[JAVA环境配置 2](#_Toc799760563)

[Windows下环境配置 2](#_Toc1840493859)

[Linux下环境配置 3](#_Toc1809870013)

[JAVA基础语法 4](#_Toc1563701534)

[Java关键字 5](#_Toc556313989)

# JAVA环境配置

## Windows下环境配置

下载java开发工具包JDK，下载地址：

[http://www.oracle.com/technetwork/java/javase/downloads/index.html](http://www.oracle.com/technetwork/java/javase/downloads/index.html" \t "http://www.runoob.com/java/_blank)



在"系统变量"中设置3项属性，JAVA\_HOME,PATH,CLASSPATH(大小写无所谓),若已存在则点击

编辑"，不存在则点击"新建"。变量设置参数如下：

变量名：**JAVA\_HOME**

变量值：**C:\Program Files (x86)\Java\jdk1.8.0\_91**

// 要根据自己的实际路径配置

变量名：**CLASSPATH**

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

//记得前面有个"."

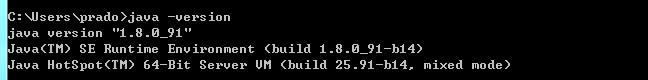
变量名：**Path**

变量值：**%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;**

***注意：****如果使用1.5以上版本的JDK，不用设置CLASSPATH环境变量，也可以正常编译和运行Java程序*

1、"开始"->"运行"，键入"cmd"；

2、键入命令: **java -version**、**java**、**javac** 几个命令，出现以下信息，说明环境变量配置成功；



## Linux下环境配置

***PATH环境变量：***作用是指定命令搜索路径，在shell下面执行命令时，它会到PATH变量所指定的路径中查找看是否能找到相应的命令程序。我们需要把 jdk安装目录下的bin目录增加到现有的PATH变量中，bin目录中包含经常要用到的可执行文件如javac/java/javadoc等，设置好 PATH变量后，就可以在任何目录下执行javac/java等工具了。   
***CLASSPATH环境变量：***作用是指定类搜索路径，要使用已经编写好的类，前提当然是能够找到它们了，JVM就是通过CLASSPTH来寻找类的。我们需要把jdk安装目录下的lib子目录中的dt.jar和tools.jar设置到CLASSPATH中，当然，当前目录“.”也必须加入到该变量中。   
***JAVA\_HOME环境变量：***它指向jdk的安装目录，Eclipse/NetBeans/Tomcat等软件就是通过搜索JAVA\_HOME变量来找到并使用安装好的jdk。

1. 修改.bash\_profile文件

这种方法更为安全，它可以把使用这些环境变量的权限控制到用户级别，如果你需要给某个用户权限使用这些环境变量，你只需要修改其个人用户主目录下的.bash\_profile文件就可以了。   
·用文本编辑器打开用户目录下的.bash\_profile文件   
·在.bash\_profile文件末尾加入：   
export JAVA\_HOME=/usr/share/jdk1.6.0\_14   
export PATH=$JAVA\_HOME/bin:$PATH   
export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

重新登陆

1. 修改/etc/profile文件   
   如果你的计算机仅仅作为开发使用时推荐使用这种方法，因为所有用户的shell都有权使用这些环境变量，可能会给系统带来安全性问题。   
   ·用文本编辑器打开/etc/profile   
   ·在profile文件末尾加入：   
   export JAVA\_HOME=/usr/share/jdk1.6.0\_14   
   export PATH=$JAVA\_HOME/bin:$PATH   
   export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

重新登陆

1. 直接在shell下设置变量

不赞成使用这种方法，因为换个shell，你的设置就无效了，因此这种方法仅仅是临时使用，以后要使用的时候又要重新设置，比较麻烦。   
只需在shell终端执行下列命令：   
export JAVA\_HOME=/usr/share/jdk1.6.0\_14   
export PATH=$JAVA\_HOME/bin:$PATH   
export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

