3月读书笔记与小结

学习内容

《大话设计模式》第1，2章，附录， 《深入探索Android热修复技术原理》第1，2，6章，附录

\*\*第一周\*\*：

1.了解了解工厂模式的设计思路，使用场景，注意事项，并结合项目中子VM创建使用的工厂模式，理论+实践相结合去学习

2.了解到好的设计设计并不是“代码无错就是优？”，初学者代码容易犯的错及代码规范，附录回顾面向对象编程，回顾UML图画法

3.了解面向对象编程中，如何进行业务的封装，复用和耦合，结合项目代码和demo去理解效果，此外，大话设计模式这本书，

通过对话的方式，简单易懂

4.后期会在项目或demo中去实践，排除使用误区和初学者易犯错的地方

5.另外，略读了《深入探索Android热修复技术原理》，了解热修复技术，当前最热门的热修复技术

6.了解当前最热门的热修复技术：Sophix，Dexposed，AndFix，Amigo，腾讯系热修复方案

7.后期会去了解各个热修复技术的技术原理，应用

\*\*第二周\*\*：

1.通过一个商场收银软件来了解策略模式的设计思想，场景，由简单的工厂模式过度到策略模式的实现

2.“面向对象的编程，并不是类越多越好，类的划分是为了封装，但分类的基础是抽象，具有相同属性和功能的对象的抽象集合才是类”

3.策略模式：定义了算法家族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化，不会影响到使用算法的客户，工厂模式虽然能够解决商场打折问题，但是如果商场经常出现促销打折，使用工厂模式就会遇到这样的弊端，经常改写算法或者添加属性，动则牵连其身

4.策略模式的实现，通过分析核心代码，了解策略模式的实现方式，理解其精髓

5.“策略模式是一种定义一系列算法的方法，从概念上来看，所有这些算法完成的都是相同的工作，只是实现不同，它可以以相同的方式调用所有的算法，减少了各种算法类与使用算法类之间的耦合”

6.策略模式的优点是简化了单元测试，因为每个算法都有自己的类，可以通过自己的接口单独测试

\*\*第三周\*\*

1.了解简单工厂模式与策略模式的结合，从简单工厂模式知道两个类(Operation/OperationFactory)，而策略模式只需要知道一个类(CashContext)，耦合度降低，使用的时候只需要实例化一个CashContext对象，并调用改类中的getResult方法即可然收费算法与客户端分离

