1. Java基础知识
2. Java Web
3. **名词解释**
4. JDK(Java Development Kit)

是Java开发工具包，它提供了Java的开发环境(提供了编译器javac等工具，用于将java文件编译为class文件)和运行环境(提供了JVM和Runtime辅助包，用于解析class文件使其得到运行)。如果你下载并安装了JDK，那么你不仅可以开发Java程序，也同时拥有了运行Java程序的平台。JDK是整个Java的核心，包括了Java运行环境(JRE)，一堆Java工具tools.jar和Java标准类库 (rt.jar)。

1. JRE(Java Runtime Enviroment)

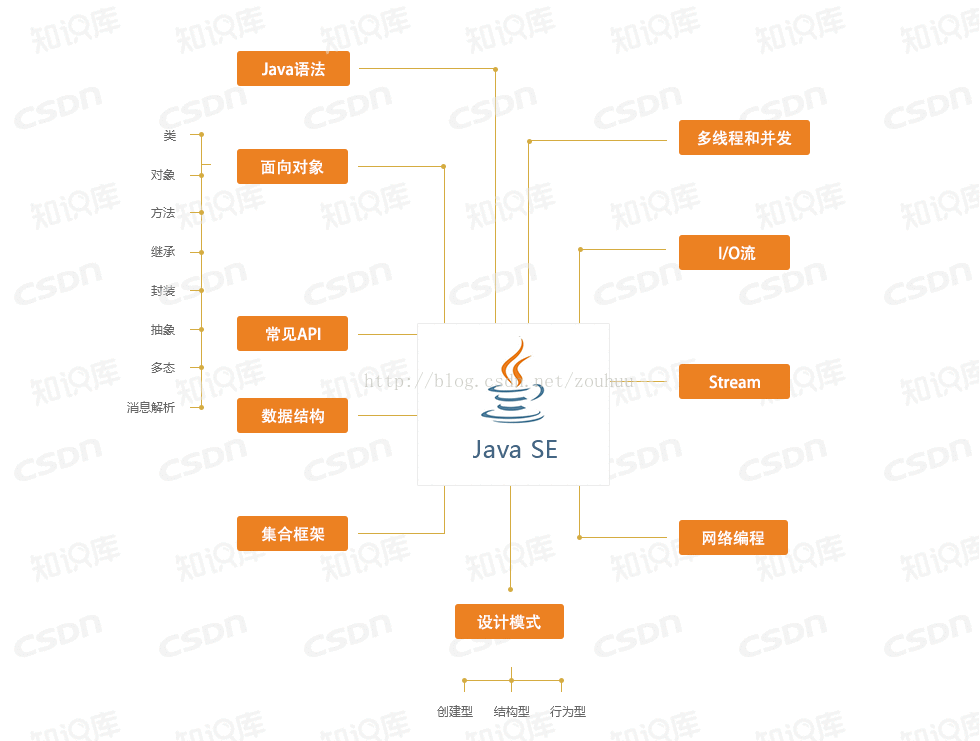
是Java的运行环境。面向Java程序的使用者，而不是开发者。如果你仅下载并安装了JRE，那么你的系统只能运行Java程序。JRE是运行Java程序所必须环境的集合，包含JVM标准实现及 Java核心类库。它包括Java虚拟机、Java平台核心类和支持文件。它不包含开发工具(编译器、调试器等)。

1. JVM(Java Virtual Machine)

字面意思“Java虚拟机”，简单来说就是一个可以运行Java的软件。我们在将.java编译后，会生成相应的.class文件， JVM就是加载并运行.class文件的软件。

