Java关键字汇总说明

author：尚硅谷教育

50 character sequences, formed from ASCII letters, are reserved for use as keywords and cannot be used as identifiers (§3.8).

Keyword:
(one of)

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

The keywords `const` and `goto` are reserved, even though they are not currently used. This may allow a Java compiler to produce better error messages if these C++ keywords incorrectly appear in programs.

While `true` and `false` might appear to be keywords, they are technically boolean literals (§3.10.3). Similarly, while `null` might appear to be a keyword, it is technically the null literal (§3.10.7).

> 关键字一共50个，其中const和goto是保留字。
>
> 此外，还有3个特殊值：true,false,null。它们看起来像关键字，但从技术角度，它们是特殊的布尔值和空值。

一、基本数据类型相关关键字(8个)

（1）byte：单字节类型

1个字节（8位），数据范围：[-128,127]

> 注意：byte类型与byte类型做算术运算结果升级为int

（2）short：短整型

2个字节（16位），数据范围：[-32768,32767]

> 注意：short类型与short类型做算术运算结果升级为int

（3）int：整型

4个字节（32位），数据范围：[-2的31次方 , 2的31次方 -1]

（4）long：长整型

8个字节（64位），数据范围：[-2的63次方 , 2的63次方 -1]

> 注意：long类型的数字后面需要加L或小写l

（5）char：单字符类型

2个字节（16位），无符号整数范围：[0,65535]，使用Unicode字符集

其中标准ASCII字符集：[0,127]，数字字符['0','9']的编码值是[48,57]，大写字母字符['A','Z']的编码值是[65,90]，小写字母字符['a','z']的编码值是[97,122]。

char类型的字符常量有三种表示方式：

（1）'a'、'尚'

（2）转义字符：

EscapeSequence:

\ b (backspace BS, Unicode \u0008)
\ t (horizontal tab HT, Unicode \u0009)
\ n (linefeed LF, Unicode \u000a)
\ f (form feed FF, Unicode \u000c)
\ r (carriage return CR, Unicode \u000d)
\ " (double quote ", Unicode \u0022)
\ ' (single quote ', Unicode \u0027)
\ \ (backslash \, Unicode \u005c)

（3）使用Unicode编码值十六进制形式表示

例如：'\u5c1a' 表示 '尚'

> 注意：char类型与char类型做算术运算结果是int类型

（6）float：单精度浮点型

4个字节，表示最大精度范围：科学记数法表示数字后的小数点后6~7位

> 注意：float类型的数字后面需要加F或f

（7）double：双精度浮点型

8个字节，表示最大精度范围：科学记数法表示数字后的小数点后15~16位

（8）boolean：布尔类型

它只能有两个值：true和false

二、声明引用数据类型相关(3个)

（9）class：声明类

语法格式：

```
【修饰符】 class 类名{
    ...
}
【修饰符】 class 类名 【extends 父类】  【implements 父接口们】{
    ...
}
```

类中可以有5个成分：

（1）成员变量：属性

（2）成员方法

（3）构造器

（4）代码块

（5）成员内部类

（10）interface：声明接口

语法格式：

```
【修饰符】 interface 接口名{
    ...
}
【修饰符】 interface 接口名 【extends 父接口们】{
    ...
}
```

接口的特点：

（1）接口对成员有严格的要求：

在JDK1.8之前，接口只能有全局的静态的常量(public static final)和公共的抽象方法(public abstract)；

在JDK1.8之后，接口中可以有公共的静态方法（public static）和公共的默认方法（public default）；

接口没有构造器、代码块能其他成员，但是接口中可以声明成员内部类、成员内部接口。

（2）接口不能直接实例化，但是接口类型的变量可以与其实现类的对象之间构成多态引用

（3）接口可以继承多个父接口

（4）Java类可以同时实现多个父接口，实现类实现接口时，如果实现类不是抽象类，必须实现接口的所有抽象方法。

（11）enum：声明枚举

语法格式：

```
【修饰符】 enum 枚举类型名{
    ...
}
【修饰符】 enum 枚举类型名 【implements 父接口们】{
    ...
}
```