2.策略模式的解析，通过博客或网上资源去更深入了解工厂模式和策略模式

3.热修复在编译器与语言特性的影响

4.了解Sophix方案纵横方向的比较

5.热修复的介绍，技术概览与沉淀

6.底层热替换原理，突破底层差异的方法

## 学习小结

《大话设计模式》----读书小结

**工厂模式**：Java中最常用的设计模式之一，属于创建型模式，类似工厂加工一样，它提供

了一种创建对象的最佳方式，在该模式中创建对象时不会对客户端暴露创建的逻辑，并且通过使用一个共同的接口来指向新创建的对象

**意图**：定义一个创建对象的接口，让其子类自己决定实例化哪一个工厂类，工厂模式使其创建过程延迟到子类进行

**主要解决问题**：解决接口选择问题

**使用时机**：不同条件下创建不同实例

**解决方式**：让其子类实现工厂接口，返回一个抽象的产品

**8系产品应用**：VM创建与渲染

接口：IViewModelFactory 接口实现：DefaultViewModelFactory

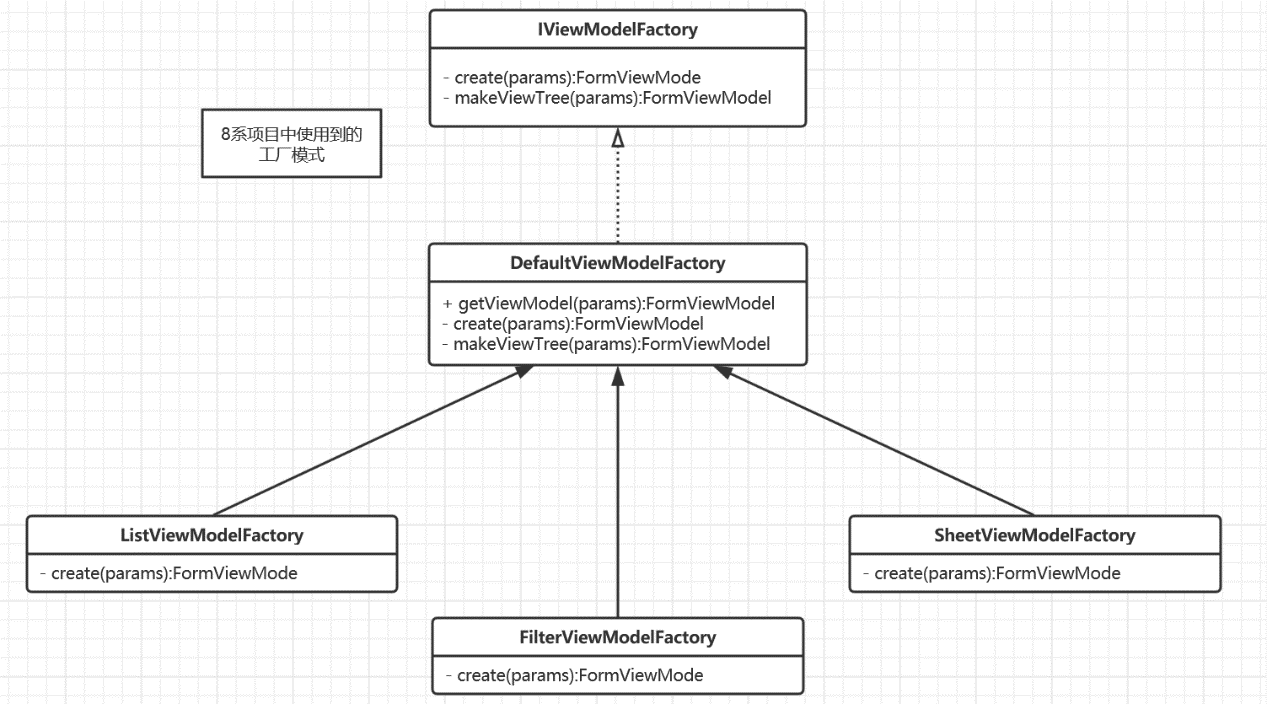
类继承：

FilterViewModelFactory

ListViewModelFactory

SheetViewModelFactory

**工厂模式类图**：



项目中使用的工厂模式还有很多地方，这里列举部分，帮助我理解项目代码及设计思路，为以后的功能代码设计作良好的铺垫，以后也会巧妙地运用该种模式来设计代码。

**优点**：

1、一个调用者想创建一个对象，只要知道其名称就可以了。

2、扩展性高，如果想增加一个产品，只要扩展一个工厂类就可以。

3、屏蔽产品的具体实现，调用者只关心产品的接口。

**缺点**：

每次增加一个产品时，都需要增加一个具体类和对象实现工厂，使得系统中类的个数成倍增加，在一定程度上增加了系统的复杂度，同时也增加了系统具体类的依赖。这并不是什么好事。

**使用场景**：

1、日志记录器：记录可能记录到本地硬盘、系统事件、远程服务器等，用户可以选择记录日志到什么地方。

2、数据库访问，当用户不知道最后系统采用哪一类数据库，以及数据库可能有变化时。

3、设计一个连接服务器的框架，需要三个协议，"POP3"、"IMAP"、"HTTP"，可以把这三个作为产品类，共同实现一个接口。

4、VM渲染与生成UI

**需要注意的地方**：

作为一种创建类模式，在任何需要生成复杂对象的地方，都可以使用工厂方法模式。有一点需要注意的地方就是复杂对象适合使用工厂模式，而简单对象，特别是只需要通过 new 就可以完成创建的对象，无需使用工厂模式。如果使用工厂模式，就需要引入一个工厂类，会增加系统的复杂度。