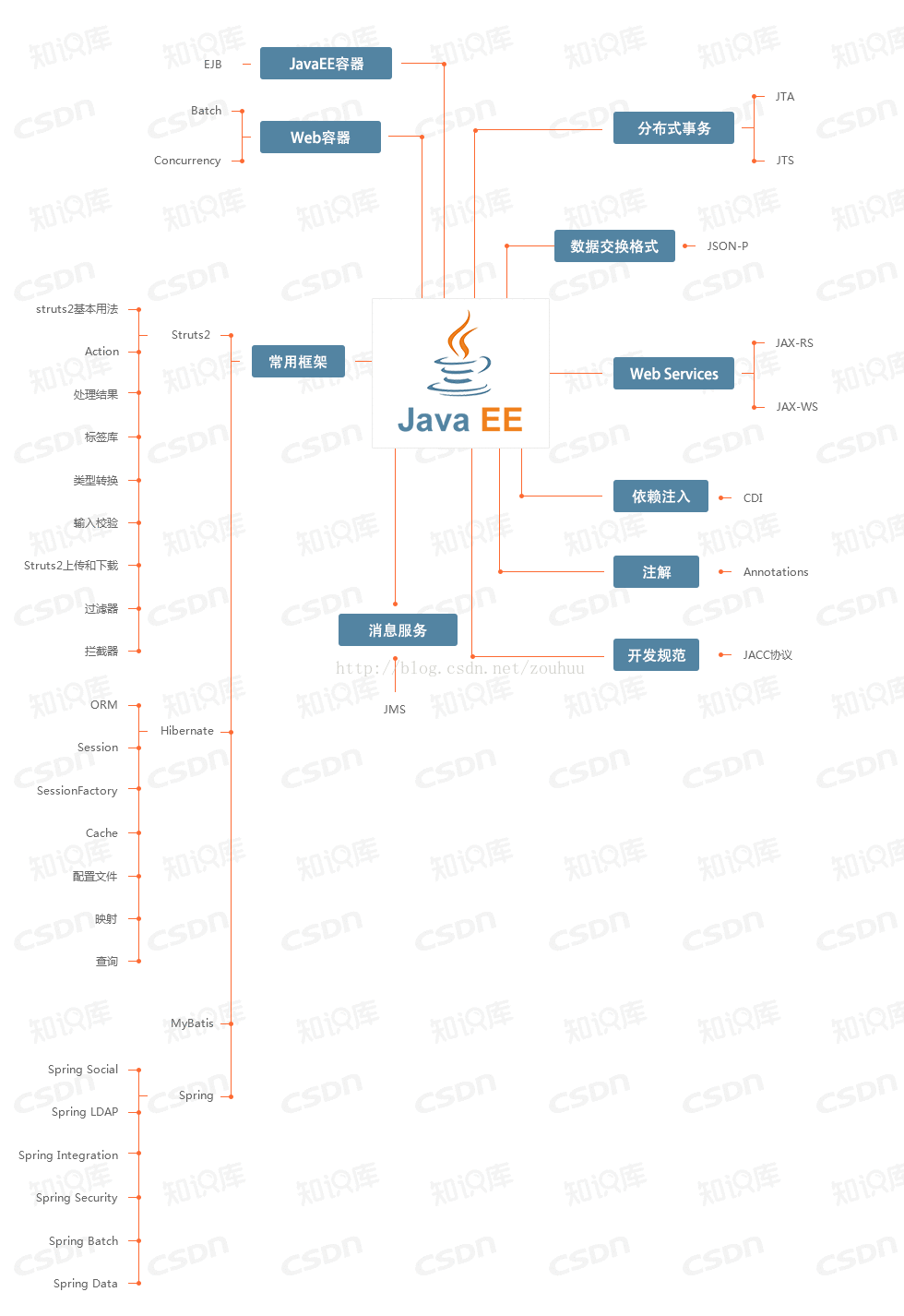
1. JAVASE

Java Standard Edition，Java标准版，就是一般Java程序的开发就可以(如桌面程序)，可以看作是JavaEE的子集。



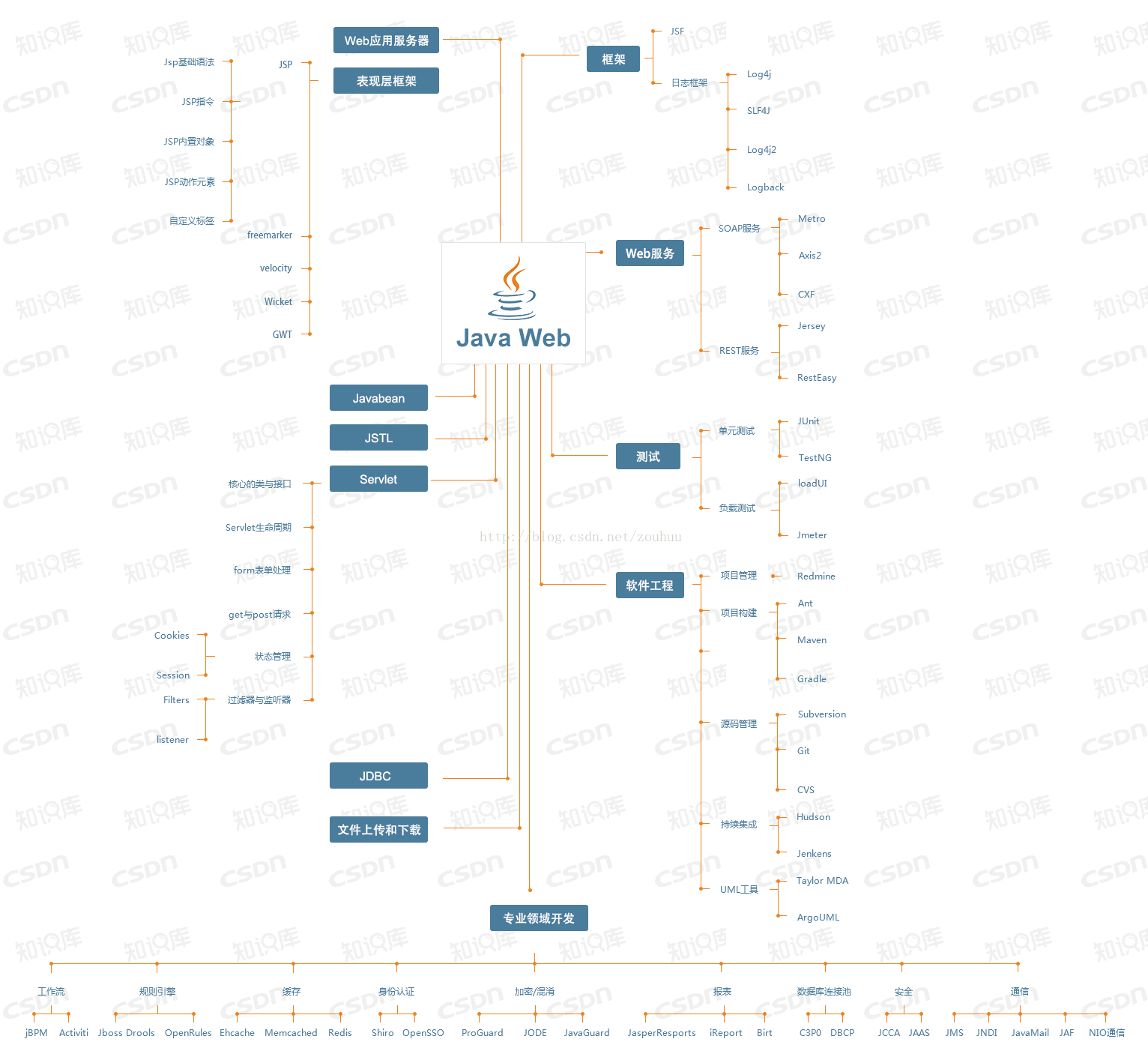
1. JAVAEE

Java Enterprise Edition，Java企业版，多用于企业级开发，包括web开发等等。



1. JAVAWEB

Java Web是指使用Java体系开发网站类应用



1. 数据库原理

**1、基本语法注意事项**

（1）大小写敏感

（2）项目名和包名（如com.demo）全部小写，“package 包名” 放在源程序第一句，之后是import语句，然后定义类

（3）类名：类名的首字母应该大写，例如 MyFirstJavaClass 。

（4）方法名：所有的方法名都应该以小写字母开头，后面的单词首字母大写，如setAge()。

（5）常量名全部大写。

（6）源文件名：源文件名必须和类名相同，文件名的后缀为.java。（如果文件名和类名不相同则会导致编译错误）。

（7）main方法入口：所有的Java 程序由public static void main(String []args)方法开始执行。main方法必须是静态的。

（8）Java中所有函数都属于某个类的方法，因此main方法必须有一个外壳类。

（9）一个Java文件中只能有一个public类，可以有多个非public类。

（10）

**2、Java变量（局部变量、成员变量、静态变量）**

Java中主要有如下几种类型的变量

1. 局部变量

在方法、构造方法、语句块中定义的变量。其声明和初始化在方法中实现，在栈上分配，在方法结束后自动销毁。局部变量没有默认值，所以局部变量被声明后，必须经过初始化，才可以使用。

1. 成员变量（实例变量，非静态变量）

定义在类中，方法体之外。变量在创建对象时实例化，具有默认值。成员变量可被类中的方法、构造方法以及特定类的语句块访问。

特点：

——成员变量随着对象的创建而存在，随着对象的回收而释放。

——成员变量只能被对象调用。

——成员变量存储在堆内存的对象中，所以也叫对象的特有数据。

（3）类变量（静态变量）

定义在类中，方法体之外，但必须要有 static 来声明变量类型。静态成员属于整个类，可通过对象名或类名来调用。想要实现对象中的共性数据的对象共享，可以将这个数据进行静态修饰

