

[

# Java千百问\_02基本使用（001）\_如何用记事本编写Java程序

，  
[点击进入\\_更多\\_Java千百问](#)

## 如何用记事本编写Java程序

在安装完Jdk之后，我们就想编写java程序，但是，有了Jdk之后就够了吗？就能够写我们的程序实现我们想要的功能吗？

答案是：**当然！**

在**远古时代**，那时候我们还没有集成好的开发软件，没有各种个样的IDE，我们只能靠着**记事本**来一点一点码我们的代码，经历过那段时光的同学，但凡你敢说你能写代码，就会有各种牛逼企业请你去，风光无限啊。

回到正题，通过记事本写Java，首先你要有一个**记事本**，这当然不是废话，windows自带的可以，linux的vim可以，mac的文本编辑可以，只要能打开文本文件、支持另存为的软件都可以。

1.首先新建记事本，输入我们的**代码**：

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

将文本另存为**.java**文件，**HelloWord.java**。

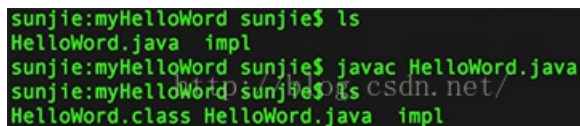
2.然后我们对它进行**编译**和**运行**：

首先，使用**cmd/终端**进入该文本存放的目录。

然后，使用jdk提供的工具**javac**编译代码，这个工具就是为了**编译代码**存在的。

**cmd/终端**中输入：

```
javac HelloWorld.java
```



```
sunjie:myHelloWord sunjie$ ls  
HelloWord.java  impl  
sunjie:myHelloWord sunjie$ javac HelloWorld.java  
sunjie:myHelloWord$ ls csdn.net/  
HelloWord.class HelloWorld.java  impl
```

等待完成后，会在同级目录生成同名的**HelloWord.class**。

3.最后，使用jdk提供的工具java运行代码，被运行的类需要有main方法才能通过java工具运行。

cmd/终端中输入（如果报错，请看：[为什么会报"错误:找不到或无法加载主类 HelloWorld.class"](#)）：

```
java HelloWorld
```

```
sunjie:myHelloWord sunjie$ java HelloWorld
Hello word
```

查看运行结果，成功打印“Hello word”，本文完结。

什么！！你告诉我这就完了？当然不是，下面我们看一下jar包的编译和运行。[不使用IDE如何打jar包](#)

[点击进入ooppookid的博客](#)

]

[

## Java千百问\_02基本使用（002）\_为什么会报"错误: 找不到或无法加载主类 HelloWorld.class"

,  
[点击进入\\_更多\\_Java千百问](#)

### 为什么会报"错误: 找不到或无法加载主类 HelloWorld.class"

java工具只认识类名，而不认识文件，执行时，它会遍历所在路径的类，如果找不到就会报：

"错误: 找不到或无法加载主类 HelloWorld.class"

```
sunjie:myHelloWord sunjie$ java HelloWorld.class  
错误: 找不到或无法加载主类 HelloWorld.class
```

解决非常简单，在cmd/终端中使用java运行工具时，是不能带有.class的。

当然，如果对于jar包，执行时需要带有.jar，如：

```
java -jar HelloWorld.jar
```

解决办法：将java HelloWorld.class改为：

```
java HelloWorld
```

```
sunjie:myHelloWord sunjie$ java HelloWorld  
Hello word
```

[点击进入ooppookid的博客](#)

]

[

# Java千百问\_02基本使用（003）\_不使用IDE如何打jar包

,

[点击进入\\_更多\\_Java千百问](#)

## 不使用IDE如何打jar包

### 1.什么是jar包

jar包，最直白的感受就是后缀是.jar的一种压缩文件，它是以zip文件格式为基础的压缩包。

与zip文件不同的是，jar文件不仅用于压缩和发布，而且还用于部署和封装库、组件和插件程序。

jar包可以被类似Jvm这样的工具直接使用，通过MANIFEST、部署描述符等特殊文件，来指示工具处理特定的jar。

jar包大体分为2种，可执行jar包，不可执行jar包。

目前大多数的jar包都是不可被执行的，这类jar包就是提供给其他应用显现某些特定功能的。而可执行jar包可以在java环境上直接运行，大多数体现为小型工具。

### 2.不使用IDE如何打jar包

本文是延续：[如何用记事本编写Java程序](#)，所以还是通过记事本来完成教程。

1.首先，为了区别单文件，这次我们在不同的包下编写了两个类。类的具体功能非常明确，不再多说。

```
package com.helloworld;

import com.helloworld.impl>HelloWordImpl;

public class HelloWord {

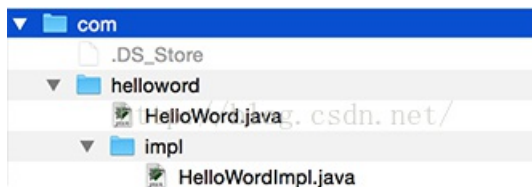
    public static void main(String[] args) {
        HelloWordImpl helloWordImpl = new HelloWordImpl();
        System.out.println(helloWordImpl.getText());
    }
}

package com.helloworld.impl;

public class HelloWordImpl {

    public String getText() {
        return "Hello word jar";
    }
}
```