**策略模式**：在策略模式（Strategy Pattern）中，一个类的行为或其算法可以在运行时更改。这种类型的设计模式属于行为型模式，在策略模式中，我们创建表示各种策略的对象和一个行为随着策略对象改变而改变的 context 对象。策略对象改变 context 对象的执行算法。

**意图**：定义一系列的算法,把它们一个个封装起来, 并且使它们可相互替换。 主要解决问题：在有多种算法相似的情况下，使用 if...else 所带来的复杂和难以维护。 何时时机：一个系统有许多许多类，而区分它们的只是他们直接的行为。

**如何解决**：将这些算法封装成一个一个的类，任意地替换。 关键代码：实现同一个接口。

**应用实例**：

1、诸葛亮的锦囊妙计，每一个锦囊就是一个策略。

2、旅行的出游方式，选择骑自行车、坐汽车，每一种旅行方式都是一个策略。

3、JAVA AWT 中的 LayoutManager。

4、8系项目中使用到glide库中的DiskCacheStrategy缓存策略模式。

**优点**：

1、算法可以自由切换。

2、避免使用多重条件判断。

3、扩展性良好。

**缺点**：

1、策略类会增多。

2、所有策略类都需要对外暴露。

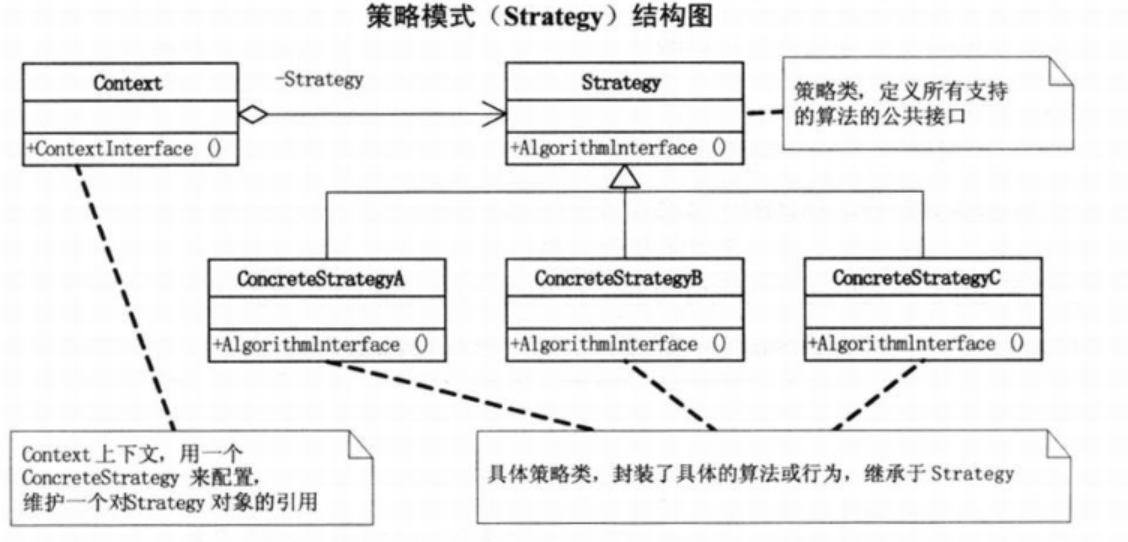
**使用场景**：

1、如果在一个系统里面有许多类，它们之间的区别仅在于它们的行为，那么使用策略模式可以动态地让一个对象在许多行为中选择一种行为。

2、一个系统需要动态地在几种算法中选择一种。

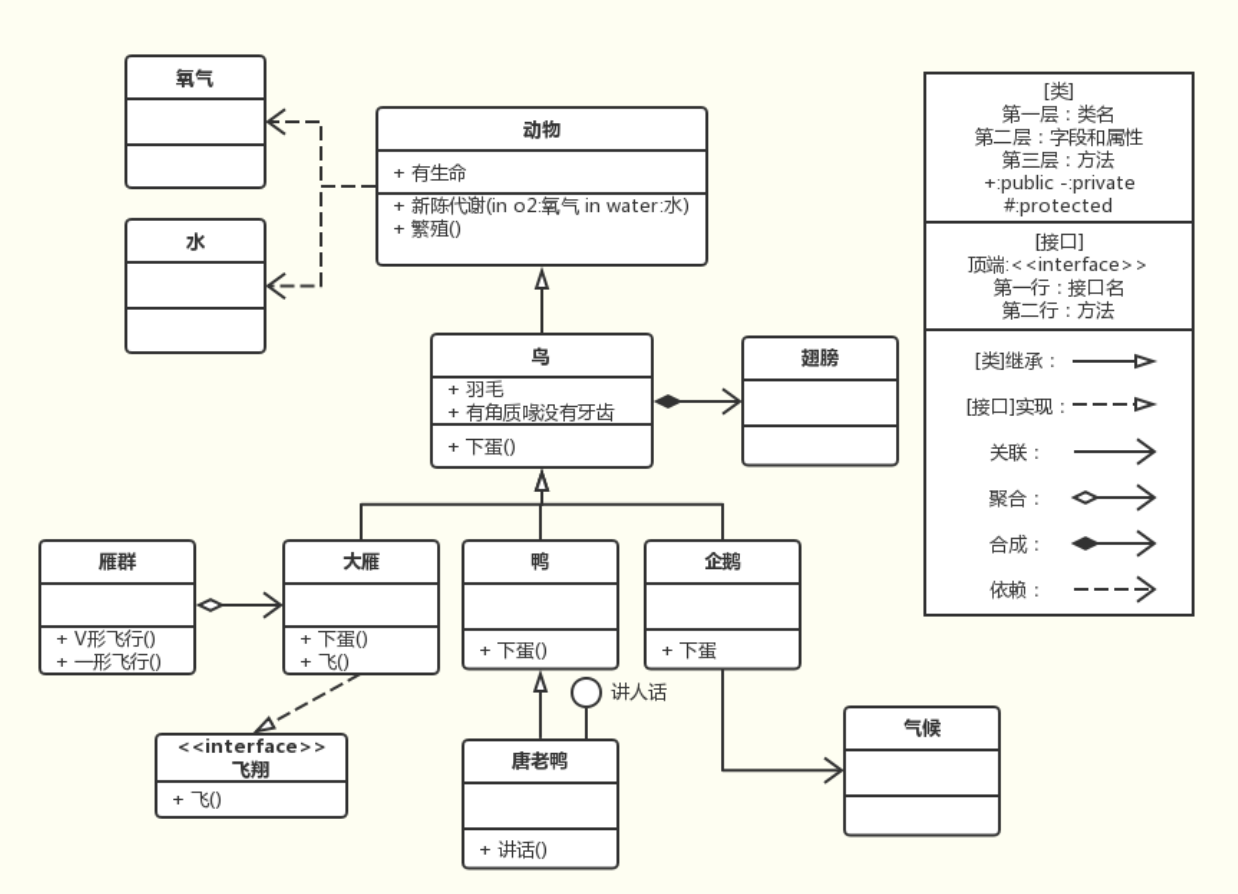
3、如果一个对象有很多的行为，如果不用恰当的模式，这些行为就只好使用多重的条件选择语句来实现。 注意事项：如果一个系统的策略多于四个，就需要考虑使用混合模式，解决策略类膨胀的问题。

