**一．JAVA基础**

1995年Sun，2009年ORACLE

Java SE（基础核心标准开发）

Java EE（企业应用开发）

Java ME（嵌入式开发、移动设备基于Android的手机游戏）

API[应用程序](http://baike.baidu.com/view/330120.htm" \t "http://baike.baidu.com/_blank)编程**接口**(预先定义的[函数](http://baike.baidu.com/view/15061.htm" \t "http://baike.baidu.com/_blank)，基础类库)Application Programming Interface

JVM Java虚拟机，跨平台基础

JDK是开发工具包，包含JRE,JVM JRE=JVM+lib

运行java程序的最小环境是JRE：Java安装目录下的公用运行jre

开发java程序的最小环境是JDK：JDK目录下的jdk开发专用JRE

编译运行过程：

编译期-JDK把.Java源代码文件翻译为.class字节码文件

运行期-JRE调用JVM将字节码翻译成可供机器执行的程序

Javac.exe Java语言编译器

程序=算法+数据结构

**环境变量**

JAVA\_HOME变量值：D:\Java\jdk1.7.0\_75

Path变量值：%JAVA\_HOME%\bin;

CLASSPATH变量值： .;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar

测试：cmd java、javac

CLASSPATH环境变量是用来指定Java程序搜索类的路径的，当前目录 . 是一个常用的路径，其实Java程序在编译和运行的时候，不仅在CLASSPATH中搜索类，系统还会在JRE目录下找一个rt.jar的文件，路径为jre/lib/rt.jar，搜索顺序为先搜索rt.jar，再搜索CLASSPATH指定的路径

-cp / -classpath 动态的为java程序指定类加载路径

**Eclipse：**一个IDE（集成开发环境）工具，开发环境和程序调试环境集合，提升开发效率，**运行前至少要有JRE**

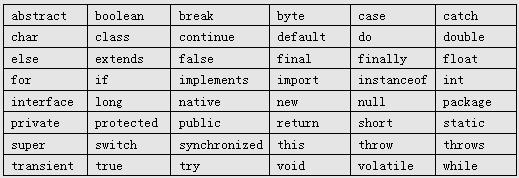
**MyEclipse是**eclipse的扩展，用于开发java、javaEE的插件集。

**Eclipse**相当于是开源可拓展容器

**Main函数后的(String[] args)不可修改 args是arguments字符串数组**

1. **变量与常量**

**Java常用关键字**（区分大小写）



**Java标识符**： Java程序中**包**、**类、变量、方法等的名字**。

1.标识符可以由**字母、数字、下划线（\_）、美元符（$）**组成，**不能以数字开头**。

2.**标识符不能是 Java 关键字**和保留字，但可以包含关键字和保留字，Myvoid 可以。

3.标识符是**严格区分大小**写的。

4.标识符的命名最好能反映出其作用。

**Java变量：**变量类型、变量名以及变量值

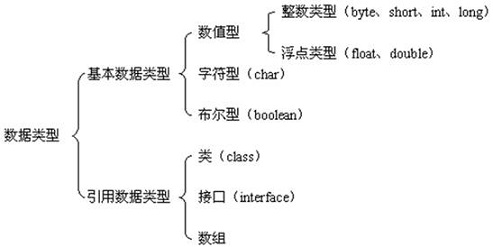
**变量的作用域：从声明开始到包含它最近的大括号结束**

1、变量名由多单词组成时，第一个单词的首字母小写，其后单词的首字母大写，俗称骆驼式命名法（也称驼峰命名法），如 myAge

2、见名知意

**Java中的数据类型**

Java 语言是一种**强类型语言**。在 Java 中存储的数据都是有类型的，而且必须在编译时就确定其类型。 Java 中有两类数据类型：

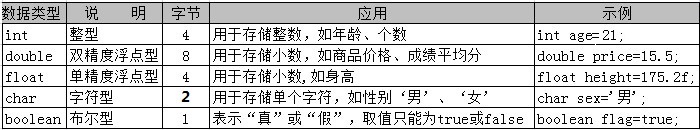


在 Java 的领域里，**基本类型**变量存的是数据本身，而**引用类型**存的是保存数据的**空间地址**。

基本类型：对一个变量的修改不会影响另一个变量

引用类型：对其中一个引用的修改影响另外一个引用

Null：空，没有指向任何对象 基本类型不允许设置为null



Int:

**整数直接默认为int类型，最大值是-2/2的31/31-1次方，运算时防止溢出**

float有效小数位6-7位

double有效小数位15位，Double类型只有 + \* 可以精确计算

若需要及其精确的运算，使用BigDecimal类：

**BigDecimal a = new BigDecimal(“5.21”)**

**BigDecimal a = new BigDecimal(“1.25”)**

System.***out***.println(a.subtract(b));减法 add加法

System.***out***.println(a.divide(b,3,RoundingMode.UP)); multiply 乘

保留三个小数，四舍五入

小数默认为double，float后+f/F

float x = 4.0f或者float x = (float)4.0

其中4.0f后面的f只是为了区别double，并不代表任何数字上的意义

char表示字符，只能存储一个字符

Java中的char采用unicode编码，一个字符char对应一个码

字符char是表现形式，实质上是码

特殊符号需通过转义符（\）来转义

基本类型从小到大：byte-short-int-long-float-double

**整数直接量**可以直接赋值给byte-short-char,但不能超范围

byte-short-char型**变量**参与四则或位运算时，底层一律先转换为int再运算，返回类型也为int

**char** a ='程'; **int** b = a/2; System.***out***.println(b); = 15621

**short** i = 120; **int** j = i>>1;

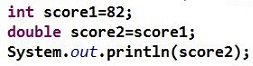
**整数直接量**的位运算或四则运算，编译期进行运算，根据=右边运算结果范围决定数据类型，如果超范围可以左边数据类型向上，也可以右边强制向下数据类型转换

**byte** w = (511>>2);

String 是一个类，具有面向对象的特征，可以调用方法，如name.length()获取字符串的长度

**Java中变量的使用规则：**需要先声明后使用

**Java中的自动类型转换**

****

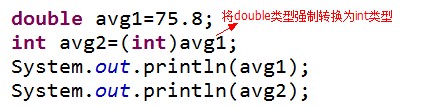
1.目标类型能与源类型兼容，如 double 型兼容 int 型

2.目标类型大于源类型，如 double 类型长度为 8 字节， int 类型为 4 字节

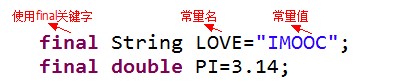
**Char类型可以向上转为int类型，但是int类型不能转为char；**

**char** a ='程'; **int** b = a;

**Java中的强制类型转换**

****

**Java常量的应用** 语法：final 常量名 = 值 （常量名使用大写字符）



**Java 中注释有三种类型：单行注释、多行注释、文档注释**

javadoc 命令从文档注释中提取内容，生成程序的 API 帮助文档。

Export-JAVA-doc

文档注释写在类的开头和成员变量

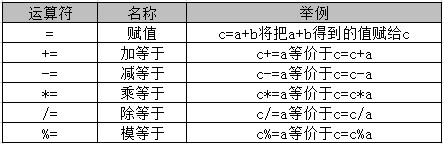
1. **Java常用运算符：算数运算符、赋值运算符、比较运算符、逻辑运算符、条件运算符、位运算**

**自增++和自减--运算符只能用于操作变量，不能直接用于操作数值或常量**

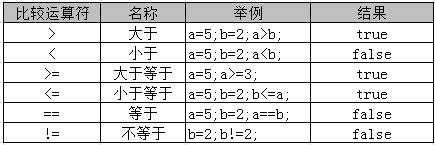
**任何复杂的逻辑都可以通过三种结构来实现：顺序，分支，循环**

% 用来求余数，也称为”取模运算符“

**常用赋值运算符**



**常用比较运算符**

****

1、  > 、 < 、 >= 、 <= 只支持左右两边操作数是数值类型

2、  == 、 != 两边的操作数既可以是数值类型，也可以是引用类型

**常用逻辑运算符**

****

**简述逻辑操作(&,|,^)与条件操作(&&,||)的区别**。

1. 条件操作只能操作布尔型的,而逻辑操作不仅可以操作布尔型,而且可以操作数值型

b.逻辑操作不会产生短路

短路现象：( one > two ) | | ( one < three ) 中，如果能确定左边 one > two 运行结果为 true , 则系统就认为已经没有必要执行右侧的 one < three 啦。

