2.6.3 Java 函数式 API 的使用

现在我们已经学习了 Kotlin 中函数式 API 的用法，但实际上在 Kotlin 中调用 Java 方法时也可以使用函数式 API，只不过这是有一定条件限制的。具体来讲，如果我们在 Kotlin 代码中调用了一个 Java 方法，并且该方法接收一个 Java 单抽象方法接口参数，就可以使用函数式 API。Java 单抽象方法接口指的是接口中只有一个待实现方法，如果接口中有多个待实现方法，则无法使用函数式 API。

如果你觉得上面的描述有些模糊的话，没关系，下面我们通过一个具体的例子来学习一下，你就能明白了。Java 原生 API 中有一个最为常见的单抽象方法接口——Runnable 接口。这个接口中只有一个待实现的 run()方法，定义如下：

```
public interface Runnable {
    void run();
}
```

根据前面的讲解，对于任何一个 Java 方法，只要它接收 Runnable 参数，就可以使用函数式 API。那么什么 Java 方法接收了 Runnable 参数呢？这就有很多了，不过 Runnable 接口主要还是结合线程来一起使用的，因此这里我们就通过 Java 的线程类 Thread 来学习一下。

Thread 类的构造方法中接收了一个 Runnable 参数，我们可以使用如下 Java 代码创建并执行一个子线程：

```
new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("Thread is running");
    }
}).start();
```

注意，这里使用了匿名类的写法，我们创建了一个 Runnable 接口的匿名类实例，并将它传给了 Thread 类的构造方法，最后调用 Thread 类的 start()方法执行这个线程。

而如果直接将这段代码翻译成 Kotlin 版本，写法将如下所示：

```
Thread(object : Runnable {
    override fun run() {
        println("Thread is running")
    }
}).start()
```

Kotlin 中匿名类的写法和 Java 有一点区别，由于 Kotlin 完全舍弃了 new 关键字，因此创建匿名类实例的时候就不能再使用 new 了，而是改用了 object 关键字。这种写法虽然算不上复杂，但是相比于 Java 的匿名类写法，并没有什么简化之处。

但是别忘了，目前 Thread 类的构造方法是符合 Java 函数式 API 的使用条件的，下面我们就看看如何对代码进行精简，如下所示：