策略模式类图：



策略模式类图

熟悉UML类图画法（以超哥的ppt复习）：



类图画法

关联（association）：当一个类“知道”另一个类时，可以用关联。企鹅“知道”气候的变化。

聚合（Aggregation）：表示一种弱的“拥有”关系，体现的是A对象可以包含B对象，但B对象不是A对象但一部分。一个雁群可以有多只大雁，满足聚合关系。

合成（Composition）：是一种强的“拥有”关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样。 依赖（Dependency）：动物依赖氧气和水，它们是依赖关系。

工厂模式和策略模式的思考：目前项目中工厂模式用的地方比较多，用起来也很方便，代码也很简洁，后期的开发会适当采用该模式去处理一些经常会有需求变动造成改动比较打的地方，抽象接口，以不动应万变，实际的开发中可能会有策略和工厂模式的结合，通过结合可以弥补各自的劣势，从而达到互补的效果。后期还会学习其他的设计模式，能够在后面的工作或学习中，通过不同的场景，来熟练选择一种合适的模式，达到因地制宜的效果。

《深入探索Android热修复技术原理》----读书小结

**什么是热修复**？

就给项目打补丁

正常的开发流程：

用户安装

重现发版1.1上线

版本1.0上线

用户登录

发现bug

紧急修复

热修复流程：

自动获取补丁修复

打出补丁推送给用户

紧急修复

发现bug

用户安装

版本1.0上线

热修复优势：

1. 无须重新发版，实时高效热修复。
2. 用户无感知修复bug,无须下载新的应用，代价小。
3. 修复bug成功率高，把损失降到最低。

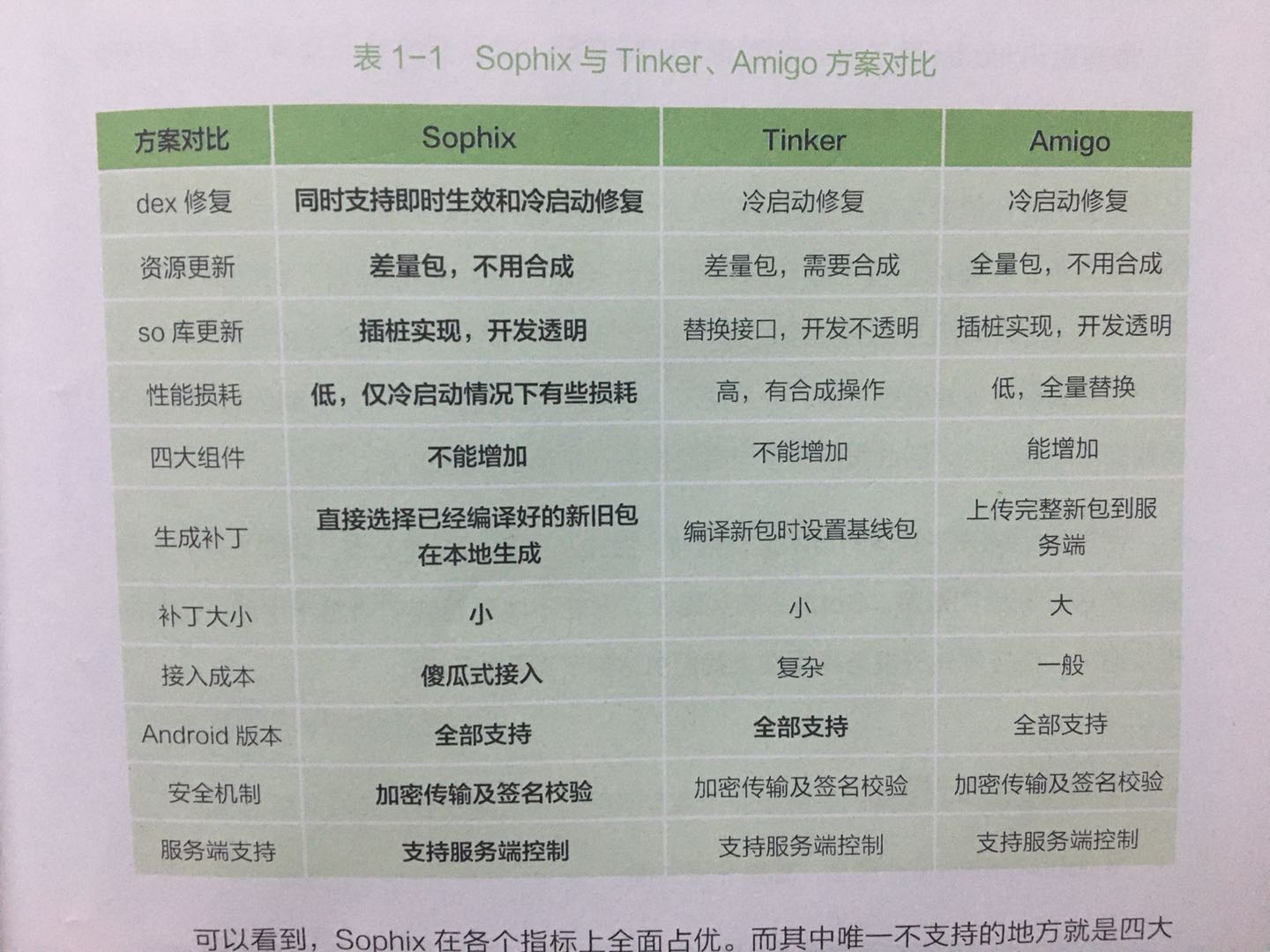
**修复的是什么？**

资源修复，代码修复，SO库修复



目前的主流热修复技术：

Sophix Tinker Amigo



Sophix，Tinker，Amigo对比

**热修复概述**：

1. assets中的文件也是资源，只是这些资源是不带资源ID的原始文件，因此可以直接指定路径来访问这些资源。