特点：

——无论一个类创建了多少个对象，类只拥有类变量的一份拷贝。

——静态变量随着类的加载而存在，随着类的消失而消失。

——静态变量可以被对象调用，还可以被类名调用。

——静态变量数据存储在方法区（共享数据区）的静态区，所以也叫对象的共享数据。

——静态随着类的加载而加载，而且优先于对象存在。

——静态方法只能访问静态成员，不可以访问非静态成员。因为静态方法加载时，优先于对象存在，所以没有办法访问对象中的成员。

——静态方法中不能使用this，super关键字。因为this代表对象，而静态在时，有可能没有对象，所以this无法使用。



**3、判断成员变量、成员函数是否需要静态修饰**

（1）成员变量（数据共享时静态化）

该成员变量的数据是否是所有对象都一样：

如果是，那么该变量需要被静态修饰，因为是共享的数据。

如果不是，那么就说这是对象的特有数据，要存储到对象中。

静态变量并不是说其就不能改变值，不能改变值的量叫常量。 其拥有的值是可变的 ，而且它会保持最新的值。

（2）成员函数（方法中没有调用特有数据时就定义成静态）

该函数内是否访问了对象中的特有数据：

如果有访问特有数据，那方法不能被静态修饰。

如果没有访问过特有数据，那么这个方法需要被静态修饰。

**4、堆、栈和方法区（Java内存管理）**

（1）Java堆

——堆内存用来存放由new创建的对象实例和数组。

——为编译器自动分配和释放，是垃圾收集器GC管理的主要区域。GC会在不确定的时间将没有任何引用的对象所占的内存释放掉。

——所有线程共享，在虚拟机启动时创建。

——如果堆中没有内存完成实例分配，并且堆也无法完成扩展时，将会抛出OutOfMemoryError异常。

（2）Java栈

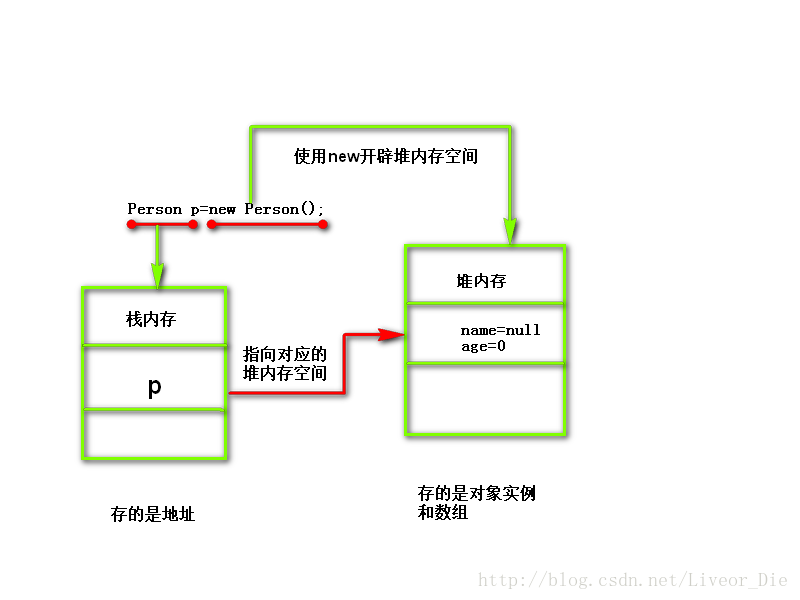
——栈内存中保存的是堆内存空间的访问地址（首地址），栈中的变量就成了堆内存中的变量的引用变量（Java中的指针）

——引用变量在程序运行到其作用域之外后被释放。

（3）方法区

方法区是各个线程共享的内存区域，它用于存储已被虚拟机加载的类信息、常量、静态变量等数据。

https://blog.csdn.net/Liveor\_Die/article/details/77895631



例题：JVM如何管理内存，分成几个部分？分别有什么用途？说出下面代码的内存实现原理：

Foo foo = new Foo();

foo.f();

参考答案：

　　JVM内存分为“堆”、“栈”和“方法区”三个区域，分别用于存储不同的数据。

　　堆内存用于存储使用new关键字所创建的对象；栈内存用于存储程序运行时在方法中声明的所有的局部变量；方法区用于存放类的信息，Java程序运行时，首先会通过类装载器载入类文件的字节码信息，经过解析后将其装入方法区。类的各种信息（包括方法）都在方法区存储。

以上代码的内存实现原理为：

　　1.Foo类首先被装载到JVM的方法区，其中包括类的信息，包括方法和构造等。

　　2.在栈内存中分配引用变量foo。

　　3.在堆内存中按照Foo类型信息分配实例变量内存空间；然后，将栈中引用foo指向foo对象堆内存的首地址。

　　4.使用引用foo调用方法，根据foo引用的类型Foo调用f方法。

**5、Java的两大数据类型（内置、引用）**

（1）内置数据类型

Java语言提供了八种基本类型。

1. **byte：**

byte 数据类型是8位、有符号的，以二进制补码表示的整数；

最小值是 -128（-2^7）；最大值是 127（2^7-1）；

默认值是 0；

byte 类型用在大型数组中节约空间，主要代替整数。

1. **short：**

short 数据类型是 16 位、有符号的以二进制补码表示的整数

最小值是 -32768（-2^15）；最大值是 32767（2^15 - 1）；

默认值是 0；

Short 数据类型也可以像 byte 那样节省空间。

1. **int：**

int 数据类型是32位、有符号的以二进制补码表示的整数；

最小值是 -2,147,483,648（-2^31）；最大值是 2,147,483,647（2^31 - 1）；

默认值是 0；

一般地整型变量默认为 int 类型；

1. **long：**

long 数据类型是 64 位、有符号的以二进制补码表示的整数；

最小值是 -9,223,372,036,854,775,808（-2^63）；

最大值是 9,223,372,036,854,775,807（2^63 -1）；

默认值是 0L；

这种类型主要使用在需要比较大整数的系统上；

例子： long a = 100000L，Long b = -200000L。

Java 里使用 long 类型的数据一定要在数值后面加上 L，否则将作为整型解析

1. **float：**

float 数据类型是单精度、32位、符合IEEE 754标准的浮点数；

float 在储存大型浮点数组的时候可节省内存空间；

默认值是 0.0f；

浮点数不能用来表示精确的值，如货币；

1. **double：**

double 数据类型是双精度、64 位、符合IEEE 754标准的浮点数；

浮点数的默认类型为double类型；

double类型同样不能表示精确的值，如货币；

默认值是 0.0d；

1. **boolean：**

boolean数据类型表示一位的信息；

只有两个取值：true 和 false；

默认值是 false；

1. **char：**

char类型是一个单一的 16 位 Unicode 字符；

最小值是 \u0000（即为0）；最大值是 \uffff（即为65,535）；

char 数据类型可以储存任何字符；

例如 char a = 'S'; //char 后面赋值要用单引号

注意：String a = "I AM FINE"; //String 后面赋值要用双引号，因为是字符串数据类型

（2）引用数据类型

引用类型是一个对象类型，它的值是地址（类似于C++指针），所指向的内存中保存着变量所表示的一个值或一组值。

对象、数组都是引用数据类型。

所有引用类型的默认值都是null。

MyDate a，b; // 在内存开辟两个引用空间

a = new MyDate(); // 开辟MyDate对象的数据空间，并把该空间的首地址赋给a

b = a; // 将a存储空间中的地址写到b的存储空间中

* **Java常量**

常量在程序运行时是不能被修改的。使用 final 关键字来修饰常量，通常使用大写字母表示，声明方式和变量类似：

final double PI = 3.1415927;

byte、int、long、和short都可以用十进制、16进制以及8进制的方式来表示。当使用常量的时候，前缀 0 表示 8 进制，而前缀 0x 代表 16 进制, 例如：

int decimal = 100;

int octal = 0144;

int hexa = 0x64;

* **数据类型转换**

（a）隐含强制类型转换

在Java中，如果比int类型小的类型做运算，在编译的时候就会将它们统一强转成int类型。当是比int类型大的类型做运算，就会自动转换成它们中最大类型那个。

（b）强制类型转换

int a = 123;

byte b = (byte)a; //强制类型转换为byte

（c）包装类过渡类型转换

首先声明一个变量，然后生成一个对应的包装类，就可以利用包装类的各种方法进行类型转换了。

当希望把float型转换为double型时：

float f1=100.00f;

