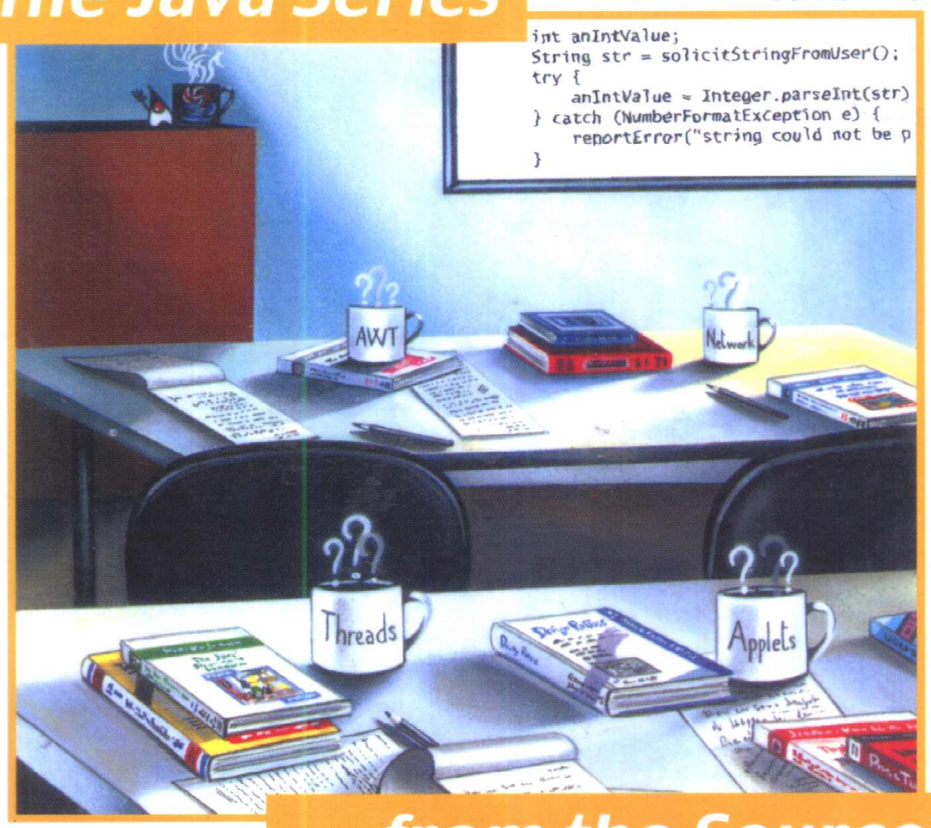
The Java FAQ

Java经典问答

Jonni Kanerva 著
陈霞 译

The Java Series



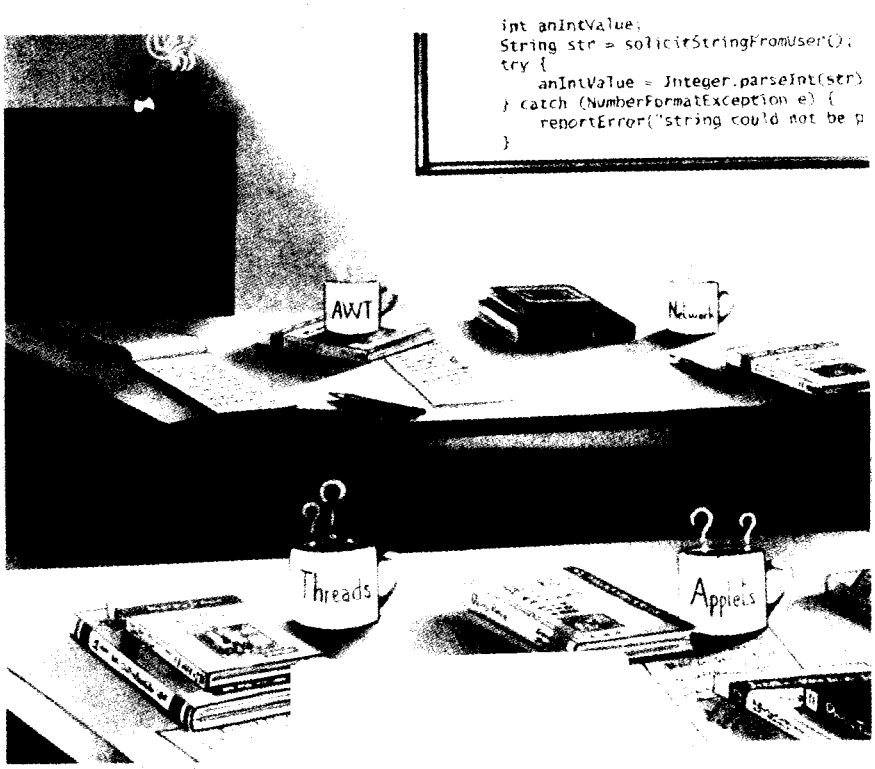
...from the Source™

Addison-Wesley

The Java FAQ

Java经典问答

Jonni Kanerva 著
陈霞 译



中国电力出版社

内 容 提 要

Java 系列丛书是关于 Java 的最完整、专业和指定的官方教材，它们直接来自 Sun Microsystems 公司的 Java 技术创始人。本书由 Sun JavaSoft 工作组的成员编写，提供了大量翔实的信息，它们都是您在学习 Java 过程中需要理解，以便可以用它迅速开发出灵活、健壮、方便、安全的 Java 程序和 Internet Applet 所必须的。Java 系列丛书是 Java 学习者不可缺少的参考资料。

本书适合软件设计开发人员及大专院校师生阅读。

图书在版编目（CIP）数据

Java 经典问答/（美）强尼茨诺编著；陈霞译.-北京：中国电力出版社，2001.12

ISBN 7-5083-0841-7

Ⅰ.J… Ⅱ.①强…②陈… Ⅲ.Java 语言-程序设计-问答 Ⅳ.TP312.44

中国版本图书馆 CIP 数据核字（2001）第 077983 号

北京版权局著作权登记号 **图字 01-2001-2224**

中国电力出版社出版、发行

（北京三里河路 6 号 100044 http://www.infopower.com.cn）

实验小学印刷厂印刷

各地新华书店经售

2002 年 1 月第一版 2002 年 1 月北京第一次印刷

787 毫米×1092 毫米 16 开本 17 印张 384 千字

