**Java中的50个关键字**

    关键字也称为保留字，是指java语言中规定了特定含义的标示符。对于保留字，用户只能按照系统规定的方式使用，不能自行定义。Java中有50个常用关键字：

**与数据类型相关（10）**

|  |  |  |  |
| --- | --- | --- | --- |
| 关键字 | 类型 | 占字节数 | 备注 |
| **boolean** | 布尔数据类型 | 1 | 有true和false两个值 |
| **int** | 整型数据类型 | 4 |  |
| **long** | 长整型数据类型 | 8 |  |
| **short** | 短整型数据类型 | 2 |  |
| **byte** | 字节数据类型 | 1 | 大小范围为-27—27-1 |
| **float** | 浮点型（实数型） | 4 |  |
| **double** | 双精度型数据类型 | 8 |  |
| **char** | 字符型数据类型 | 2(Unicode码) |  |
| **class** | 表示一个类 | 不确定 | 可用于定义一个类。 |
| **interface** | 表示一个接口 | 不确定 | 可用于定义一个接口 |

**与流程控制相关（13）**

**if：** 表示条件判断，一般用法if（关系表达式），后跟else或{……}

else： 条件转折，如if （关系表达式）{语句块1}else{语句块2}，如果关系表达式的值为true，则执行语句块1，否则执行语句块2.

do……while……：do和while一般一起使用，用于表示循环语句。do{……}while（关系表达式）……；当关系表达式的值为true是继续循环。

**for：**用于表示循环，for循环是最常使用的循环，格式for（表达式a; 表达式b; 表达式c）括号里面的书通常用于控制循环的次数，一般会用一个int类型的变量类计数，如（int i=0; i<10; i++）表达式a用于流程控制的开始值，表达式b表示循环终止条件，表达式c用于计数。

**switch（条件a）case……：**switch和case合起来用于表示条件分支流程。如：  
while(int c) {  
case 1: {语句块1}   
case 2: {语句块2}  
……?????? ……  
case n: {语句块n}  
default：exit(0);  
}

如果c为1，则执行语句块1；如果c为2，则执行语句块2；以此类推，如果c为n，则执行语句块n。default表示除case以外出现的情况。

default：在switch……case……分支语句可知，default是在所有case条件下都不成立时使用。用于包表示s“rc”文件夹下的当前包；如果用于类，表示只可被本文件内的其它类访问。

**break：**用于结束本层循环，或跳出某层循环。

continue：用于跳出本次循环，而break跳出本层循环。Break和continue可以实现类似于C\C++中goto语句的用法：  
label0:  
{  
　for (int k = 0; k < 10; k++) {  
　..........   
　label1:  
　for (int j = 0; j < 10; j++) {  
　　................  
　　break label0;　　//跳转至label0  
　}  
　...........  
　label2:  
　for (int kk = 0; kk < 10; kk++) {  
　　..............  
　　break label0;　　//跳至label2

　}  
}

**return:**返回一个值，通常用于函数中，返回一个具有特定类型的值。如:

public int fuction()  
{  
　　int a;   
　　……  
　　return a;//返回int型的值  
}

**try……catch……finally……：**用于异常处理，使用形式如：

try{  
　……  
}catch(类型b){  
　……  
}finally{  
　……  
}

try{……}中放置可能会发生异常的的语句块，catch(){……}用于抓住异常，{}定义当出现异常时的处理方法。finally{……}表示不管异常是否发生，都得进行finally{}中的处理。

**与修饰符相关（12）**

**不同访问修饰符的访问权限**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 范围 | 本类 | 同软件包 | 不同包中子类 | 不同包且无继承 | 作用(含义) |
| **public** | 可以 | 可以 | 可以 | 可以 | 公有的 |
| **protect** | 可以 | 可以 | 可以 |  | 受保护的 |
| **default(无权限修饰符)** | 可以 | 可以 |  |  | 当前的 |
| **private** | 可以 |  |  |  | 私有的 |

如：

一个".java"后缀名的文件里面只能一个publi型的cass类。如c13.swing包下有HelloWorld.java 文件，该文件下有如下代码：

package c13.swing;

public class HelloWorld {

public void show() {

System.out.println("HelloWorld!");

}

}

在default包下有如下代码：

import c13.swing.\*;//引入所需要的包里的所有类

public class Test {

public static void main(String args[]) {

HelloWorld h = new HelloWorld();

h.show();

}

}

此时运行正常，结果：HelloWorld!