Float F1=new Float(f1);

double d1=F1.doubleValue(); //F1.doubleValue()为Float类的返回double值型的方法

**6、Java修饰符（访问控制修饰符、非访问修饰符）**

（1）访问控制修饰符

default (即缺省，什么也不写）: 在同一包内可见，不使用任何修饰符。使用对象：类、接口、变量、方法。

private : 在同一类内可见。使用对象：变量、方法。 注意：不能修饰类（外部类）

public : 对所有类可见。使用对象：类、接口、变量、方法

protected : 对同一包内的类和所有子类可见。使用对象：变量、方法。 注意：不能修饰类（外部类）。



**访问控制和继承规则：**

父类中声明为 public 的方法在子类中也必须为 public。

父类中声明为 protected 的方法在子类中要么声明为 protected，要么声明为 public，不能声明为 private。

父类中声明为 private 的方法，不能够被继承。

（2）非访问修饰符

——static 修饰符，使用对象：类、变量、方法、初始化函数

——final 修饰符，使用对象：类、变量、方法。final 修饰的类不能够被继承；修饰的方法可以被继承，但不能被继承类重新定义（声明 final 方法的主要目的是防止该方法的内容被修改）；修饰的变量为常量，是不可修改的（只能初始化一次）。final 修饰符通常和 static 修饰符一起使用来创建类常量。

——abstract 修饰符，用来创建抽象类和抽象方法。使用对象：类、接口、方法。抽象类不能用来实例化对象，声明抽象类的唯一目的是为了将来对该类进行扩充；抽象方法是一种没有任何实现的方法，声明以分号结尾，如：public abstract sample();，该方法的的具体实现由子类提供；任何继承抽象类的子类必须实现父类的所有抽象方法，除非该子类也是抽象类；抽象类可以不包含抽象方法；如果一个类包含抽象方法，那么该类一定要声明为抽象类，否则将出现编译错误。

——synchronized 和 volatile 修饰符，主要用于线程的编程。synchronized 关键字声明的方法同一时间只能被一个线程访问。

——transient 修饰符：序列化（写入字节序列到目标文件）的对象包含被 transient 修饰的实例变量时， JVM跳过该特定的变量。该修饰符包含在定义变量的语句中，用来预处理类和变量的数据类型。当对象被反序列化时（从源文件读取字节序列进行重构），这样的实例变量值不会被持久化和恢复。例如：

public transient int limit = 55; // 不会持久化

public int b; // 持久化

——volatile修饰符：volatile修饰的成员变量在每次被线程访问时，都强制从共享内存中重新读取该成员变量的值。而且，当成员变量发生变化时，会强制线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的同一个值。使用的场景之一，单例模式中采用DCL双锁检测（double checked locking）机制，在多线程访问的情况下，可使用volatitle修改，保证多线程下的可见性。缺点是性能有损失，因此单线程情况下不必用此修饰符。

**7、Java运算符**

算术运算符、关系运算符、位运算符、逻辑运算符、赋值运算符、其他运算符

（1） 位运算符

作用于每一位，将int,long,short,char,byte类型转为二进制，

& ：相对应位都是1则为1，否则为0

| ：相对应位有1则为1，都是0则为0

^ ：相对应位值相同则为0，否则为1

~ ：按位补运算符翻转每一位，1变为0，0变为1

<< ：按位左移，左边的操作数按右操作数指定的位数左移

>> ：按位右移

>>> ：按位右移补零，移动得到的空位以0填充

（2） 逻辑运算符

与&&、或||、非！

（3） 条件运算符 （?:）

variable x = (expression) ? value if true : value if false

（4） instanceof 运算符

该运算符用于操作对象实例，检查该对象是否是一个特定类型（类类型或接口类型）。

( Object reference variable ) instanceof (class/interface type)

如果运算符左侧变量所指的对象，是操作符右侧类或接口(class/interface)的一个对象，那么结果为真。例如：

String name = "James";

boolean result = name instanceof String; // 由于 name 是 String 类型，所以返回真

注意：在判断一个实例引用的类型时，使用的是实际类型，而不是声明的类型。子类是父类的类型，但父类不是子类的类型。

**8、Java循环结构**

（1） while循环

while( 布尔表达式 ) {

//循环内容 }

（2）do…while 循环

不同于while，**至少会执行一次**。

do {

//代码语句

}while(布尔表达式);

所以语句块在检测布尔表达式之前已经执行了。 如果布尔表达式的值为 true，则语句块一直执行，直到布尔表达式的值为 false。

（3）for循环

for(初始化; 布尔表达式; 更新) {

//代码语句

}

**几点说明：**

最先执行初始化步骤。可以声明一种类型，但可初始化一个或多个循环控制变量，也可以是空语句。

然后，检测布尔表达式的值。如果为 true，循环体被执行。如果为false，循环终止，开始执行循环体后面的语句。

执行一次循环后，更新循环控制变量。

再次检测布尔表达式。循环执行上面的过程。

（4）Java增强for循环

for(声明语句 : 表达式)

{ //代码句子 }

声明语句：声明新的局部变量，该变量的类型必须和数组元素的类型匹配。其作用域限定在循环语句块，其值与此时数组元素的值相等。

表达式：表达式是要访问的数组名，或者是返回值为数组的方法。

如：

int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ){ System.out.print(x) }

String [] names ={"James", "Larry", "Tom", "Lacy"};

for( **String name : names** ) { System.out.print(name); }

（5）break关键字

break 主要用在循环语句或者 switch 语句中，用来跳出整个语句块。

break 跳出最里层的循环，并且继续执行该循环下面的语句。

（6）continue关键字

continue 适用于任何循环控制结构中。作用是让程序立刻跳转到下一次循环的迭代。

在 for 循环中，continue 语句使程序立即跳转到更新语句。

在 while 或者 do…while 循环中，程序立即跳转到布尔表达式的判断语句。

**9、Java分支结构**

（1）if...else语句

if语句至多有 1 个else语句，可以有若干个elseif语句，else语句在所有的elseif语句之后。

一旦其中一个 else if 语句检测为 true，其他的 else if 以及 else 语句都将跳过执行。

（2）switch 语句

当变量的值与 case 语句的值相等时，那么 case 语句之后的语句开始执行，直到 break 语句出现才会跳出 switch 语句。

当遇到 break 语句时，switch 语句终止。程序跳转到 switch 语句后面的语句执行。case 语句不必须要包含 break 语句。如果没有 break 语句出现，程序会继续执行下一条 case 语句，直到出现 break 语句。

switch 语句可以包含一个 default 分支，该分支必须是 switch 语句的最后一个分支。default 在没有 case 语句的值和变量值相等的时候执行。default 分支不需要 break 语句。

**10、Java Number & Math 类**

（1） Number类

Number 类属于 java.lang 包。

所有的**包装类**（Integer、Long、Byte、Double、Float、Short）都是抽象类 Number 的子类。

这种由编译器特别支持的包装称为装箱，所以当内置数据类型被当作对象使用的时候，编译器会把内置类型装箱为包装类。相似的，编译器也可以把一个对象拆箱为内置类型。

常用方法：

xxxValue()：将Number对象转换为xxx数据类型的值并返回，无参数，如intValue()

compareTo()：将number对象与参数比较，相等返回0，小于参数返回-1，大于返回1

equals()：判断number对象是否与参数相等，返回True和False

valueOf()：返回给定参数的原生Number对象值，参数可以是原生数据类型， String等.

toString()：以字符串形式返回值。

parseInt()：将字符串解析为int类型，类似的还有parseDouble()