定价 **28.00** 元

目　　录

第 1 章

类、接口和包

对象、类和方法（Q1.1~Q1.10）
子类、重载和覆盖（Q1.20~Q1.28）
接口和抽象类（Q1.29~Q1.30）
包和访问修饰符（Q1.29~Q1.34）

📖 对象、类和方法

Q1.1

什么是对象？

一个对象就是一个程序单元，它将一组结构化数据和对这些数据进行的各种操作结合在一起。

面向对象程序设计将我们熟悉的非面向对象程序设计语言（如 C 语言）中的函数与数据之间的关系倒了过来。在 C、Pascal 或 Basic 中，程序常常是建立在对数据的直接处理之

上：首先要确定数据结构，然后给出可以从程序的任何地方对这些数据进行处理（如查询或改变其值）的函数。因此与数据结构有关的内容会出现在程序的各个角落，如果要改变程序某一部分中的数据结构，势必会因此而影响到程序的其他部分。

面向对象程序设计方法恰恰为降低这种程序代码之间的相互依赖提供了标准的工具和技术解决方案：一个对象由一组结构化数据的定义开始，并且还包含了对这些数据进行的所有可能的存取操作。在对象中，与数据结构有关的所有操作都和这个结构紧密地结合在一起，而不是被分散在程序的各个角落。

类（Q1.2）实际上是具有相同性质的一类对象的集合，它同时定义了对象拥有的数据和操作这些数据的一系列方法（Q1.3）。我们在程序设计中接触的每一个对象都隶属某个类并被称为那个类的实例（Q1.7）。

面向对象的程序设计将操作和它们的数据紧密地结合在一起，从而使我们可以通过向对象请求服务而不是直接操作它的数据来完成对数据的存取。这种小小的不同可以使申请服务方从服务提供方那得到极大的好处：请求方（程序的某一部分）只需知道请求什么样的服务就可以了，而其他的事情则全部由执行方（对象）负责完成。现在，只有部分程序是相互依赖的，不是通过数据结构，而是通过该对象声明所能完成的功能。