枚举类型的特点：

（1）枚举类型有已知的几个常量对象，必须在枚举类型的首行声明；

（2）枚举类型不能显式声明父类，因为它默认继承java.lang.Enum类型

三、特殊空类型（1个）

（12）void：无返回值类型

语法格式：

```
//抽象方法
【修饰符】 void 方法名（【形参列表】）【throws 异常列表】;

//非抽象方法
【修饰符】 void 方法名（【形参列表】）【throws 异常列表】{

}
```

> 注意：如果在返回值类型是void的方法中，想要提前结束方法运行，可以使用return;

四、流程控制语句相关（10个）

（13）if：条件判断分支结构

（14）else：条件判断分支结构

语法格式：

```
#单分支条件判断
if(条件表达式){
    语句块;
}
```

```
#双分支条件判断
if(条件表达式){
    语句块1;
}else{
    语句块2;
}

#多分支条件判断
if(条件表达式1){
    语句块1;
}else if(条件表达式2){
    语句块2;
}else if(条件表达式3){
    语句块3;
}
...
else{
    语句块n+1;
}
```

（15）switch：选择结构

（16）case：选择结构

语法格式：

```
switch(表达式){
    case 常量值1:
        语句块1;
        【break;】
    case 常量值2:
        语句块2;
        【break;】
    ...
    【default:
        语句块n+1;
        【break;】
        】
}
```

执行过程：

（1）入口

①当switch(表达式)中表达式的值与case后面的某个常量值匹配了，就从这个case进入；

②当switch(表达式)中表达式的值与所有case后面的常量值都不匹配，就从default进入，不管default部分在哪里。

（2）一旦找到入口，就会顺序往下执行，直到遇到出口

（3）出口

①自然出口：switch的结束}

②中断出口：break;

> 注意：

> ①switch(表达式)中表达式的结果的类型有要求：四种基本数据类型(byte,short,int,char)和两种引用数据类型(JDK1.7后支持String，JDK1.5后支持枚举)
>
> ②case后面必须是常量值或常量表达式，绝对不能是变量
>
> ③case后面的常量值不能重复

（17）for：循环

1、普通for循环

语法格式：

```
for(【初始化表达式】；【循环条件】；【迭代表达式】){
    循环体语句块；
}
```

执行过程：

（1）先执行【初始化表达式】

（2）判断【循环条件】

（3）如果【循环条件】成立，就执行{循环体语句块；}，之后执行【迭代表达式】，然后回到（2）

如果【循环条件】不成立，就直接结束for循环

> 注意：
>
> （1）for(;;)两个分号不能省略，但也不能多了
>
> （2）如果for(;;)中间的循环条件没写，就是表示条件永远成立，如果{}中没有break，return等就会死循环

2、增强for循环

语法格式：

```
for(元素数据类型  元素名 ： 数组或集合等容器名){
    循环体语句块；
}
```

说明：

（1）常用于遍历数组和集合等容器，其实只要是实现了java.lang.Iterable接口的容器对象都可以使用foreach来进行遍历，其实调用的是java.util.Iterator迭代器的hasNext()和next()方法

（2）在使用foreach遍历集合时，不要同时调用集合涉及到修改元素个数的方法

错误示例：

```
ArrayList list = new ArrayList();
for(Object obj : list){
    list.add(xx);//错误
    list.remove(xx);//错误
}
```

（18）while：循环

语法格式：

```
while(循环条件){
    循环体语句块;
}
```

执行过程：

（1）先判断循环条件

（2）如果条件成立，执行{循环体语句块;}，然后回到（1）

如果条件不成立，直接结束while循环。

（19）do

语法格式：

```
do{
    循环体语句块;
}while(循环条件);
```

执行过程：

（1）先上来就执行一次{循环体语句块;} 说明do...while系列的循环，至少执行一次循环体语句块;

