# 第3章 世界，你好！

**本章内容**

- 类和构件
- 静态模型和动态模型
- 模型间的联系
- 对UML的扩展

Brian Kernighan和Dennis Ritchie是编程语言C的作者，他们指出，学习一门新的编程语言的唯一方法是用它编写程序。对于UML也是如此，学习UML的唯一方法是用它绘制模型。

当开始学习一门新的编程语言时，很多开发者写的第一个程序是简单的，只包含了一些打印串“世界，你好！”之类的语句。这是一个合理的出发点，因为掌握这种微小的应用可以立见成效。同时，它也覆盖了使某些东西运行所需要的全部基础设施。

现在我们开始使用UML。对“世界，你好！”的建模大概是你曾见到的UML的最简单的应用。然而，这个应用的简单只是一种假象，因为在整个应用的下面有一些使之工作的有趣的机制。用UML可以很容易地对这些机制建模，UML对这种简单的应用提供了较为丰富的视图。

## 3.1 关键抽象

在Web浏览器中，打印“世界，你好！”的Java程序是很简单的：

```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet {
  public void paint (Graphics g) {
    g.drawString("世界,你好!", 10, 10);
  }
}
```

第一行代码：

```
import java.awt.Graphics;
```

使得下面的代码可以直接使用类Graphics。前缀java.awt表明了类Graphics所在的Java包。

第二行代码：

```
class HelloWorld extends java.applet.Applet {
```

引入了一种名为HelloWorld的新类，并说明它是一种像Applet那样的类，Applet位于包java.applet中。

其余的三行代码：

```
public void paint (Graphics g) {
  g.drawString("世界,你好!", 10, 10);
}
```

声明了一个名为 paint的操作，它的执行调用另一个名为 drawString的操作，drawString操作负责在指定的位置上打印“世界，你好！”。在通常的面向对象的方式下，drawString是一个名称为g的参数中的一个操作，g的类型是类Graphics。

在UML中，对这种应用的建模是简单的。如图3-1所示，把类HelloWorld用一个矩形图标表示。类HelloWorld的paint操作也展示在这里，它的所有形式参数已被省略掉，在一个附属的注解中详述了该操作的实现。【在第4章和第9章中讨论类。】

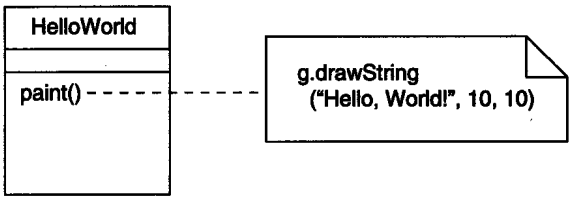


图3-1 对HelloWorld 的关键抽象

> **注释** 如图中所示的那样，虽然UML允许（但不要求）紧密地与各种语言（如Java）相结合，但UML不是一种可视化的编程语言。UML被设计为：允许把模型转换成代码，也允许把代码通过逆向工程转换为模型。在UML中，像数学表达式这样的事物最好用文字性的编程语言语法来书写；而像类层次这样的事物最好以图形的方式来可视化。

这个类图反映出了“世界，你好！”这个应用的基本部分，但还遗漏一些事物。按上述代码的描述，这个应用还涉及其他两个类，即 Applet和Graphics，而且二者的使用方式不同。类Applet是类HelloWorld的父类，类Graphics则是在类HelloWorld的一个操作paint的特征标记和实现中被使用。可以在类图中表示这些类及其他们与类 HelloWorld的不同关系，如图3-2所示。

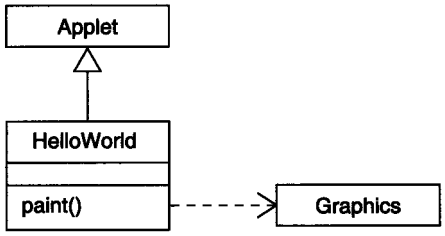


图3-2 与HelloWorld 直接相关的类

用矩形图标表示类Applet和类Graphics。因为不显示它们的任何操作，因此对它们的图标进行了省略。从HelloWorld到Applet的带有空心箭头的有向线段表示的是泛化关系，在这

里它意味着HelloWorld是Applet的子类。从HelloWorld到Graphics的有向虚线段表示的是依赖关系，它意味着HelloWorld使用Graphics。【在第5章和第10章中讨论关系。】

至此，被HelloWorld建造于其上的框架还没有完工。如果针对Applet和Graphics研究Java库，将会发现这两个类是一个更大的类层次的一部分。跟踪类Applet扩展和实现的那些类，能够产成另一个类图，如图3-3所示。

