移动开发总结

- 根据自己移动开发从业经验整理
- 从移动开发的生命周期去做一些分析和总结

赵辛贵

移动产品生命周期

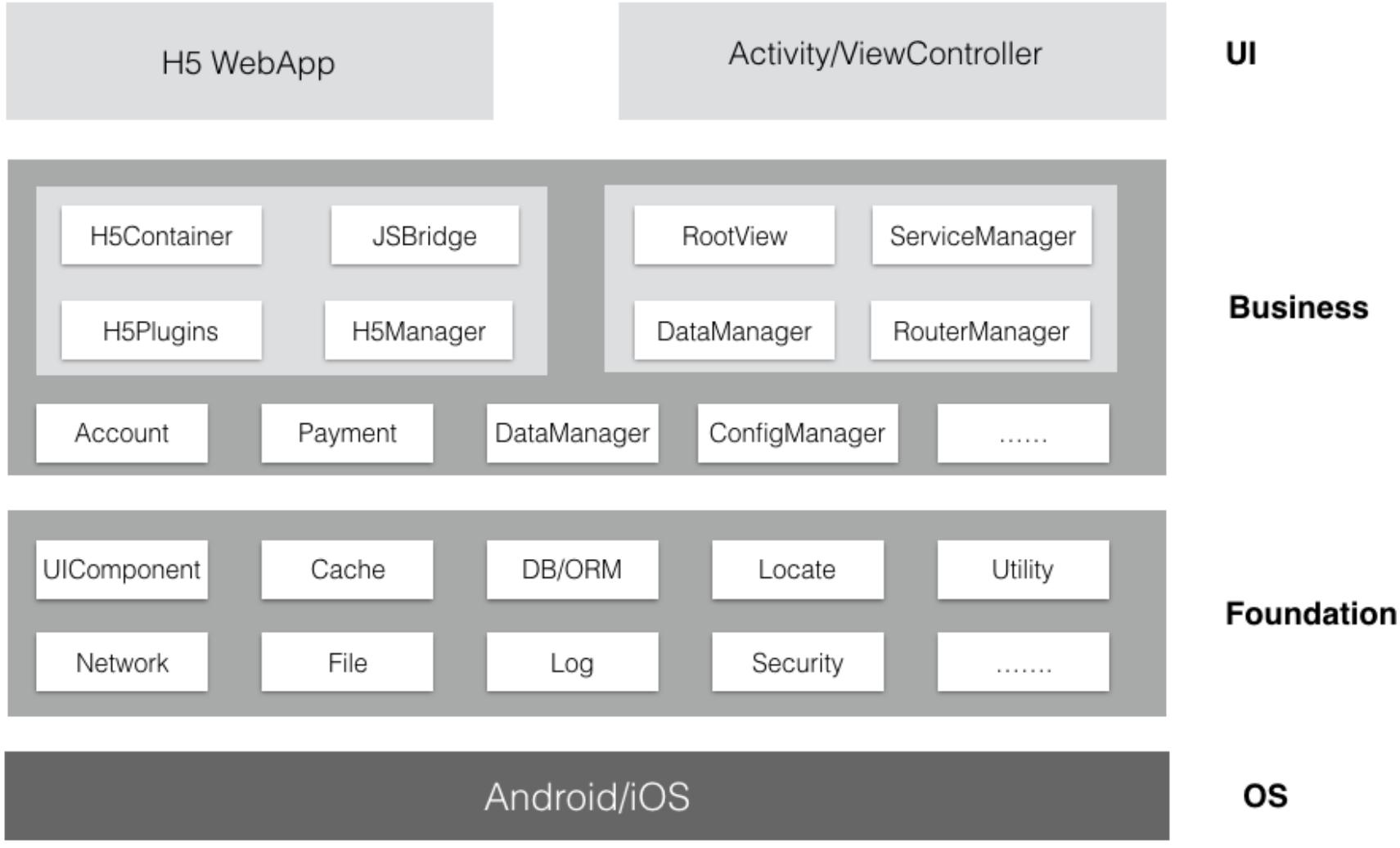
1. 需求
2. 开发
3. 测试
4. 发布
5. 运营

开发

今天主要就开发方面的一些问题进行探讨总结

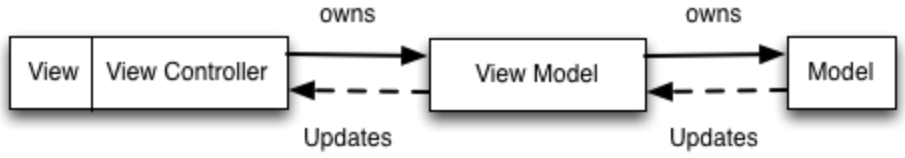
- 框架设计
