第一章 Java语言简介

认识Java

Java是心啊在最流行的编程语言之一，并且自身拥有庞大且完善的生态系统，可以实现你的任何需求（但是你不可能要求一门语言实现的功能过多，所以有些时候java不擅长的部分会被其他的语言所顶替），在国内而言，之所以java使用广泛，更多的主要原因有许多的大户都在使用java实现其各自的的核心业务。

java是由sun公司开发的一套编程语言，其前身并不a是java。

在1991年时候，当时有一个Green项目，这个项目的核心功能在于使用Email控制家电产品的运行（智能家居），在80年代Email流行时侯，当时可以发送email是一件很牛的事情，最初SUN公司打算竞争此项目，但考虑但c++的复杂性，所以利用c++开发了一套OAK（橡树）平台，并利用此平台进行了项目竞标，但是失败了。但是这个时候世界上有一个最早的最牛x的软件公司诞生了：网景（第一家依靠技术上市的公司，1年内上市。第二家ba，orcle收购）。受浏览器技术的启发，推出了一个HotJava的浏览器。那么后来在1995.5.2时候正式推出了Java编程语言，同时推出了JDK1.0开发包（1996年开始陆续下载使用，1997年传到了中国，电脑报）。

（OAK不能申请版权，java可以。所以后面用了java）

SUN（Standford University NetWork）公司是一家从事与硬件开发的技术公司，SUN最早的带早点代表性产品：小型机（被广泛应用在了amazon）。世界上三种OS（win unix 类unix），最初的电子商务由IBM提出来的（第一代电商当当，卓越。淘宝03年），后来由于网络经济发展问题（任何经济模式一定会由瓶颈）但是对于90年代末的互联网低潮而言，这就是一个严重的伤害（典型李嘉诚的tom网，新浪搜狐美上市失败，8848衰落）。而SUN公司经过一个发展后，未能恢复往日的经济市里，后来被orcle公司收购，在oracle之前最希望收购的是ibm，当时生产线上技术语言基础是java（SUN并没有用java赚多少钱，真正赚钱的是IBM及后面靠版权欺诈的oracle）。

Java语言依然是一个　程序开发任务

而从最初的时代到到现在Java语言也出现了一些技术的不同发展：

- java标准开发 （J2SE JAVASE ）提供底层支持，实现桌面程序的开发
- java嵌入式开发（J2ME. JAVA ME ）sun公司最早就是想做嵌入式开发，后面被当年nokia折腾够了，后来被andriod替代了，再后来由于Oracle和google的撕逼大战，导致Android发展遇到了瓶颈，后来google使用了自己的专属语言进行Android开发（Kotlin）
- Java企业开发（J2EE JAVA EE)：主要进行企业平台的搭建，现在已经主要开发是互联网平台。

Java语言特点

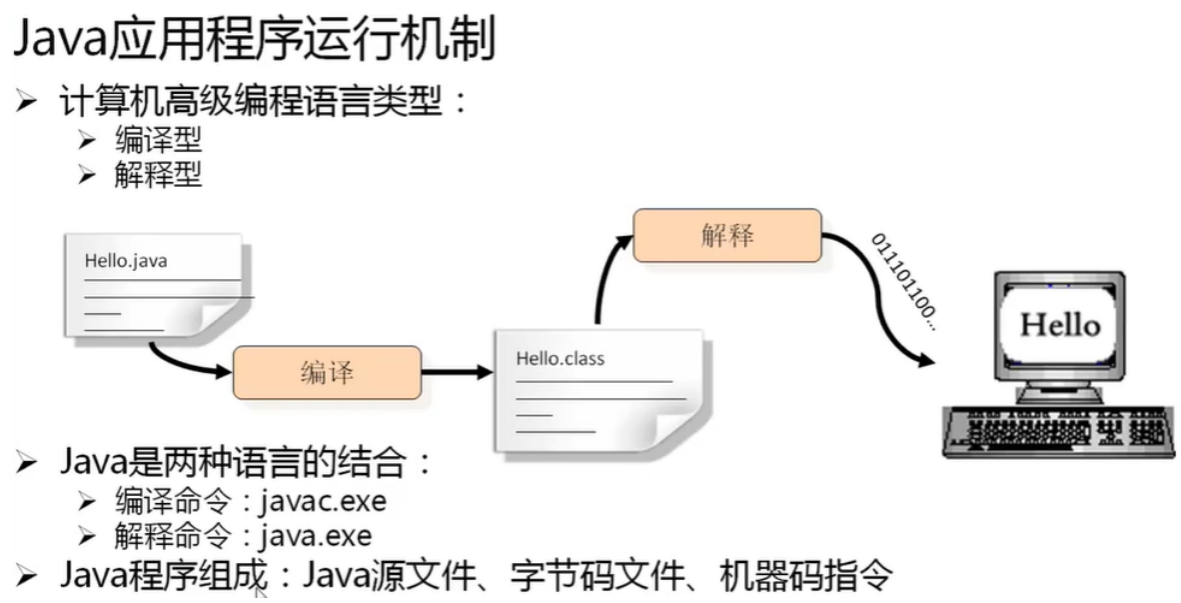
Java之所以可以得到持续发展和良好生态系统，取决于Java的生态特征：

1. 行业内通用的技术实现标准：
    - Java本身也算是一个半开源产品，很多厂商可以接触到Java底层，使得Java开发的更加透明。（对比.net）
2. 一门面向对象的编程语言：是的Java语言语法结构更加方便开发者接受，这些面对对象的设计思想还在不断地进行着扩充