Java™ 是面向对象的程序设计语言，这就意味着类、实例、方法构成了整个程序设计的核心内容。

请参看：Q1.2、Q1.3、Q1.7

Q1.2

什么是类？

类是对象的设计蓝图：它根据对象持有的数据和对数据进行的操作的特点而定义类对象的设计模版。

类是面向对象程序设计中最基本的单元，它定义了类对象的设计模版：不仅为对象确定了一个数据结构，而且还为存取这些数据定义了一系列操作。类通常也是制造对象的工厂，当程序运行时，可以通过类来创建一个或多个对象——我们把它称为类的“实例”（Q1.7）。类的每个实例都拥有它自己的数据，因此，我们可以修改一个实例的某些内容而保持其他实例不变。类只是详细定义了对象所具有的各种功能，而具体的实现则由对象本身去完成。

例如 Java 中的 String 类就是为所有字符串定义的一个对象类型。一个 String 实例由一组 Unicode 字符序列和可以从该字符序列中提取有关信息的方法（Q1.3）组成。尽管只有一个 String 类，但在一个 Java 程序中却可以拥有许多 String 类的实例。

Java 中类的声明看起来更像 C 语言中的结构定义，它至少要包括一个关键字“Class”、一个类名、一对花括号的开始“{”和结束“}”。此外，类声明中还包括许多可选部分如：访问修饰符（Q1.32）、父类定义（Q1.12）、接口定义（Q1.20）以及类体中的其他内容等等。

下面的例子定义了一个只有一个数据（又称域）和一个方法的类：

```
class SimpleClass{
    int count;  //field
    void incrementCount(){
        ++count;
    }
}
```

为学好面向对象程序设计方法，我们必须要充分理解类与实例这两个概念上的区别和联系，表 1.1 通过生活中的一些常识为我们类比了这两个概念的不同之处。

表 1.1　类和实例

类	实　例
小甜点的烹饪法和刀具	小甜点
建房方案	建好的房子
橡皮图章	印出的图案
底片	洗好的照片

注意

从现在开始，本书将按照 JLS《Java 语言规范》（第 38 页）中所使用的“对象”这一术语来指代我们将要讨论的所有类的实例和数组（Q2.19）。虽然这种叫法可能需要一点时间来适应，但是它在意思表达上却更加清楚和一致。

请参看：Q1.1、Q1.3、Q1.7、Q1.20、Q1.32、Q2.19

Q1.3

什么是方法?

在面向对象程序设计语言中，方法是体现对象功能的最基本的可执行程序设计单元，它隶属于某一个类，并且可以被特定的对象和类本身调用。

方法相当于非面向对象语言中的函数或子程序，它和函数一样有以下几个组成部分：

- 名字
- 一个可以为空的输入参数及其类型名表
- 一个可选的返回类型（关键字 Void 表示方法没有返回类型）
- 可执行代码体

下面的程序片段声明了一个方法，方法的输出结果是两个输入值（Double 型）的平均值（Double 型）：

```
double average(double a, double b) {
    return(a+b)/2.0;
}
```

因为方法是隶属于某些类（Q1.2）的，所以只可以在该类的定义中定义您的方法。值得注意的是，本书为方便起见除特殊情况，源程序代码片段都没有包括类定义部分。

通常可以通过方法来检查或处理对象中所存储的数据，例如，Java 的 String 类，它为确定一个字符串实例的长度提供了一个叫 length()的方法，它没有参数，返回值是代表调用它的字符串实例长度的 int 类型的数，请看下面的例子：

```
String aString="Bicycle Shop Quarterly News";
int howLong=aString.length();
```

方法不是简单地被激活，而是通过一个确定的对象（或类 Q1.6）来调用。让我们再来思考一下下面的程序片段，它创建了两个 AWT——抽象窗口工具集（Abstract Window Toolkit）——按钮实例并且为每个按钮设置了要显示的文本内容：

```
/* in Java: */
Button button1=new Button();
Button button2=new Button();
```

```
button1.setText("Push Me First);
button2.setText("Push Me Second");
```

button1 调用 setText 方法就等于让 Button 类去查找关于 setText 的定义，然后使用 button1 中的有关数据去执行方法体中的代码。在 button2 上调用 setText 方法导致同样的结果，只是同样的程序片段是作用于 button2 而不是 button1。