2.其次，把这两个类放在对应目录下，所谓"对应"及要求类的包与实际路径相符，只有这样，运行时才能找到正确路径下的类。

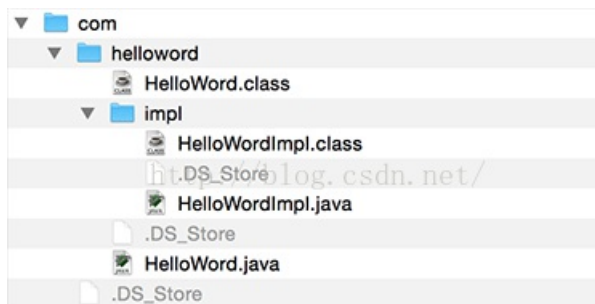


3.然后，使用`javac`工具进行批量编译，如果需要编译的量很大的话，需写一些工具脚本进行大量编译（TODO，以后会有文章介绍）。

```
javac com/helloworld/impl/HelloWordImpl.java com/helloworld/HelloWord.java
```

```
sunjie:Downloads sunjie$ javac com/helloworld/impl/HelloWordImpl.java com/helloworld/HelloWord.java
sunjie:Downloads sunjie$
```

编译后会在对应目录生成`.class`文件。



4.最后，对其进行打包。在打包之前，我们需要把我们的整个文件夹`移动`到一个新的文件夹下，这里将`com`放在了一个叫`bin`的文件夹下（如果不想把源代码打入其中，需要将`.java`文件`移除`）。

打包需要使用`jar`工具，这个工具会将所制定的文件夹里的内容打包成`jar`。`cmd/终端`进入`bin`所在的路径，输入：

```
jar cvf myHelloWord.jar -C bin .
```

其中`"myHelloWord.jar"`是生成`jar`包的名称，`"bin"`是我们的文件夹目录。

P.S.

该方式打的`jar`包是`没有办法运行`的。其中，`cvf`是`jar`工具提供的几个参数，下面是`jar`命令的帮助说明：

用法: `jar {ctxui}[vfm0Mc] [jar-file] [manifest-file] [entry-point] [-C dir] files ...`

选项包括：

- c 创建新的归档文件
- t 列出归档目录
- x 解压缩已归档的指定（或所有）文件
- u 更新现有的归档文件
- v 在标准输出中生成详细输出
- f 指定归档文件名
- m 包含指定清单文件中的清单信息
- e 为捆绑到可执行 `jar` 文件的独立应用程序，指定应用程序入口点
- 0 仅存储；不使用任何 ZIP 压缩
- M 不创建条目的清单文件
- i 为指定的 `jar` 文件生成索引信息

-C更改为指定的目录并包含其中的文件，如果有任何目录文件，则对其进行递归处理。

清单文件名、归档文件名和入口点名的指定顺序与 "m"、"f" 和 "e" 标志的指定顺序相同。

**示例 1：**将两个类文件归档到一个名为 classes.jar 的归档文件中：

```
jar cvf classes.jar A.class B.class
```

**示例 2：**使用现有的清单文件 "mymanifest" 并将 my/ 目录中的所有文件归档到 "classes.jar" 中：

```
jar cvfm classes.jar mymanifest -C my/ .
```

5.我们需要的是一个能够运行的jar，所以我们需要编写并指定MANIFEST.MF，告诉java工具jar包的执行入口（即我们的main方法）。

在com的同级目录（bin下面）下创建名称为"MANIFEST.MF"的文件，并输入如下内容（还可以指定更多jar包的信息，这里我们只指定它的main类）：

```
Manifest-Version: 1.0
Main-Class: com.helloworld>HelloWord
```

保存之后，通过jar工具打包，这里要指定MANIFEST.MF文件：

```
jar cvfm counter.jar MANIFEST.MF -C bin .
```