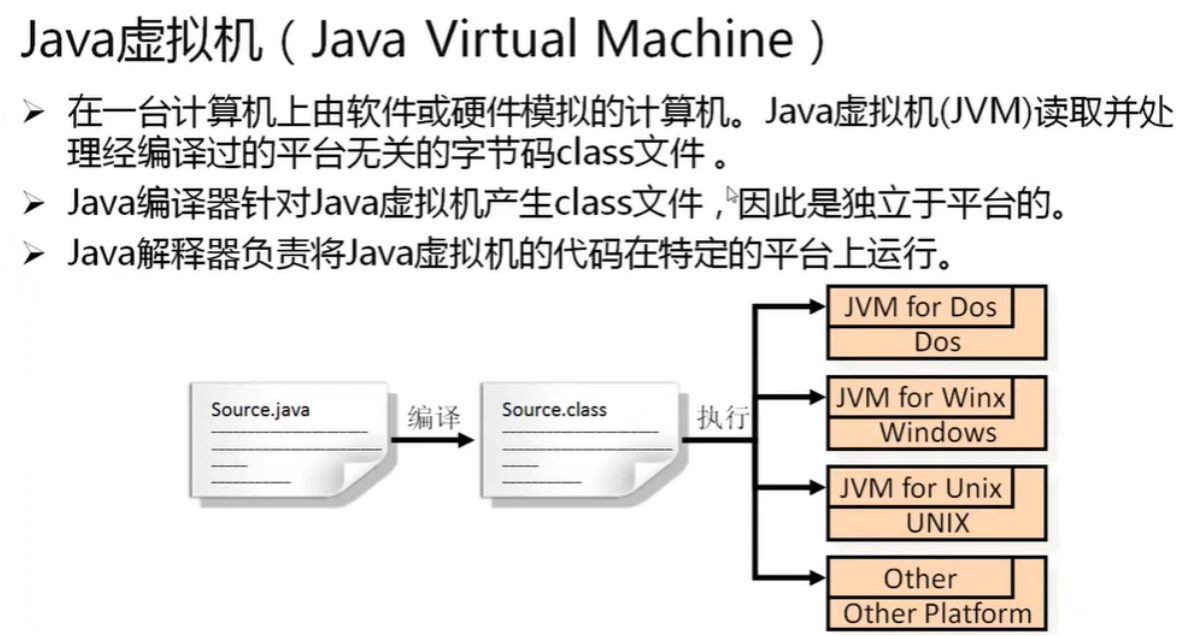
3. 提供方便的内存回收处理机制：想一些编程语言里里面需要明确的手工进行对象的回收和释放，否则你的程序将无法正常提供支持，Java提供自动的内存回收操作，这样处理更方便一些（牵扯到优化问题—）
4. 避免了复杂的指针问题i，而使用更加简单的引用来代替指针：指针虽然以一种高效的内存处理模式，需要较强的逻辑分析，而Java在设计的时候充分的考虑到了这一点，多以开发者直接利用引用就可以简化指针的处理，而引用是初学过程中最为难以理解的部分；
5. Java是位数不多hi吃多线程编程的开发语言，可以是的单位时间内，处理的性能得到提升（提升不是绝对的），多线程也是Java开发之中最为难以理解的部分，而正确的多线程处理次啊是提升处理能力的核心所在；
6. Java提供有高效的网络处理能力，可以基于NIO实现更加高效的数据的传输处理
7. Java具有良好的可移植性，可以提升一个程序的适用范围；
8. Java语言足够简单。

Java可移植性

Java语言之中最大的特点在于其可移植性的支持，所谓可移植性指的是同一个程序可以在不同操作系统之间进行部署，减少开发的难度，在Java里实现可移植性的控制，主要依靠jvm，是一个由软件和硬件模拟出来的计算机，所由的程序只要由Java虚拟机的支持，那么就可以实现程序的执行，不通的操作系统上有不通jvm，所以实现了在不同系统的执行。



所有Java程序的解释都要放在java虚拟机执行。



所有的*.java的源代码需要编译完成侯才可执行，但是编译完成侯的程序不是绑定机器的，是可在任何机器和系统上上执行的，是一种通用性程序，而这种通用性的程序就是JVM所能够识别的代码。

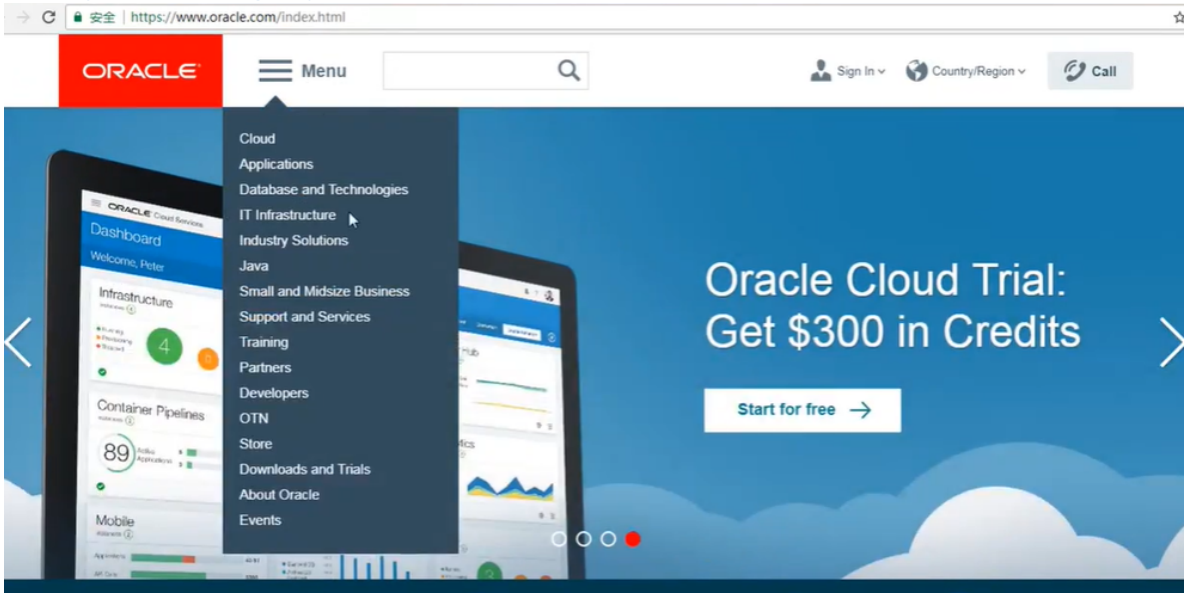
所有程序不是直接运行在操作系统上，而通过jvm执行，这样一来比直接执行在操作系统上要慢，先期是一个问题，但是后面由于硬件的发展，这些问题已经可以忽略了，但是依然会存在有JVM调优问题。（内置应用发挥操作系统的全部性能）

第二章 搭建java开发系统

JDK简介

Java语言属于编译型与解释型的开发语言，如果需要开发，则一定要有jdk的安装配置，需要官方网站+本机配置。

最早jdk都是sun公司提供（www.sun.com)被Oracle收购后，需要登陆Oracle网站(https://www.oracle.com/java/technologies/javase-downloads.html)



