# 第一章 对象导论

## 1.1抽象过程

纯粹的面向对象程序设计方式

1. 万物皆为对象
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
3. 美个对象都有自己的有其他对象构成的存储
4. 每个对象都拥有其类型
5. 某一个特定类型的所有对象都可以接受同样的消息

## 1.2每个对象都有一个接口

## 1.3每个对象都提供服务

# 第二章一切都是对象

## 2.1 用引用操纵对象

## 2.2 必须由你创建所有对象

### 2.2.1 存储到什么地方

1. 寄存器

这是最快的存储区，因为它位于不同于其它存储区的地方--处理器内部。但是寄存器的数量极其有限，所以寄存器根据需求进行分配。你不能直接控制，也不能在程序中感觉到寄存器存在的任何迹象（另一方面，C和C++允许您向编译器建议寄存器的分配方式）。

1. 堆栈

位于通用RAM中，但通过栈堆栈指针可以从处理器那里获得直接支持。堆栈指针若向下移动，则分配新的内存；若向上移动，则释放那些内存。这是一种快速有效的分配存储方法，仅次于寄存器。创建程序时，Java系统必须知道存储在堆栈内所有项的确切生命周期，以便于上下移动堆栈指针。这一约束限制了程序的灵活性，所以虽然某些Java数据存储于对战中-特别是对象引用，但是Java对象并不存储于其中。

1. 堆

一种通用的内存池（也位于RAM区），用于存放所有的Java对象。堆不同于堆栈的好处：编译器不需要知道存储的数据在堆里存活多长时间。因此，在堆里分配存储有很大的灵活性。当需要一个对象时，只需要new写一行简单代码，当执行这行代码时，会主动在堆里进行存储分配。当然，为这种灵活性必须付出代价：用堆进行分配和清理可能比用堆栈进行存储分配需要更多的时间（如果确实可以再Java中像C++中一样在栈中创建对象）。

1. 常量存储

常量值通常直接存放在程序代码内部，这样做是安全的，因为它们永远不会被改变。有时，在嵌入式系统中，常量本身会与其它部分隔离开，所以在这种情况下，可以选择将其存放在ROM（只读存储器）中。

1. 非RAM存储

如果数据完全存活于程序之外，那么它可以不受程序的任何控制，在程序没有运行时也可以存在。其中两个基本的例子是流对象和持久化对象。在流对象中，对象转化成字节流，通常被发送到另一台机器。在“持久化对象”中，对象呗存放于磁盘，因此，即使程序终止，他们仍可以保持自己的状态。这种存储方式的技巧在于：把对象转化成可以存放在其它媒介上的事务，在需要时可恢复成常规的、基于RAM的对象。Java提供了对轻量级持久化的支持，而诸如JDBC和Hibernate这样的机制提供了更加复杂的对在数据库中存储和读取对象信息的支持。

### 2.2.2 特例：基本类型

在程序设计经常用到一系列类型，他们需要特殊对待。可以把它们想象成基本类型。之所以特殊对象，是因为new将对象存储在堆里，故用new创建一个对象--特别是小的、简单的变量，往往不是很有效。因此，对于这些类型，Java才去与C和C++相同的方法。也就是说，不用new来创建变量，而是创建一个并非是引用的自动变量。这个变量直接存储“值”，并置于堆栈中，因此更加高效。

Java要确定每种基本类型所占存储空间的大小。它们并不像其他大多数语言那样随机器硬件架构的变化而变化。这种所占存储空间大小而定不变性是Java程序比用其他大多属于盐编写的程序更具有可移植性的原因之一。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 基本类型 | 大小 | 最小值 | 最大自 | 包装器类型 |
| boolean | - | - | - | Boolean |
| char | 16-bit | Unicode 0 | Unicode216-1 | Char |
| byte | 8 bits | -27 | 27-1 | Byte |
| short | 16 bits | -215 | 215-1 | Short |
| int | 32 bits | -231 | 231-1 | Integer |
| long | 64 bits | -263 | 263-1 | Long |
| float | 32 bits | IEEE754 | IEEE754 | Float |
| double | 64 bits | IEEE754 | IEEE754 | Double |
| void | - | - | - | Void |

所有数值类型都有正负号，所以不要去寻找无符号的数值类型。

boolean类型所占存储空间的大小并没有明确指定，仅定义为能够去字面值true或false

基本类型具有的包装器类