在非面向对象语言如 C 中，您可能会使用一个类似的 setButtonText 函数并把按钮作为其中的一个参数：

```
/* in C: */
Struct Button *button1=makeButton();
Struct Button *button2=makeButton();
SetButtonText(button1,"Push Me First");
SetButtonText(button2,"Push Me Second");
```

（关于 C 中的结构和 Java 中类之间关系的讨论可以参看问题 Q2.10。）

因为每个类只负责自己的方法，所以不同类可以使用具有相同名字、相同参数和返回类型的方法。例如，AWT 中 Label 类也定义了一个叫 setText 的方法，参数是一个 String 类型字符串：

```
/ * using Label's setText method rather than Button's: */
Label aLabel=new Label();
alabel.setText("Read me—I'm a label.");
```

在这个例子中，Label 类提供了它自己对 setText 的定义，这个定义完全不同于 Button 类给出的定义。

从调用者的角度看，虽然方法不同，但它们的作用看起来却非常相似，这也是面向对象程序设计的一个关键部分：它把对动作的请求和请求是如何完成的细节完全分离开。只要对象的方法名字、参数表和返回类型均相同（Q1.14），就可以直接调用该方法而无需知道对象到底属于哪个类。而到底是哪段程序被执行则由目标对象的类来决定，而不是调用者。

请参看：Q1.2、Q1.4、Q1.6、Q1.14、Q2.10

Q1.4

什么是方法的声明?

一个方法的声明是由方法名和方法输入参数（包括个数以及它们的类型）组成。

一个方法的声明足以表明该方法在所属类中的惟一性。换句话说，当一个类要从方法调用映射到相应的可执行方法体时，方法声明成了一个独一无二的查找键。

下面是 AWT 中 Rectangle 类的方法声明：

```
a. isEmpty()
b. intersection(Rectangle)
c. setLocation(int,int)
d. contains(int,int)
e. contains(Point)
```

（请注意，在这里我们引用的方法声明并不是合法的 Java 源代码，它们只代表方法声明本身。）a、b、c 三个声明在名称、参数个数和类型上都各不相同；c 和 d 只在名字上不同；d 和 e 虽然在名字上相同，但在参数个数和类型上却不相同。

在这里大家有必要记住，一个方法的输出——返回值或声明的例外——对方法的声明并没有多大的影响，尽管编译器会利用这些信息来检查方法是否被覆盖(Overriding)（Q1.14、Q2.24）

请参看：Q1.13、Q1.14、Q2.24

Q1.5

实例变量和类变量有什么不同?

一个实例变量代表的是类定义中的一个独立的数据项，类的每个实例中都拥有一个属于自己的拷贝；而一个类变量则代表的是可以被这个类和它所有的实例共享的一个数据项。

一个实例变量代表的数据表示每个实例都有它自己的一份拷贝。实例变量相当于 C 或

C++中结构类型的某个域。比如，如果程序有 20 个 Button 按钮实例，每个实例都有它自己的 label 标签，实例变量为按钮保存标签上的字符串信息。

相反，类变量则属于类本身，该类的所有实例访问的都是该变量的同一个拷贝。因此，一个类变量可以充当一个共享资源和可以提供实例之间通信的一种方式。类变量通常用关键字 static 声明。

下面的程序段声明了一个有两个实例变量和一个类变量的类：

```
class Example {
   int i1;   // no static keyword,therefore instance variable
   String s;
   Static int i2;  // static keyword,therefore class variable
}
```

请参看：Q1.6

Q1.6

实例方法和类方法有什么不同？

实例方法有“this”引用，而类方法没有

实例方法总是和类实例一起出现。一个实例方法必须被一个特定的类实例调用，它提供了访问该实例中各种信息的一种独特方式。例如：

```
String s1="abcde" ;    // one String instance
String s2="fgh";       // another String instance
int i1=s1.length();    // i1=5
int i2=s2.length();    // i2=3
```

length 这个方法返回的是目标对象（如 s1，s2）中的字符个数，就好像那个目标是 length 函数的一个参数（C 语言中的 cf.strlen（aString））。但定义一个实例方法时，可以使用“this”这个关键字来指代目标实例。