（2） Math类

Math 的方法都被定义为static形式，通过Math类可以在主函数中直接调用。

常用方法：

abs(),floor(),round(), min(), max(),log(),sqrt(), sin(),

ceil()：向上取整，返回类型为double

exp()：用于返回自然数底数e的参数次方

rint()：返回与参数最接近的整数。返回类型为double。

pow()：返回第一个参数的第二个参数次方。

asin()：求指定double类型参数的反正弦值。

toDegrees()：将参数转化为角度，返回double

toRadians()：将角度转换为弧度，返回double

random()：返回一个随机数，无参数，返回double

**11、Java Character类**

java.lang.Character

用于对单个字符进行操作，Character类在对象中包装一个基本类型char的值。

实例：

char ch = 'a'; // Unicode 字符表示形式

char uniChar = '\u039A'; // 字符数组

char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };

Character ch = new Character('a');

**12、Java String类**

Java中字符串属于对象，String类用于创建和操作字符串。

（1）创建字符串

String类有11中构造方法，提供不同参数来初始化字符串，如：提供字符数组参数

char[] helloArray = { 'r', 'u', 'n', 'o', 'o', 'b'};

String helloString = new String(helloArray);

注意：

String类是***不可改变***的，一旦创建了String对象，它的值就无法改变了。

如果需要对字符串做很多修改，应该使用***StringBuffer & StringBuilder***类。

（2）字符串长度

String 是被 final 修饰的，他的长度是不可变的，调用String的concat方法是把字符串拼接起来并重新创建一个对象，把拼接后的String的值赋给新创建的对象。

访问器方法：获取有关对象的信息的方法。

String 类的一个访问器方法是length()方法，返回字符串对象包含的字符数。

String site = "www.runoob.com";

int len = site.length();

（3）连接字符串

两个方法：concat()方法（将指定字符串连接到此字符串的结尾），或使用“+”操作符连接字符串

string1.concat(string2);

"我的名字是 ".concat("Runoob");

"Hello," + " runoob" + "!"

（4）创建格式化字符串

输出格式化数字可以使用 printf() 和 format() 方法。

String 类使用静态方法 format() 返回一个String 对象而不是 PrintStream 对象。

String 类的静态方法 format() 能用来创建可复用的格式化字符串，而不仅仅是用于一次打印输出。

System.out.printf("浮点型变量的值为 " +

"%f, 整型变量的值为 " +

" %d, 字符串变量的值为 " +

"is %s", floatVar, intVar, stringVar);

也可以这样写：

String fs;

fs = String.format("浮点型变量的值为 " +

"%f, 整型变量的值为 " +

" %d, 字符串变量的值为 " +

" %s", floatVar, intVar, stringVar);

**String类支持的方法**：详见http://www.runoob.com/java/java-string.html

-- [int compareTo(Object o)](http://www.runoob.com/java/java-string-compareto.html) 或 [int compareTo(String anotherString)](http://www.runoob.com/java/java-string-compareto.html)

用于两种方式的比较，字符串与对象进行比较，或按字典顺序比较两个字符串（ASCII码顺序）。若相等，返回0；否则返回第一个不等的字符的差值，参数值较小，返回值>0，反之<0

-- [char charAt(int index)](http://www.runoob.com/java/java-string-charat.html)

返回指定索引处的 char 值。

-- [static String copyValueOf(char[] data)](http://www.runoob.com/java/java-string-copyvalueof.html) 或 [static String copyValueOf(char[] data, int offset, int count)](http://www.runoob.com/java/java-string-copyvalueof.html)

返回指定数组中表示该字符序列的 String。

-- [int hashCode()](http://www.runoob.com/java/java-string-hashcode.html)

返回此字符串的哈希码。

-- [String toUpperCase()](http://www.runoob.com/java/java-string-touppercase.html)

使用默认语言环境的规则将此 String 中的所有字符都转换为大写。

-- [String trim()](http://www.runoob.com/java/java-string-trim.html)

返回字符串的副本，忽略前导空白和尾部空白。

-- [static String valueOf(primitive data type x)](http://www.runoob.com/java/java-string-valueof.html)

返回给定data type类型x参数的字符串表示形式，x可以为boolean, char, char[], double, float, int, long, Object。

**13、Java StringBuffer 和 StringBuilder 类**

当对字符串进行修改的时候，需要使用 StringBuffer 和 StringBuilder 类。

和 String 类不同的是，StringBuffer 和 StringBuilder 类的对象能够被多次的修改，并且不产生新的未使用对象。

StringBuilder 的方法不是线程安全的（不能同步访问）,但有速度优势，所以多数情况下建议使用 StringBuilder 类。然而在应用程序要求线程安全的情况下，则必须使用 StringBuffer 类。如：

StringBuffer sBuffer = new StringBuffer("菜鸟教程官网：");

sBuffer.append("www");

sBuffer.append(".runoob");

sBuffer.append(".com");

System.out.println(sBuffer);

StringBuffer方法：

append(String s), reverse(),

delete(int start, int end), insert(int offset, int i),

replace(int start, int end, String str)

许多和String类的方法类似，详见http://www.runoob.com/java/java-stringbuffer.html

***总结：***

StringBuffer 和 StringBuilder 长度可变；

StringBuffer 线程安全 StringBuilder 线程不安全；

StringBuilder 速度快。

**14、Java数组**

用来存储固定大小的同类型元素。

（1）声明数组变量

首先必须声明数组变量，才能在程序中使用数组。

dataType[] arrayRefVar;

（2）创建数组

使用new操作符创建数组：arrayRefVar = new dataType[arraySize];

数组变量的声明，和创建数组可以用一条语句完成，如下所示：

dataType[] arrayRefVar = new dataType[arraySize];

dataType[] arrayRefVar = {value0, value1, ..., valuek};

（3）处理数组

使用基本循环或foreach循环，foreach循环能在不使用下标的情况下遍历数组。如：

double[] myList = {1.9, 2.9, 3.4, 3.5};

for (double element: myList) {

System.out.println(element);

}

数组可以作为函数的参数和返回值。

（4）多维数组

动态初始化：直接为每一维分配空间，或从最高维开始，分别为每一维分配空间。如：

int a[][] = new int[2][3];

或

String s[][] = new String[2][];

s[0] = new String[2];

s[1] = new String[3];

s[0][0] = new String("Good");

s[0][1] = new String("Luck");

s[1][0] = new String("to");

s[1][1] = new String("you");

s[1][2] = new String("!");

多维数组的引用：

num[1][0];

（5）Arrays类

java.util.Arrays类，它提供的所有方法都是静态的。

具有以下功能：

给数组赋值：通过 fill 方法。

对数组排序：通过 sort 方法,按升序。

比较数组：通过 equals 方法比较数组中元素值是否相等。

查找数组元素：通过 binarySearch 方法能对**排序好的**数组进行二分查找法操作。

-- public static int binarySearch(Object[] a, Object key)

-- public static boolean equals(long[] a, long[] a2)

-- public static void fill(int[] a, int val)

-- public static void sort(Object[] a)

**15、Java日期时间**

java.util.Date类，封装当前的日期和时间，两种构造函数：

Date()：使用当前日期和时间来初始化对象

Date(long millisec)：该参数是从1970年1月1日起的毫秒数

（1）方法：

-- String toString( )

把此 Date 对象转换为以下形式的 String： dow mon dd hh:mm:ss zzz yyyy 其中： dow 是一周中的某一天 (Sun, Mon, Tue, Wed, Thu, Fri, Sat)。

-- void setTime(long time)

用自1970年1月1日00:00:00 GMT以后time毫秒数设置时间和日期。

-- boolean after(Date date)