2. AndroidManifest出现的BUG是无法修复的，因为它是由系统进行解析的，系统会直接获取安装包里唯一的AndroidManifest文件，在解析过程中不会访问补丁包信息。
3. 如果想要增加四大组件，通常需要预先在安装包的AndroidManifest里边埋入代理的组件，在每次新增组件时，进行偷梁换柱，通过预埋的代理组件实现与系统进程间的通信。
4. 代码的修复，由于所有的Java代码最终都编译为classes.dex格式文件，因此任何的热修复方案，想要改变代码逻辑，都需要在补丁里包含一个新的dex文件。然后再程序运行的时候加载这个dex文件，并且改变执行流，从执行原有安装包里的classes.dex文件引导到执行新的dex文件。
5. 资源的修复，主要需要修改资源包的内容。正常情况下，资源包就是整个apk安装包，如果想要新增一个原有安装包里不存在的资源，就必须修改资源包的内容。所以，就必须想办法把原有安装包替换为新资源包，或者把新的资源包插入程序的查找过程中。
6. so库的修复，在Android系统中，所有的so库都是由System.load进行加载的，因此只要找到办法在加载的时候优先加载补丁包的so库，而不是原有安装包的so库，就能进行完整的底层代码替换了。

代码修复主要有两大方案：

一种是阿里系的底层替换方案，另一种是腾讯系的类加载方案。

a.底层替换方案限制颇多，但时效性最好，加载轻快，立即见效；

b.类加载方案时效性差，需要重新冷启动才能见效，但修复范围广，限制少。

**底层替换方案**：是在已经加载的类中直接替换原有方法，是在原有类的基础上进行修改的，因而无法实现对原有类进行方法和字段的增减，因为这样会破坏原有类的结构；方法和字段的增加或减少，会导致方法和字段的索引发生变化；如果程序运行中某个类突然增加了一个字段，那么对于已经产生这个类的示例，它们还是原有的结构，这是无法改变的。

**类加载方案**：类加载方案的原理是在app重新启动后让ClassLoader去加载新的类。因为在app启动到一半的时候，所有需要发生变更的类已经被加载过了，在Android系统上是无法对一个类进行卸载的，如果不重启，原有的类还在虚拟机中，就无法加载新类。因此，只有在下次启动的时候，在还没有运行到业务逻辑之前抢先加载补丁包中的新类，这样后续访问这个类时，就会被解析为新类。

