Robolectric 框架设计调研

何伟[1)]

[1)](南京大学 计算机科学与技术系, 南京市 中国 210046)

摘 要 移动应用平台已经深入人们生活，而安卓作为主流的移动平台被人们广泛使用。安卓应用开发也变得热门起来，但是对安卓应用进行测试确实比较麻烦的事，应用需要先安装到真机或者模拟器上，才能进行功能测试。Robolectric 是一款针对安卓应用的单元测试框架，能够在不安装应用的情况下对应用代码进行单元测试，进而提高开发者的开发效率。

关键词 Robolectric，安卓测试

中图法分类号 **** DOI 号 *投稿时不提供 DOI 号*

Robolectric Framework Design Study

Wei He[1)]

[1)](Department of Computer Science and Technology, Nanjing University, Nanjing 210046, China)

Abstract Mobile application platforms have penetrated into people's lives, and Android is widely used as a mainstream mobile platform. Android application development has also become popular, but it is really troublesome to test Android applications. The application needs to be installed on a real machine or an emulator before functional testing can be performed. Robolectric is a unit testing framework for Android applications, which can unit test the application code without installing the application, thereby improving the development efficiency of developers.

Key words Robolectric, Android Testing

1. 引言

移动互联网的兴起使得移动软件成为人们生活中重要的组成部分。安卓作为最流行的移动平台之一，为人们生活带来巨大便利。为了抢占市场，应用开发者们也不停地更新应用，开发更多的功能，但是对应用代码的测试确实比较麻烦的。想要测试刚编写的代码，需要先编译应用，在将应用安装到安卓设备上，启动应用然后才能测试。Robolectric 是 Google 开发的一款针对安卓应用的单元测试框架，它是基于 Junit 开发的，提供了安卓应用运行时所需要的 Framework 函数库支持。

我们知道，安卓应用的运行需要大量库函数支持，即 Android Framework，这些库函数只能在安卓设备上才能正常运行，Robolectric 构造了一种 Shadow 机制来处理这种问题。本文调

收稿日期：2022年7月7日；何伟，南京大学计算机系硕士生

研了 Robolectric 框架的多种重要设计，如使用了 InvokeDynamic 指令，构造了 Shadow 机制，创建了新的 ClassLoader 对字节码插桩等以及它是如何通过这些设计在 JVM 中运行安卓代码的。

2. InvokeDynamic

Java 7 版本之前，Java 中有四个与分发分派有关的字节码，涉及 Java 中不同的方法调用，即 invokevirtual，invokestatic，invokeinterface 和 invokespecial。四条方法调用指令的第一个参数都是被调用方法的符号引用，这个引用在编译时期产生，而动态类型语言的只有在运行时才能确定方法分派的接收者，因此 Java 在对动态类型语言的支持上有所欠缺。为了从虚拟机层面增强 Java 对动态类型语言的支持，Java 中添加了一条新的指令 invokedynamic 以及 java.lang.invoke 包。

Java.lang.invoke 包的目的是在之前的仅通过符号引用确定调用的目标方法之外，提供一种新的动态确定目标方法的机制，称为方法句柄。方法句柄的查找需要知道方法类型，java.lang.invoke 包中提供的 MethodType 需要方法的返回类型以及参数类型作为参数。之后可以通过 lookup 方法在指定类中找到符合要求的方法句柄。

方法句柄与 Java 中的反射机制又有不同之处，反射是在模拟 Java 代码层面的方法调用，而方法句柄是在模拟字节码层面的方法调用。由于方法句柄的设计是基于 Java 虚拟机的，所以它的应用范围更广，不仅仅包含了 Java 语言，还能用于一些基于 Java 虚拟机的其他语言等。

invokedynamic 和方法句柄的作用是一样的，不过一个是使用代码层面的 API 实现，一个使用字节码实现功能。该指令允许用户级别的代码来确定执行哪一个方法调用，方法分派的决定权从虚拟机转嫁到用户代码中。