打包结果如下：

```
sunjie:myHelloWord sunjie$ jar cvfm myHelloWord.jar MANIFEST.MF -C bin .
已添加清单
正在添加: com/(输入 = 0) (输出 = 0)(存储了 0%)
正在添加: com/.DS_Store(输入 = 6148) (输出 = 219)(压缩了 96%)
正在添加: com/helloworld/(输入 = 0) (输出 = 0)(存储了 0%)
正在添加: com/helloworld/.DS_Store(输入 = 6148) (输出 = 198)(压缩了 96%)
正在添加: com/helloworld/HelloWord.class(输入 = 520) (输出 = 333)(压缩了 35%)
正在添加: com/helloworld/HelloWord.java(输入 = 257) (输出 = 148)(压缩了 42%)
正在添加: com/helloworld/impl/(输入 = 0) (输出 = 0)(存储了 0%)
正在添加: com/helloworld/impl/.DS_Store(输入 = 6148) (输出 = 178)(压缩了 97%)
正在添加: com/helloworld/impl/HelloWordImpl.class(输入 = 313) (输出 = 233)(压缩了 25%)
正在添加: com/helloworld/impl/HelloWordImpl.java(输入 = 130) (输出 = 106)(压缩了 18%)
```

这样就能在bin下看到打好的myHelloWord.jar包。

### 3.如何运行jar包

cmd/终端进入jar包所在的路径，输入：

```
java -jar myHelloWord.jar
```

运行结果：

```
sunjie:myHelloWord sunjie$ java -jar myHelloWord.jar
Hello word jar
```

[点击进入oopppookid的博客](#)



[

# Java千百问\_02基本使用（004）\_java开发应该使用什么工具

,

[点击进入\\_更多\\_Java千百问](#)

## java开发应该使用什么工具

如果想编写大量的java代码，使用记事本开发费时又费力，而且非常容易出错，也不能很方便的编译运行，效率非常低（使用记事本开发：[如何用记事本编写Java程序](#)）。在这种情况下，一款快捷、宜用的开发工具就非常必要了。开发Java，根据开发方向的不同，我们会选择不同的工具，最为普遍的就是免费的Eclipse、NetBeans，收费的MyEclipse、JBuilder等，这些都属于集成开发环境（即IDE），我们首先看看什么是开发领域的IDE。

### 1.什么是IDE

IDE，即Integrated Development Environment，即集成开发环境，软件开发领域的专属工具，可以辅助开发程序的应用软件。

IDE的构成：

一般包括代码编辑器、编译器、调试器和图形用户界面工具。即集成了代码编写功能、分析功能、编译功能、调试功能、运行功能等一体化的开发软件套装（目前大部分IDE还包括代码生成、建模功能等）。如微软的VisualStudio系列，Borland的C++Builder，Delphi系列，Eclipse等。

IDE的使用：

可以独立运行，也可以和其它程序并用。例如，BASIC语言在微软办公软件中可以使用，可以在微软Word文档中编写WordBasic程序。IDE为用户使用VisualBasic、Java和PowerBuilder等现代编程语言提供了方便。

IDE的体系：

IDE的体系可以按照高级语言划分，例如C++、VB、C#、Java、Html等语言的集成开发环境。

### 2.什么是Eclipse

Eclipse对于绝大多数Java开发者来说，犹如吃饭的碗、代步的车、居住的房一样密不可分，作为Java最主流的IDE，我们没有理由怀疑它的贡献。

虽然大多数用户很乐于将Eclipse当作Java集成开发环境（IDE）来使用，但Eclipse的目标却不仅限于此。Eclipse还包括插件开发环境（Plug-in Development Environment，PDE），这个组件主要针对希望扩展Eclipse的软件开发人员，因为它允许他们构建与Eclipse环境无缝集成的工具。由于Eclipse中的每样东西都是插件，对于给Eclipse提供插件，以及给用户提供一个一致和统一的集成开发环境而言，所有工具开发人员都具有同等的发挥场所。

这种平等和一致性并不仅限于Java开发工具。尽管Eclipse是使用Java语言开发的，但它的用途并不限于Java语言；例如，支持诸如C/C++、COBOL、PHP、Android等编程语言的插件已经可用，或预计将会推出。Eclipse框架还可用来作为与软件开发无关的其他应用程序类型的基础，比如内容管理系统。

基于Eclipse的应用程序的一个突出例子是IBM Rational Software Architect，它构成了IBM Java开发工具系列的基础。



### 3.Eclipse是如何发展的

Eclipse 最初由OTI和IBM两家公司的IDE产品开发组创建，起始于1999年4月。作为Visual Age for Java的替代品，IBM提供了最初的Eclipse代码基础，包括Platform、JDT（Java Development Tools，是一组为Eclipse平台添加了功能齐全的Java集成开发环境功能的插件）和PDE（plug-in development environment，插件开发环境）。最初主要用来Java语言开发，通过安装不同的插件Eclipse可以支持不同的计算机语言，比如C++和Python等开发工具。