（2）判断循环条件

（3）如果条件成立，再次执行{循环体语句块;}，然后回到（2）

如果条件不成立，那么直接结束do..while

> 注意：
>
> （1）do...while系列的循环，至少执行一次循环体语句块;
>
> （2）while(循环条件); 后面的;不能省略
>
> （3）如果是在{循环体语句块;}中声明的局部变量，是不能在while()中使用的，如果要在while()中使用这个变量，那么需要提取到do{}上面声明;

（20）break

用法：

（1）switch：结束当前switch

（2）循环：结束当前（层）循环

如果break在内循环中，只能结束内循环;

如果break在外循环中内循环外，可以结束外循环;

如果break结合标签，那么可以直接结束标签对应的循环;

```
//用于switch结构
```

```
switch(表达式){
    case 常量值1:
        语句块1;
        【break;】
    case 常量值2:
        语句块2;
        【break;】
    ...
    【default:
        语句块n+1;
        【break;】
        】
}

//用于for,while,do...while循环
while(true){
    ...
    if(xx){
        break;//结束while循环
    }
}

for(初始化表达式; 循环条件; 迭代表达式){
        ...
    if(xx){
        break;//结束for循环
    }
}


for(初始化表达式; 循环条件; 迭代表达式){

    for(初始化表达式; 循环条件; 迭代表达式){
        if(xx){
            break; //结束的是内循环
        }
    }
}

out:for(初始化表达式; 循环条件; 迭代表达式){

    for(初始化表达式; 循环条件; 迭代表达式){
        if(xx){
            break out; //结束的是out标记的外循环
        }
    }
}
```

(21) continue

用法:

只能用在循环中:提取结束本次循环,跳过了本次循环剩下的循环体语句

使用示例:

```
for(初始化表达式; 循环条件; 迭代表达式){
      ...//上面的循环体语句
    if(xx){
        continue;//提前结束本次循环，本次循环“下面的循环体语句"被跳过了
    }
    ...//下面的循环体语句
}


for(初始化表达式; 循环条件; 迭代表达式){
    ...//上面的外循环体语句
    for(初始化表达式; 循环条件; 迭代表达式){
        ...//上面的内循环体语句
        if(xx){
            continue; //提前结束本次内循环，本次内循环“下面的内循环体语句"被跳过了
        }
        ...//下面的内循环体语句
    }
     ...//下面的外循环体语句
}

out:for(初始化表达式; 循环条件; 迭代表达式){

    for(初始化表达式; 循环条件; 迭代表达式){
        if(xx){
            continue out; //提前结束的是out标记的外循环剩下的语句，相当于提前结束了本轮内循环，直接准备下一次外循环
        }
    }
}
```

（22）return

形式：

（1）return ;

用于提前结束返回值类型是void的方法。

可选：返回值类型为void方法中，可能有return;，也可能没有;

（2）return 返回值;

用于提前结束返回值类型不是void的方法，并且会返回结果。

必选：返回值类型不是void的方法中，必须有return 返回值;语句

五、default（1个）

（23）default：默认的

用法：

1、switch...case流程控制语句结构中

2、JDK1.8之后接口中用于声明默认方法

3、在注解中用于声明某个配置参数的默认值

```
//用于switch结构
switch(表达式){
    case 常量值1:
        语句块1;
        【break;】
    case 常量值2:
        语句块2;
        【break;】
    ...
    【default:
        语句块n+1;
        【break;】
        】
}

//用于接口中
【修饰符】 interface 接口名{
    【权限修饰符】 default 返回值类型 方法名（【形参列表】）{
    }
}

//用于注解中
@元注解
【修饰符】 @interface 注解名{
    数据类型   配置参数名   default 默认值;
}
```

六、关系（5个）

（24）extends：继承关系

语法格式：

```
【修饰符】 class 子类  extends 父类{

}
【修饰符】 interface 子接口  extends 父接口们{

}
```

