而从 Map 中读取一条数据就可以这么写：

```
val number = map["Apple"]
```

因此，上述代码经过优化过后就可以变成如下形式：

```
val map = HashMap<String, Int>()
map["Apple"] = 1
map["Banana"] = 2
map["Orange"] = 3
map["Pear"] = 4
map["Grape"] = 5
```

当然，这仍然不是最简便的写法，因为 Kotlin 毫无疑问地提供了一对 mapOf()和 mutableMapOf()函数来继续简化 Map 的用法。在 mapOf()函数中，我们可以直接传入初始化的键值对组合来完成对 Map 集合的创建：

```
val map = mapOf("Apple" to 1, "Banana" to 2, "Orange" to 3, "Pear" to 4, "Grape" to 5)
```

这里的键值对组合看上去好像是使用 to 这个关键字来进行关联的，但其实 to 并不是关键字，而是一个 infix 函数，我们会在本书第 9 章的 Kotlin 课堂中深入探究 infix 函数的相关内容。

最后再来看一下如何遍历 Map 集合中的数据吧，其实使用的仍然是 for-in 循环。在 main()函数中编写如下代码：

```
fun main() {
    val map = mapOf("Apple" to 1, "Banana" to 2, "Orange" to 3, "Pear" to 4, "Grape" to 5)
    for ((fruit, number) in map) {
        println("fruit is " + fruit + ", number is " + number)
    }
}
```

这段代码主要的区别在于，在 for-in 循环中，我们将 Map 的键值对变量一起声明到了一对括号里面，这样当进行循环遍历时，每次遍历的结果就会赋值给这两个键值对变量，最后将它们的值打印出来。重新运行一下代码，结果如图 2.25 所示。

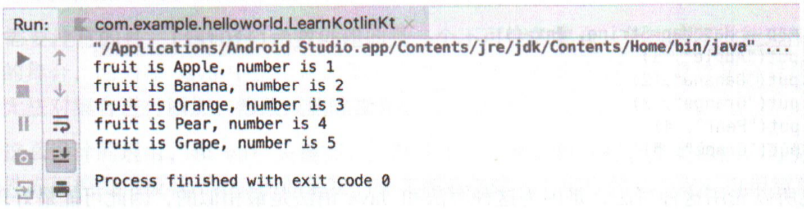

图 2.25　遍历 Map 中的数据

好了，关于集合的创建与遍历就学到这里，接下来我们开始学习集合的函数式 API，从而正式入门 Lambda 编程。