资源修复基本都参考了Instant Run的实现（需要处理兼容性问题和搜索到AssetManger所有的引用处），简单说，Instant Run中的资源热修复分为两步：

a.构造一个新的AssetManager，并通过反射调用addAssetPath函数，然后把完整的新资源包加载到AssetManager中。这样就得到了一个含有所有新资源的AssetManager。

b.找到所有之前引用原有AssetManager的地方，通过反射，把引用出替换为新的AssetManager。

**so库的修复**： so库的修复本质是对native方法的修复和替换。可以采用类似类修复反射注入的方式。把补丁so库的路径插入到nativeLibraryDirectories数组的最前边。

**热替换代码修复：**

 1.每一个Java方法在Art虚拟机中都对应着一个ArtMethod，ArtMethod记录了这个Java方法的所有信息，包括所属类、访问权限、代码执行地址等；

 2.通过env->FromRefletedMethod，可以由Method对象得到这方法所对应的ArtMethod的真正起始地址，然后就可以把它强转为ArtMethod指针，从而对其包含的内容进行修改；

 3.把旧方法的ArtMethod对象中所有的字段都换为新方法ArtMethod的成员字段后，执行时所有的数据就可以保持和新方法的数据一致。

 4.dex2oat生成的AOT机器码时是有做一些检查和优化的，由于dex2oat编译机器码时确认了两个方法同属于一个类，所以机器码中就不存在权限检查的相关代码了。

 5.加载新类的ClassLoader需要与老的类的ClassLoader一致，通过反射依赖注入即可：

Field classLoaderField = Class.class.getDeclaredField("classLoader");

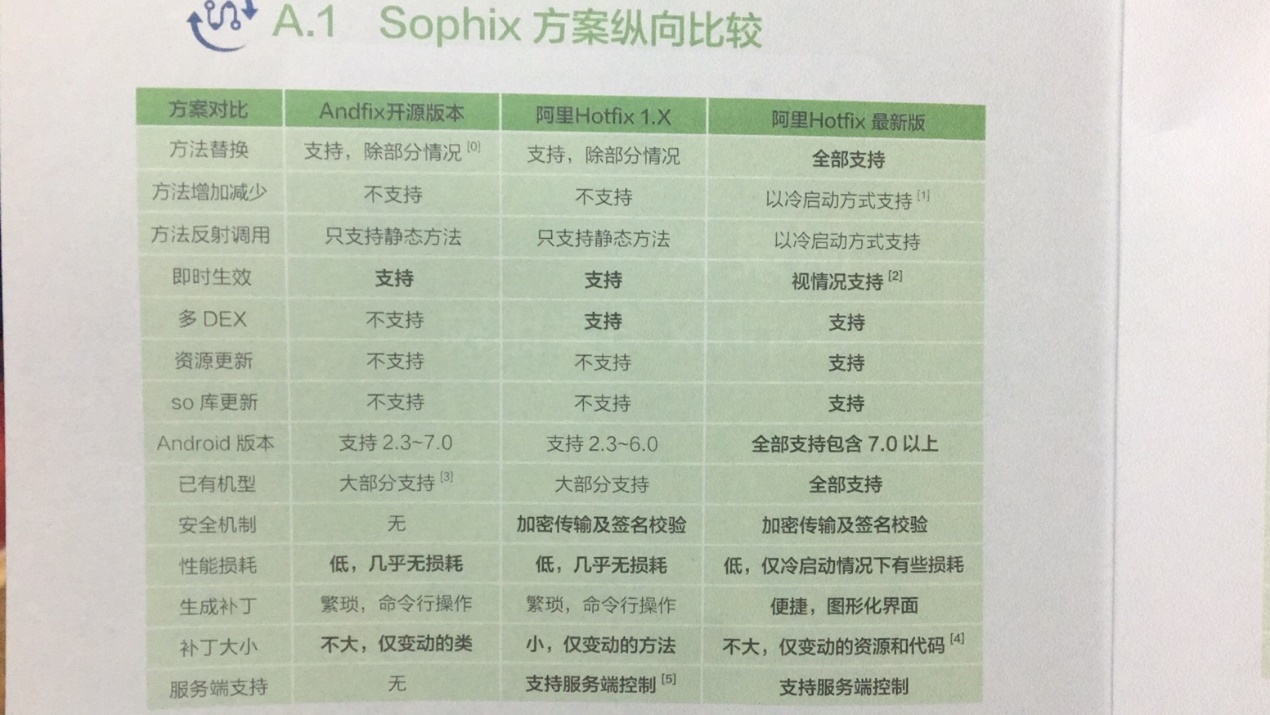
classLoaderField.setAccessible(true);