类方法没有目标实例，所以不需要“this”引用。可以随时随地调用类方法，即使还没有创建该类的实例。类方法如同类变量一样，在声明的时候需加上关键字“static”。

例如，Thread 线程类中的 currentThread 方法就是一个类方法。可以通过 Thread 类本身来调用它，它确定当前正在执行的线程并返回对该线程的一个引用。

```
Thread activeThread=Thread.currentThread();
```

现在有许多 Java 程序员更喜欢用名词对象、域、静态域、方法、静态方法来代替实例变量、实例方法、类变量、类方法。表 1.2 简要给出了这些名词之间的联系。

表 1.2 类中各元素命名

成员	域	实例变量（或非静态域）
		类变量（或静态域）
	方 法	实例方法（或非静态方法）
		类方法（或静态方法）

注意

术语“域”在区别类中定义的变量和方法中的变量时非常有用。

VariableExample.html

请参看：Q1.2、Q1.3、Q1.5

Q1.7

如何创建类实例？

创建类实例的方法通常是调用该类的构造函数。

在 Java 中，每当创建对象（如实例和数组）时系统都会从它管理的内存中分配一定的空间给这个对象。虽然您永远无法直接访问那些堆栈，但是 Java 语言和 Java 虚拟机会通过关键字 new 和 newInstance 方法以及自动垃圾收集器为您处理这一切。

创建实例的标准方法是用关键字 new，然后加上类名以及类的某个构造函数所需要的参数，例如：

```
Button myButton = new Button (" Press Me" );
```

这行代码声明了一个叫 myButton 的变量，创建了一个叫做“Press Me”的新按钮，并将 myButton 指向这个按钮。Java 常用的编码风格是将类变量的声明和初始化放在同一个程

序行中，也可以简单地声明一个 Button 类型的实例变量，如

```
Button myButton;
```

这样就创建了一个叫做 myButton 的变量，它被缺省初始化（参看 Q1.10）为 null；或者如果声明的是一个本地变量（即方法内部的变量），系统会要求给它先赋初值后才可以使用。

还可以通过调用方法来创建或得到类的实例。有些方法的返回值通常会是一个新的实例，例如 Class 类（Q2.15、Q3.7）中的 newInstance、Integer 类（Q10.4）中的 valueOf；此外，一些方法如 InetAddress 的 getByName（Q10.16）也能返回一个实例，只是返回的实例与原来的相同。

请参看：Q1.10、Q2.15、Q3.5、Q3.7、Q10.4、Q10.16

Q1.8

什么是抽象方法？

一个抽象方法包括了定义此方法的各方面内容，只是不包括实现方法的程序代码，即方法体。

一个抽象的方法声明了方法的名字、参数类型、返回值以及例外事件等，只是没有提供方法的实现方案。可以通过加关键字 abstract 并在方法体的位置上用一个分号代替来声明这种抽象方法。如：

```
public abstract void drawFigure();  // abstract method declaration
```

抽象方法允许您设计一个类而把部分或所有的代码实现留给子类去完成。包含有抽象方法的类被称做抽象类，但也必须在声明时加上关键字 abstract。一旦类中包含了抽象的方法，所有继承此类的子类将不得不为这些方法加上具体的程序代码以实现方法定义的所有功能。

例如，java.lang 包中的 Number 类就全部是由抽象方法组成的，完整的类定义如下：

```
/* in Number,java (JDK 1.0.2 and 1.1) */
public abstract class Number {
    public abstract int intValue();
    public abstract long longValue();
```

```
    public abstract float floatValue();
    public abstract double doubleValue();
}
```

java.lang 包中也包含了 Number 的一些具体子类：Integer、Long、Float、Double、Byte 和 Short，并且每个子类中都定义了上述抽象方法的实现代码。