若当调用此方法的Date对象在指定日期之后返回true,否则返回false。

-- int compareTo(Date date)

比较当调用此方法的Date对象和指定日期。两者相等时候返回0。调用对象在指定日期之前则返回负数。调用对象在指定日期之后则返回正数。

-- boolean equals(Object date)

当调用此方法的Date对象和指定日期相等时候返回true,否则返回false。

（2）获取当前日期时间：

Date date = new Date();

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

运行结果：Mon May 04 09:51:52 CDT 2013

（3）日期比较

三种方法：

-使用getTime()方法获取两个日期（自1970年1月1日经历的毫秒数值），然后比较这两个值。

-使用方法before()，after()和equals()。例如，一个月的12号比18号早，则

new Date(99, 2, 12).before(new Date (99, 2, 18)) 返回true。

-使用compareTo() 方法，它是由Comparable接口定义的，Date类实现了这个接口。

（4）使用SimpleDateFormat格式化日期

Date dNow = new Date( );

SimpleDateFormat ft=new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

System.out.println("Current Date: " + ft.format(dNow));

运行结果：

Current Date: Wed 2016.11.09 at 08:23:19 AM UTC

“日期和时间的格式化编码”“使用printf格式化日期”及“解析字符串为时间”详见

http://www.runoob.com/java/java-date-time.html

（5）Java 休眠(sleep)

sleep()使当前线程进入停滞状态（阻塞当前线程），让出CPU的使用、目的是不让当前线程独自霸占该进程所获的CPU资源，以留一定时间给其他线程执行的机会。

你可以让程序休眠一毫秒的时间或者到您的计算机的寿命长的任意段时间。

try {

System.out.println(new Date( ) + "\n");

Thread.sleep(1000\*3); // 休眠3秒

System.out.println(new Date( ) + "\n");

} catch (Exception e) {

System.out.println("Got an exception!"); }

运行结果：

Thu Sep 17 10:20:30 CST 2015

Thu Sep 17 10:20:33 CST 2015

（6）测量时间

下面的一个例子表明如何测量时间间隔（以毫秒为单位）：

try {

long start = System.currentTimeMillis( );

System.out.println(new Date( ) + "\n");

Thread.sleep(5\*60\*10);

System.out.println(new Date( ) + "\n");

long end = System.currentTimeMillis( );

long diff = end - start;

System.out.println("Difference is : " + diff);

} catch (Exception e) {

System.out.println("Got an exception!"); }

运行结果：

Fri Jan 08 09:48:47 CST 2016

Fri Jan 08 09:48:50 CST 2016

Difference is : 3019

（7）Calendar类

使用Calendar 类可以设置和获取日期数据的特定部分（小时、日等），在日期的这些部分加上或者减去值。Calendar类比Date类更强大复杂，是**抽象类**，在实际使用时实现特定的子类的对象，用**getInstance**方法创建。

Calendar c = Calendar.getInstance(); //默认是当前日期

或：

//创建一个代表2009年6月12日的Calendar对象

Calendar c1 = Calendar.getInstance();

c1.set(2009, 6 - 1, 12);

**Calendar类对象字段类型**

Calendar.YEAR Calendar.MONTH Calendar.DATE Calendar.HOUR（12小时制）

Calendar.MINUTE Calendar.SECOND

Calendar.DAY\_OF\_MONTH Calendar.HOUR\_OF\_DAY（24小时制）

Calendar.DAY\_OF\_WEEK（星期几）

**Calendar类对象信息的设置**

**Set设置：**

Calendar c1 = Calendar.getInstance();

c1.set(2009, 6 - 1, 12); //把Calendar对象c1的年月日分别设这为：2009、6、12

或只设定某个字段，如：

c1.set(Calendar.YEAR,2008);

**Add设置：**

c1.add(Calendar.DATE,10);

c1.add(Calendar.DATE,-10);

**Calendar类对象信息的获得**

int year = c1.get(Calendar.YEAR);

（8）GregorianCalendar类

Calendar类实现了公历日历，GregorianCalendar是Calendar类的一个具体实现。

GregorianCalendar定义了两个字段：AD和BC。这是代表公历定义的两个时代。

有多种方法，可检测闰年等。

16、Java 正则表达式

java.util.regex包

遇到需要匹配、查找、替换、判断字符串的情况时，可以使用正则表达式。

正则表达式是一种可以用于模式匹配和替换的规范，一个正则表达式就是由普通的字符（例如字符a到z）以及特殊字符（元字符）组成的文字模式，它用以描述在查找文字主体时待匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

java.util.regex包主要包括以下三个类：

**-Pattern类：**

Pattern类用于创建一个正则表达式，也可以说创建一个匹配模式,它的构造方法是私有的,不可以直接创建,但可以通过Pattern.complie(String regex)简单工厂方法创建一个正则表达式。

Pattern p=Pattern.compile("//w+");

p.pattern(); //返回 /w+

pattern() 返回正则表达式的字符串形式,其实就是返回Pattern.complile(String regex)的regex参数。

**-Matcher类：**

Pattern类只能做一些简单的匹配操作,要想得到更强更便捷的正则匹配操作,那就需要将Pattern与Matcher一起合作。Matcher类提供了对正则表达式的分组支持,以及对正则表达式的多次匹配支持。Pattern.matcher(CharSequence input)返回一个Matcher对象，代码示例：

Pattern p=Pattern.compile("//d+");

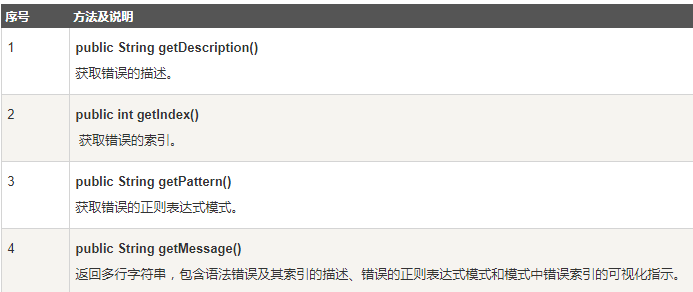
Matcher m=p.matcher("22bb23");

m.matches(); //返回false,因为bb不能被/d+匹配,导致整个字符串匹配未成功.

**-PatternSyntaxException类：**

PatternSyntaxException 是一个非强制异常类，它指示一个正则表达式模式中的语法错误。

PatternSyntaxException 类提供了下面的方法来帮助我们查看发生了什么错误。



**17、Java方法**

常用的System.out.println()语句中，println() 是一个方法，System 是系统类，out 是标准输出对象。

（1）方法的命名规则：

方法的名字的第一个单词应以小写字母作为开头，后面的单词则用大写字母开头写，不使用连接符。例如：addPerson。

下划线可能出现在 JUnit 测试方法名称中用以分隔名称的逻辑组件。一个典型的模式是：test<MethodUnderTest>\_<state>，例如 testPop\_emptyStack。