 4.卸载jdk   
找到jdk安装目录的\_uninst子目录   
在shell终端执行命令./uninstall.sh即可卸载jdk。

# JAVA基础语法

**一个Java程序可以认为是一系列对象的集合，而这些对象通过调用彼此的方法来协同工作。下面简要介绍**

**类、对象、方法和实例变量的概念。**

**对象**：对象是类的一个实例，有状态和行为。例如，一条狗是一个对象，它的状态有：颜色、名字、品种；

行为有：摇尾巴、叫、吃等。

**类**：类是一个模板，它描述一类对象的行为和状态。

**方法**：方法就是行为，一个类可以有很多方法。逻辑运算、数据修改以及所有动作都是在方法中完成的。

**实例变量**：每个对象都有独特的实例变量，对象的状态由这些实例变量的值决定。

## Java关键字

下面列出了Java保留字。这些保留字不能用于常量、变量、和任何标识符的名称。

abstract: 抽象方法，抽象类的修饰符

assert: 断言条件是否满足

boolean: 布尔数据类型

break: 跳出当前循环

byte: 8-bit有符号类型

case: switch语句的一个条件

catch: 和try搭配，用来捕获异常信息

char: 16-bit Unicode字符数据类型

class: 定义类

constant：定义常量

continue: 不执行循环体剩余部分

default: switch语句中默认分支

do: 循环语句，循环体至少会执行一次

double: 64-bit双精度浮点数

else: if条件不成立时执行的分支

enum: 枚举类型

extends: 表示一个类是另一个类的子类

final: 表示一个值在初始化之后就不能再改变了，表示方法不能被重写，或者一个类不能有子类

finally: 无论有没有异常发生都会执行的代码，实质是为了完成执行的代码而设计的，主要是为了程序的健壮性和完整性而设计的

float: 32-bit单精度浮点数

for: for循环语句

goto：未使用，其他条件分支语句可以代替

if: 条件语句

implements: 表示一个类实现了接口

import: 导入类

instanceof: 测试一个对象是否是某个类的实例

int: 32位整型数

interface: 接口，一种抽象的类型，仅有方法和常量的定义

long: 64位整型数

native: Java语言本身不能对操作系统底层进行访问和操作，但是可以通过调用其它语言来实现对底层的访问，但是可以通过JNI接口调用其它语言来实现对底层的访问。native关键字修饰的方法能够扩展JAVA程序的功能

new: 分配新的类的实例

package：一系列相关类组成一个包

private: 表示私有字段，或者方法等，只能从类内部访问

protected: 表示字段只能通过类或者其子类访问，子类或者在同一个包内的其他类

public: 表示共有属性或者方法

return: 方法返回值

short: 16位数字

static:

strictfp:

super: 表示基类

switch: 选择语句

synchronized: 表示同一时间只能由一个线程访问的代码块

this: 表示调用当前实例，或者调用另一个构造函数

throw: 抛出异常

throws:

transient:

try: 表示代码块要做异常处理

void: 标记方法不返回任何值

volatile:

while: while循环

# 字符串

## 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类在Java 5中被提出，它和StringBuffer之间的最大不同在于StringBuilder的方法不是线程安全的（不能同步访问）。

由于StringBuilder相较于StringBuffer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。

## finalize() 方法

Java允许定义这样的方法，它在对象被垃圾收集器析构(回收)之前调用，这个方法叫做finalize( )，它用来清除回收对象。

例如，你可以使用finalize()来确保一个对象打开的文件被关闭了。

在finalize()方法里，你必须指定在对象销毁时候要执行的操作。

finalize()一般格式是：

protected void finalize(){

// 在这里终结代码}

数组

容器

JAVA对象和类

枚举

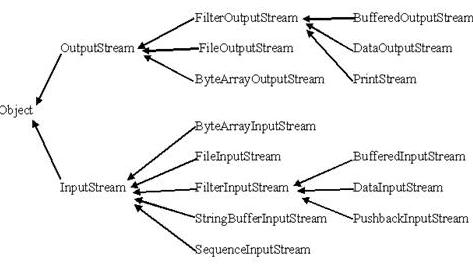
接口

泛型

注解

异常处理

JAVA I/O系统



## FileInputStream

该流用于从文件读取数据，它的对象可以用关键字new来创建。

有多种构造方法可用来创建对象。

可以使用字符串类型的文件名来创建一个输入流对象来读取文件：

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

也可以使用一个文件对象来创建一个输入流对象来读取文件。我们首先得使用File()方法来创建一个文件对象：

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

## FileOutputStream

该类用来创建一个文件并向文件中写数据。

如果该流在打开文件进行输出前，目标文件不存在，那么该流会创建该文件。

有两个构造方法可以用来创建FileOutputStream 对象。

使用字符串类型的文件名来创建一个输出流对象：

OutputStream f = new FileOutputStream("C:/java/hello")