对于jdk而言的发展历史。标志性版本：

1. 1995.05.23 jdk1.0 开发包，1996年正式下载，标志java诞生
2. 1992.12.04 jdk1.2推出，java更名java2（java的升级版本）
3. 2005.05.23 java十周年大会，jdk1.5 ，带来新特性和开发支持最多的历史性版本，决定了后续十年
4. 2014年 java提供了jdk1.8 支持Lombda表达式，支持函数编程
5. 2017 java提供了jdk1.9 提升1.8的稳定性
6. 2018 java提升了jdk1.10，是属于jdk1.9的稳定版

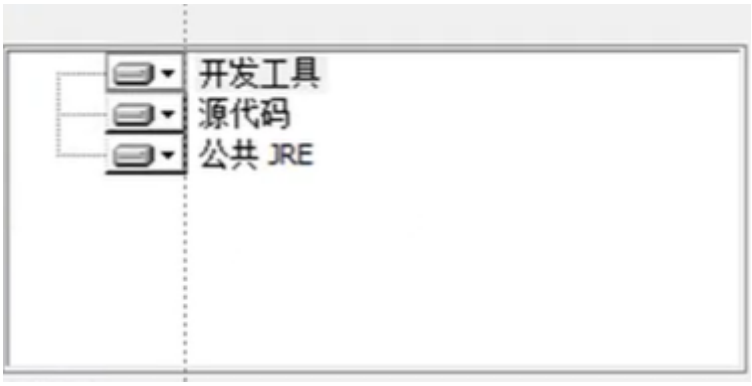
如果是生产环境部署还是以jdk1.8为主，1.9和1.10相差不大。本次演示为1.10

jre 指java运行时环境，只提供程序解释功能，不提供开发功能。装了jdk后会自动进行jre的更新处理

实际上三类系统：win unix 类unix（linux macOS），本次使用win

jdk安装和配置

jdk下载完成后，直接执行exe安装。



jdk之中默认支持jre。选择jre目录

安装完后需要注意，jdk配置处理。在jdk里所有可执行程序路径为：....\bin。重点使用Java.exe javac.exe

如果需要命令行里使用，需要配置

【高级系统设置】-【环境变量】-【系统变量】path

如果命令行打开了，无法进行新的环境属性读取，需要把cmd关了再打开

Java编程起步

几乎所有语言的第一个程序都是"hello world"，因为最早的c语言诞生的第一个程序就是这个

从记事本开始，命名.java

"Hello.java"

记事本或者editPlus，不然可能有编码问题

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

java需要两次处理才能执行

- 编译： javac Hello.java 出现*.class
- 在JVM上进行程序的解释执行： java Hello
    - 无需后缀，就是字节码文件

基本单元就是类

```
[public] class 类名称{ }
```

- 如果是public class定义，类名称必须和文件名一致。一个文件里只能有一个public class
- 如果class 定义，可以类名称和文件不一致，生成出来时是按class名称生成*.class文件。
- jvm执行时执行*.class文件的名字
- 在一个java文件中可以有多个class定义，并且编译后会形成不同的*.class文件
- 比较少一个*.java源代码里有多个class。
- Java语言有严格的明确要求，命名类时必须首字母大写

主方法：

```
public static void main(String [] args) {
    程序代码由此开始执行
}
//或者
public static void main(String args[]) {
    程序代码由此开始执行
}
```

java主方法定义很长，具体解释后面说明，在以后的课程讲解。称为主类，所有主类采用public class定义

屏幕打印：

输出后换行 System.out.println(打印内容)

输出后不换行 System.out.print(打印内容)

JSheel工具

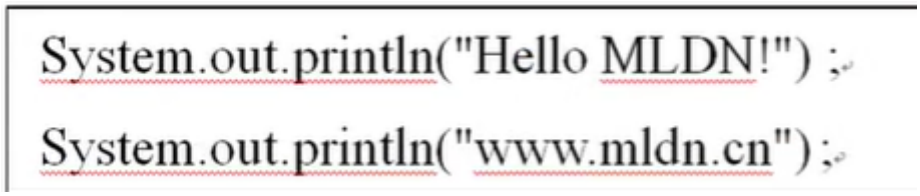
shell是脚本程序的含义。提供shell交互式环境。

避免编写结构代码，验证代码（jdk 1.9之后提供），类似python的idle

```
jshell> "Hello World !"
$1 ==> "Hello World !"

jshell> 1 + 1
$2 ==> 2
```

希望直接验证文本里的代码




```
jshell> /open d:/mldn.txt
Hello MLDN!
www.mldn.cn

jshell> /exit
|  再见
```

直接编写核心代码就可验证

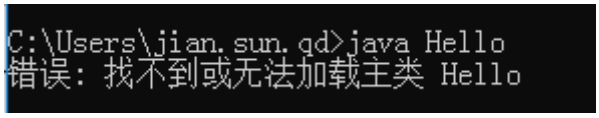
CLASSPATH环境属性

需要完整的解释需要很多知识，只是简单的介绍

del *.class 删除无用class

Hello.class

- 如果当前用户目录为class所在目录，可以直接java Hello执行
- 如果当前目录为其他路径，如果执行java Hello会有如下错误（1.8）



1.9及以后显示

ClassNotFoundException

> 需求：在任何目录都可以执行java Hello 执行*.class
>
> 措施：定义classpath环境属性
>
> SET CLASSPATH=E:\workspace\ideaproject

设置了CLASSPATH后，在java程序解释时候会自动的通过该环境设置的路径进行加载，所以：JVM解释程序时需要得到CLASSPATH的支持。

有一个问题，默认的加载都是从当前的目录加载，如果导出乱设置，会有问题，所以必须要包含当前目录。

> 设置加载当前目录（避免其他程序修改了该路径）
>
> SET CLASSPATH=.
>
> 该set设置为cmd命令行里的设置，如果关闭cmd会消失

==》设置全局的环境变量

Q： PATH和CLASSPATH区别

- PATH： 操作系统提供的路径配置，定义可执行程序的路径
- CLASSPATH：有jre提供的，用于定义Java程序解释时类加载的路径，默认设置为当前目录加载，可以通过"SET CLASSPATH=路径"的形式定义
- 关系：JVM-->CLASSPATH定义的路径

第四章 java 基本概念

注释

只是时程序开发中的重要组成技术，合理的注释可以使项目维护更加容易

注释的本质在于,编译器进行程序编译的时候如果发现有注释的内容将不对此部分进行编译:

- 单行注释: //
- 多行注释: /* ... */
- 文档注释: / ** ... */,文档注释里还有很多选项

如果使用开发工具开发,使用单行注释方便,对于一些重要部分加注释.类名方法名需要用文档注释

标识符与关键字

任何一个程序都是一个结构的整合体,在java语言里有不同的结构,例如:类,方法,变量结构等,一定要有不同的说明.对于结构的说明实际上就是标识符,是有命名要求的,一般是要求有意义的单词所组成,同时标识符的组成在Java之中的定义如下: 由字母,数字, _ , $ 所组成,其中不能使用保留字(关键字).