（25）implements：实现关系

语法格式：

```
【修饰符】 class 子类  【extends 父类】 implements 接口们{

}
```

（26）instanceof：所属关系判断

语法格式：

```
if(对象  instanceof 类型名){
}
只有这个对象属于这个类型，就返回true
```

> 提示：一般用于在向下转型之前，加这个判断可以避免ClassCastException异常

（27）this：引用当前对象的

this关键字：当前对象 （1）在构造器或非静态代码块中，表示正在创建的对象 （2）在成员方法中，表示正在调用当前方法的对象

this的用法： （1）this.属性 或 this.成员变量 当局部变量与成员变量重名时，那么可以在成员变量的前面加this.进行区分

> this.属性，如果在本类中没有找到，也可能表示引用从父类继承到子类中可见的属性

（2）this.成员方法() 可以省略this.，表示访问当前对象的其他成员方法

> this.方法，如果在本类中没有找到，也可能表示引用从父类继承到子类中可见的方法

（3）this()或this(实参列表) 当需要调用本类的其他构造器，可以在当前构造器的首行用this()或this(实参列表)进行调用

> this()和this(实参列表)只会在本类中查找

（28）super：引用父类的

super关键字：引用父类的，找父类的xx

super的用法：

1、super.属性

如果子类声明了一个属性，它和从父类继承的属性同名了，这个时候从父类继承的属性会被hidng(隐藏)，如果此时在子类中想要使用它，那么就要用super.属性。

> super.属性也是访问不了父类的私有的属性，如果跨包，super.属性也访问不了父类权限修饰符缺省的属性
>
> super.属性，如果在直接父类中没有找到，还会向上往间接父类继续查找

2、super.方法

如果子类重写了从父类继承的某个方法后，在子类中又想要调用父类被重写的方法时，那么就可以使用super.方法

> 父类私有的方法是不能被重写的，父类私有的方法通过super.方法也是无法访问的
>
> 如果跨包，父类权限修饰符缺省的方法也不能被重写和通过super.方法访问到
>
> super.方法，如果在直接父类中没有找到，还会向上往间接父类继续查找

3、super()或super(实参列表)

在子类的构造器的首行，

通过super()调用父类的无参构造器，super()这个可以省略。

通过super(实参列表)调用功能父类的有参构造，super(实参列表)不能省略。

> 注意：super()和super(实参列表)只能从直接父类找，不能跨过直接父类去引用间接父类的

七、创建对象（1个）

（29）new：创建

用于创建数组对象、类的对象

创建的对象会存储在堆中。

八、和包相关的（2个）

（30）package：声明包

1、包的作用： （1）避免类的重名 （2）控制某些类、成员的可见范围 （3）分包进行组织管理众多的类

2、声明包的语法格式：

```
package 包名;
```

> 注意：package语句必须在.java源文件的首行

3、包的命名规范和习惯： （1）所有单词都小写，每个单词之间使用.分割 （2）习惯用公司域名倒置 + 模块名

（31）import：导包

4、使用其他包的类： （1）使用全名称：包.类名 （2）使用import语句 + 简名称

5、import语句

```
import 包.类名;
import 包.*;
import static 包.类名.静态成员名;
import static 包.类名.*;
```

> 注意：当使用两个不同包的同名类时，例如：java.util.Date和java.sql.Date，只能一个使用全名称，一个使用导包。

九、修饰符

1、权限修饰符（3个）

（32）private：私有的

（33）protected：受保护的

（34）public：公共的

权限修饰符	本类	本包中其他类	其他包的子类	其他包的非子类（任意位置）	可以修饰
private	√	×	×	×	属性、方法、构造器、成员内部类
缺省	√	√	×	×	属性、方法、构造器、成员内部类、外部类
protected	√	√	√	×	属性、方法、构造器、成员内部类
public	√	√	√	√	属性、方法、构造器、成员内部类、外部类