**条件运算符** 三元运算符 三目运算符

（ ? :  ）

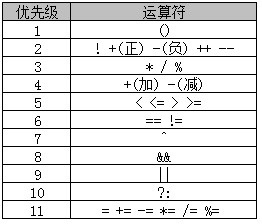
语法形式：布尔表达式 ？ 表达式1 ：表达式2

运算过程：如果布尔表达式的值为 true ，则返回 表达式1 的值，否则返回 表达式2 的值

535f7d130001a84f04340051

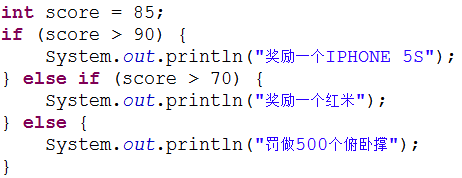
因为，表达式 8>5 的值为 true ，所以，返回： 8大于5

**运算符优先级 先求值，再比较**

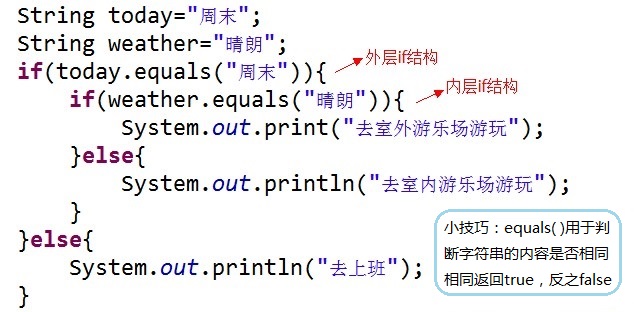
****

**流程控制语句**

**Java条件语句之多重if【if成功告别else】**

****

**Java条件语句之嵌套if**

****

**因为string类内部已经重写了equals方法，比的是内容**

**Java条件语句之Switch**

执行过程：当 switch 后表达式的值和 case 语句后的值相同时，从该位置开始向下执行，直到遇到 break 语句或者 switch 语句块结束；如果没有匹配的 case 语句则执行 default 块的代码。

1、 switch 后面小括号中表达式的值必须是**整型或字符型**

2、 case 后面的值不能是变量或带有变量的表达式，如 a \* 2

3、 case 匹配后，执行匹配块里的程序代码，如果没有遇见 break 会继续执行下一个的 case 块的内容，直到遇到 break 语句或者 switch 语句块结束

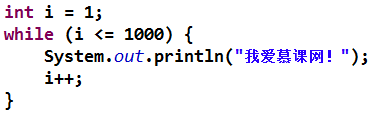
4、 default 块可以出现在任意位置，也可以省略

5、 可以把功能相同的 case 语句合并起来

****

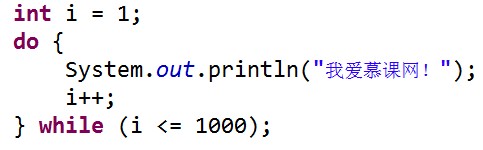
**Java常用的三种循环：while、do...while、for**

**while: 先判断，后执行**

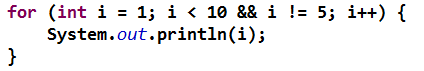
****

输出1000遍“我爱慕课网！”

**do...while： 先执行，后判断。保证循环至少被执行一次**

****

**for: 结构更加简洁易读**

****

**java循环跳转语句之break**

break结束的是{}内自己所在的的本层循环。

**java循环跳转语句之continue**

continue跳过的是{}内自己所在的的本层循环后续语句，开始下次本层循环。

不能出现在switch中

**java循环跳转语句之多重循环**

二重循环中，外层循环每执行一次，内层循环要执行一圈。

**\* print输出不换行，println输出换行**

1. **数组**

Arrays.sort(arr)；数组的排序

**Java中数组**

**int**[] score={58,84,86,33,12};

**int** score1[]={58,84,86,33,12};

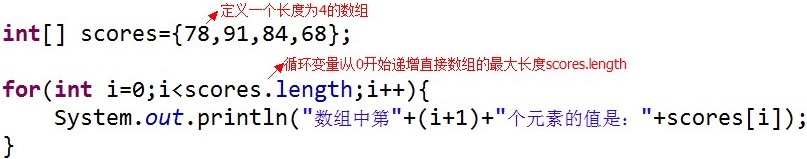
**int**[] score2=**new** **int**[]{58,84,86,33,12};

String names[];

names=**new** String[5];

names[0]="Clear";

