**Java 抽象类**

在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来，并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。

抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。

由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。也是因为这个原因，通常在设计阶段决定要不要设计抽象类。

父类包含了子类集合的常见的方法，但是由于父类本身是抽象的，所以不能使用这些方法。

在Java中抽象类表示的是一种继承关系，一个类只能继承一个抽象类，而一个类却可以实现多个接口。

**抽象类**

在Java语言中使用abstract class来定义抽象类。如下实例：

Employee.java 文件代码：

/\* 文件名 : Employee.java \*/

public abstract class Employee {

private String name;

private String address;

private int number;

public Employee(String name, String address, int number) {

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public double computePay() {

System.out.println("Inside Employee computePay");

return 0.0;

}

public void mailCheck() {

System.out.println("Mailing a check to " + this.name + " " + this.address);

}

public String toString() {

return name + " " + address + " " + number;

}

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public void setAddress(String newAddress) {

address = newAddress;

}

public int getNumber() {

return number;

}

}

注意到该Employee类没有什么不同，尽管该类是抽象类，但是它仍然有3个成员变量，7个成员方法和1个构造方法。 现在如果你尝试如下的例子：

AbstractDemo.java 文件代码：

/\* 文件名 : AbstractDemo.java \*/

public class AbstractDemo {

public static void main(String [] args) {

/\* 以下是不允许的，会引发错误 \*/

Employee e = new Employee("George W.", "Houston, TX", 43);

System.out.println("\n Call mailCheck using Employee reference--"); e.mailCheck();

}

}

当你尝试编译AbstractDemo类时，会产生如下错误：

Employee.java:46: Employee is abstract; cannot be instantiated

Employee e = new Employee("George W.", "Houston, TX", 43);

^

1 error

**继承抽象类**

我们能通过一般的方法继承Employee类：

Salary.java 文件代码：

/\* 文件名 : Salary.java \*/

public class Salary extends Employee {

private double salary; //Annual salary

public Salary(String name, String address, int number, double salary) {

super(name, address, number); setSalary(salary);

}

public void mailCheck() {

System.out.println("Within mailCheck of Salary class ");

System.out.println("Mailing check to " + getName() + " with salary " + salary);

}

public double getSalary() {

return salary;

}

public void setSalary(double newSalary) {

if(newSalary >= 0.0) {

salary = newSalary;

}

}

public double computePay() {

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

}

尽管我们不能实例化一个Employee类的对象，但是如果我们实例化一个Salary类对象，该对象将从 Employee 类继承7个成员方法，且通过该方法可以设置或获取三个成员变量。

AbstractDemo.java 文件代码：

/\* 文件名 : AbstractDemo.java \*/

public class AbstractDemo {

public static void main(String [] args) {

Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00); Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Call mailCheck using Salary reference --");

s.mailCheck();

System.out.println("\n Call mailCheck using Employee reference--"); e.mailCheck();

}

}

以上程序编译运行结果如下：

Constructing an Employee

Constructing an Employee

Call mailCheck using Salary reference --

Within mailCheck of Salary class

Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--

Within mailCheck of Salary class

Mailing check to John Adams with salary 2400.

**抽象方法**

如果你想设计这样一个类，该类包含一个特别的成员方法，该方法的具体实现由它的子类确定，那么你可以在父类中声明该方法为抽象方法。

Abstract关键字同样可以用来声明抽象方法，抽象方法只包含一个方法名，而没有方法体。

抽象方法没有定义，方法名后面直接跟一个分号，而不是花括号。

public abstract class Employee { private String name; private String address; private int number; public abstract double computePay(); //其余代码 }

声明抽象方法会造成以下两个结果：

* 如果一个类包含抽象方法，那么该类必须是抽象类。
* 任何子类必须重写父类的抽象方法，或者声明自身为抽象类。

继承抽象方法的子类必须重写该方法。否则，该子类也必须声明为抽象类。最终，必须有子类实现该抽象方法，否则，从最初的父类到最终的子类都不能用来实例化对象。

如果Salary类继承了Employee类，那么它必须实现computePay()方法：

Salary.java 文件代码：

/\* 文件名 : Salary.java \*/

public class Salary extends Employee {

private double salary; // Annual salary

public double computePay() {

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

//其余代码

}

**抽象类总结规定**

* 1. 抽象类不能被实例化(初学者很容易犯的错)，如果被实例化，就会报错，编译无法通过。只有抽象类的非抽象子类可以创建对象。
* 2. 抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。
* 3. 抽象类中的抽象方法只是声明，不包含方法体，就是不给出方法的具体实现也就是方法的具体功能。
* 4. 构造方法，类方法（用static修饰的方法）不能声明为抽象方法。
* 5. 抽象类的子类必须给出抽象类中的抽象方法的具体实现，除非该子类也是抽象类。