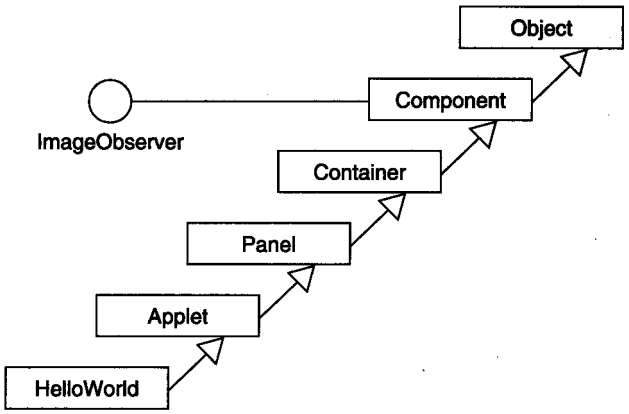


图3-3 HelloWorld 的继承层次

注释 通过对一个已经存在的系统进行逆向工程可以产生图，图3-3就是一个很好的例子。逆向工程是从代码创建模型。

可以清楚地从本图中看出，HelloWorld仅是这个较大类层次的一个叶子。HelloWorld是Applet的子类；Applet是Panel的子类；Panel是Container的子类；Container是Component的子类；Component是Object的子类；Object是Java中各类的父类。因而，这个模型与Java库（每个子类都扩展了某些父类）相匹配。

ImageObserver和Component之间的关系有点不同，这在类图中已经反应出来。在Java库中，ImageObserver是一个接口，它在其他事物中的含义是它没有实现，而需要其他类实现它。如图所示，在UML中把接口表示成一个圆。Component实现ImageObserver这件事是用一条从实现（Component）到接口（ImageObserver）的实线来表示。【在第11章中讨论接口。】

如这些图所示，HelloWorld只与两个类（Applet和Graphics）直接协作，这两个类是已预定义的Java大类库的一小部分。为了管理如此大规模的类层次图，Java用一些不同的包组织它的接口和类。在Java环境中，命名根包为Java（这不奇怪）。被嵌套在这个包中的是几个其他的包，每个包还含有其他的包、接口和类。 Object位于包lang中，它的全程路径名为java.lang.Object。类似地，Panel、Container和Component位于包awt中；类Applet位于包applet中。接口ImageObserver位于包image中，依次下去，image位于包awt中，它的完整路径名是一个相当长的串：java.awt.image.ImageObserver。

可以把这个包在一个类图中可视化，如图3-4所示。

像该图所显示的，在UML中包被表示成带有标签的文件夹。包可以被嵌套，有向的虚线段

描述了包之间的依赖。例如， HelloWorld依赖包java.applet，java.applet依赖包java.awt。【在第12章中讨论包。】

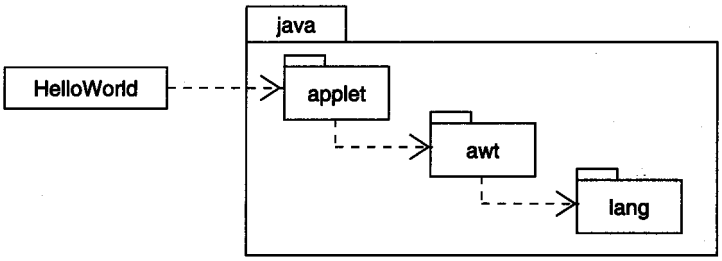


图3-4 HelloWorld 包

## 3.2 机制

掌握像Java类库这样的内容丰富的类库的最难的部分是理解各部分如何协同工作。例如，如何调用HelloWorld的操作paint？若想改变这段程序的行为（例如以不同的颜色打印字符串），必须要使用什么样的操作？为了回答这样和那样的问题，必须有一个描述这些类动态协同工作方式的概念模型。【在第28章讨论模式和框架。】

对Java库的研究揭示了HelloWorld的操作paint是从Component继承来的。这仍然要问如何调用该操作的问题。回答是： paint是作为在该程序内运行的线程的一部分被调用的，如图3-5所示。【在第22章中讨论进程和线程。】