另一个例子是 java.io 包中的 InputStream 类，它为庞大的、可以处理特殊类型的面向字节输入流的子类家族提供了一个基础。JDK 1.0.2 类定义又实现了除一个方法之外的所有方法（在 JDK 1.1 中,InputStream's available 方法也是抽象定义的）但是负责从输入设备中读入一个字节的无参数方法 read 仍然是一个抽象的方法，因此它的子类必须定义它：

```
public abstract int read () throws IOException;
```

InputStream 类中的其他读方法都被设计成最后也转去调用 read()来实现。这样，当在子类中实现 read()方法后，类中的其他方法就已经随着新方法的定义而实现了。

请参看：Q1.9、Q1.11

Q1.9

什么是抽象类？

一个抽象类是一个没有程序实现部分而需要由其子类去填充完整的类。

一个抽象类是被有意设计成未完成的样子。它只定义了一个框架，不同的子类（Q1.11）可以根据自己的情况去填充不同的内容。不同的现实子类（非抽象的）提供一组变体。

一个抽象类必须用关键字 abstract 明确地声明。注意声明一个抽象类非常简单，它甚至不需要任何抽象方法。通过 abstract 的声明就等于告诉子类它在功能上是不完整的，所以不能创建它的实例。

例如，AWT 中的 Component 类对于所有 AWT 用户界面元素来说是一个抽象父类。尽管它为所有的方法都提供了缺省的实现方案，但它却依然是抽象的。Component 不能被直接实例化，相反它却提供了一个通用的体系结构，因此它的许多子类如 Button, TextField 可以被定义和初始化。

请参看：Q1.8、Q1.11、Q5.1

Q1.10

什么是方法的引用？

一个对象的引用从本质上讲是一个具有很强的完整和安全方面约束能力的对象指针。

作为 Java 程序员，可以通过对象的引用来存取对象实例或对象数组。对象的引用为运行系统提供了两类信息：

- 一个指向对象实例信息——数据的指针
- 一个指向对象类信息——运行类型和方法表的指针

尽管指针只存在于 Java 运行系统中，但是却不能直接操作它。Java 语言不支持引用或指针的数字化概念：不能对指针进行计算，也不能通过数据去造指针。

我们可以把对象的引用看做一个实体来对它进行操作，表 1.3 选自《Java 语言规范》一书（第 39 页），它简要介绍了建立在对象引用之上的操作。

表 1.3　建立在对象引用之上的操作

将对象的引用赋值给变量
在引用上调用方法
从引用中提取某个域
将引用强制转换成另一个类型
用 instanceof 操作来检测该引用的运行状态
将引用和字符串实例连接在一起
使用“==”和“！=”来检测两个引用是否相等
在条件操作？：中将引用作为一个值

Java 中对象引用的完整性是其安全性（防止蓄意破坏）和保险性（防止错误发生）的基石。

想一想下面的程序片段：

```
String s1;
String s2;
s1= "a string";
s2= s1;
```

头两行声明了两个 String 变量；这时，s1 和 s2 只是两个尚未初始化的局部变量。第三行将 s1 指向一个字符串实例“a string”（Q2.8）。在第四行中，程序将 s2 指向同一个对象 s1。现在，虽然只有 String 类的实例，却有两个不同的引用指向它。

当我们声明类、接口或数组类型变量时，它的值总是一个对象的引用或 null。下表将 Java 和 C++中这种定义的相似之处进行了一个比较。

Java	C++
Button b;	Button *b;
String s;	String *s;

请参看：Q2.8

子类、重载和覆盖

Q1.11

什么是子类？

子类被定义成对某个已存在类的修改或扩展。

除了 java.lang 中的 Object 类，Java 语言中所有的类都是其他一些类（我们称它们为直接父类或父类）的子类。除了某些特殊的扩展和更改之外，子类在功能和使用上和它的父类一样。

一个子类可以通过为自己的实例增加额外的实例变量和方法来扩展它的父类，或者为自身定义一些特殊的类变量和类方法；子类还可以通过改写父类的方法来改变父类的行为。无论哪种情况，Java 语言都通过关键字 extends 来声明这种关系。例如，在 JDK 中，java.io 包中的 FileInputStream 被定义为 InputStream 的一个子类：

```
/* in FileInputStream.java(JDK 1.0.2 and 1.1): */
public class FileInputStream extends InputStream {
```