围绕着Eclipse项目已经发展成为了一个庞大的Eclipse联盟，有150多家软件公司参与到Eclipse项目中，其中包括Borland、Rational Software、Red Hat及Sybase等。由于其开放源码，任何人都可以免费得到，并可以在此基础上开发各自的插件，因此越来越受人们关注。随后还有包括Oracle在内的许多大公司也纷纷加入了该项目，Eclipse的目标是成为可进行任何语言开发的IDE集成者，使用者只需下载各种语言的插件即可。目前已经有许多软件开发商以Eclipse为框架开发自己的IDE。

里程碑：

2003年，Eclipse 3.0选择osgi服务平台规范为运行时架构。

2007年6月，稳定版3.3发布；

2008年6月发布代号为Ganymede的3.4版；

2009年6月发布代号为Galileo的3.5版；

2010年6月发布代号为Helios的3.6版；

2011年6月发布代号为Indigo的3.7版；

2012年6月发布代号为Juno的4.2版；

2013年6月发布代号为Kepler的4.3版；

2014年6月发布代号为Luna的4.4版；

2015年6月项目发布代号为Mars的4.5版。

从2006年起，Eclipse基金会每年都会安排同步发布（simultaneous release）。至今，同步发布主要在6月进行，并且会在接下来的9月及2月释放出SR1及SR2版本。

[点击进入oopbookid的博客](#)

# Java千百问\_02基本使用（005）\_Mac环境下无法打开eclipse怎么办

[点击进入\\_更多\\_Java千百问](#)

## Mac环境下无法打开eclipse怎么办

有很大的可能是eclipse（什么是Eclipse: [java开发应该使用什么工具](#)）没有指定jdk版本（如何安装Jdk: [如何安装和配置Jdk](#)）路径，启动时候加载不到jdk导致的闪退。废话不多说，直接说解决办法。

1、打开\$ECLIPSE\_HOME/Eclipse.app/Contents/MacOS/eclipse.ini文件：

在Finder中右键或者Ctrl+点击Eclipse应用程序，然后点击“显示包内容”，进入目录Contents/MacOS即可找到。

2、通过ls-ltr/Library/Java/JavaVirtualMachines/列出已经安装好的各个JDK版本的路径，然后在eclipse.ini文件中指定之，比如：

```
-vm
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java

系统默认则是：
/System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands/java
```

附带给出查询JAVA\_HOME指向的查询方法，如下：

```
/usr/libexec/java_home -V
```

可以查看所有的JAVA\_HOME指向哪：

```
Matching Java Virtual Machines (4):
1.8.0_40, x86_64: "Java SE 8" /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home
1.7.0_79, x86_64: "Java SE 7" /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home
1.6.0_65-b14-468, x86_64: "Java SE 6" /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
1.6.0_65-b14-468, i386: "Java SE 6" /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home

/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home
```

```
sunjie:conf sunjie$ /usr/libexec/java_home -V
Matching Java Virtual Machines (4):
 1.8.0_40, x86_64: "Java SE 8" /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home
 1.7.0_79, x86_64: "Java SE 7" /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home
 1.6.0_65-b14-468, x86_64: "Java SE 6" /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home
 1.6.0_65-b14-468, i386: "Java SE 6" /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home

/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home
```

显示文件信息，如下：

```
ls -ltr /usr/libexec/java_home
```

结果：

```
lrwxr-xr-x 1 root wheel 79 12 30 10:36 /usr/libexec/java_home -> /System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands/java_home
```

```
sunjie:conf sunjie$ ls -ltr /usr/libexec/java_home
lrwxr-xr-x 1 root wheel 79 12 30 10:36 /usr/libexec/java_home -> /System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands/java_home
http://blog.csdn.net/
```

[点击进入ooppookid的博客](#)

[

# Java千百问\_02基本使用（006）\_eclipse如何保存时格式化

,

[点击进入\\_更多\\_Java千百问](#)

## 1、eclipse如何打开保存时格式化

我们都知道，为了让代码可读性更高，eclipse提供了**代码格式化**的功能，一般指定了对应format文件（eclipse会默认自带）后，通过快捷键**Ctrl+Shift+F**来进行格式化。

然而，很多时候并不方便，而且在修改完代码后容易忘记。其实还有一种更好的方法：**保存时自动格式化**。每次保存都会自动格式化，十分方便。

方法如下：

具体方法：windows→Preferences→Java → Editor→Save Actions

开启”Perform the selected actions on save”中的**“Format source code”**

]

[

## Java千百问\_02基本使用（007）\_eclipse变量高亮如何打开

,

[点击进入\\_更多\\_Java千百问](#)

### 1、eclipse变量高亮如何打开

在开发中，我们经常会有这样的需求：想看某个变量或者方法[在哪里被使用](#)。

eclipse提供了这样的功能，在选中变量后，会在同一文本中显示所有该关键字出现的地方那个，加上[灰色背景高亮](#)。

[开启/关闭该功能：](#)

windows-> preferences->java->Editor->Mark Occurrences

选中“[Mark Occurrences of the selected element in the current file](#)”即可。

或者使用[Ctrl+Alt+O](#)快捷键。当然在工具栏也会有该操作（一般是一个黄色笔头的毛笔图标）。

[改变成其颜色：](#)

windows-> preferences->java->Editor->Mark Occurences

第一行有个“[Annotations](#)”[链接](#)，点击进入后，显示出列表，变更Occurrence annotation右边[Color](#)即可。