字母开头,数字 下划线辅助.

关键字是系统对于一些结构的描述处理,有特殊的含义.

Java中的关键字

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

对于所有关键字的定义不需要背的,有几个简短的说明

- JDK 1.4 出现assert关键字,用于异常处理
- JDK 1.5 出现enum关键字,用于枚举定义
- 未使用的关键字:goto, const
- 有些不是关键字,有特殊含义:true false

第五章 Java数据类型划分

Java数据类型简介

程序是一套数字处理的游戏框架,也就是说在整个程序的开发过程之中所达到的最终目的就是对数据的处理过程,所以就需要有各种类型的数据定义

数据分类

- 基本数据类型:具体的数字单元,1　1.1(有默认值)
  - 数值型:
    - 整型:byte short int long　(0)
    - 浮点型: float double　　(0.0)
  - 布尔型:boolean　　(false)
  - 字符型:char　　('\u0000')
- 引用数据类型:牵扯到内存关系的使用(默认值null)
  - 数组 类　接口(包含包装类)

讨论基本类型

基本类型有对应的范围

No.	数据类型	大小/位	可表示的数据范围
1	long（长整数）	64	-9223372036854775808 ~ 9223372036854775807
2	int（整数）	32	-2147483648 ~ 2147483647
3	short（短整数）	16	-32768~32767
4	byte（位）	8	-128 ~ 127
5	char（字符）	16	0 ~ 65536
6	float（单精度）	32	-3.4E38（-3.4×10^{38}） ~ 3.4E38（3.4×10^{38}）
7	double（双精度）	64	-1.7E308（-1.7×10^{308}）~ 1.7E308（1.7×10^{308}）

不同类型保存不同范围的数据,需要涉及类型选择

参考原则:

- 描述数字,首选int double
- 数据传输或者文字编码转换使用byte型(二进制操作)
- 处理中文使用方便的是用字符char
- 描述内存或大小,描述表的主键列(自动增长),用long

- 时间戳用long

整型数据类型

共四种 byte short int long,任何一个整型常量,默认都是int型

```
// int 变量名称 = 常量(常量是int型)
int x =10;
x = 20;
//int型变量 * int型变量 = int型数据
Sout(x*x)      //400
```

任何数据都有可保存的范围,一般不会超过范围,如果超过了?

```
int max = Integer.MAX_VALUE;  //2147483647
int min = Integer.MIN_VALUE;  //-2147483648
Sout(max+1)    //-2147483648
Sout(min-1)    //2147483647
```

32位,首位符号位,为1则是负数,按位取反+1.

出现时表示发生数据溢出,处理方式:

- 预估范围,如果不够则加大范围

  ```
  long max = Integer.MAX_VALUE;  //2147483647
  long min = Integer.MIN_VALUE;  //-2147483648
  //long型 ± int 型 = long型
  Sout(max+1)                    //2147483648
  Sout(min-1)                    //-2147483649
  ```

- 也可以在常量上处理,默认整数常量都是int型,可以追加L,或转换为long

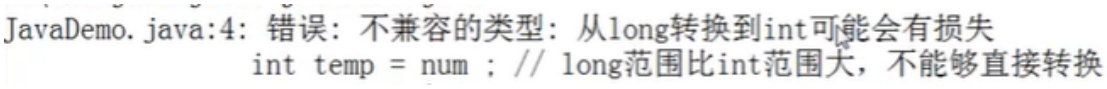
  ```
  int max = Integer.MAX_VALUE;  //2147483647
  int min = Integer.MIN_VALUE;  //-2147483648
  Sout(max+1L)    //2147483648
  Sout(max+1l)    //不要写小写l,容易看错
  Sout((long)min-1)    //-2147483649
  ```

  强制类型转换,范围大的转换范围小的,可能会导致数据丢失

  ```
  long num = 2147483649;
  int temp = num;
  ```

  ```
  JavaDemo.java:3: 错误: 过大的整数: 2147483649
                  long num = 2147483649 ; // 此数据已经超过了int范围
                             ^
  ```

  ```
  long num = 2147483649L;
  int temp = num;
  ```




```
long num = 2147483649L;
int temp = (int)num;
```

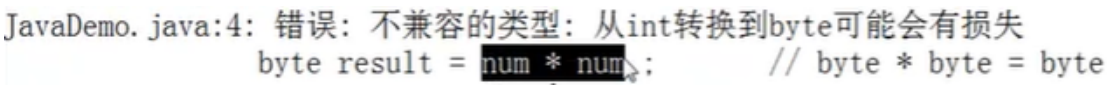
溢出了.

程序支持数据转换处理,如果不是必须的,不建议这种转换

在进行整型处理时,byte类型要注意,-128~127

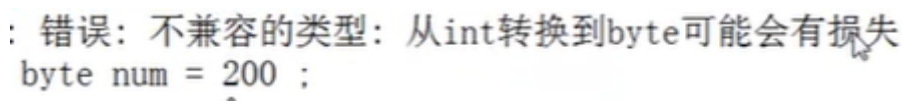
范例:定义byte变量

```
//byte型 = int 型
byte num = 10;
Sout(num * num )
//结果400 √
byte num = 10;
byte result = num * num;
Sout(result);
```



正常java程序里20是int型,为byte赋值时并没有报错: java做了自动转换,如果常量没超过范围可以自动由int变为byte. 变量还是需要强转

```
byte num = 200;
Sout(num)
```




```
byte num = (byte) 200;
Sout(num)
// -56 溢出
```

由于200超过了byte范围,产生溢出.

