedWriter |

|  |
| --- |
| FilterWriter |

|  |
| --- |
| PipedWriter |

|  |
| --- |
| StringWriter |

|  |
| --- |
| FileWriter |

|  |
| --- |
| PrintWriter |

|  |
| --- |
| OutputStreamWriter |

|  |
| --- |
| CharArrayWriter |

如果你刚刚接触Java的IO部分， 你可能会感觉无从入手， 确实Java提供了过多的类，反而让人感到很乱。

可将Java库的IO类分为输入与输出两个部分， 在1.0版本中提供了两个抽象基类， 所有输入的类都从InputStream继承， 所有输出的类都从OutputStream继承， 1.1提供了两个新的基类， 负责输入的Reader与输出的Writer， 但它们并不是用来替换原来老的InputStream与OutputStream， 它们主要是让Java能更好的支持国际化的需求。 原来老的IO流层只支持8位字节流， 不能很好地控制16位Unicode字符。 Java内含的char是16位的Unicode， 所以添加了Reader和Writer层次以提供对所有IO操作中的Unicode的支持。 除此之外新库也对速度进行了优化， 可比旧库更快地运行。

InputStream的类型：

1．  字节数组

2．  String对象

3．  文件

4．  管道， 可以从另外一个输出流得到一个输入流

5．  一系列的其他流， 可以将这些流统一收集到单独的一个流内。

6．  其他起源（如socket流等）

还有一个是File类， Java中一个目录也是一个文件，可以用file.isFile()和file.isDirectory()来进行判断是文件还是目录。 File 对象可能作为参数转换为文件流进行操作。 具体操作参照Java IO API。

**常用方法：**

/\*\*

 \* 拼文件名, 在windows(\)跟unix(/)上的文件分割符是不一样的

 \* 可以通过File类的静态成员变量separator取得

 \*/

public static String concatFileName(String dir, String fileName) {

String fullFileName = "";

if (dir.endsWith(File.separator)) {fullFileName = dir + fileName;

} else {fullFileName = dir + File.separator + fileName;}

return fullFileName; }

/\*\*

 \* 从控制台读取输入的数据, System.in (InputStream) 先转换至InputStreamReader再用

 \* BufferedReader进行读取.

 \*

 \*/

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

String str = "";

while (str != null) {str = in.readLine();// process(str); }

/\*\*

 \* 从文件中按行进行读取数据处理

 \*/

BufferedReader in = new BufferedReader(new FileReader("infilename"));

String str;

while ((str = in.readLine()) != null) {// process(str); }

in.close();

/\*\*

 \* 写数据至一个新的文件中去.

 \*/

BufferedWriter out = new BufferedWriter(new FileWriter("outfilename"));

out.write("a String");

out.close();

/\*\*

 \* 追加新的数据到一个文件中去, 如果原文件不存在

 \* 则新建这个文件.

 \*/

BufferedWriter out = new BufferedWriter(new FileWriter("filename", true));

out.write("aString");

out.close();

/\*\*

 \* 将一个可序列化的Java的object以流方式写到一个文件中去中.

 \*/

ObjectOutput out = new ObjectOutputStream(new FileOutputStream("filename.ser"));

out.writeObject(object);

out.close();

/\*\*

 \* 从文件中恢复序列化过的Java Object

 \*/

ObjectInputStream in = new ObjectInputStream(new FileInputStream("filename.ser"));

Object object = (Object) in.readObject();

in.close();

/\*\*

 \* 以指定的编码方式从文件中读取数据

 \*/

BufferedReader in = new BufferedReader(new InputStreamReader(

 new FileInputStream("infilename"), "UTF8"));

String str = in.readLine();

/\*\*

 \* 以指定的编码方式写数据到文件中

 \*/

Writer out = new BufferedWriter(new OutputStreamWriter(

new FileOutputStream("outfilename"), "UTF8"));

out.write("a String");

out.close();

**由上面的例子可以看出， 流之间可以互相转换， 如果想对流进行字符操作最好将之转换成BufferedReader与BufferedWriter这样可以提高读写的效率。**

需要注意的是一般来说流的大小是得不到的， 虽然一般的InputStream都有一个available()的方法可以返回这个流可以读到的字节数， 不过这个方法有时候不会很准确， 比如在读取网络传输的流的时候， 取到的长度并不一定是真实有效的长度。

大家在写程序的时候可能已经注意到一些类都继承了java.io.Serializable这个接口， 其实继承了这个接口之后这个类的本身并不做任何事情， 只是这个类可以被序列化并通过流来进行传输， 并可被还原成原Object。 这个流可以通过网络传输（EJB 中进行传递参数和返回值的Data Class）， 也可以保存到文件中去（ObjectOutputStream）。

### 7. Java集合类

在写程序的时候并不是每次只使用一个对象， 更多的是对一组对象进行操作， 就需要知道如何组合这些对象， 还有在编码的时候我们有时并不知道到底有多少对象，它们需要进行动态的分配存放。