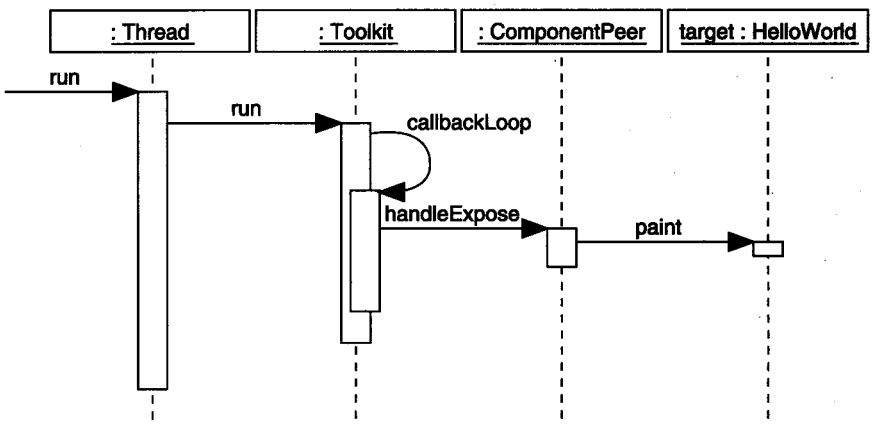


图3-5 painting机制

该图展示了几个对象间的协作，包括类 HelloWorld的一个实例。其他的对象是 Java环境的一部分，其中的大多数对象都处于所创建的程序的背景中。在 UML中，把实例表示得像类一样，只是实例的名称下面有下划线，以示与类的区别。这个图的前 3个对象是匿名的，它们没有

唯一的名称。对象HelloWorld有一个名称（target），这个名称为对象ComponentPeer所知。【在第11章中讨论实例。】

如图3-5所示，可以使用顺序图对事件的顺序建模。这里，顺序从运行对象 Thread开始，它调用Toolkit的一个操作run。对象Toolkit调用它自己的操作（callbackLoop），然后它调用ComponentPeer的操作handleExpose。对象ComponentPeer调用它的目标操作paint。对象ComponentPeer假设它的目标是Component，但在这种情况下，目标实际是Component的子类（即HelloWorld），因此，HelloWorld的操作paint被多态地处理。【在第18章中讨论顺序图。】

## 3.3 构件

“世界，你好！”被实现为一个程序，但它不是单独存在的，它通常是一些 Web页的一部分。打开含有该程序的网页，并由一些运行该程序的对象 Thread的浏览器机制触发，程序就开始执行。然而，类HelloWorld并不直接作为Web页的一部分，而应是由Java编译器（它把描述该类的源代码转换成可执行的构件）产生的该类的二进制形式。这提出了对系统的非常不同的观察角度。所有早期的图描述的是程序的逻辑视图，这里要做的是小应用程序的物理构件的视图。

可以用构件图对这个物理视图建模，如图 3-6所示。【在第25章中讨论构件。】

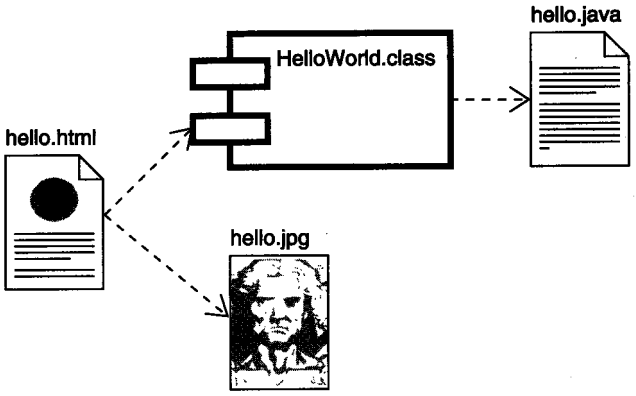


图3-6 HelloWorld 构件

这个图中的每个图符都表示了系统实现视图中一个 UML元素。名为hello.java的构件表示了逻辑类HelloWorld的源代码，它是可以由开发环境和配置管理工具操纵的文件。用 Java编译器可把这段源代码转换成二进制程序 HelloWorld.class，以使得它可在计算机的Java虚拟机上执行。

构件的规范图符是带有两个标签的矩形。二进制程序 HelloWorld.class是该基本图符的变体，把图符的线条加黑，是为了指明它是一个可执行的构件（正像主动类一样）。构件hello.java的图符已经用一个用户定义的图符所代替，表示它是一个文本文件。通过扩展UML的表示法，类似地制作 Web页hello.html的图符。如图表示的那样，该 Web页有另一个

构件hello.jpg，它由用户定义的构件图标表示，这里它提供了一个图像的缩影。由于后 3个构件使用的是用户定义的图符，故它们名称是在图符的外面。

**注释** 难以明确地对类（HelloWorld）、类的源代码（hello.java）和类的对象代码（HelloWorld.class）之间的关系建模，尽管为了可视化系统的物理配置，有时这样做还是有用的。换句话说，人们经常对像这种基于 Web的系统的组织进行可视化，这要用构件图对它的网页和其他可执行的构件进行建模。