也可以使用一个文件对象来创建一个输出流来写文件。我们首先得使用File()方法来创建一个文件对象：

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

# Java Scanner 类

java.util.Scanner是Java5的新特征，我们可以通过 Scanner 类来获取用户的输入。

下面是创建 Scanner 对象的基本语法：

Scanner s = new Scanner(System.in);

接下来我们演示一个最简单的的数据输入，并通过 Scanner 类的 next() 与 nextLine() 方法获取输入的字符串，在读取前我们一般需要 使用 hasNext 与 haxNextLine 判断是否还有输入的数据：

### **使用 next 方法：**

import java.util.Scanner;

public class ScannerDemo {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

// 从键盘接收数据

//next方式接收字符串

System.out.println("next方式接收：");

// 判断是否还有输入

if(scan.hasNext()){

String str1 = scan.next();

System.out.println("输入的数据为："+str1);

}

} }

执行以上程序输出结果为：

$ javac ScannerDemo.java

$ java ScannerDemonext方式接收：

runoob com输入的数据为：runoob

可以看到 com 字符串并未输出，接下来我们看 nextLine。

### **使用 nextLine 方法：**

import java.util.Scanner;

public class ScannerDemo {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

// 从键盘接收数据

//nextLine方式接收字符串

System.out.println("nextLine方式接收：");

// 判断是否还有输入

if(scan.hasNextLine()){

String str2 = scan.nextLine();

System.out.println("输入的数据为："+str2);

}

} }

**执行以上程序输出结果为：**

$ javac ScannerDemo.java

$ java ScannerDemo

nextLine方式接收：

runoob com输入的数据为：runoob com

**可以看到 com 字符串输出。**

### **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。
* 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。
* next()不能得到带有空格的字符串。

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。

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：

import java.util.Scanner;

public class ScannerDemo {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

// 从键盘接收数据

int i = 0 ;

float f = 0.0f ;

System.out.print("输入整数：");

if(scan.hasNextInt()){

// 判断输入的是否是整数

i = scan.nextInt() ;

// 接收整数

System.out.println("整数数据：" + i) ;

}else{

// 输入错误的信息

System.out.println("输入的不是整数！") ;

}

System.out.print("输入小数：");

if(scan.hasNextFloat()){

// 判断输入的是否是小数

f = scan.nextFloat() ;

// 接收小数

System.out.println("小数数据：" + f) ;

}else{

// 输入错误的信息

System.out.println("输入的不是小数！") ;

}

} }

执行以上程序输出结果为：

$ javac ScannerDemo.java

$ java ScannerDemo输入整数：12整数数据：12输入小数：1.2小数数据：1.2

以下实例我们可以输入多个数字，并求其总和与平均数，每输入一个数字用回车确认，通过输入非数字来结束输入并输出执行结果：

import java.util.Scanner;

class ScannerDemo {

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

double sum = 0;

int m = 0;

while(scan.hasNextDouble())

{

double x = scan.nextDouble();

m = m + 1;

sum = sum + x;

}

System.out.println(m+"个数的和为"+sum);

System.out.println(m+"个数的平均值是"+(sum/m));

} }

执行以上程序输出结果为：

$ javac ScannerDemo.java

$ java ScannerDemo12231521.4end4个数的和为71.44个数的平均值是17.85

# Java 数据结构

Java工具包提供了强大的数据结构。在Java中的数据结构主要包括以下几种接口和类：

* 枚举（Enumeration）
* 位集合（BitSet）
* 向量（Vector）
* 栈（Stack）
* 字典（Dictionary）
* 哈希表（Hashtable）
* 属性（Properties

线程

并发

网络编程

函数式编程