Java的集合类只能容纳对象句柄， 对于简单类型的数据存放， 只能通过数据来存放， 数组可以存放简单类型的数据也能存放对象。

Java提供了四种类型的集合类： Vector(矢量)， BitSet(位集)， Stack(堆栈)， Hashtable(散列表)。

1.         矢量： 一组有序的元素， 可以通过index进行访问。

2.         位集： 其实就是由二进制位构成的Vector， 用来保存大量”开-关”信息， 它所占的空间比较小， 但是效率不是很高， 如果想高效率访问， 还不如用固定长度的数组。

3.         堆栈： 先入后出（LIFO）集合， java.util.Stack类其实就是从Vector继承下来的， 实现了pop, push方法。

4.         散列表： 由一组组“键--值”组成， 这里的键必须是Object类型。 通过Object的hashCode进行高效率的访问。

对于这些集合之间的关联关系见下图， 其中标色的部分为我们常用的类。

由上图可以看出， 基本接口有两个：

**Collection：** 所有的矢量集合类都从它继承下去的， 但并不直接从它继承下去的。 List与Set这两个接口直接继承了Collection， 他们的区别是List里面可以保存相同的对象句柄， 而Set里面的值是不重复的。 我们经常用的Vector与ArrayList就是从List继承下去的， 而HashSet是从Set继承的。

**Map：**散列表的接口， Hashtable与HashMap继承了这个接口。

下面给出常用集合类的常用方法。

/\*\*

 \* Vector 与 ArrayList的操作几乎是一样的

 \* 常用的追加元素用add(), 删除元素用remove()

 \* 取元素用get(), 遍历它可以循环用get()取. 或者

 \* 先得到一个Iterator, 然后通过遍历Iterator的方法

 \* 遍历Vector或ArrayList

 \*/

// 生成一个空的Vector

Vector vector = new Vector();

// 在最后追加一个元素。

vector.add("one");

vector.add("two");

// 在指定的地方设置一个值

vector.set(0, "new one");

// 移走一个元素或移走指定位置的元素

vector.remove(0);

// 用for循环遍历这个Vector

for (int i = 0; i < vector.size(); i++) {

String element = (String) vector.get(i); }

// 用枚举器(Enumeration)遍历它(只有Vector有，ArrayList没有)

Enumeration enu = vector.elements();

while (enu.hasMoreElements()) {

enu.nextElement();}

// 用反复器(Iterator)遍历它

Iterator it = vector.iterator();

while (it.hasNext()) {it.next();}

/\*\*

 \* Hashtable与HashMap的操作, 追加元素用put(不是add)

 \* 删除元素用remove, 遍历可以用Iterator 既可以遍历

 \* 它的key, 也可以是value

 \*/

// 生成一个空的Hashtable或HashMap

Hashtable hashtable = new Hashtable();

// 追加一个元素

hashtable.put("one", "one object value");

// 删除一个元素

hashtable.remove("one");

// 用Iterator遍历

Iterator keyIt = hashtable.keySet().iterator();

while (keyIt.hasNext()) {

Object keyName = keyIt.next();

String value = (String) hashtable.get(keyName); }

Iterator valueIt = hashtable.values().iterator();

while (valueIt.hasNext()) {

valueIt.next();}

// 用Enumeration遍历, 只有Hashtable有, HashMap没有.

Enumeration enu = hashtable.elements();

while (enu.hasMoreElements()) {

enu.nextElement();}

**说明：**Enumeration是老集合库中的接口， 而Iterator是新集合（1.2）中出现的， 而Vector与Hashtable也都是老集合中的类， 所以只有Vector与Hashtable可以用Enumeration。

**Vector与ArrayList对比：**

虽然在使用的时候好象这两个类没什么区别， 它们都是从List继承下来的， 拥有相同的方法， 但它们的内部还是有些不同的，

         首先Vector在内部的一些方法作了线程同步(synchronized)。 同步的代价就是降低了执行效率， 但提高了安全性。而ArrayList则是线程不同步的， 可以多线程并发读写它。

         内部数据增长率。 所有的这些矢量集合在内部都是用Object的数组进行存储和操作的。 所以也就明白了为什么它可以接受任何类型的Object， 但取出来的时候需要进行类型再造。 Vector与ArrayList具有自动伸缩的功能， 我们不用管它size多大， 我们都可以在它的后面追加元素。 Vector与ArrayList内部的数组增长率是不一样的， 当内部的数组不能容纳更多元素的时候， Vector会自动增长到原两倍大小， ArrayList会变为原一倍半大小， 而不是我们所想象的一个元素一个元素的增长。