2、其他修饰符（8个）

（35）static：静态的

static：静态的，可以修饰成员变量、成员方法、代码块、成员内部类 （1）成员变量，属性： static修饰的成员变量称为“类变量，或静态变量”，

它的值是该类所有对象的共享的，存储在方法区，

它的get/set方法也是静态的，如果在静态方法中，类变量与局部变量重名时，使用“类名.”进行区别。

（2）成员方法： static修饰的成员方法称为“类方法，或静态方法”，它不能被重写，可以被继承，

调用它可以用“类名.”进行调用，

在静态方法中，不能出现this,super,不能直接使用本类的非静态的属性、非静态的方法、非静态的成员内部类。

（3）代码块： static修饰的代码块称为“静态代码块”，它是在类初始化时执行，因为它的代码会被编译器合并到()类初始化方法中，它只执行一次，子类的初始化时，如果发现父类没有初始化，会先初始化父类。

（4）成员内部类

static修饰的成员内部类称为“静态内部类”，

静态内部类中可以包含静态成员；

静态内部类在外部类外面使用时，使用“外部类名.静态内部类"即可。

语法格式：

```
【修饰符】 class 类{
    【修饰符】 static 数据类型 静态变量;

    static{
        静态代码块;
    }

    【修饰符】 static 返回值类型 方法名(【形参列表】)【throws 异常列表】{

    }

    【修饰符】 static class 静态内部类{

    }
}
```

（36）native：原生的

native：本地的，原生的 只能修饰方法，表示这个方法的方法体不是用Java语言实现的， 但是可以和普通的Java方法一样去调用和重写它。

（37）final：最终的

final：最终的，可以修饰类、方法、变量 （1）类（包括外部类、内部类）：不能被继承 （2）方法：不能被子类重写，可以被继承 （3）变量（包括成员变量和局部变量）：值不能修改，即为常量 建议常量名大写，每个单词之间使用_分割，形式：XXX_YYY_ZZZ

（38）abstract：抽象的

abstract：抽象的，只能修饰类、方法

（1）方法：

abstract修饰的方法称为“抽象方法”

```
【权限修饰符】 abstract class 抽象类{
    【修饰符】 abstract 返回值类型  方法名(【形参列表】)【throws 异常列表】;
}

【权限修饰符】 interface 接口名{
    public abstract 返回值类型  方法名(【形参列表】)【throws 异常列表】;
}
```

特点：

抽象方法没有方法体，

包含抽象方法的类必须是抽象类，

子类继承抽象类或实现类实现接口时，如果该子类或实现类不是抽象类，那么必须实现抽象父类和接口的所有抽象方法。

（2）类：

abstract修饰的类称为“抽象类”。

语法格式：

```
【权限修饰符】 abstract class 抽象类{

}
【权限修饰符】 abstract class 抽象类 【extends 父类】 【implements 父接口们】{

}
```

特点：

（1）拥有一个或者多个抽象方法的类“必须”是抽象类

（2）抽象类不能直接创建对象，即不能实例化，但是抽象类的变量可以与子类的对象构成多态引用。

（3）有时抽象类中没有抽象方法，（目的只有一个：不让你创建对象）

（4）抽象类就是用来被继承的，子类继承抽象类时，必须对父类的抽象方法进行实现，否则子类也得是抽象类

（5）抽象类也是类，因此原来类中可以有的5大成员，抽象类都可以有

（39）synchronized：同步的

synchronized：同步的，可以修饰方法和同步代码块

（1）同步方法

```
【修饰符】 synchronized 返回值类型 方法名(【形参列表】)【throws 异常列表】{
    ...
}
```

同步方法的锁对象：

静态方法：当前类.class

非静态方法：this

（2）同步代码块

```
synchronized(锁对象){

}
```

（40）volatile：易变的

volatile：易变的，不定性的，可以修饰成员变量

表示该成员变量的值是易变的，每一次获取它的值都要从主存中重新读取，以保证在多线程中，不同线程读取到的该值都是最新的。

因为Java中多线程读取某个成员变量时，发现一段时间内它的值都未发生变化，Java执行引擎就会把这个值放在缓存中，以后的线程读取，就会读取这个缓存值，即使这个时候某个线程修改了该变量主存中的值，Java执行引擎仍然会去读取缓存的值，而如果希望线程总是读取最新的该变量的值，那么可以在变量前面加volatile，使得Java执行引擎都从主存中读取，而不缓存。