**使用循环操作Java中的数组**

****

**数组名.length  用于获取数组的长度**

**Arrays类操作Java中的数组**

**排序Arrays.sort(数组名)**

**数组转换字符串Arrays.toString(数组名);**

**foreach操作数组**

foreach 并不是 Java 中的关键字，是 for 语句的特殊简化版本

foreach只能用来遍历元素,循环内部无法使用 删除等方法

**Java中的二维数组**

**1、 声明数组并分配空间**

IMG_256

IMG_256

1. **赋值**

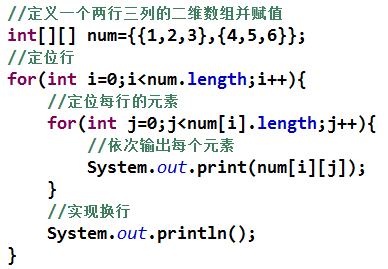
**53756c6100017e3e02810027**

**53757d490001ea5d02100048**

1. **处理数组**

二维数组中，num.length代表第一维的长度,即有几行

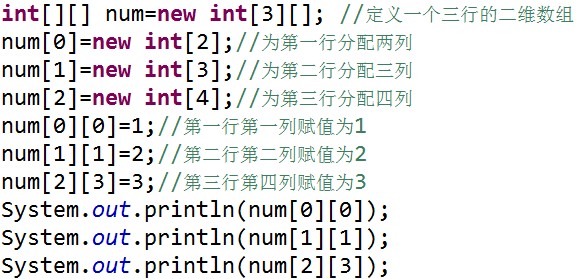
二维数组中，num[i].length代表第二维的长度，有几列

****

**123**

**456**

**创建不规则二维数组：**

****

**1 2 3**

1. **定义Java中的方法**

Ø 无参无返回值方法

Ø 无参带返回值方法

Ø 带参无返回值方法

Ø 带参带返回值方法

\* 方法参数列表可以是任意的基本类型数据或引用类型数据。

**Java中方法的重载【只关注参数列表，其他必须完全一样】**

1. **UUID**

**UUID：【通用唯一识别码 (Universally Unique Identifier)】**

是一个软件建构的标准UUID是指在一台机器上生成的数字，它保证对在同一时空 中的所有机器都是唯一的。通常平台会提供生成的API

UUID 的目的，是让分布式系统中的所有元素，都能有唯一的辨识资讯，而不需要透过中央 控制端来做辨识资讯的指定

GUID：目前最广泛应用的 UUID，即是微软的 Microsoft's Globally Unique Identifiers (GUIDs)

UUID由以下几部分的组合当前日期和时间，时钟序列，全局唯一的IEEE机器识别号

分布式系统（distributed system）是建立在网络之上的软件系统。正是因为软件的特性， 所以分布式系统具有高度的内聚性和透明性。

内聚性是指每一个数据库分布节点高度自治，有本地的数据库管理系统。

透明性是指每一个数据库分布节点对用户的应用来说都是透明的，看不出是本地还 是远程

--Java中生成UUID的方式：

--java中生成的是36位，每8个数字中间有一个-

--import java.util.UUID

--String uuid=UUID.randomUUID().toString();

--uuid=uuid.replace("-","");

1. **ThreadLocal**

**ThreadLocal：线程本地变量，该类提供了线程局部变量。**这些变量不同于它们的普通对应物，因为访问一个变量（通过其 get 或 set 方法）的每个线程都有自己的局部变量，它独立于变量的初始化副本。**ThreadLocal 实例通常是类中的私有静态字段，它们希望将状态与某一个线程（例如，用户 ID 或事务 ID）相关联。**

        事务是一种机制、是一种操作序列，它包含了一组[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/andydreaming/article/details/_blank)操作命令，这组命令要么全部执行，要么全部不执行。因此事务是一个不可分割的工作逻辑单元。**在数据库系统上执行并发操作时事务是作为最小的控制单元来使用的。这特别适用于多用户同时操作的数据通信系统。**

       在业务逻辑中我们要统一管理事务，但是在一个事务中会涉及到多个Dao，不同的DAO取得的是不同的Connection对象，这样就不能实现统一管理事务。

       那么怎么才能实现两个或多个DAO用同一个事务来控制，来统一管理事务那？

        首先就要解决一个根本问题，我们在一个事务中，不同的DAO取得的是同一个Connection,这个要怎么实现呢。我们想到了TheadLocal.

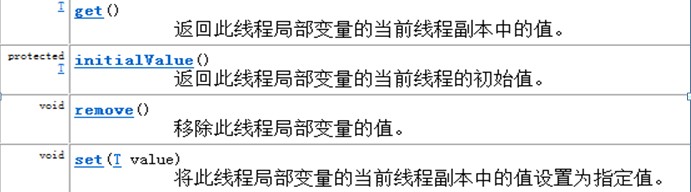
        因为当一个业务逻辑要对多个DAO进行处理的时候，此操作属于同一个线程内。所以，我们希望能把此Connection放在线程内，或者和线程有关联。

**在任何一个线程内都可以用ThreadLocal来保存一个变量的copy,这样，如果此对象存在，就可以直接取用。**

**如果ThreadLocal中放置connection， 保证事务中的DAO类获取的都是同一个connection,这样才保证事务。所以，在同一个线程内，不同的DAO就可以保证取得到的是同一个Connection**

**这样就实现了，事务的统一管理**

      ThreadLocal的API如下图：

  
**private** **static** ThreadLocal<Session> *tl*= **new** ThreadLocal<Session>();

/\*ThreadLocal 实例通常是类中的私有静态字段，它们希望将状态与某一个线程（例如，用户 ID 或事务 ID）相关联。 \*/

**public** **static** Session getSession(){

Session session=*tl*.get();

**if**(session==**null**){

session=*sf*.openSession();

*tl*.set(session);

}

**return** session;

}

**public** **static** **void** closeSession(){

Session session=*tl*.get();

**if**(session!=**null**){

session.close();

*tl*.set(**null**);

}

}