**Hashtable与HashMap对比：**

Hashtable与HashMap都是从Map继承下来的， 方法几乎都一样， 它们内部有两个不同点：

         与Vector和ArrayList一样， 它们在线程同步是不同的， Hashtable在内部做了线程同步， 而HashMap是线程不同步的。

         HashMap的键与值都可以为null， 而Hashtable不可以， 如果你试图将一个null值放到Hashtable里面去， 会抛一个NullPointException的。

**性能对比：**

抛开不常用的集合不讲， 每种集合都应该有一个我们常用的集合类， 而在不同的场合下应该使用效率最高的一个。 一般来说我推荐尽量使用新的集合类， 除非不得已， 比如说需要用用了老集合类写的产品的程序。 也就是说尽量使用ArrayList与HashMap， 而少使用Vector与Hashtable。

         在单线程中使用ArrayList与HashMap， 而在多线程中如果需要进行线程同步可以使用Vector与Hashtable， 但也可以用synchronized对ArrayList与HashMap进行同步， 不过同步后的ArrayList与HashMap是比Vector与Hashtable慢的。 不过我认为需要进行线程同步的地方并不多。 如果一个变量定义在方法内部同时只可能有一个线程对之进行操作， 就不必要进行同步， 如果定义在类的内部并且不是静态的， 属于实例变量， 而这个类并没有被多线程使用也就不必要同步。  
一般自己写的程序很少会自己去另开线程的， 但在Web开发的时候， 如果用了Servlet， 则每个request都是一个线程， 也就是说每个Servlet都是在多线程环境下运行的， 如果Servlet中使用了全局静态的成员变量就得小心点儿， 如果需要同步就得在方法上加上synchronized修饰符， 如果允许多个线程操作它， 并且你知道不会有什么冲突问题就可以大胆的使用ArrayList与HashMap。 另外如果在多线程中有线程在对ArrayList或HashMap进行修改（结构上的修改）， 而有一个线程在用Iterator进行读取操作， 这个时候就有可能会抛ConcurrentModificationException, 因为用Iterator的时候， 不允许原List的结构改变。但可以用get方法来取。

**常用技巧：**

1.         采用面向接口的编程技巧， 比如现在需要写一个共通函数，对矢量集合类诸如Vector，ArrayList，HashSet等等进行操作， 但我并不知道最终用户会具体传给我什么类型的类， 这个时候我们可以使用Collection接口， 从而使代码具有很大的灵活性。 代码示例如下：

/\*\*

 \* 将list里面的所有元素用sep连接起来，

 \* list可以为Vector, ArrayList, HashSet等。

 \*/

public static String join(String sep, Collection list) {

   StringBuffer sb = new StringBuffer();

   Iterator iterator = list.iterator();

   while (iterator.hasNext()) {

          sb.append(iterator.next());

          if (iterator.hasNext()) { sb.append(sep); }

   }

   return sb.toString();

}

2.         利用Set进行Unique， 比如有一组对象（其中有对象是重复的）， 但我们只对不同的对象感兴趣， 这个时候可以使用HashSet这个集合类， 然后可以通过覆盖Object的equals方法来选择自定义判断相等的rule。 缺省的是地址判断。 例：

class DataClass {

   private String code = null;

   private String name = null;

   public void setCode(String code) {this.code = code; }

   public String getCode() {return this.code; }

   public void setName(String name) {this.name = name; }

   public String getName() {return this.name; }

   public boolean equals(DataClass otherData) {

          if (otherData != null) {

                 if (this.getCode() != null&& this.getCode().equals(otherData.getCode()) {

                        return true;

                 }

          }

          return false;

   }}

DataClass data1 = new DataClass();

DataClass data2 = new DataClass();

data1.setCode("1");

data2.setCode("1");

HashSet singleSet = new HashSet();

singleSet.add(data1);

singleSet.add(data2);

结果singleSet里面只有data1， 因为data2.equals(data1)， 所以data2并没有加进去。

3.         灵活的设计集合的存储方式， 以获得较高效的处理。 集合里面可以再嵌套集合， 例：在ArrayList里面存放HashMap， HashMap里面再嵌套HashMap。

### 8．ResourceBundle, Properties

**ResourceBundle：**开发一个项目， 配置文件是少不了的， 一些需要根据环境进行修改的参数， 都有得放到配置文件中去， 在Java中一般是通过一个properties文件来实现的， 这个文件以properties结尾。 内部结构是二维的， 以key=value的形式存在。 如下：

options.column.name.case=1

options.column.bean.serializable=1

options.column.bean.defaultconstructor=1

options.column.method.setter=1

options.general.user.version=1.0

database.connection[0]=csc/csc@localhost\_oci8

database.connection[1]=cscweb/cscweb@localhost\_thin

ResourceBundle用来解析这样的文件， 它的功能是可以根据你的Locale来进行解析配置文件， 如果一个产品需要进行多语言支持， 比如在不同语种的系统上， 会显示根据它的语言显示相应的界面语言， 就可以定义多份的properties文件， 每个文件的key是一样的， 只是value不一样， 然后在application起动的时候， 可以判别本机的Locale来解析相应的properties文件。 Properties文件里面的数据得要是Unicode。 在jdk下面可以用native2ascii这个命令进行转换。 例： native2ascii Message.txt Message.properties 会生成一个Unicode的文件。