]

[

# Java千百问\_02基本使用（008）\_eclipse如何关闭代码验证

,

[点击进入\\_更多\\_Java千百问](#)

## 1、eclipse如何关闭代码验证

我们知道eclipse会在编译时对java代码、文本、标记文本等等进行验证，但是，这个过程在文件多的时候非常缓慢，通常会影响我们的开发效率。这里我们介绍一下如何选择性关闭这个功能。

选择性关闭：

具体方法：windows→Preferences→Validation

在下面有一个列表，可以根据自己需要选择关闭验证功能（选中状态是需要验证）。当然这里还可以自定义验证规则。其中：

- “Build”列是指文件在编译时会先校验。
- “Manual”列是指需要手工校验文件。

通常会关闭占用资源的jsp、xml等文件的验证，以保证编译时的效率。

]

[

# Java千百问\_02基本使用（009）\_eclipse如何设置BuildPath

,

[点击进入\\_更多\\_Java千百问](#)

## 1、eclipse如何设置BuildPath

我们知道运行java应用需要依赖jre以及程序中使用的第三方jar包，我们使用java命令运行时，一般会手工指定-classpath（默认为操作系统的CLASSPATH环境变量），eclipse提供了方便的引入功能，针对不同工程可以选择加载不同的classpath。

了解java如何运行看这里：[\[java程序如何运行\]\[2\]](#)  
[2]:

具体配置方法如下：

右击工程→Preferences→Java Build Path

其中有4个tab选项卡，分别如下：

“Source”

用来指定本工程下源文件的路径，选择路径下的源文件都会被自动编译。下面的Default output folder指定编译后文件的输出目录。

“Project”

用来指定本工作空间内需要引入的工程，引入后相当于把该工程打jar包引入。

“Libraries”

用来引入jre或者第三方jar包（也可以引入单独的.class文件）。这里相当于制定了该工程的classpath。

“Order and Export”

指定各路径的编译顺序。

]

[

# Java千百问\_02基本使用（010）\_java、javax、sun、org包有什么区别

,

[点击进入\\_更多\\_Java千百问](#)

## 1、java、javax、sun、org包有什么区别

java、javax、org、sun包都是jdk提供的类包，且都是在rt.jar中。rt.jar是JAVA基础类库（java核心框架中很重要的包），包含lang在内的大部分功能，而且rt.jar默认就在根classloader的加载路径里面，所以放在classpath是多此一举。他们之间的区别具体如下：

了解java核心框架看这里：[java核心框架是什么样的](#)

- [java.\\*](#)  
java SE的**标准库**，是**java标准**的一部分，是**对外承诺的java开发接口**，通常要保持**向后兼容**，一般不会轻易修改。包括其他厂家的在内，所有jdk的实现，在java.\*上都是一样的。
- [javax.\\*](#)  
也是**java标准**的一部分，但是没有包含在标准库中，一般属于**标准库的扩展**。通常属于**某个特定领域**，不是一般性的api。  
所以以扩展的方式提供api，以避免jdk的标准库过大。当然某些早期的javax，后来被并入到标准库中，所有也应该属于新版本JDK的标准库。比如jmx，java 5以前是以扩展方式提供，但是jdk5以后就做为标准库的一部分了，所有javax.management也是jdk5的标准库的一部分。
- [com.sun.\\*](#)  
是sun的hotspot虚拟机中**java.\***和**javax.\***的实现类。因为包含在rt中，所以我们可以调用。但是因为**不是sun对外公开承诺的接口**，所以根据根据实现的需要**随时增减**，因此在不同版本的hotspot中可能是不同的，而且在其他的jdk实现中是没有的，调用这些类，可能不会向后兼容，所以**一般不推荐使用**。
- [org.omg.\\*](#)  
是由**企业或者组织**提供的java类库，大部分不是sun公司提供的，同com.sun.\*，**不具备向后兼容性**，会根据需要**随时增减**。其中比较常用的是w3c提供的对XML、网页、服务器的类和接口。

]

# Java千百问\_02基本使用（011）\_如何编写单线程Socket程序

[点击进入\\_更多\\_Java千百问](#)

## 1、如何编写单线程Socket程序

了解Socket看这里：[Socket是什么](#)

编写Socket最简单的就是单线程的Socket，但基本上是没有实用意义的，因为在实际的应用中基本上是不止于一个Client的。一般都是多线程Socket程序。

了解多线程Socket看这里：[如何编写多线程Socket程序](#)