如果把public class HelloWorld改成privateclass HelloWorld则会运行错误，因为private是私有的不能被其他包内的对象所引用。

**final：(**类似于C++中的const)，用法如下：  
a，final的变量的值不能被改变；1），final的成员变量； 2）final的局部变量；  
b，final的方法不能被重写；  
c，final的类不能被继承。

void：空类型，用于函数前表示空返回值。

**static：**可用于定义静态成员变量或方法

strictfp：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你满意。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。因此如果你想让你的浮点运算更加精确，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。    
你可以将一个类、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，例如下面的代码：

1. 合法的使用关键字strictfp   
strictfp interface A {}

public strictfp class FpDemo1 {   
strictfp void f() {}   
}   
2. 错误的使用方法   
interface A {   
strictfp void f();   
}

public class FpDemo2 {   
strictfp FpDemo2() {}   
}

一旦使用了关键字strictfp来声明某个类、接口或者方法时，那么在这个关键字所声明的范围内所有浮点运算都是精确的，符合IEEE-754规范的。例如一个类被声明为strictfp，那么该类中所有的方法都是strictfp的。

**abstract：**抽象的意思，可以用定义抽象类，抽象函数。抽象类的用法规则如下：  
a，用abstract关键字来修饰一个类时，这个类叫做抽象类；用abstract来修饰一个方法时，这个方法叫做抽象方法；  
b，含有抽象方法的类必须被声明为抽象类，抽象类必须被继承，抽象方法必须被重写；  
c，抽象类不能被实例化；  
d，抽象方法只需声明，而不需实现。

transient: 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。  
 使用对象：字段  
介绍：字段不是对象持久的一部分，不应该字段和对象一起串起。

synchronized：锁定当前对象，在执行当前这个对象时不应许其他线程打断插入。使用的方式有：

这是我在暑假学习做的笔记

（1），class Timer {  
private static int num = 0;  
public synchronized void add(String name) { //执行这个方法的过程中锁定当前对象。  
num ++;  
try {  
Thread.sleep(1);  
} catch( InterruptedException e) {  
return;  
}  
System.out.println(name + ",你是第" + num +"使用Timer的线程。");  
}  
}  
（2），class Timer {  
private static int num = 0;  
public void add(String name) {   
synchronized(this) { //锁定大括号这个区域的代码段  
num ++;  
try {  
Thread.sleep(1);  
} catch( InterruptedException e) {  
return;  
}  
System.out.println(name + ",你是第" + num +"使用Timer的线程。");  
}  
}  
}

**volatile：**volatile变量一个时间只有一个线程可以访问。 保证线程安全。   
可以被写入 volatile 变量的这些有效值独立于任何程序的状态，包括变量的当前状态。   
所以，Volatile 变量是一种非常简单但同时又非常脆弱的同步机制，它在某些情况下将提供优于锁的性能和伸缩性。

**native：**是用作java 和其他语言（如c++）进行协作时用的也就是native 后的函数的实现不是用java写的， native的意思就是通知操作系统，这个函数你必须给我实现，因为我要使用。所以native关键字的函数都是操作系统实现的， java只能调用。

**与动作相关（10）**

package：打包，把java程序写在一个包中。

import：引入报名，用法：   
a，写全包名：jsxt.java140.Cat c = new com.bjsxt.java140.Cat  
a，引入一个包中的具体某个对象：xt.java140.Cat;  
b，引入一个包中的所有象：import com.bjsxt.java140.Cat.\*;

**throw：**抛出用户自己定义的异常

**throws：**抛出系统异常

extends：继承（类继承类，或接口继承接口）

implements：实现，通过类来实现接口

this：是指向该对象自身的标示符

super：调用基类的构造函数

instanceof：判断某个对象是否为自己所要的对象

**new：**用于新建一个对象，即类的实例化

**其他（5）**

**true：**表示boolean类型的真值

false：表示boolean类型的假值，即不成立的情况

null：表示空值，如果是对象，则表示此对象为空，如果是数据类型，在表示相应数据类型的默认值，如int的默认值为0。

goto：C\C++中实现无条件转向语句，为了结构化程序设计java中现在一般不用。

const：和const一样，在Java中,const是作为保留字以备扩充。可以用final替换const，一般C++中用const，java中用final。