**Properties:**Properties这个类其实就是从Hashtable继承下来的， 也就是说它是一个散列表， 区别在于它的key与value都是String型的， 另外也加了几个常用的方法：

         String getProperty(String key)  取得一个property

         String getProperty(String key, String defaultValue)  取property， 如果不存在则返回defaultValue。

         void list(PrintStream out)  向out输出所有的properties

         void list(PrintWriter out)

         Enumeration propertyNames()  将所有的property key名以Enumeration形式返回。

         Object setProperty(String key, String value) 设置一个property。

ResourceBundle与Properties一般结合起来使用。 它们的用法很简单， 由ResourceBundle解析出来的key与value然后放至到一个静态的Properties成员变量里面去， 然后就可以通过访问Properties的方法进行读取Property。 下面给个简单的例子：

public class PropertyManager implements Serializable {

/\*\* 定义一个静态的Properties变量 \*/

private static Properties properties = new Properties();

/\*\*

 \* 通过一个类似于类名的参数进行Property文件的初期化

 \* 比如现在有一个文件叫Message.properties， 它存放在

 \* ejb/util下面并且， 这个目录在运行的classpath下面

 \* 则in就为ejb.util.Message

 \*

 \*/

public static void init(String in) throws MissingResourceException {

    ResourceBundle bundle = ResourceBundle.getBundle(in);

    Enumeration enum = bundle.getKeys();

    Object key = null;

    Object value = null;

    while (enum.hasMoreElements()) {

        key = enum.nextElement();

        value = bundle.getString(key.toString());

        properties.put(key, value);

    }}

/\*\*

 \* 取得一个Property值

 \*/

public static String getProperty(String key) {return properties.get(key); }

/\*\*

 \* 设置一个Property值

 \*/

public static void setProperty(String key, String value) {properties.put(key, value); }

}

不过现在的Java产品中，越来越倾向于用XML替换Properties文件来进行配置。 XML配置具有层次结构清楚的优点。

### 9. Exceptions

|  |
| --- |
| Throwable |

|  |
| --- |
| Exception |

|  |
| --- |
| Error |

|  |
| --- |
| RuntimeException |

|  |
| --- |
| ClassNotFoundException |

|  |
| --- |
| IOException |

|  |
| --- |
| SQLException |

|  |
| --- |
| IndexOutOfBoundsException |

|  |
| --- |
| ClassCastException |

|  |
| --- |
| NullPointerException |

|  |
| --- |
| OutOfMemoryError |

|  |
| --- |
| StackOverflowError |

|  |
| --- |
| NoClassDefFoundError |

Java采用违例(Exception)处理机制来进行错误处理。 违例机制的一个好处就是能够简化错误控制代码， 我们再也不用检查一个特定的错误， 然后在程序的多处地方对其进行控制。 此外， 也不需要在方法调用的时候检查错误（因为保证有人能够捕获这里的错误）。 我们只需要在一个地方处理问题：”违例控制模块”或者”违例控制器”。 这样可有效减少代码量， 并将那些用于描述具体操作的代码与专门纠正错误的代码分隔开。

一个完整的违例例子：

public void throwTest() throws MyException {

try {

        ...

} catch (SQLException se) {

        cat.error("", se);

        throw new MyException(se.getMessage());

} catch (Exception e) {cat.error("", e);

} finally {

        ...

}

}

如果一段代码有可能会抛出违例可以用try {} catch {}来处理。 被catch到的违例可以再抛出， 也可以转换为其它类型的Exception抛出。 finally块里面的代码总会被执行到的， 不管前面是否已经throw或return了。

Throwable是所有违例的基类， 它有两种常规类型。 其中， Error代表编绎期和系统错误， 我们一般不必特意捕获它们。 Exception是可以从任何标准Java库的类方法中掷出的基本类型。

看上面的图， 如果是Error的子类或是RuntimeException的子类这种违例有一定的特殊性， 可以说我们可以当它们不存在， 当这种违例抛出的时候， 我们可以不catch它， 也可以不在方法上throws它。 RuntimeException一般代表的是一个编程错误， 是完全可以避免的。