它的目的在于由用户通过方法句柄 API 在运行时确定如何分派，同时避免反射带来的性能惩罚和安全问题。invokedynamic 指令执行时，通过用户代码决定方法解析，找到要调用的函数，该用户代码被称为引导方法。引导方法会返回一个调用点对象，调用点对象包含了一个方法句柄，指向真正执行的方法。每个 invokedynamic 都会有一个引导方法关联，遇到 invokedynamic 指令时，引导方法会被调用。引导方法的参数包含了调用者信息，调用方法名以及方法类型。通过这些信息查找实际分派的方法，获得方法句柄，封装在调用点对象中返回。

3. Shadow 机制

安卓应用的运行需要有库函数的支持，但是由于其中有些函数的实现与安卓系统服务有关或者涉及到了 Native 方法调用，而在电脑上我们无法使用类似的服务或者调用相关的本地函数，直接将库函数的代码移植到电脑上无法正常运行。当然，库函数中有大量的纯 Java 方

法能够直接在电脑上运行。Robolectric 构造了一种 Shadow 机制，对于不能直接运行的库函数，使用功能类似的代码模拟；能正常运行的库函数就直接调用。Robolectric 项目中包含了大量的对库函数模拟的类，并且维护了这些类与库函数之间的映射，为了将对无法正常运行的那部分库函数的调用转到对这部分模拟函数的调用。Shadow 机制的原理很简单，但是为了便捷高效的实现这一机制，需要用到前面提到的 invokedynamic 指令以及 Java 的类加载机制。

4. SandboxClassLoader

我们知道 Java 的类加载采用了双亲委派机制。JVM 的类加载器具有父子关系，当某个类加载器需要加载一个类时，它首先将加载任务传递给父加载器，如果父加载器不能成功加载再由自己加载。Robolectric 通过破坏这一机制来实现 Shadow 机制。首先运行安卓应用代码前将它的类加载器改为 SandboxClassLoader。这是一个特殊的类加载器，它违反了双亲委派机制，并能对字节码插桩。这个类加载器再加载代码时，先判断自己能否加载，如果字节能加载，就不会委派给父加载器。在加载代码时会进行判断，判断是否需要对字节码进行插桩。这里需要插桩的代码大部分都是安卓相关代码。插桩使用了 ASM 框架，插桩的时候对原类中的每一个方法进行“备份”。将原来的方法名加上一个特殊的前缀，然后生成一个与原来这个方法签名相同的方法，但是方法体变了。方法体中只有一个动态调用，即插入了 invokedynamic 指令。

这样如果安卓应用代码调用了库函数的代码，会执行这个只包含了 invokedynamic 指令的方法，然后 Robolectric 框架处理这个动态调用指令。处理的方法是查找建立好的映射，看这个方法是否是需要模拟的。如果找到了映射就返回一个指向模拟方法的调用点，否则返回指向“备份”方法的调用点，即会执行原方法。

不过这个机制仍有一些缺陷。首先是应用依然不能拉起系统服务，当应用希望启动某个服务时，使用 Shadow 机制执行代码，并欺骗应用服务已经成功启动，实际上服务并没有真正启动。另一个问题是当应用代码执行到 Native 函数调用时，无法获取安卓系统中的本地库函数并无法执行该 Native 方法。对于部分 Native 方法，Robolectric 通过 Java 实现等价于 C++/C 库函数的方法，并调用这个方法。

5. 依赖注入与反射

Robolectric 中使用了大量的依赖注入与 Java 反射机制。Robolectric 框架中有很多类相互依赖并且，使用依赖注入可以很方便地创建出这些类的实例。由于 Robolectric 是一个针对单元测试的框架，有些库函数的类被 Shadow 是为了保存一些测试信息，保存完信息之后需要调用原来的方法，会通过 Java 反射机制直接调用。