浮点型数据

任意一个小数常量,对应的是double,所以定义小数时建议使用double

```
double x =10.2;
int y = 10;
//double * int = double型
Sout(x * y)   //102.0
```

自动转型,都是由小到大转.默认用double,也可以用较小的float,所以赋值时需要强制转换

```
float x = 10.2
```

```
JavaDemo.java:3: 错误: 不兼容的类型: 从double转换到float可能会有损失
                float x = 10.2 ;
```

```
float x = (float) 10.2;
float y = 0.1F;
Sout(x * y)    //float型  103.02004
```

.00004时浮点型带来的精度损失

```
int x = 10;
int y = 4;
Sout(x/y);    //int型,不保存小数点后的   2
```

```
int x = 10;
int y = 4;
Sout((double)x/y);    //double型  2.5
```

字符型

使用' '定义的内容就是一个字符

```
char c = 'B';
Sout(c);     //B
```

任何编程语言,字符可以与int相互转换.也就是字符对应的内容,可以转换成系统所对应的系统代码

```
char c = 'A';
int num = c;
Sout(num);         //65
```

char范围

- 大写字母: A(65)~~Z(90)
- 小写字母: a(97)~~z(122)
- 数字范围: '0'(48)~~'9'(97)

大小写差了32个数,实现大小写转换

```
char c = 'x';
int num = c + 32;
Sout((char)num);          //x
```

到目前与c一致,但char可以保存中文字符

```
char c = '仁';
int num = c;
Sout(num);          //20161
```

java可以使用char保存中文,是因为java使用的Unicode的编码,主要特点是可以保存任意语言.

以前情况.换行时中文容易出乱码,字母1字节,中文2字节

布尔型

布尔时数学家名字,描述逻辑处理结果,java中只有true false

```
boolean flag = true;
if(flag) {
    Sout("符合");
}
```

部分编程语言,会用0代表false,1代表true. java不能混用

String字符串

在任何语言里都没有提供所谓的字符串这种数据,但从实际的使用上来讲,各个编程语言都会提供字符串的相应描述.在Java里使用String作为字符串定义

由于String类的存在较为特殊,所以其可以像普通变量那样采用直接赋值进行定义,使用" " 进行表示

```
String str = "Hello World!";
Sout(str);
```

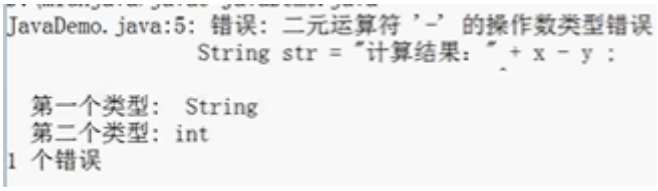
在进行字符串变量使用时也可以使用"+" 来进行字符串连接处理

```
String str = "Hello";
str = str + "World";
str+="!!!";
Sout(str);
```

需要考虑零一点,对于+有了两种描述(字符串连接, 加法计算)

```
double x = 10.1;
int y = 20;
String str = "计算结果:" + x + y;
Sout(str);
//计算结果:10.120  非期望结果
```

```
double x = 10.1;
int y = 20;
String str = "计算结果:" + x - y;
Sout(str);
```



数据范围大的与数据范围小的进行计算时,所有类型小的数据会变成大的数据类型. + 连接的有字符串,则全部变成String型运算.

```
double x = 10.1;
int y = 20;
String str = "计算结果:" + (x + y);
Sout(str);
//计算结果:30.1  期望结果
```

可以使用转义字符进行一些处理,例如:TAB(\t), "(\ "), ' (\ '), 换行(\n) ,\ (\ \ )

```
String str = "Hello \n World!";
Sout(str);
//Hello
//World
```

第六章 Java运算符

运算符简介

所有程序开发都是一种数字处理的游戏,那么对于数字的处理一定会有所谓的操作符.

对于程序开发而言,会提供有大量的基础运算符,那么这些运算符也都会提供各自的优先顺序,不建议取记优先级.可以加括号,括号优先级最高.

优先级	运算符	类	结合性
1	()	括号运算符	由左至右
1	[]	方括号运算符	由左至右
2	!、+（正号）、-（负号）	一元运算符	由右至左
2	~	位逻辑运算符	由右至左
2	++、--	递增与递减运算符	由右至左
3	*、/、%	算术运算符	由左至右
4	+、-	算术运算符	由左至右
5	<<、>>	位左移、右移运算符	由左至右
6	>、>=、<、<=	关系运算符	由左至右
7	==、!=	关系运算符	由左至右
8	&（位运算符AND）	位逻辑运算符	由左至右
9	^（位运算符号XOR）	位逻辑运算符	由左至右
10	|（位运算符号OR）	位逻辑运算符	由左至右
11	&&	逻辑运算符	由左至右
12	||	逻辑运算符	由左至右
13	?:	三目运算符	由右至左
14	=	赋值运算符	由右至左

对于程序开发而言,个人不建议编写太复杂的运算.

```
public class Demo {
    public static void main(String args[]) {
        int x = 10;
        int y = 20;
        System.out.println(x -- + y ++ * --y /x /y * ++ x - --y + y ++);
    }
}
```

```
E:\software\javadev\Java\jdk-14.0.1\bin>javac Demo.java

E:\software\javadev\Java\jdk-14.0.1\bin>java Demo
30
```

如果在项目代码里按这样逻辑写了代码,会被骂,写简单易懂的代码.

数学运算符

在Java中数学运算都提供标准支持.包括四则运算等

实现简单的四则运算

```
public class Demo {
    public static void main(String args[]) {
        int result = 10;
        result = result + 20;
        System.out.println(result);
    }
}
```

支持简化运算符(+= -= *= /= %=)

```
public class Demo {
    public static void main(String args[]) {
        int result = 10;
        result += 20;
        System.out.println(result);
    }
}
```

结果一样

最头疼的时"++" "--",有两类使用方法

- ++变量:先自增/减,再运算
- 变量++:先运算,再自增减

```
public class Demo {
    public static void main(String args[]) {
        int x = 10;
        int y = 20;
        int result = ++ x - y --;
        System.out.println("计算结果:" + result);  //-9
        System.out.println("x=" + x);               //11
        System.out.println("y=" + y);               //19
    }
}
```