**性能注意点：**因为使用了Exception之后是要影响一些效率的， 所以Exception不能滥用。一般的不要用Exception来控制业务流程， 其次不要循环体内使用。

**技巧：**我们可以从Exception或直接从Throwable继承写我们自己的Exception， 然后根据业务需要抛不同种类的Exception。

### 10. JDBC类库

有了 JDBC，向各种关系数据库发送 SQL 语句就是一件很容易的事。换言之，有了 JDBC API，就不必为访问 Sybase 数据库专门写一个程序，为访问 Oracle 数据库又专门写一个程序，为访问 Informix 数据库又写另一个程序，等等。您只需用 JDBC API 写一个程序就够了，它可向相应数据库发送 SQL 语句。而且，使用 Java 编程语言编写的应用程序，就无须去忧虑要为不同的平台编写不同的应用程序。将 Java 和 JDBC 结合起来将使程序员只须写一遍程序就可让它在任何平台上运行。

下面为常用的处理流程：

|  |
| --- |
| DriverManager |

|  |
| --- |
| Connection |

|  |
| --- |
| Statement |

|  |
| --- |
| PreparedStatement |

|  |
| --- |
| CallableStatement |

|  |
| --- |
| ResultSet |

简单地说，JDBC 可做三件事：

1.         与数据库建立连接

2.         发送 SQL 语句

3.         处理结果

下列代码段给出了以上三步的基本示例：

Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();

Connection conn = DriverManager.getConnection (

"jdbc:oracle:thin:@eai-sol:1521:eai\_db", "csc2", "csc2");

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT CONTACTID FROM CONTACTINFO");

while (rs.next()) {long contactID = rs.getLong("CONTACTID");}

下面对常用的几个类和接口做些简单的说明。

**DriverManager:**

DriverManager类是 JDBC 的管理层，作用于用户和驱动程序之间。它跟踪可用的驱动程序，并在数据库和相应驱动程序之间建立连接。另外，DriverManager 类也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等事务。对于简单的应用程序，一般程序员需要在此类中直接使用的唯一方法是 DriverManager.getConnection。正如名称所示，该方法将建立与数据库的连接。

Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();

Connection conn = DriverManager.getConnection (

"jdbc:oracle:thin:@eai-sol:1521:eai\_db", "csc2", "csc2");

其中第一句话的作用是在当前的环境中load一个DB Driver， 有人可能觉得奇怪， 这句话执行完之后， 后面怎么知道去用这个Driver呢？ 其实DriverManager可以从load的classes里面找到注册过的driver，然后使用它所找到的第一个可以成功连接到给定 URL 的驱动程序。 第二句话的三个参数分别是URL, User, Password。Ｄriver不一样，　URL可能也不一样。

**Statement:**

Statement 对象用于将 SQL 语句发送到数据库中。实际上有三种 Statement 对象，它们都为在给定连接上执行 SQL 语句的包容器：Statement、PreparedStatement（它从 Statement 承而来）和 CallableStatement（它从 PreparedStatement 继承而来）。它们都专用于发送定类型的 SQL 语句： Statement 对象用于执行不带参数的简单 SQL 语句；PreparedStatement 对象用于执行带或不带 IN 参数的预编译 SQL 语句；CallableStatement 对象用于执行对数据库已存储过程的调用。

Statement 接口提供了执行语句和获取结果的基本方法。PreparedStatement 接口添加了处理 IN 参数的方法；而 CallableStatement 添加了处理 OUT 参数的方法。

         创建 Statement 对象  
建立了到特定数据库的连接之后，就可用该连接发送 SQL 语句。Statement 对象用 Connection 的方法 createStatement 创建，如下列代码段中所示：  
Statement stmt = conn.createStatement();  
  
为了执行 Statement 对象，被发送到数据库的 SQL 语句将被作为参数提供给 Statement 的方法：   
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM table1");

         使用 Statement 对象执行语句

Statement 接口提供了三种执行 SQL 语句的方法：executeQuery、executeUpdate 和execute。使用哪一个方法由 SQL 语句所产生的内容决定。  
方法 executeQuery 用于产生单个结果集的语句，例如 SELECT 语句。

方法 executeUpdate 用于执行 INSERT、UPDATE 或 DELETE 语句以及 SQL DDL（数据定义语言）语句，例如 CREATE TABLE 和 DROP TABLE。INSERT、UPDATE 或 DELETE 语句的效果是修改表中零行或多行中的一列或多列。executeUpdate 的返回值是一个整数，指示受影响的行数（即更新计数）。对于 CREATE TABLE 或 DROP TABLE 等不操作行的语句，executeUpdate 的返回值总为零。

方法 execute 用于执行返回多个结果集、多个更新计数或二者组合的语句。

         语句完成

当连接处于自动提交模式时，其中所执行的语句在完成时将自动提交或还原。语句在已执行且所有结果返回时，即认为已完成。对于返回一个结果集的 executeQuery 方法，在检索完 ResultSet 对象的所有行时该语句完成。对于方法 executeUpdate，当它执行时语句即完成。但在少数调用方法 execute 的情况中，在检索所有结果集或它生成的更新计数之后语句才完成。

         关闭 Statement 对象

Statement 对象将由 Java 垃圾收集程序自动关闭。而作为一种好的编程风格，应在不需要 Statement 对象时显式地关闭它们。这将立即释放 DBMS 资源，有助于避免潜在的内存问题。 关闭Statement用 stmt.close()　方法。