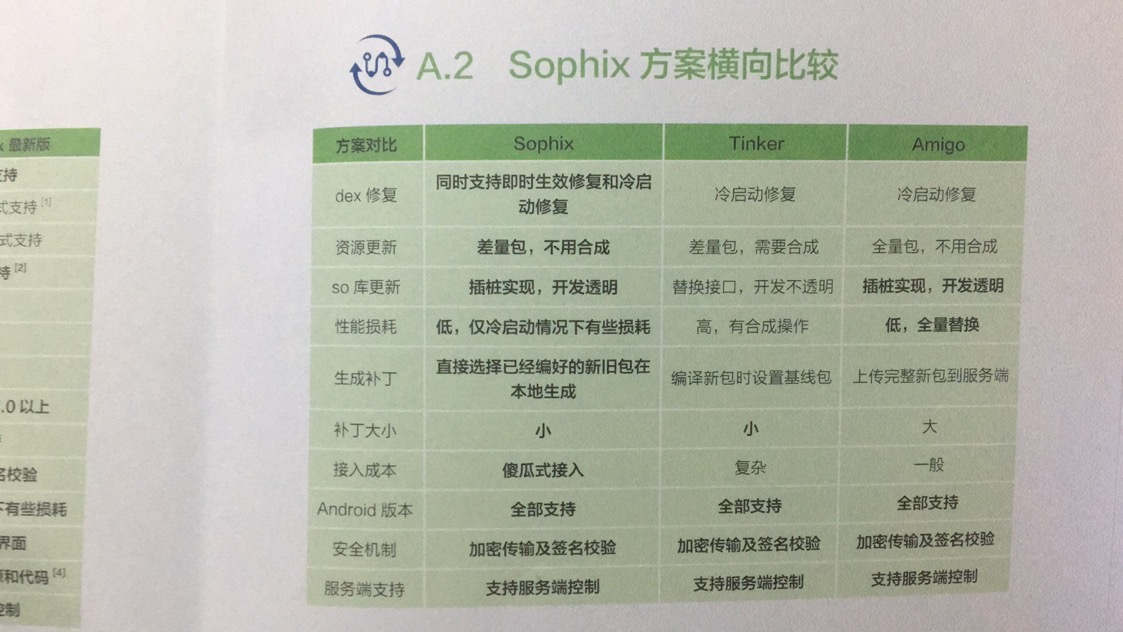
classLoaderField.set(newClass, oldClass.getClassLoader);

反射调用非静态方法时，由于方法所属类被替换成新的类，但是调用这个方法的类的实例可能还是老的类而导致出现非法参数异常。

* o表示Method.invoke传入的第一个参数，也就是作用的对象；
* c表示ArtMethod所属的类型；
* 只有o是c的一个实例才能够通过验证，才能继续执行后面的反射调用流程；

Sophix方案纵横比较：





关于热修复的一点思考：目前针对我们的产品，每次线上有bug出现，用户吐槽修复进度慢，周期长，是否可以引入热修复技术，采用补丁包的方式，达到快速修复，用户无感知升级的效果。