这是当初内存量不大时用的,现在不建议这么写,如果要写可以考虑分开

```
x++;
result = x - y;
y--;
```

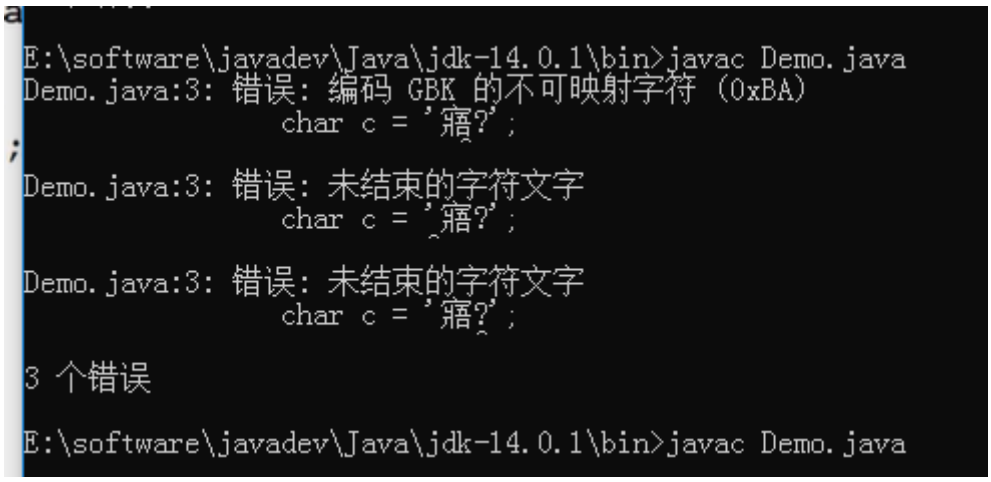
关系运算符

主要特征是大小的处理:> , < , >= , <= , != , ==.所有的关系运算返回的结果都是布尔类型

在关系判断时,特别注意相等的判断问题.在Java里,=表示赋值,内容的比较用==

```
boolean flag = 1 == 2;   //false
```

eg:用notepad++ 编辑过的.java




```
public class Demo {
    public static void main(String args[]) {
        char c = '建';
        System.out.println((int)c);    //24314
        }
}
```

```
public class Demo {
    public static void main(String args[]) {
        char c = '建';
        boolean flag = 24314 = 'c';
        System.out.println(flag);    //true
        }
}
```

三目(赋值)运算符

在程序开发中,三目运算用的比较多,合理的使用可以避免一些大范围的程序编写. 是一种设置逻辑关系的判断后才可进行赋值的操作.

> 关系运算符 ? 满足时的运算 : 不满足时的运算

eg:比较两个大小,来返回max

```
public class Demo {
    public static void main(String args[]) {
        int x =10;
        int y =20;
        int max = x>y ? x : y;
        System.out.println(max);
    }
}
```

传统方式;

```
public class Demo {
    public static void main(String args[]) {
        int x =10;
        int y =20;
        int max;
        if(x > y) {
            max = x;
        } else{
            max = y;
        }
        System.out.println(max);
    }
}
```

代码长

三目运算允许嵌套,但是可读性差

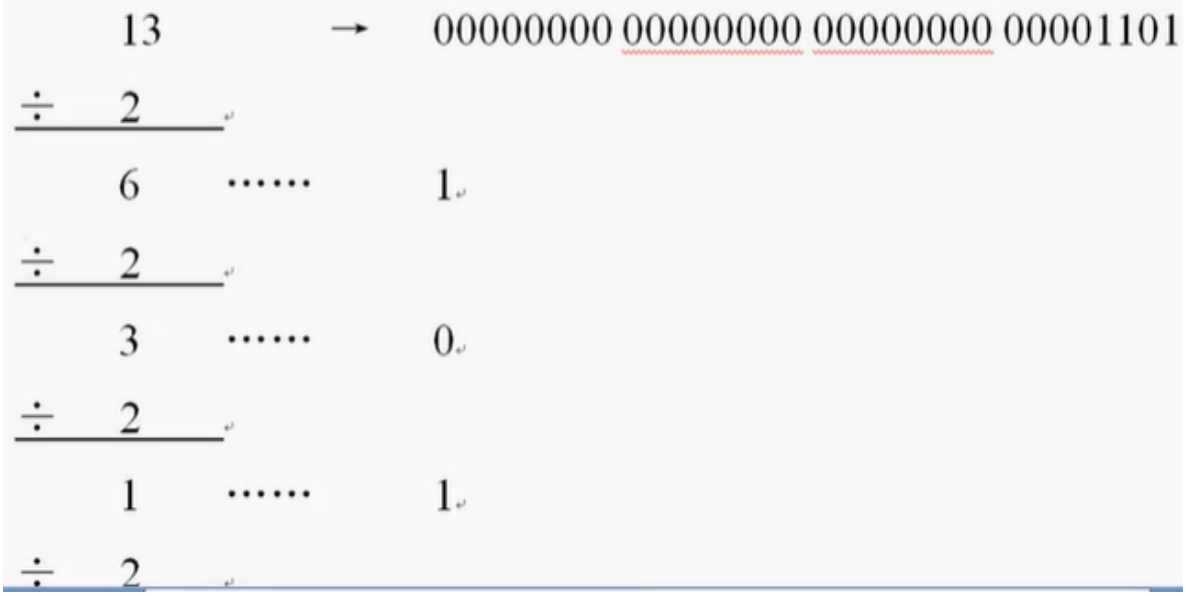
```
public class Demo {
    public static void main(String args[]) {
        int x = 10;
        int y = 20;
        int z = 15;
        int max = x>y ? (x>z ? x : z) : (y>z ? y : z);
        System.out.println(max);
    }
}
```

根据实际情况决定是否使用

位运算

指可以直接进行二进制数据的计算处理,主要有与(&) 或(|) 异或(^) 取反(~) 移位处理(>> <<)

10进制-->2进制, 处以2取余



1 2 4 8 16

1 0 1 1

观察与操作

```
public class Demo {
    public static void main(String args[]) {
        int x =13;
        int y =7;
        System.out.println(x&y);
        System.out.println(x|y);
    }
}
```

13 00000000 00000000 00000000 00001101

7 00000000 00000000 00000000 00000111

& 00000000 00000000 00000000 00000101

| 00000000 00000000 00000000 00001111

移位有乘除属性,求2的3次方

```
public class Demo {
    public static void main(String args[]) {
        int x =2;
        System.out.println(x << 2 );  //8
        System.out.println(x);        //2
    }
}
```

2的2进制: 00000000 00000000 00000000 00000010

向左移2位: 00000000 00000000 00000000 00001000

之前在内存有限时,很少有人使用移位来操作乘除

> Q: 请解释& 和 &&,| 和 ||的区别
>
> - & 和 | 两个运算符可以进行位运算与逻辑运算:
>   - 在进行逻辑运算时所有的判断条件都要执行
>   - 在进行位运算时只针对当前的数据进行与和或处理
> - 在逻辑运算上还可以使用&&, ||
>   - &&:在若干个条件判断,如果前面返回false,后续条件都不执行和判断
>   - 在进行位运算时只针对当前的数据进行与和或处理
>
> ||: 在在若干个条件判断,如果前面返回true,后续条件都不执行和判断