**ResultSet:**

ResultSet 包含符合 SQL 语句中条件的所有行，并且它通过一套 get 方法（这些 get 方法可以访问当前行中的不同列）提供了对这些行中数据的访问。ResultSet.next 方法用于移动到 ResultSet 中的下一行，使下一行成为当前行。

结果集一般是一个表，其中有查询所返回的列标题及相应的值。例如，如果查询为 SELECT a, b, c FROM Table1，则结果集将具有如下形式：

a        b         c

-------- --------- --------

12345    Cupertino CA

83472    Redmond   WA

83492    Boston    MA  
下面的代码段是执行 SQL 语句的示例。该 SQL 语句将返回行集合，其中列 1 为 int，列 2 为 String，而列 3 则为日期型：

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

while (rs.next()) {

int i = rs.getInt("a");

String s = rs.getString("b");

Timestamp t = rs.getTimestamp("c");

}

         行和光标

ResultSet 维护指向其当前数据行的光标。每调用一次 next 方法，光标向下移动一行。最初它位于第一行之前，因此第一次调用 next 将把光标置于第一行上，使它成为当前行。随着每次调用 next 导致光标向下移动一行，按照从上至下的次序获取 ResultSet 行。在 ResultSet 对象或其父辈 Statement 对象关闭之前，光标一直保持有效。

         列

方法 getXXX 提供了获取当前行中某列值的途径。在每一行内，可按任何次序获取列值。但为了保证可移植性，应该从左至右获取列值，并且一次性地读取列值。列名或列号可用于标识要从中获取数据的列。例如，如果 ResultSet 对象 rs 的第二列名为“title”，并将值存储为字符串，则下列任一代码将获取存储在该列中的值：

String s = rs.getString("title");

String s = rs.getString(2);

注意列是从左至右编号的，并且从列 1 开始。同时，用作 getXXX 方法的输入的列名不区分大小写。 **为了代码的可维护性与可读性，　应该禁止用index的方法来取值，　要用读列名的方法，　如上面的第一行取值方法。**

         数据类型和转换

对于 getXXX 方法，JDBC 驱动程序试图将基本数据转换成指定 Java 类型，然后返回适合的 Java 值。例如，如果 getXXX 方法为 getString，而基本数据库中数据类型为 VARCHAR，则 JDBC 驱动程序将把 VARCHAR 转换成 Java String。getString 的返回值将为 Java String 对象。

         NULL 结果值

要确定给定结果值是否是 JDBC NULL，必须先读取该列，然后使用 ResultSet.wasNull 方法检查该次读取是否返回 JDBC NULL。

**PreparedStatement:**

该 PreparedStatement 接口继承 Statement，并与之在两方面有所不同：

PreparedStatement 实例包含已编译的 SQL 语句。这就是使语句“准备好”。 包含于 PreparedStatement 对象中的 SQL 语句可具有一个或多个 IN 参数。IN 参数的值在 SQL 语句创建时未被指定。相反的，该语句为每个 IN 参数保留一个问号（“？”）作为占位符。每个问号的值必须在该语句执行之前，通过适当的 setXXX 方法来提供。

由于 PreparedStatement 对象已预编译过，所以其执行速度要快于 Statement 对象。因此，多次执行的 SQL 语句经常创建为 PreparedStatement 对象，以提高效率。

作为 Statement 的子类，PreparedStatement 继承了 Statement 的所有功能。另外它还添加了一整套方法，用于设置发送给数据库以取代 IN 参数占位符的值。同时，三种方法 execute、 executeQuery 和 executeUpdate 已被更改以使之不再需要参数。

         创建 PreparedStatement 对象

以下的代码段（其中 conn 是 Connection 对象）创建包含带两个 IN 参数占位符的 SQL 语句的 PreparedStatement 对象：

PreparedStatement pstmt=conn.prepareStatement("UPDATE table1 SET a = ? WHERE b = ?");

pstmt 对象包含语句 " UPDATE table1 SET a = ? WHERE b = ?"，它已发送给 DBMS，并为执行作好了准备。

         传递 IN 参数

在执行 PreparedStatement 对象之前，必须设置每个 ? 参数的值。这可通过调用 setXXX 方法来完成，其中 XXX 是与该参数相应的类型。例如，如果参数具有 Java 类型 long，则使用的方法就是 setLong。setXXX 方法的第一个参数是要设置的参数的序数位置，第二个参数是设置给该参数的值。例如，以下代码将第一个参数设为 123456789，第二个参数设为 100000000：

pstmt.setLong(1, 123456789);

pstmt.setLong(2, 100000000);