# Java 接口

接口（英文：Interface），在JAVA编程语言中是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。

接口并不是类，编写接口的方式和类很相似，但是它们属于不同的概念。类描述对象的属性和方法。接口则包含类要实现的方法。

除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。

接口无法被实例化，但是可以被实现。一个实现接口的类，必须实现接口内所描述的所有方法，否则就必须声明为抽象类。另外，在 Java 中，接口类型可用来声明一个变量，他们可以成为一个空指针，或是被绑定在一个以此接口实现的对象。

### 接口与类相似点：

* 一个接口可以有多个方法。
* 接口文件保存在 .java 结尾的文件中，文件名使用接口名。
* 接口的字节码文件保存在 .class 结尾的文件中。
* 接口相应的字节码文件必须在与包名称相匹配的目录结构中。

### 接口与类的区别：

* 接口不能用于实例化对象。
* 接口没有构造方法。
* 接口中所有的方法必须是抽象方法。
* 接口不能包含成员变量，除了 static 和 final 变量。
* 接口不是被类继承了，而是要被类实现。
* 接口支持多继承。

### 接口特性

* 接口中每一个方法也是隐式抽象的,接口中的方法会被隐式的指定为 **public abstract**（只能是 public abstract，其他修饰符都会报错）。
* 接口中可以含有变量，但是接口中的变量会被隐式的指定为 **public static final** 变量（并且只能是 public，用 private 修饰会报编译错误）。
* 接口中的方法是不能在接口中实现的，只能由实现接口的类来实现接口中的方法。

### 抽象类和接口的区别

* 1. 抽象类中的方法可以有方法体，就是能实现方法的具体功能，但是接口中的方法不行。
* 2. 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 **public static final** 类型的。
* 3. 接口中不能含有静态代码块以及静态方法(用 static 修饰的方法)，而抽象类是可以有静态代码块和静态方法。
* 4. 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

## 接口的声明

接口的声明语法格式如下：

[可见度] interface 接口名称 [extends 其他的类名] { // 声明变量 // 抽象方法 }

Interface关键字用来声明一个接口。下面是接口声明的一个简单例子。

NameOfInterface.java 文件代码：

/\* 文件名 : NameOfInterface.java \*/

import java.lang.\*; //引入包

public interface NameOfInterface {

//任何类型 final, static 字段 //抽象方法

}

接口有以下特性：

* 接口是隐式抽象的，当声明一个接口的时候，不必使用**abstract**关键字。
* 接口中每一个方法也是隐式抽象的，声明时同样不需要**abstract**关键字。
* 接口中的方法都是公有的。

### 实例

Animal.java 文件代码：

/\* 文件名 : Animal.java \*/

interface Animal {

public void eat();

public void travel();

}

## 接口的实现

当类实现接口的时候，类要实现接口中所有的方法。否则，类必须声明为抽象的类。

类使用implements关键字实现接口。在类声明中，Implements关键字放在class声明后面。

实现一个接口的语法，可以使用这个公式：

Animal.java 文件代码：

...implements 接口名称[, 其他接口名称, 其他接口名称..., ...] ...

### 实例

MammalInt.java 文件代码：

/\* 文件名 : MammalInt.java \*/

public class MammalInt implements Animal{

public void eat(){

System.out.println("Mammal eats");

}

public void travel(){

System.out.println("Mammal travels");

}

public int noOfLegs(){

return 0;

}

public static void main(String args[]){

MammalInt m = new MammalInt();

m.eat();

m.travel();

}

}

以上实例编译运行结果如下:

Mammal eats

Mammal travels

重写接口中声明的方法时，需要注意以下规则：

* 类在实现接口的方法时，不能抛出强制性异常，只能在接口中，或者继承接口的抽象类中抛出该强制性异常。
* 类在重写方法时要保持一致的方法名，并且应该保持相同或者相兼容的返回值类型。
* 如果实现接口的类是抽象类，那么就没必要实现该接口的方法。

在实现接口的时候，也要注意一些规则：

* 一个类可以同时实现多个接口。
* 一个类只能继承一个类，但是能实现多个接口。
* 一个接口能继承另一个接口，这和类之间的继承比较相似。

## 接口的继承

一个接口能继承另一个接口，和类之间的继承方式比较相似。接口的继承使用extends关键字，子接口继承父接口的方法。

下面的Sports接口被Hockey和Football接口继承：

// 文件名: Sports.java

public interface Sports {

public void setHomeTeam(String name);

public void setVisitingTeam(String name);

}

// 文件名: Football.java

public interface Football extends Sports {

public void homeTeamScored(int points);

public void visitingTeamScored(int points);

public void endOfQuarter(int quarter); }

// 文件名: Hockey.java

public interface Hockey extends Sports {

public void homeGoalScored();

public void visitingGoalScored();

public void endOfPeriod(int period);

public void overtimePeriod(int ot);