（41）strictfp：严格遵循FP模式

strictfp：表示要求严格遵循FP模式，可以修饰类、接口、方法

使用 strictfp 关键字声明一个方法时，该方法中所有的float和double表达式都严格遵守FP-strict的限制,符合IEEE-754规范。

当对一个类或接口使用 strictfp 关键字时，该类中的所有代码，包括嵌套类型中的初始设定值和代码，都将严格地进行计算。严格约束意味着所有表达式的结果都必须是 IEEE 754 算法对操作数预期的结果，以单精度和双精度格式表示。 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，可以用关键字strictfp。

（42）transient：瞬时的

transient：表示瞬时的，临时的，短暂的，转瞬即逝的;

用于修饰成员变量;

transient修饰的成员变量的值，如果该类实现的是java.io.Serializable接口，那么在序列化过程中该成员变量不会参与序列化。

十、和异常处理相关的（5个）

（43）try：尝试执行

（44）catch：尝试捕获异常对象

（45）finally：最终块

```
//形式1：
try{
    语句块;
}catch(异常类型1 异常对象名1){
    处理异常代码块1;
}catch(异常类型1 异常对象名1){
    处理异常代码块1;
}
...

//形式2：
try{
    语句块;
}catch(异常类型1 异常对象名1){
    处理异常代码块1;
}catch(异常类型1 异常对象名1){
    处理异常代码块1;
}
...
finally{
```

```
    最终语句块;
}

//形式3:
try{
    语句块;
}finally{
    最终语句块;
}
```