**CallableStatement:**

CallableStatement 对象为所有的 DBMS 提供了一种以标准形式调用已储存过程(也就是ＳＰ)的方法。已储存过程储存在数据库中。对已储存过程的调用是 CallableStatement 对象所含的内容。有两种形式：一种形式带结果参数，另一种形式不带结果参数。结果参数是一种输出 (OUT) 参数，是已储存过程的返回值。两种形式都可带有数量可变的输入（IN 参数）、输出（OUT 参数）或输入和输出（INOUT 参数）的参数。问号将用作参数的占位符。

在 JDBC 中调用已储存过程的语法如下所示。注意，方括号表示其间的内容是可选项；方括号本身并不是语法的组成部份。

{call 过程名[(?, ?, ...)]}

返回结果参数的过程的语法为：

{? = call 过程名[(?, ?, ...)]}

不带参数的已储存过程的语法类似：

{call 过程名}

通常，创建 CallableStatement 对象的人应当知道所用的 DBMS 是支持已储存过程的，并且知道这些过程都是些什么。然而，如果需要检查，多种 DatabaseMetaData 方法都可以提供这样的信息。例如，如果 DBMS 支持已储存过程的调用，则 supportsStoredProcedures 方法将返回 true，而 getProcedures 方法将返回对已储存过程的描述。

CallableStatement 继承 Statement 的方法（它们用于处理一般的 SQL 语句），还继承了 PreparedStatement 的方法（它们用于处理 IN 参数）。CallableStatement 中定义的所有方法都用于处理 OUT 参数或 INOUT 参数的输出部分：注册 OUT 参数的 JDBC 类型（一般 SQL 类型）、从这些参数中检索结果，或者检查所返回的值是否为 JDBC NULL。

         创建 CallableStatement 对象

CallableStatement 对象是用 Connection 方法 prepareCall 创建的。下例创建 CallableStatement 的实例，其中含有对已储存过程 Csc\_ GetCustomId调用。该过程有两个变量，但不含结果参数：

CallableStatement cstmt = con.prepareCall("{call CSC\_GetCustomId (?, ?, ?)}");

其中 ? 占位符为 IN、 OUT 还是 INOUT 参数，取决于已储存过程 Csc\_ GetCustomId。

         IN 和 OUT 参数

将IN 参数传给 CallableStatement 对象是通过 setXXX 方法完成的。该方法继承自 PreparedStatement。所传入参数的类型决定了所用的 setXXX 方法（例如，用 setFloat 来传入 float 值等）。

如果已储存过程返回 OUT 参数，则在执行 CallableStatement 对象以前必须先注册每个 OUT 参数的 JDBC 类型（这是必需的，因为某些 DBMS 要求 JDBC 类型）。注册 JDBC 类型是用 registerOutParameter 方法来完成的。语句执行完后，CallableStatement 的 getXXX 方法将取回参数值。正确的 getXXX 方法是为各参数所注册的 JDBC 类型所对应的 Java 类型也就是说， registerOutParameter 使用的是 JDBC 类型（因此它与数据库返回的 JDBC 类型匹配），而 getXXX 将之转换为 Java 类型。下面给出CSC中的一个例子：

String sqlSp = "{call CSC\_GetCustomId(?, ?, ?)}";

cstmt = conn.prepareCall(sqlSp.toString());

cstmt.registerOutParameter(1, Types.NUMERIC);

cstmt.registerOutParameter(2, Types.NUMERIC);

cstmt.registerOutParameter(3, Types.VARCHAR);

cstmt.execute();

long customerID = cstmt.getLong(1);

long lRet = cstmt.getLong(2);

String sErr = cstmt.getString(3);

         INOUT 参数

既支持输入又接受输出的参数（INOUT 参数）除了调用 registerOutParameter 方法外，还要求调用适当的 setXXX 方法（该方法是从 PreparedStatement 继承来的）。setXXX 方法将参数值设置为输入参数，而 registerOutParameter 方法将它的 JDBC 类型注册为输出参数。setXXX 方法提供一个 Java 值，而驱动程序先把这个值转换为 JDBC 值，然后将它送到数据库中。这种 IN 值的 JDBC 类型和提供给 registerOutParameter 方法的 JDBC 类型应该相同。然后，要检索输出值，就要用对应的 getXXX 方法。例如，Java 类型为 byte 的参数应该使用方法 setByte 来赋输入值。应该给 registerOutParameter 提供类型为 TINYINT 的 JDBC 类型，同时应使用 getByte 来检索输出值。下例假设有一个已储存过程 reviseTotal，其唯一参数是 INOUT 参数。方法 setByte 把此参数设为 25，驱动程序将把它作为 JDBC TINYINT 类型送到数据库中。接着，registerOutParameter 将该参数注册为 JDBC TINYINT。执行完该已储存过程后，将返回一个新的 JDBC TINYINT 值。方法 getByte 将把这个新值作为 Java byte 类型检索。