（2）命令行参数的使用：

运行一个程序时候再传递给它消息。这要靠传递命令行参数给main()函数实现。

命令行参数是在执行程序时候紧跟在程序名字后面的信息。

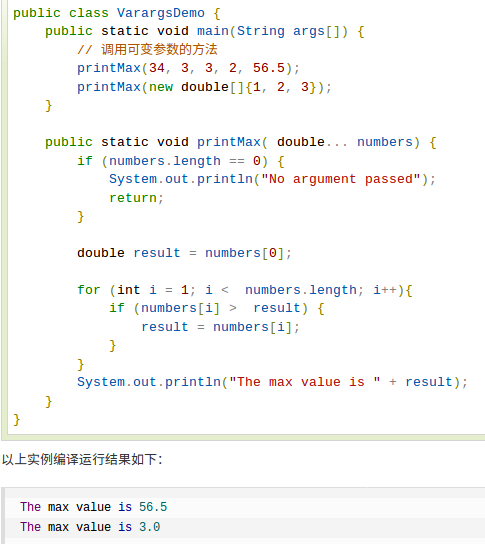
（3）构造方法：

当一个对象被创建时候，构造方法用来初始化该对象。构造方法和它所在类的名字相同，没有返回值。 通常会使用构造方法给一个类的实例变量赋初值，或者执行其它必要的步骤来创建一个完整的对象。 所有的类都有构造方法，Java自动提供了一个默认构造方法，它把所有成员初始化为0。 一旦定义了自己的构造方法，默认构造方法就会失效。

（4）可变参数：

Java支持给一个方法传递同类型的可变参数，在方法声明中，在指定参数类型后加一个省略号(...) 。 typeName... parameterName

一个方法中只能指定一个可变参数，它必须是方法的最后一个参数。任何普通的参数必须在它之前声明。



（5）finalize() 方法：

在对象被垃圾收集器析构(回收)之前调用，用来清除回收对象。例如，可以使用 finalize() 来确保一个对象打开的文件被关闭了。

在 finalize() 方法里，必须指定在对象销毁时候要执行的操作。

（6）Java里的形参和实参：

定义一个函数void add（int a, int b），这里的a和b就是形参。

进行[函数调用](http://www.so.com/s?q=函数调用&ie=utf-8&src=internal_wenda_recommend_text)的时候，add（1, 2），这里的1和2就是实参。

形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只在函数内部有效。

实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须有确定的值，以便把这些值传送给形参。因此应预先用赋值，输入等办法使参数获得确定值。

在一般传值调用的机制中只能把实参传送给形参，而不能把形参的值反向地传送给实参。因此在函数调用过程中，形参值发生改变，而实参中的值不会变化。而在引用调用的机制当中是将实参引用的地址传递给了形参，所以任何发生在形参上的改变实际上也发生在实参变量上。

**18、Java 流(Stream)、文件(File)和IO**

Java.io包

JDK 5 后的版本我们也可以使用 [Java Scanner](http://www.runoob.com/java/java-scanner-class.html) 类来获取控制台的输入。

（1）控制台输入输出

* 读取控制台输入

Java 的控制台输入由 System.in 完成。为了获得一个绑定到控制台的字符流，可以把 System.in 包装在一个 BufferedReader 对象中来创建一个字符流。

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

BufferedReader 对象创建后，可以使用 read() 方法从控制台读取一个字符，或用 readLine() 方法读取一个字符串。

* 从控制台读取多字符输入

用read()方法从BufferedReader对象读取一个字符。语法如下：

int read() throws IOException

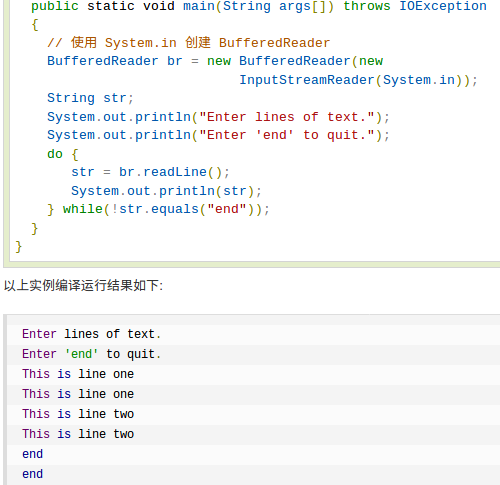
每次调用 read() 方法，它从输入流读取一个字符并把该字符作为整数值返回。 当流结束的时候返回 -1。该方法抛出 IOException。



* 从控制台读取字符串

使用 BufferedReader 的 readLine() 方法。语法如下：

String readLine() throws IOException



* 控制台输出

控制台的输出由 print()和println() 完成。这些方法都由类 PrintStream 定义，System.out 是该类对象的一个引用。

PrintStream 继承了 OutputStream类，并且实现了方法 write()，可以用来往控制台写操作。该方法将 byteval 的低八位字节写到流中。格式如下：

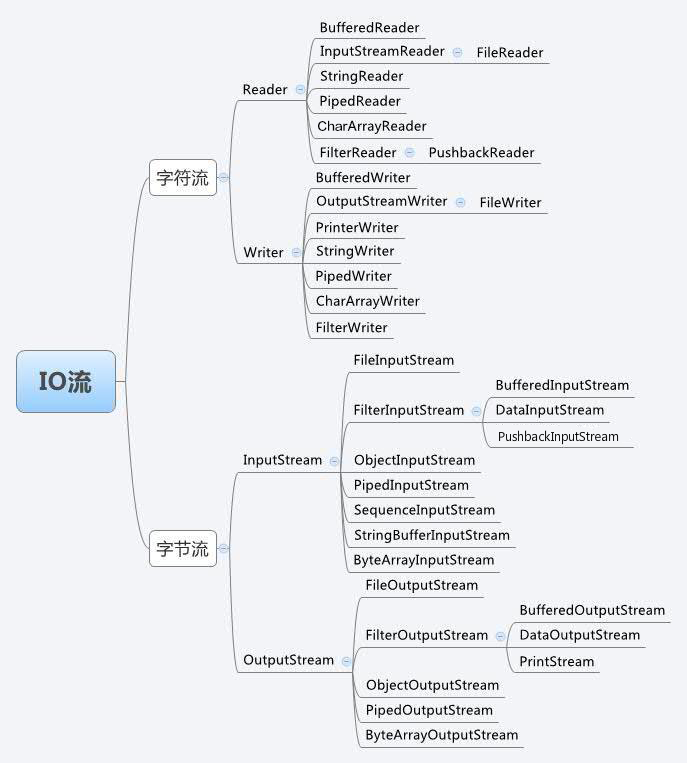
void write(int byteval)

write() 方法不经常使用，因为 print() 和 println() 方法用起来更为方便。

（2）读写文件

字节（Byte）是一种计量单位，表示数据量多少，它是计算机信息技术用于计量存储容量的一种计量单位。

字符是指计算机中使用的文字和符号，比如1、2、3、A、B、C、~！·#￥%……—\*（）——+、等等。



* FileInputStream

该流用于从文件读取数据，构造方法如下：

InputStream f = new FileInputStream("C:/java/hello");

或：

File f = new File("C:/java/hello");

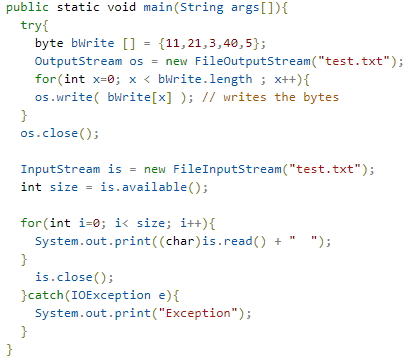
InputStream out = new FileInputStream(f);



* FileOutputStream

该类用来创建一个文件并向文件中写数据。构造方法与其他方法与FileInputStream类似，与read()对应的方法为write()