第七章 Java程序控制逻辑

程序开发的三种逻辑结构: 顺序结构, 分支结构, 循环结构.大部分 结构都是顺序结构的程序

IF 分支结构

主要针对于关系表达式进行判断处理的分支结构.主要三类使用形式,使用if　else

语法	描述
if	满足时执行一次
if else	如果 否则
if else if else	可以多检测条件判断

```
if (布尔表达式) {
    条件满足时执行 ;
} else if (布尔表达式) {
    条件满足时执行 ;
} else if (布尔表达式) {
    条件满足时执行 ;
} [else {
    条件不满足时执行 ;
}]
```

观察if

```
public class JavaDemo {
    public static void main(String args[]) {
        int age = 22 ;
        if (age >= 18 && age <= 20) {
            System.out.println("光荣的参与为人民服务的志愿活动！") ;
        }
        System.out.println("回到正常的生活轨迹！") ;
    }
}
```

观察if else,不满足的执行

```
public class JavaDemo {
    public static void main(String args[]) {
        double money = 20.00 ;
        if (money >= 19.8) {
            System.out.println("很牛X走到售卖出，很霸气的拿出20元，说不用找了。") ;
        } else {
            System.out.println("在灰暗的角落等待着别人剩下的东西。") ;
        }
        System.out.println("好好吃饭，好好的喝！") ;
    }
}
```

- 建议都写else,不然可能会留bug\

观察多条件判断,只有if else可以实现

```
public class JavaDemo {
    public static void main(String args[]) {
        double score = 90.00 ;
        if (score >= 90.00 && score <=100) {
            System.out.println("优等生。") ;
        } else if(score >=60 && score < 90) {
            System.out.println("良等生。") ;
        } else {
            System.out.println("差等生。") ;
        }
    }
}
```

多条件判断时可以不写else语句,好的习惯一定要写上else

SWITCH开关语句

根据内容进行判断,可以判断的只能是数据,只能是(int, char, enum(jdk1.5), Stirng(jdk 1.7))

```
switch(数据) {
    case 数值 : {
        数值满足时执行 ;
        [break ;]
    }
    case 数值 :
        数值满足时执行 ;
        [break ;]
    [default:
        所有判断数值不满足时的执行 ;
        [break ;]
    ]
}
```

case后可不加{},但是建议加

```
public class JavaDemo {
    public static void main(String args[]) {
        int ch = 2 ;
        switch (ch) {
            case 2 :
                System.out.println("设置的内容是2。") ;
            case 1 : {
```

```
                System.out.println("设置的内容是1。") ;
            }
            default : {
                System.out.println("没有内容满足。") ;
            }
        }
    }
}
```

执行结果

> 设置的内容是2。
> 设置的内容是1。
> 没有内容满足。

增加 break后

> 设置的内容是2。

重要语句break;

在switch设计时,如果每一个case里没有加break,会继续执行后面的语句,一直到遇到break

JDK 1.7时代,Oracle推出的JDK 1.7里面将开发者呼吁的10年以上的字符串判断功能增加了

```
public class JavaDemo {
    public static void main(String args[]) {
        String str = "hello" ;
        switch(str) {
            case "Hello":{
                System.out.println("Hello") ;
                break ;
            }
            case "hello": {
                System.out.println("hello") ;
                break ;
            }
            default: {
                System.out.println("NoMatch") ;
            }
        }
    }
}
```

> hello

while循环结构

某一段代码被重复执行的操作.有两种使用形式

while(布尔表达式)	**while (布尔表达式) { 条件满足时执行 ; 修改循环条件 ; }**
do while(布尔表达式)	do { 条件满足时执行 ; 修改循环条件 ; } while (布尔表达式) ;

```
public class JavaDemo {
    public static void main(String args[]) {
        int sum = 0 ;   // 保存最终的计算总和
        int num = 1 ; // 进行循环控制
        while (num <= 100) {    // 循环的执行条件
            sum += num ;    // 累加
            num ++ ;    // 修改循环条件
        }
        System.out.println(sum) ;
    }
}
```

如果不修改循环条件,会进入死循环

dowhile

```
public class JavaDemo {
    public static void main(String args[]) {
        int sum = 0 ;   // 保存最终的计算总和
        int num = 1 ; // 进行循环控制
        do {    // 循环的执行条件
            sum += num ;    // 累加
            num ++ ;    // 修改循环条件
        } while (num <= 100) ;
        System.out.println(sum) ;
    }
}
```

区别

- 如果num = 200,while不执行,do while 执行一次后停止
- 开发使用do while基本没有

for循环

```
for (定义循环的初始化数值 ; 循环判断 ;  修改循环数据) {
    循环语句的执行 ;
}
```

实现1-100累加

```
public class JavaDemo {
    public static void main(String args[]) {
        int sum = 0 ;   // 保存最终的计算总和
        for (int x = 1 ; x <= 100 ; x ++) {
            sum += x ;  // 累加
        }
        System.out.println(sum) ;
    }
}
```

如果观察三个操作的定义

```
public class JavaDemo {
    public static void main(String args[]) {
        int sum = 0 ;   // 保存最终的计算总和
        int x = 1 ; // 循环条件初始化
        for ( ; x <= 100 ; ) {
            sum += x ;  // 累加
            x ++ ; // 修改循环条件
        }
        System.out.println(sum) ;
    }
}
```

不建议这么写

对于while和for循环的选择标准:

- 明确循环次数情况下for优先
- 在不知道循环次数,但知道循环结束条件时使用while循环

循环控制

两个控制语句:break , continue

```
public class JavaDemo {
    public static void main(String args[]) {
        for (int x = 0 ; x < 10 ; x ++) {
            if (x == 3) {
                break ; // 循环结束
            }
            System.out.print(x + "、") ;
        }
    }
}
```

- break结束整个循环

  > 0、1、2、

  注意!!!! *.forEach 如果break则是中断全部

- continue 结束当前循环

  将break更换为continue

  > 0、1、2、4、5、6、7、8、9、

c语言中有个goto的指令,这个指令会直接造成代码的混乱,所以在开发中之中一般都对其深恶痛绝. Java里可以用continue实现goto的功能.不建议这么用

```
public class JavaDemo {
    public static void main(String args[]) {
        point: for (int x = 0 ; x < 10 ; x ++) {
            for (int y = 0 ; y < 3 ; y ++) {
                if (x == y) {
                    continue point ;// 循环结束
                }
                System.out.print(x + "、") ;
            }
            System.out.println() ;
        }
    }
}
```