CallableStatement cstmt = con.prepareCall("{call reviseTotal(?)}");

cstmt.setByte(1, 25);

cstmt.registerOutParameter(1, java.sql.Types.TINYINT);

cstmt.executeUpdate();

byte x = cstmt.getByte(1);

### 11. 常用设计模式

**1．Singleton模式**

Singleton模式主要作用是保证在Java应用程序中，一个Class只有一个实例存在。一般有两种方法：

         定义一个类，它的构造函数为private的，所有方法为static的。其他类对它的引用全部是通过类名直接引用。例如：

private SingleClass() {}

public static String getMethod1() {}

public static ArrayList getMethod2() {}

         定义一个类，它的构造函数为private的，它有一个static的private的该类变量，通过一个public的getInstance方法获取对它的引用,继而调用其中的方法。例如：

private staitc SingleClass \_instance = null;

private SingleClass() {}

public static SingleClass getInstance() {

    if (\_instance == null) {\_instance = new SingleClass();}

    return \_instance; }

public String getMethod1() {}

public ArrayList getMethod2() {}

**2．Prototype模式**

Prototype模式用于创建对象，尤其是当创建对象需要许多时间和资源时。在Java中，Prototype模式的实现是通过方法clone(),该方法定义在Java的根对象Object中, 因此，Java中的其他对象只要覆盖它就行了。通过clone(),我们可以从一个对象获得更多的对象，

并请可以按照我们的需要修改他们的属性。

public class Prototype implements Cloneable {

  private String Name;

  public rototype(String Name) {this.Name = Name; }

  public void setName(String Name) {this.Name = Name; }

  public String getName() {return Name; }

  public Object clone() {

      try{return super.clone();

      }catch(CloneNotSupportedException cnse){

          cnse.printStackTrace();

          return null;

      }

  }}

Prototype p = new Prototype("My First Name");

Prototype p1 = p.clone();

p.setName("My Second Name");

Prototype p2 = p.clone();

**3. Factory模式和Abstract Factory模式**

         Factory模式

利用给Factory对象传递不同的参数，以返回具有相同基类或实现了同一接口的对象。

         Abstract Factory模式

先利用Factory模式返回Factory对象，在通过Factory对象返回不同的对象！

下面给出Sun XML Parser中的例子：

// 1. Abstract Factory模式

SAXParserFactory spf = SAXParserFactory.newInstance();

String validation = System.getProperty ("javax.xml.parsers.validation", "false");

if (validation.equalsIgnoreCase("true")) {spf.setValidating (true); }

// 2. Factory模式

SAXParser sp = spf.newSAXParser();

parser = sp.getParser();

parser.setDocumentHandler(this);

parser.parse (uri);

1. SAXParserFactory中的静态方法newInstance()根据系统属性javax.xml.parsers.SAXParserFactory不同的值生成不同的SAXParserFactory对象spf。然后SAXParserFactory对象又利用方法newSAXParser()生成SAXParser对象。

注意：

SAXParserFactory 的定义为：

public abstract class SAXParserFactory extends java.lang.Object

SAXParserFactoryImpl 的定义为：

public class SAXParserFactoryImpl extends javax.xml.parsers.SAXParserFactory

public static SAXParserFactory newInstance() {

String factoryImplName = null;

try {

     factoryImplName =System.getProperty("javax.xml.parsers.SAXParserFactory",

"com.sun.xml.parser.SAXParserFactoryImpl");

} catch (SecurityException se) {

factoryImplName = "com.sun.xml.parser.SAXParserFactoryImpl";

}

SAXParserFactory factoryImpl;

try {

     Class clazz = Class.forName(factoryImplName);

     factoryImpl = (SAXParserFactory) clazz.newInstance();

} catch (ClassNotFoundException cnfe) {

     throw new FactoryConfigurationError(cnfe);

} catch (IllegalAccessException iae) {

     throw new FactoryConfigurationError(iae);

} catch (InstantiationException ie) {

     throw new FactoryConfigurationError(ie);

}

return factoryImpl;

}

2. newSAXParser() 方法在SAXParserFactory定义为抽象方法，

SAXParserFactoryImpl继承了SAXParserFactory，它实现了方法newSAXParser()：

public SAXParser newSAXParser()

throws SAXException, ParserConfigurationException {

SAXParserImpl saxParserImpl = new SAXParserImpl(this);

return saxParserImpl;

}

注意：

SAXParserImpl的定义为：

public class SAXParserImpl extends javax.xml.parsers.SAXParser

SAXParserImpl的构造函数定义为：

public SAXParserImpl(SAXParserFactory spf)

throws SAXException, ParserConfigurationException {

super();

this.spf = spf;

if (spf.isValidating ()) {

     parser = new ValidatingParser();

     validating = true;

} else { parser = new Parser();}

if (spf.isNamespaceAware ()) {

     namespaceAware = true;

     throw new ParserConfigurationException(

"Namespace not supported by SAXParser");

}

}