例如把给定的数字以**二进制形式**写进该文件，同时输出到控制台上，会有乱码：



解决乱码问题：



* 文件和I/O

还有一些关于文件和I/O的类：

File Class(类)

FileReader Class(类)

FileWriter Class(类)

（3）Java中的目录

* 创建目录

**File类**中有两个方法可以用来创建文件夹：

mkdir( )方法创建一个文件夹，成功则返回true，失败则返回false。失败表明File对象指定的路径已经存在，或者由于整个路径还不存在，该文件夹不能被创建。

mkdirs()方法创建一个文件夹和它的所有父文件夹。

File d = new File("/tmp/user/java/bin");

d.mkdirs(); // 现在创建目录

* 读取目录

一个目录其实就是一个 File 对象，它包含其他文件和文件夹。

如果创建一个 File 对象并且它是一个目录，那么调用 isDirectory() 方法会返回 true。

可以通过调用该对象上的 list() 方法，来提取它包含的文件和文件夹的列表。



* 删除目录或文件

删除文件可以使用 java.io.File.delete() 方法。



**19、Java Scanner类**

java.util.Scanner

Scanner s = new Scanner(System.in);

通过Scanner类的next()与nextLine()方法获取输入的字符串。如果要输入int或float类型的数据，在Scanner 类中也有支持，但是在输入之前最好先使用**hasNextXxx()**方法进行验证，再使用nextXxx()来读取。

next() 与 nextLine() 区别：

next():

1、一定要读取到有效字符后才可以结束输入，即不能获得空白。

2、对输入有效字符之前遇到的空白，next() 方法会自动将其去掉。

3、只有输入有效字符后才将其后面输入的空白作为分隔符或者结束符。

4、next() 不能得到带有空格的字符串。

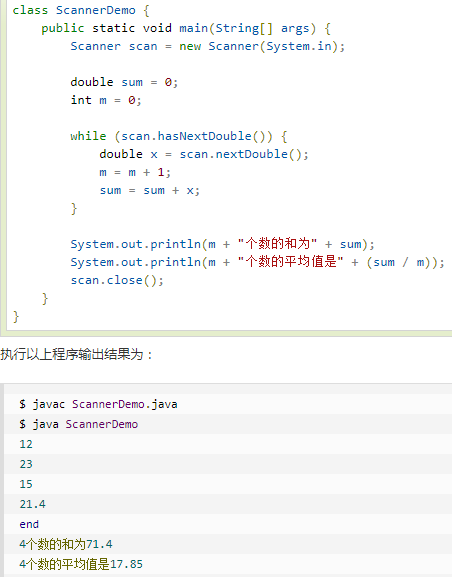
例：输入“ my book”，得到“my”

nextLine()：

1、以Enter为结束符,也就是说 nextLine()方法返回的是输入回车之前的所有字符。

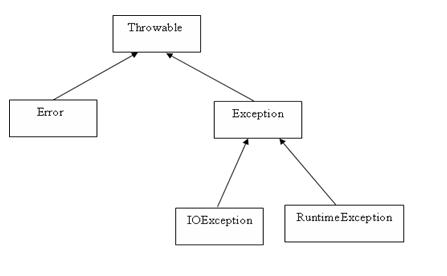
2、可以获得空白。

例：输入“ my book”，得到“ my book”



**20、Java异常处理**

所有的异常类是从 java.lang.Exception 类继承的子类。异常类有两个主要的子类：IOException 类和 RuntimeException 类。



需要掌握以下三种类型的异常：

**检查性异常**：最具代表的检查性异常是用户错误或问题引起的异常，这是程序员无法预见的。例如要打开一个不存在文件时，一个异常就发生了，不处理编译不能通过。如：ClassNotFoundException，InterruptedException，NoSuchFieldException等

**运行时异常**： 运行时异常是可能被程序员避免的异常。与检查性异常相反，不处理编译可以通过，如果有抛出直接抛到控制台。如：ArrayIndexOutOfBoundsException，IllegalArgumentException，NullPointerException等

**错误**： 错误不是异常，而是脱离程序员控制的问题。错误在代码中通常被忽略。例如，当栈溢出时，一个错误就发生了，它们在编译也检查不到的。

* 捕获异常

try 代码后不能既没 catch 块也没 finally 块。

try{

// 程序代码

}catch(异常类型1 异常的变量名1){

// 程序代码

}catch(异常类型2 异常的变量名2){

// 程序代码

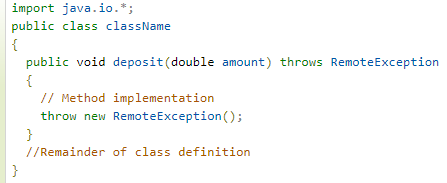
}catch(异常类型2 异常的变量名2){

// 程序代码

}

* throws/throw 关键字：

如果一个方法没有捕获一个检查性异常，那么该方法必须使用 throws 关键字来声明。throws 关键字放在方法签名的尾部。也可以使用 throw 关键字抛出一个异常，无论它是新实例化的还是刚捕获到的。一个方法可以声明抛出多个异常，多个异常之间用逗号隔开。



* finally关键字

finally 关键字用来创建在 try 代码块后面执行的代码块。无论是否发生异常，finally 代码块中的代码总会被执行。finally 代码块出现在 catch 代码块最后。

在 finally 代码块中，可以运行清理类型等收尾善后性质的语句。

* 声明自定义异常

注意：所有异常都必须是 Throwable 的子类。检查性异常类需要继承 Exception 类；运行时异常类需要继承 RuntimeException 类。一个异常类和其它任何类一样，包含有变量和方法。

1. 设计模式

**1、Java继承**

**2、Java Override/Overload**

**3、Java多态**

**4、Java抽象类**

**5、Java封装**

**6、Java接口**

**7、Java包（package）**

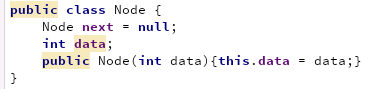
1. 数据结构与算法

**8.1 链表**

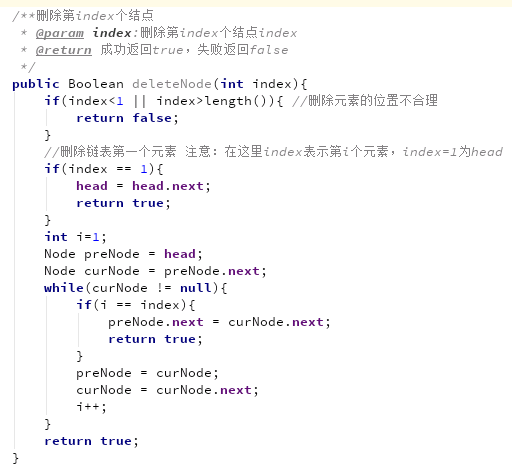
8.1.1 单链表的增删操作

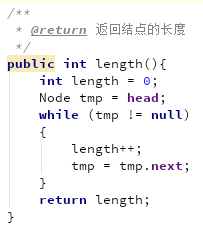
当结点只包含其后继结点的信息的链表成为单链表。

在Java语言中，可定义如下数据类来存储结点信息。



删除第index个结点





8.1.2 从链表中删除重复数据

**8.2 栈与队列**

**8.3 排序**

**8.4 位运算**

**8.5 数组**

**8.6 字符串**

**8.7 二叉树**

**8.8 其他**

1. 海量数据处理