}

Hockey接口自己声明了四个方法，从Sports接口继承了两个方法，这样，实现Hockey接口的类需要实现六个方法。

相似的，实现Football接口的类需要实现五个方法，其中两个来自于Sports接口。

## 接口的多继承

在Java中，类的多继承是不合法，但接口允许多继承，。

在接口的多继承中extends关键字只需要使用一次，在其后跟着继承接口。 如下所示：

public interface Hockey extends Sports, Event

以上的程序片段是合法定义的子接口，与类不同的是，接口允许多继承，而 Sports及 Event 可能定义或是继承相同的方法

## 标记接口

最常用的继承接口是没有包含任何方法的接口。

标记接口是没有任何方法和属性的接口.它仅仅表明它的类属于一个特定的类型,供其他代码来测试允许做一些事情。

标记接口作用：简单形象的说就是给某个对象打个标（盖个戳），使对象拥有某个或某些特权。

例如：java.awt.event 包中的 MouseListener 接口继承的 java.util.EventListener 接口定义如下：

package java.util; public interface EventListener {}

没有任何方法的接口被称为标记接口。标记接口主要用于以下两种目的：

* **建立一个公共的父接口：**

正如EventListener接口，这是由几十个其他接口扩展的Java API，你可以使用一个标记接口来建立一组接口的父接口。例如：当一个接口继承了EventListener接口，Java虚拟机(JVM)就知道该接口将要被用于一个事件的代理方案。

* **向一个类添加数据类型：**

这种情况是标记接口最初的目的，实现标记接口的类不需要定义任何接口方法(因为标记接口根本就没有方法)，但是该类通过多态性变成一个接口类型。

# Java 包(package)

为了更好地组织类，Java 提供了包机制，用于区别类名的命名空间。

### 包的作用

* 1、把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。
* 2、如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。
* 3、包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。

Java 使用包（package）这种机制是为了防止命名冲突，访问控制，提供搜索和定位类（class）、接口、枚举（enumerations）和注释（annotation）等。

包语句的语法格式为：

package pkg1[．pkg2[．pkg3…]];

例如,一个Something.java 文件它的内容

package net.java.util; public class Something{ ... }

那么它的路径应该是 **net/java/util/Something.java** 这样保存的。 package(包) 的作用是把不同的 java 程序分类保存，更方便的被其他 java 程序调用。

一个包（package）可以定义为一组相互联系的类型（类、接口、枚举和注释），为这些类型提供访问保护和命名空间管理的功能。

以下是一些 Java 中的包：

* **java.lang**-打包基础的类
* **java.io**-包含输入输出功能的函数

开发者可以自己把一组类和接口等打包，并定义自己的包。而且在实际开发中这样做是值得提倡的，当你自己完成类的实现之后，将相关的类分组，可以让其他的编程者更容易地确定哪些类、接口、枚举和注释等是相关的。

由于包创建了新的命名空间（namespace），所以不会跟其他包中的任何名字产生命名冲突。使用包这种机制，更容易实现访问控制，并且让定位相关类更加简单。

## 创建包

创建包的时候，你需要为这个包取一个合适的名字。之后，如果其他的一个源文件包含了这个包提供的类、接口、枚举或者注释类型的时候，都必须将这个包的声明放在这个源文件的开头。

包声明应该在源文件的第一行，每个源文件只能有一个包声明，这个文件中的每个类型都应用于它。

如果一个源文件中没有使用包声明，那么其中的类，函数，枚举，注释等将被放在一个无名的包（unnamed package）中。

### 例子

让我们来看一个例子，这个例子创建了一个叫做animals的包。通常使用小写的字母来命名避免与类、接口名字的冲突。

在 animals 包中加入一个接口（interface）：

Animal.java 文件代码：

/\* 文件名: Animal.java \*/

package animals;

interface Animal {

public void eat();

public void travel();

}

接下来，在同一个包中加入该接口的实现：

MammalInt.java 文件代码：

package animals;

/\* 文件名 : MammalInt.java \*/

public class MammalInt implements Animal{

public void eat(){

System.out.println("Mammal eats");

}

public void travel(){

System.out.println("Mammal travels");

}

public int noOfLegs(){

return 0;

}

public static void main(String args[]){

MammalInt m = new MammalInt();

m.eat();

m.travel();

}

}

然后，编译这两个文件，并把他们放在一个叫做animals的子目录中。 用下面的命令来运行：

$ mkdir animals

$ cp Animal.class MammalInt.class animals

$ java animals/MammalInt

Mammal eats

Mammal travel

## import 关键字

为了能够使用某一个包的成员，我们需要在 Java 程序中明确导入该包。使用 "import" 语句可完成此功能。

在 java 源文件中 import 语句应位于 package 语句之后，所有类的定义之前，可以没有，也可以有多条，其语法格式为：

import package1[.package2…].(classname|\*);