try：尝试执行某些代码，如果发生异常，将会抛出异常对象，让catch去捕获；

catch：尝试捕获try中抛出的异常对象，如果类型匹配，就可以捕获，当前方法不会结束；如果所有catch都无法捕获，将会结束当前方法。

finally：无论try中是否有异常抛出，也无论是否catch捕获了该异常，也无论try和catch中是否有return语句都会执行。

（46）throw：抛出

用于手动抛出异常

```
throw 异常对象;
```

throw可以用于抛出异常对象，如果它抛出的异常，没有被catch的话，可以代替return语句结束当前方法，并将把异常对象带回调用处。

用户自定义异常只能使用throw语句手动抛出。

（47）throws：抛出异常列表

用于在方法签名中，声明该方法将抛出哪些类型的异常。

```
【修饰符】 返回值类型 方法名(【形参列表】) throws 异常类型列表{

}
```

表示这些异常在当前方法中没有处理，交给调用者进行处理。

十一、assert（1个）

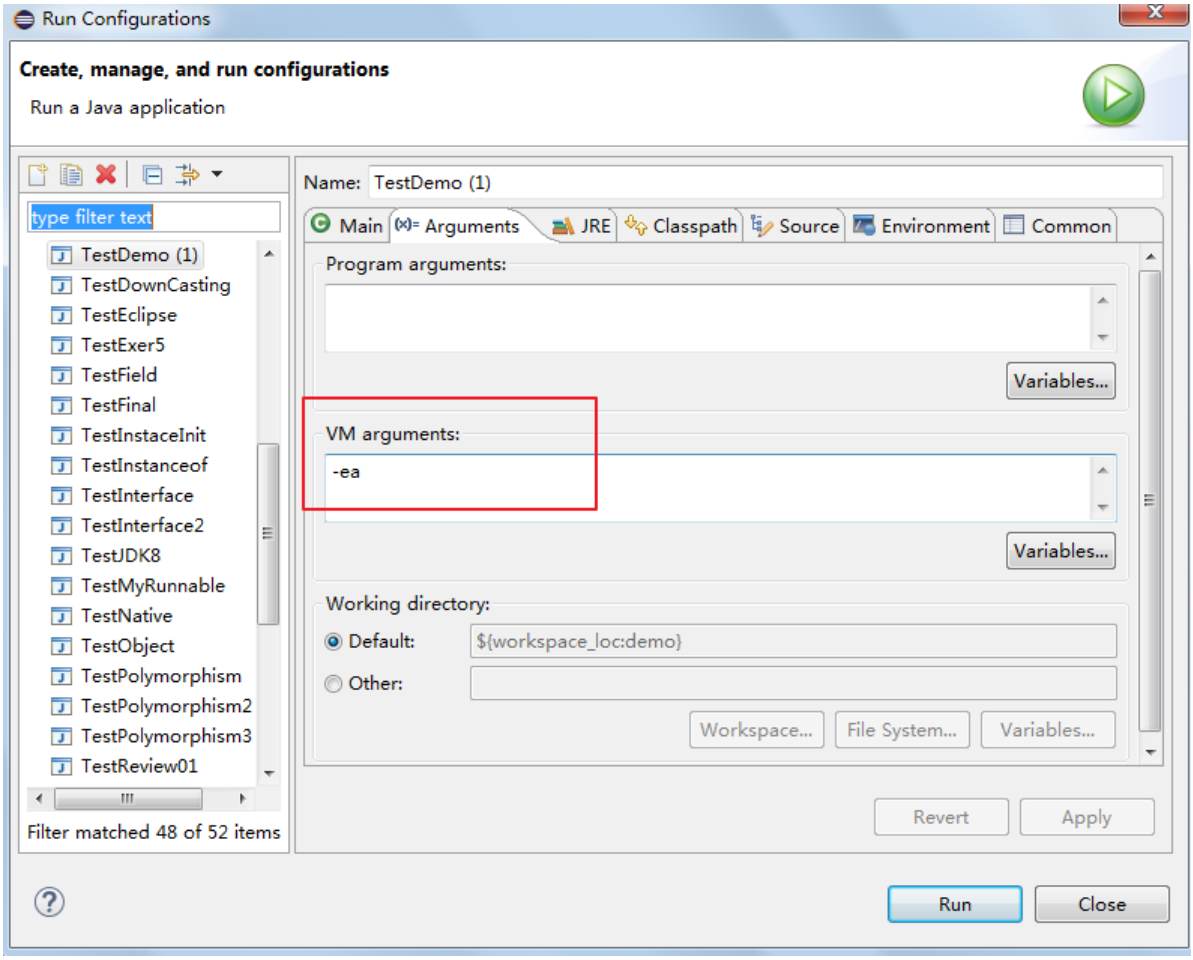
（48）assert：断言

如果它断言的表达式为false，将会抛出java.lang.AssertionError对象。

语法格式：

```
assert 布尔表达式;
或
assert 布尔表达式 : "错误信息";
```

> 注意：要开启断言功能，在eclipse中需要加JVM参数 -ea



示例：

```
public class TestDemo {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        assert a/b == 0.5 :"结果不正确";
    }
}
```

```
Console
<terminated> TestDemo (1) [Java Application] D:\ProgramFiles\Java\jdk1.8.0_141\bin\javaw.exe (2019年4月14日 下午1:30:44)
Exception in thread "main" java.lang.AssertionError: 结果不正确
        at com.atguigu.TestDemo.main(TestDemo.java:8)
```

十二、保留字（2个）

（49）const：常数，不变的

在C语言等其他很多编程语言中用于声明常量。在Java中没有使用它。

（50）goto：跳转

在C语言等其他很多编程语言中用于跳转到指定位置。在Java中没有使用它。

十三、特殊值（3个）

（51）true：真

（52）false：假

boolean值，通常用来作为条件。

boolean类型的变量，比较表达式（>,<,>=,<=,==,!=），逻辑表达式（&&,&,||,|,^,!），instanceof，这些都是boolean值。

（53）null：空，如果null调用属性和方法会报空指针异常

这三个看似关键字，但是从技术角度来说，不能算是关键字，是特殊值。