**前言**

一开始,我只是将Java看作"又一种程序设计语言".从许多方面看,它也的确如此.

但随着时间流逝,以及对Java的深入研究,我渐渐发现,与我所见过的其他编程语言相比,Java有着完全不同的核心目的.

程序设计其实是对复杂性的管理:待解决问题的复杂性,以及用来解决该问题的工具的复杂性.正是这种复杂性,导致多数程序设计项目失败.在我所知的所有程序设计语言中,几乎没有哪个将自己的设计目标专注于克服开发与维护程序的复杂性.

随着对Java的了解越来越深,Sun对Java的设计目标给我留下了最深刻印象,那就是,为程序员减少复杂性.用他们的话说就是:"我们关心的是,减少开发健壮代码所需的时间以及困难."

**绪论**

"上帝赋予人类说话的能力,而言语又创造了思想,思想是人类对宇宙的量度."

人类......极其受那些已经成为社会表达工具的特定语言的支配.想象一下,如果一个人可以不使用语言就能够从本质上适应现实世界,语言仅仅是解决具体的交流和反映问题时偶尔才用到的方式,我们会发现,这只能是一种幻想.事实上,"真实世界"在很大程度上是不知不觉地基于群体的语言习惯形成的.

如同任何人类语言一样,Java提供了一种表达概念的方式.如果使用得当,随着问题变得更庞大更复杂,这种表达工具将会比别的可供选择的语言更为简单、灵活。

我们不应该将Java仅仅看作是一些特性的集合——有一些特性在孤立状态下没有任何意义。只有在考虑到设计，而不仅仅是编码时，才能完整地运用Java的各部分。而且，要按照这种方式来理解Java，必须理解在语言和编程中经常碰到的问题。这本书讨论的是编程问题，它们为什么成为问题，以及Java已经采取什么样的方案来解决它们。因此，每章所阐述的特性集，都是基于我所看到的这一语言在解决特定类型问题时的方式。按照这种方式，我希望能够每次引导读者前进一点，直到Java思想意识成为你最自然不过的语言。

自始至终，我一直持这样的观点：你需要在头脑中创建一个模型，以加强对这种语言的深入理解；如果你遇到了疑问，就将它反馈到头脑中的模型并推断出答案。

**对象导论**

“我们之所以将自然界分解，组织成各种概念，并按其含义分类，主要是因为我们是整个口语交流社会共同遵守的协定的参与者，这个协定以语言的形式固定下来……除非赞成这个协定中规定的有关语言信息的组织和分类，否则我们根本无法交谈。”

所有编程语言都提供抽象机制。可以认为，人们所能够解决的问题的复杂性直接取决于抽象的类型和质量。

面向对象方式通过向程序员提供表示问题空间中的元素的工具而更进了一步。这种表示方式非常通用，使得程序员不会受限于任何特定类型的问题。我们将问题空间中的元素及其在解空间中的表示成为“对象”。这种思想的实质是：程序可以通过添加新类型的对象使自身适用于某个特定问题。因此，当你在阅读描述解决方案的代码的同时，也是在阅读问题的表述。相比以前我们所使用的语言，这是一种更灵活和更强有力的语言抽象。所以，OOP允许根据问题来描述问题，而不是根据运行解决方案的计算机来描述问题。

面向对象语言包含的特性：

（1）万物皆为对象。将对象视为奇特的变量，它可以存储数据，除此之外，你还可以要求它在自身上执行操作。理论上讲，你可以抽取待求解问题的任何概念化构件（狗、建筑物、服务等），将其表示为程序中的对象。

（2）程序是对象的集合，它们通过发送消息来告知彼此所要做的。要想请求一个对象，就必须对该对象发送一条消息。更具体地说，可以把消息想象为对某个特定对象的方法的调用请求。

（3）每个对象都有自己的由其他对象所构成的存储。换句话说，可以通过创建包含现有对象的包的方式来创建新类型的对象。因此，可以在程序中构件复杂的体系，同时将其复杂性隐藏在对象的简单性背后。

（4）每个对象都拥有类型。按照通用的说法，“每个对象都是某个类的一个实例”。每个类最重要的区别于其他类的特性就是“可以发送什么样的消息给它”。

（5）某一特定类型的所有对象都可以接受同样的消息。因为“圆形”类型的对象同时也是“几何形”类型的对象，所以一个“圆形”对象必定能够接受发送给“几何形”对象的消息。这意味着可以编写与“几何形”交互并自动处理所有与几何形性质相关的事物的代码。这种可替代性是OOP中最强有力的概念之一。

**每个对象都提供服务**

当正在试图开发或理解一个程序设计时，最好的方法之一就是将对象想象为“服务提供者”。程序本身将向用户提供服务，它将通过调用其他对象提供的服务来实现这一目的。你的目标就是去创建（或者最好是在现有代码库中寻找）能够提供理想的服务来解决问题的一系列对象。

着手从事这件事的一种方式就是问一下自己：“如果我可以将问题从表象中抽取出来，那么什么样的对象可以马上解决我的问题呢？”例如，假设你正在创建一个簿记系统，那么可以想象，系统应该具有某些包括了预定义的簿记输入屏幕的对象，一个执行簿记计算的对象集合，以及一个处理在不同打印机上打印支票和开发票的对象。也许上述对象中的某些已经存在了，但是对于那些并不存在的对象，它们看起来像什么样子？它们能够提供哪些服务？它们需要哪些对象才能履行它们的义务？如果持续这样做，那么最终你会说“那个对象看起来很简单，可以坐下来写代码了”，或者说“我肯定那个对象已经存在了”。这是将问题分解为对象集合的一种合理方式。

将对象看成服务提供者还有一个附带的好处：它有助于提高对象的内聚性。高内聚是软件设计的基本质量要求之一：这意味着一个软件构件的各个方面“组合”得很好。人们在设计对象时所面临的一个问题是，将过多的功能塞在一个对象中。例如，在检查打印模式的模块中，你可以这样设计一个对象，让它了解所有的格式和打印技术。你可能会发现，这些功能对于一个对象来说太多了，你需要的是三个甚至更多个对象，其中，一个对象可以是所有可能的支票排版目录，它可以被用来查询有关如何打印一张支票的信息；另一个对象（或对象集合）可以是一个通用的打印接口，它知道有关所有不同类型的打印机的信息（它更应该是一个需要购买而不是自己编写的对象）；第三个对象通过调用另两个对象的服务来完成打印任务。这样，每个对象都有一个它所能提供服务的内聚的集合。在良好的面向对象设计中，每个对象都可以很好地完成一项任务，但是它并不试图做更多的事。就像在这里看到的，不仅允许通过购买获得某些对象（打印机接口对象），而且还可以创建能够在别处复用的新对象（支票排版目录对象）。

将对象作为服务提供者看待是一件伟大的简化工具，这不仅在设计过程中非常有用，而且当其他人试图理解你的代码或重用某个对象时，如果他们看出了这个对象所能提供的服务的价值，它会使调整对象以适应其设计的过程变得简单得多。