如果在一个包中，一个类想要使用本包中的另一个类，那么该包名可以省略。

### 例子

下面的 payroll 包已经包含了 Employee 类，接下来向 payroll 包中添加一个 Boss 类。Boss 类引用 Employee 类的时候可以不用使用 payroll 前缀，Boss类的实例如下。

Boss.java 文件代码：

package payroll;

public class Boss {

public void payEmployee(Employee e) {

e.mailCheck();

}

}

如果 Boss 类不在 payroll 包中又会怎样？Boss 类必须使用下面几种方法之一来引用其他包中的类。

使用类全名描述，例如：

payroll.Employee

用 **import** 关键字引入，使用通配符 "\*"

import payroll.\*;

使用 **import** 关键字引入 Employee 类:

import payroll.Employee;

**注意：**

类文件中可以包含任意数量的 import 声明。import 声明必须在包声明之后，类声明之前。

## package 的目录结构

类放在包中会有两种主要的结果：

* 包名成为类名的一部分，正如我们前面讨论的一样。
* 包名必须与相应的字节码所在的目录结构相吻合。

下面是管理你自己 java 中文件的一种简单方式：

将类、接口等类型的源码放在一个文本中，这个文件的名字就是这个类型的名字，并以.java作为扩展名。例如：

// 文件名 : Car.java package vehicle; public class Car { // 类实现 }

接下来，把源文件放在一个目录中，这个目录要对应类所在包的名字。

....\vehicle\Car.java

现在，正确的类名和路径将会是如下样子：

* 类名 -> vehicle.Car
* 路径名 -> vehicle\Car.java (在 windows 系统中)

通常，一个公司使用它互联网域名的颠倒形式来作为它的包名.例如：互联网域名是 runoob.com，所有的包名都以 com.runoob 开头。包名中的每一个部分对应一个子目录。

例如：有一个 **com.runoob.test** 的包，这个包包含一个叫做 Runoob.java 的源文件，那么相应的，应该有如下面的一连串子目录：

....\com\runoob\test\Runoob.java

编译的时候，编译器为包中定义的每个类、接口等类型各创建一个不同的输出文件，输出文件的名字就是这个类型的名字，并加上 .class 作为扩展后缀。 例如：

// 文件名: Runoob.java

package com.runoob.test;

public class Runoob { }

class Google { }

现在，我们用-d选项来编译这个文件，如下：

$javac -d . Runoob.java

这样会像下面这样放置编译了的文件：

.\com\runoob\test\Runoob.class

.\com\runoob\test\Google.class

你可以像下面这样来导入所有**\com\runoob\test\**中定义的类、接口等：

import com.runoob.test.\*;

编译之后的 .class 文件应该和 .java 源文件一样，它们放置的目录应该跟包的名字对应起来。但是，并不要求 .class 文件的路径跟相应的 .java 的路径一样。你可以分开来安排源码和类的目录。

<path-one>\sources\com\runoob\test\Runoob.java

<path-two>\classes\com\runoob\test\Google.class

这样，你可以将你的类目录分享给其他的编程人员，而不用透露自己的源码。用这种方法管理源码和类文件可以让编译器和java 虚拟机（JVM）可以找到你程序中使用的所有类型。

类目录的绝对路径叫做 **class path**。设置在系统变量 **CLASSPATH** 中。编译器和 java 虚拟机通过将 package 名字加到 class path 后来构造 .class 文件的路径。

<path- two>\classes 是 class path，package 名字是 com.runoob.test,而编译器和 JVM 会在 <path-two>\classes\com\runoob\test 中找 .class 文件。

一个 class path 可能会包含好几个路径，多路径应该用分隔符分开。默认情况下，编译器和 JVM 查找当前目录。JAR 文件按包含 Java 平台相关的类，所以他们的目录默认放在了 class path 中。

## 设置 CLASSPATH 系统变量

用下面的命令显示当前的CLASSPATH变量：

* Windows 平台（DOS 命令行下）：C:\> set CLASSPATH
* UNIX 平台（Bourne shell 下）：# echo $CLASSPATH

删除当前CLASSPATH变量内容：

* Windows 平台（DOS 命令行下）：C:\> set CLASSPATH=
* UNIX 平台（Bourne shell 下）：# unset CLASSPATH; export CLASSPATH

设置CLASSPATH变量:

* Windows 平台（DOS 命令行下）： C:\> set CLASSPATH=C:\users\jack\java\classes
* UNIX 平台（Bourne shell 下）：# CLASSPATH=/home/jack/java/classes; export CLASSPATH