> 1、2、2、3、3、3、
> 4、4、4、
> 5、5、5、
> 6、6、6、
> 7、7、7、
> 8、8、8、
> 9、9、9、

循环嵌套

实现乘法口诀

> 1 * 1=1
> 1 * 2=2 2 * 2=4
> 1 * 3=3 2 * 3=6 3 * 3=9

```
public class JavaDemo {
    public static void main(String args[]) {
        for (int x = 1 ; x <= 9 ; x ++) {   // 口诀表最多9行
            for (int y = 1 ; y <= x ; y ++) {
                System.out.print(y + "*" + x + "=" + (x * y) + "\t") ;
            }
            System.out.println() ;// 换行
        }
    }
}
```

结果

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72      9*9=81
```

打印三角形

```
public class JavaDemo {
    public static void main(String args[]) {
        int line = 5 ; // 总体行数
        for (int x = 0 ; x < line ; x ++) {
            for (int y = 0 ; y < line - x ; y ++) {
                System.out.print(" ") ;
            }
            for (int y = 0 ; y <= x ; y ++) {
                System.out.print("* ") ;
            }
            System.out.println() ;
        }
    }
}
```

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

第八章 方法的定义及使用

在程序之中 很多情况下需要重复执行代码.在有一些书上也会把方法(method)称为函数(function),需要注意,本次进行方法定义的时候,有一个前提: 方法在主类中定义,并由主方法直接调用

方法的定义

基本定义方式

```
public static 返回值类型 方法名称([参数类型 变量,....]) {
    // 该方法要执行的代码
    [return [返回值] ;]
}
```

返回值可以用Java的数据类型(基本类型 引用类型),方法可以进行返回数据的处理

用return 返回的类型与方法返回类型一致.无返回用void声明

> 关于方法名称与变量的命名要求:
>
> - 方法名和变量名命名规则一致
> - 首单词首字母小写,后面首字母大写

- 一个方法基本不会太长
- 方法本质,就是方便使用者调用.
- 程序都是从主方法开始往下执行的

```
public class JavaDemo {
    public static void main(String args[]) {
        String result = get(20.0) ;
        System.out.println(result) ;
        System.out.println(get(1.0)) ;  // 返回值可以直接输出
    }
    public static String get(double money) {
        if (money >= 10.0) {
            return "给你带一份快餐，找零：" + (money - 10.0) ;
        } else {
            return "对不起，您的余额不足，请先充值，或者捡漏。" ;
        }
    }
}
```

> 给你带一份快餐，找零：10.0
> 对不起，您的余额不足，请先充值，或者捡漏。

如果方法返回值类型为void,可以用return; 结束当前方法调用

```
public class JavaDemo {
    public static void main(String args[]) {
        sale(3) ;
        sale(-3) ;
    }
    public static void sale(int money) {
        if (money <= 0) {   // 余额不足
            return ;    // 后续代码不执行了
        }
        for (int x = 1 ; x <= money ; x ++) {
            System.out.println("王健开笑，第" + x + "次") ;
        }
    }
}
```

良好的方法设计,需要经验的累计.需要记住一般就几十行长度

* 方法重载

方法重载定义:方法名称相同,参数的类型或个数不同时就称方法的重载

```
public class JavaDemo {
    public static void main(String args[]) {
        int resultA = sum(10,20) ;   // 调用两个int参数的方法
        int resultB = sum(10,20,30);    // 调用三个int参数的方法
        double resultC = sum(10.2,20.3) ;
        System.out.println("加法执行结果：" + resultA) ;
        System.out.println("加法执行结果：" + resultB) ;
        System.out.println("加法执行结果：" + resultC) ;
    }
    public static int sum(int x,int y) {
        return x + y ;
    }
    public static int sum(int x,int y,int z) {
        return x + y + z ;
    }
    public static double sum(double x,double y) {
        return x + y ;
    }
}
```

根据传参和返回自动选择方法体,实现方法重载

> 加法执行结果：30
> 加法执行结果：60
> 加法执行结果：30.5

- 方法重载与方法返回类型没有关系
- 但实际开发中,只要时方法重载,强烈建议其返回值类型相同

观察代码:

```
Sout(1);
Sout(1.1);
Sout(true);
Sout('A');
Sout(1);
Sout("Hello");
```

系统提供的重载System.out

方法的递归调用

方法自己调用自己的情况,利用递归可以解决一些比较麻烦的问题,需要考虑一下几个问题:

- 一定要设定方法递归调用的结束条件
- 每次调用之中一定要修改方法的传递条件

```
public class JavaDemo {
    public static void main(String args[]) {
        System.out.println(sum(100)) ;
    }
    public static int sum(int num) { // 执行累加
        if (num == 1) { // 不累加了
            return 1 ;
        }
        return num + sum(num - 1) ; // 递归调用
    }
}
```

如果不写退出条件

```
Exception in thread "main" java.lang.StackOverflowError
    at JavaDemo.sum(JavaDemo.java:7)
    at JavaDemo.sum(JavaDemo.java:7)
    at JavaDemo.sum(JavaDemo.java:7)

    .......
```

代码分析:

- [第1次调用sum(),主方法执行] return 100 + sum(99);
- [第2次调用sum(),主方法执行] return 99 + sum(98);
- ... ...
- [第99次调用sum(),主方法执行] return 2 + sum(1);
- [第100次调用sum(),主方法执行] return 1;

整体形式

return 100 + 99 + ... + 2 + 1

不建议用

- 递归操作虽然可以简化调用,但是自己写的代码很少出现有递归的情况.
- 如果处理不当可能会导致栈溢出

计算： 1!+ 2! + ... + 90!

```
public class JavaDemo {
    public static void main(String args[]) {
        System.out.println(sum(90)) ;
    }
    public static double sum(int num) {
        if (num == 1) {
            return 1 ;
        }
        return fan(num) + sum(num - 1) ;
    }
    public static double fan(int num) {// 执行累加
        if (num == 1) { // 不累加了
            return 1 ;
        }
        return num * fan(num - 1) ; // 递归调用
```

```
    }
}
```

实际上有一部分递归可以用循环来写,不如递归清晰