要编写Socket，需要了解java.net包中提供了两个类Socket和ServerSocket，他们分别用来表示Socket的客户端和服务端。我们的代码也分为客户端和服务端两部分。

服务端代码：

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketSimpleDemoServer {
    private int port = 8000;// 端口

    private ServerSocket serverSocket;

    public SocketSimpleDemoServer() throws Exception {
        serverSocket = new ServerSocket(port);
        System.out.println("waitting connet...");
    }

    public void service() throws IOException {
        Socket socket = null;
        String msg = null;
        while (true) { // 不停的监听，直到接收到请求
            try {
                socket = serverSocket.accept(); // 准备接受连接
                System.out.println("new connection: " + socket.getInetAddress() + ":" + socket.getPort());
                BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8")); // 输入流
                PrintWriter writer = new PrintWriter(socket.getOutputStream()); // 输出流
                while ((msg = reader.readLine()) != null) { // 接收消息
                    System.out.println("receive msg:" + msg);
                    writer.println(msg); // 发送消息
                    writer.flush(); // 注意，在使用缓冲流在发送消息的时候最好进行强制刷新，否则，可能会由于缓冲区不满而暂时不发送消息
                    if ("close".equals(msg)) {
                        break;
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                if (socket != null) {
                    socket.close();
                }
            }
        }
    }

    public static void main(String[] args) throws Exception {
        new SocketSimpleDemoServer().service();
    }
}
```

运行服务端代码后，程序会一直进行监听，直到接收到客户端请求为止。结果如下：

waitting connet...

客户端代码：

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```



```

import java.io.PrintWriter;
import java.net.Socket;

public class SocketDemoClient {

    private String host = "127.0.0.1"; // 要发送给服务端的ip

    private int port = 8000; // 要发送给服务端的端口

    private Socket socket;

    public SocketDemoClient() throws Exception {
        socket = new Socket(host, port); // 构造Socket客户端，并与连接服务端
    }

    public void talk() throws IOException {
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream(), "UTF-8"));
            PrintWriter writer = new PrintWriter(socket.getOutputStream());
            // 读取本地控制台的消息
            BufferedReader localReader = new BufferedReader(new InputStreamReader(System.in));
            String msg = null;
            while ((msg = localReader.readLine()) != null) {
                writer.println(msg);
                writer.flush();
                System.out.println("send msg:" + reader.readLine());
                if ("close".equals(msg)) {
                    break;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (socket != null) {
                socket.close();
            }
        }
    }

    public static void main(String[] args) throws Exception {
        new SocketDemoClient().talk();
    }
}

```

运行客户端代码后，我们查看**服务端的Console**，会出现以下结果，说明已经连接成功：

```

waitting connet...
new connection: /127.0.0.1:59349

```

我们在去**客户端的Console**中输入我们要发送的消息“维护世界和平”，回车确定后，**客户端Console**出现以下结果，消息已经发出：

```
send msg:维护世界和平
```

在**服务端的Console**中，我们会看到如下结果，说明消息已经被接受：

```

waitting connet...
new connection: /127.0.0.1:59349
receive msg:维护世界和平

```

这里要注意的是，在选择服务端口时，每一个端口提供一种特定的服务，**端口不能冲突**，包括系统保留的接口。通常0~1023的端口号为**系统所保留**，例如http服务的端口号为80，telnet服务的端口号为21，ftp服务的端口号为23。

]

[

# Java千百问\_02基本使用（012）\_如何编写非阻塞SocketChannel程序

[点击进入\\_更多\\_Java千百问](#)

## 1、如何编写非阻塞SocketChannel程序

了解Socket看这里：[Socket是什么](#)

了解 SocketChannel看这里：[Socket、SocketChannel有什么区别](#)

使用SocketChannel的最大好处就是可以进行**非阻塞IO**，每次链接后都会**直接返回，不会阻塞线程**。将需要多个线程的任务通过几个线程就能完成，降低了性能消耗。

了解阻塞、非阻塞看这里：[阻塞、非阻塞有什么区别](#)

要编写SocketChannel，需要了解**java.nio**包中如下几个类：

### 1. [ServerSocketChannel](#)

ServerSocket的替代类，支持阻塞通信与非阻塞通信。

#### 1. [SocketChannel](#)

Socket的替代类，支持阻塞通信与非阻塞通信。

#### 2. [Selector](#)

为ServerSocketChannel监控接收客户端连接就绪事件，为SocketChannel监控连接服务器读就绪和写就绪事件。

#### 3. [SelectionKey](#)

代表ServerSocketChannel及SocketChannel向Selector注册事件的**句柄**。当一个 SelectionKey对象位于Selector对象的selected-keys集合中时，就表示与这个SelectionKey对象相关的事件发生了。

我们的通过**客户端**和**服务端**两部分代码来介绍。

服务端代码：

