**总结&启发**

本文收集性能问题的方法：选定GitHub上star高的开源相关项目，阅读perf bug report（性能问题报告），收集常见性能问题和反模式。寻找性能相关的报告使用的关键字：“slow”, “performance”, “thread” and “memory”。 手动分析了问题报告中的修复，并记录了代码更改，浏览了代码注释，从而总结出与iOS相关的反模式。如何找出：确定一个关键类（如访问数据存储的NSManagedObjectContext，然后遍历，形成访问图）

**该方法的风险：**

使用keyword来筛选问题报告并人工复核，可能漏掉一些其实与性能有关的报告和模式，同时比较耗费人工。本文仅针对swift语言；本文提到其根据报告总结的一类反模式在实际中检测中没找到，可能时检测方法有误，也可能是借助一些其他工具的更新，该类反模式在成熟软件里已经都修正了。

**文章阅读笔记：**

Abstract：

Question：移动应用程序(即应用程序)中的性能问题通常会直接影响用户体验，但缺乏相关研究。

Method：手动研究了从四个开源iOS应用程序中收集的225个性能问题，发现iOS应用程序中的大多数性能问题都与低效的用户界面设计、内存问题和低效的线程处理有关。还手动发现了在**所研究的问题报告**中重复出现的四种性能反模式。

Answer：实现了一个名为iPerfDetector的静态分析工具，在八个开源和三个商业应用上评估了iPerfDetector。iPerfDetector在研究的应用程序中成功检测到34个性能反模式实例。

Introduction：

为什么研究：移动应用的性能问题通常会对用户产生直接影响，基于Swift开发的iOS的应用市场广阔。

研究现状：相关领域的研究集中于系统应用或Android应用，可能并不适用于IOS应用。由于以前的工具(如Facebook的Expressor(2017)和苹果的Instruments (2018c))专注于检测内存相关的问题，我们实现了iPerfDetector来检测低效的线程处理和低效的用户界面设计。

Research Question：

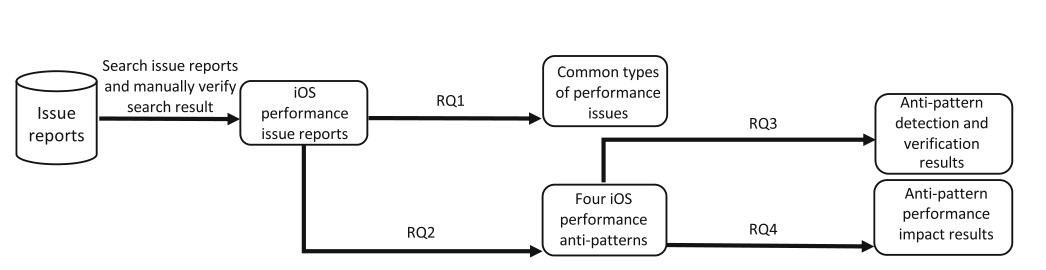
1. 三种常见的性能问题:低效的用户界面设计、内存问题和低效的线程处理。
2. 在用户界面线程上访问数据模型、在后台线程上更新用户界面控件、低效地应用用户界面模糊效果、低效地应用用户界面透明效果以及导致保留周期的对象引用是最常见的反模式。
3. 找到34个反模式，真阳性32个，被开发者接受31个。
4. 本文的案例研究表明，修复反模式可以在GPU使用方面提高5.8%至67%，在CPU使用方面提高34.15%，在响应时间方面提高80%以上。

Background：

Swift比Object-C在当下应用更广泛，iOS应用通常使用模型-视图-控制器(MVC)设计模式开发，将应用程序的数据访问和业务逻辑与视觉呈现分开。模型层是一组独立于用户界面的对象，负责管理数据库中的持久对象；视图层负责显示用户界面组件；控制器层实现应用的业务。

Method:

我们首先手动研究iOS应用程序问题报告，以找到性能问题的常见根本原因(RQ1)。其次，我们提取并表征iOS性能反模式(RQ2)。第三，我们检测商业和开源应用程序中的性能反模式，并与开发人员验证我们的发现(RQ3)。最后，我们研究了反模式(RQ4)的性能影响。



选择研究对象的4个标准：所选的应用程序应该与iOS移动应用程序开发相关；候选应用程序应该使用问题报告系统；候选应用程序应该包含与性能问题相关的问题报告。最后，app要积极维护。

寻找性能相关的报告使用的关键字：“slow”, “performance”, “thread” and “memory”。删除未修复的或关键字引起的误报。

发现20%的问题报告试图解决低效的用户界面设计。有34%与内存问题有关。线程相关问题是我们研究的问题报告中性能问题的最常见原因(36%)。大多数我们研究的问题与开发者使用iOS的特定API有关（如UI效果或线程）

RQ2 ：试图推导出与iOS开发特别相关并多次出现的性能反模式。因此，我排除了其他非iOS和特定于应用程序的性能问题(例如，显示表情符号、添加太多日志、调整数据库查询、切换到更快的数据结构等)。

下文中分类别详细介绍了每个反模式的描述、例子（含伪代码）

许多反模式都与用户定义函数和iOS特定API之间的复杂调用关系有关，如果没有工具的帮助，这些调用关系可能很难找到。

下文中分类介绍了每个反模式要如何定位。

详细介绍了报告问题后，开发人员的反馈。