```
public class SocketChannelServer {

    private int port = 8000; // 端口

    public SocketChannelServer() throws Exception {
        Selector selector = Selector.open();
        ServerSocketChannel serverChannel = ServerSocketChannel.open();
        serverChannel.configureBlocking(false); // 设置为非阻塞方式，如果为true 那么就为传统的阻塞方式
        serverChannel.socket().bind(new InetSocketAddress(port)); // 绑定IP 及 端口
        serverChannel.register(selector, SelectionKey.OP_ACCEPT); // 注册

        while (true) {
            System.out.println("Waiting accept!");
            Thread.sleep(1000);
            selector.select(); // 刚启动时，没有客户端连接时，会堵塞在这里

            Set<SelectionKey> keys = selector.selectedKeys();
            Iterator<SelectionKey> iterator = keys.iterator();
            while (iterator.hasNext()) {
                SelectionKey key = iterator.next();
                iterator.remove(); // 为了防止重复迭代
                if (key.isAcceptable()) {
                    ServerSocketChannel serverSocketChannel = (ServerSocketChannel) key.channel();
                    SocketChannel socketChannel = serverSocketChannel.accept(); // 新的连接
                    System.out.println("Client accept!" + socketChannel);
                    socketChannel.configureBlocking(false);
                    // socketChannel.register(selector,
                    // SelectionKey.OP_WRITE); // 注册write
                    socketChannel.register(selector, SelectionKey.OP_READ, ByteBuffer.allocate(1024)); // 注册read
                } else if (key.isWritable()) { // 写入
                    SocketChannel socketChannel = (SocketChannel) key.channel(); // 获得与客户端通信的信道
```



们需要多次运行客户端代码。这里我们运行两次之后（称为客户端1、客户端2），查看服务端的Console，未超过5秒时会出现以下结果，说明已经连接成功。这里需要注意，最后一句再次打印了Waiting accept!，说明并没有一直等待客户端发送请求，而是继续监听请求，即没有被阻塞：

```
Waiting accept!  
Client accept!java.nio.channels.SocketChannel[connected local=/127.0.0.1:8000 remote=/127.0.0.1:59481]  
Waiting accept!
```

这时，每个客户端Console如下，成功接收到服务端的消息，并发出给服务端的消息，之后便立刻释放了线程，并没有一直等待服务端的执行：

```
client receive msg==  
client send msg==I am a coder.
```

再回到服务端Console，5秒后，会打印我们收到的请求：

```
Waiting accept!  
Client accept!java.nio.channels.SocketChannel[connected local=/127.0.0.1:8000 remote=/127.0.0.1:59481]  
Waiting accept!  
server receive msg==I am a coder.  
Waiting accept!  
Client accept!java.nio.channels.SocketChannel[connected local=/127.0.0.1:8000 remote=/127.0.0.1:59485]  
Waiting accept!  
server receive msg==I am a coder.  
Waiting accept!
```

这里我们可以看到处理了2个客户端的请求。在服务端代码中，我们可以注册write，达到向客户端写数据的需求。

]

[

# Java千百问\_02基本使用（013）\_linux系统如何管理环境变量

,

[点击进入\\_更多\\_Java千百问](#)

## 1、linux系统如何管理环境变量

我们都知道windows的环境变量的配置已经完全被图形化了，我们可以在我的电脑的属性中简单的进行配置，那么对于linux这样没有图形化配置的操作系统应该如何配置呢？我们具体来看：

linux中环境变量包括系统级和用户级，系统级的环境变量是**每个登录到系统的用户**都要读取的系统变量，而用户级的环境变量则是该**用户使用系统时**加载的环境变量。具体配置方法如下：

### 系统级

通过修改/etc/profile、/etc/environment文件配置

**/etc/profile**，该文件是用户登录时，操作系统定制用户环境时使用的第一个文件，应用于登录到系统的每一个用户。该文件一般是调用/etc/bash.bashrc文件（系统级的bashrc文件）。

**/etc/environment**，该文件是在登录时操作系统使用的第二个文件，系统在读取你自己的profile前，设置环境文件的环境变量。

### 用户级

**~/.profile**，每个用户都可使用该文件输入专用于自己使用的shell信息。

**~/.bashrc**，该文件包含专用于自己的bash shell的bash信息，当登录时以及每次打开新的shell时，该文件被读取。

**~/.pam\_environment**，用户级的环境变量设置文件。

例子：

我们想在系统环境变量中，添加java的配置，具体如下：

1. 使用**gedit**编辑/etc/profile文件（其他编辑器也可以），终端中输入：

```
sudo gedit /etc/profile
```

2. 添加我们要加的环境变量：

```
export JAVA_HOME="/Library/Java"
export PATH="$PATH:$JAVA_HOME/bin"
export JRE_HOME="$JAVA_HOME/jre"
export CLASSPATH=".:$JAVA_HOME/lib:$JRE_HOME/lib"
```

3. 使用命令让配置生效：

```
source /etc/profile
```

]

[

# Java千百问\_02基本使用（014）\_mac系统如何管理环境变量

[点击进入\\_更多\\_Java千百问-基本使用](#)

## 1、mac系统如何管理环境变量

mac系统中管理环境变量包括系统级和用户级，系统级的环境变量是**每个登录到系统的用户**都要读取的系统变量，而用户级的环境变量则是该**用户使用系统时**加载的环境变量。具体配置方法如下：

### 系统级

通过修改`./etc/profile`文件来配置，全局（公有）配置，不管是哪个用户，登录时都会读取该文件。不建议修改这个文件。

### 用户级

`./etc/bashrc`，全局（公有）配置，bash shell执行时，不管是何种方式，都会读取此文件。

`.bash_profile`，每个用户专用于自己使用的shell信息，当用户登录时，该文件仅执行一次。

`~/pam_environment`，用户级的环境变量设置文件。

例子：

我们想在系统环境变量中，添加java的配置，具体如下：

1. 使用vi编辑`.bash_profile`文件（其他编辑器也可以），终端中输入：

```
sudo vi .bash_profile
```

2. 添加我们要加的环境变量：

```
export JAVA_HOME="/Library/Java"
export PATH="$PATH:$JAVA_HOME/bin"
export JRE_HOME="$JAVA_HOME/jre"
export CLASSPATH=".:$JAVA_HOME/lib:$JRE_HOME/lib"
```

3. 使用命令让配置生效：

```
source .bash_profile
```

]

[

# Java千百问\_02基本使用（015）\_java如何通过汇编方式运行

[点击进入\\_更多\\_Java千百问-基本使用](#)

## 1、java如何通过汇编方式运行

java本身**不能通过汇编方式运行**。但是，我们可以通过某些插件，在运行中将java代码**解释为**汇编指令，让我们能够通过分析执行的汇编指令来查找一些问题，也可以帮助我们分析和理解JVM是**如何解释和编译的**（当然java本身的编译和运行和汇编无关）。

**PrintAssembly**是JVM的一个运行参数，它允许我们获取在控制台打印java代码翻译成的**汇编指令**。使用PrintAssembly需要一些插件的支持，这些并不是JVM直接提供的，**Kenai项目**则提供了可用的插件（下载 <https://kenai.com/projects/base-hsdis/downloads>）。根据不同的环境下在对应的指令集。本人是mac系统，所以下载了**hsdis-amd64.dylib**。

下载后需要将hsdis-amd64.dylib放在\$JAVA\_PATH/jre/lib/server/中，与libjvm.dylib同目录。

之后我们就可以通过指定运行参数来运行我们的代码：

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly TestHsdis
```

例子：

```
public class TestHsdis {  
  
    public static void main(String[] args) {  
        System.out.println("1");  
    }  
}
```

编译运行时加入**-XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly**，结果如下：

```
Java HotSpot(TM) 64-Bit Server VM warning: PrintAssembly is enabled; turning on DebugNonSafepoints to  
gain additional output  
Loaded disassembler from  
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/jre/lib/server/hsdis-amd64.dylib  
Decoding compiled method 0x000000010b4fd710:  
Code:  
[Disassembling for mach='i386:x86-64']  
[Entry Point]  
[Constants]  
# {method} {0x000000011f270fc8} 'hashCode' '()'I' in 'java/lang/String'  
# [sp+0x40] (sp of caller)  
0x000000010b4fd880: mov 0x8(%rsi),%r10d  
0x000000010b4fd884: shl $0x3,%r10  
0x000000010b4fd888: cmp %rax,%r10  
0x000000010b4fd88b: jne 0x000000010b445e20 ; {runtime_call}  
0x000000010b4fd891: data32 data32 nopw 0x0(%rax,%rax,1)  
0x000000010b4fd89c: data32 data32 xchg %ax,%ax  
....
```

## 2、PrintAssembly is disabled是什么原因

我们在加入参数`-XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly`运行时，可能会报如下错误：

```
Java HotSpot(TM) 64-Bit Server VM warning: PrintAssembly is enabled; turning on DebugNonSafepoints to  
gain additional output  
Could not load hsdis-amd64.dylib; library not loadable; PrintAssembly is disabled
```

原因是对应的hsdis-amd64插件没有放入指定路径，导致运行时无法加载。我们一定要将hsdis-amd64.dylib放在\$JAVA\_PATH/jre/lib/server/中，与libjvm.dylib同目录（mac系统，如果是linux放在与libjvm.so同目录）。

]