- 工程解耦
- AppSize
- ABTest
- Crash
- 动态化
- hotfix
- android插件化

框架设计-架构



常见App架构

框架设计-MVVM

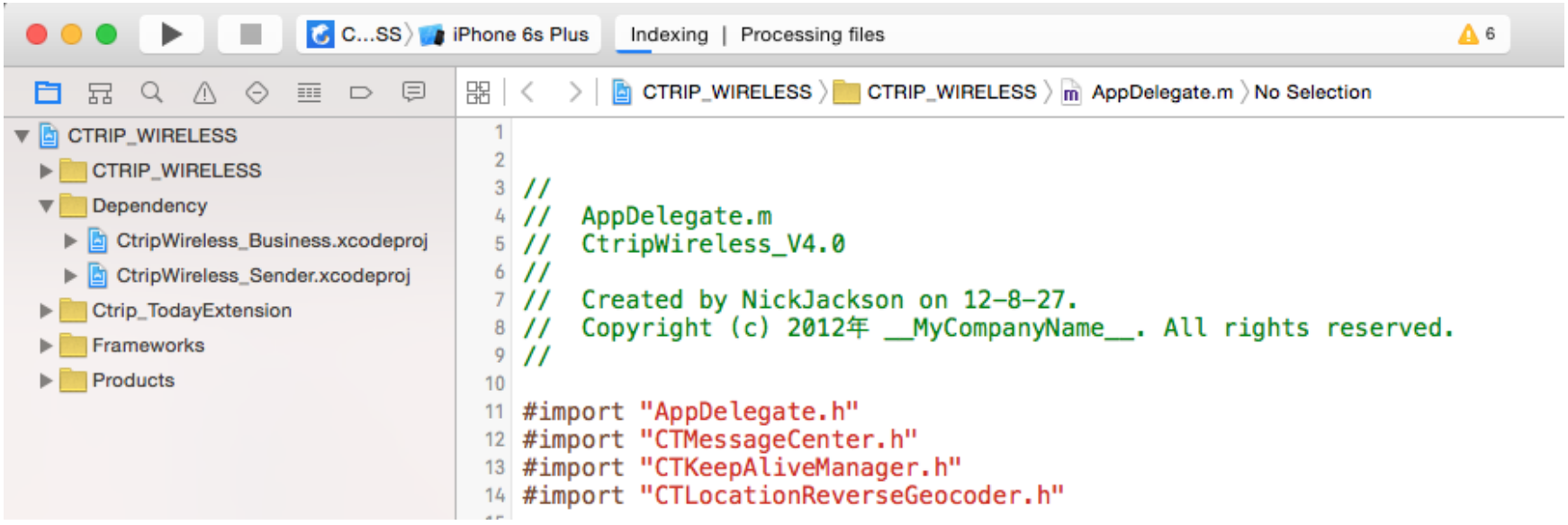


- Model 只负责原始数据模型定义
- ViewModel 负责UI组件的业务逻辑和展示数据管理
- View 负责UI渲染

工程解耦

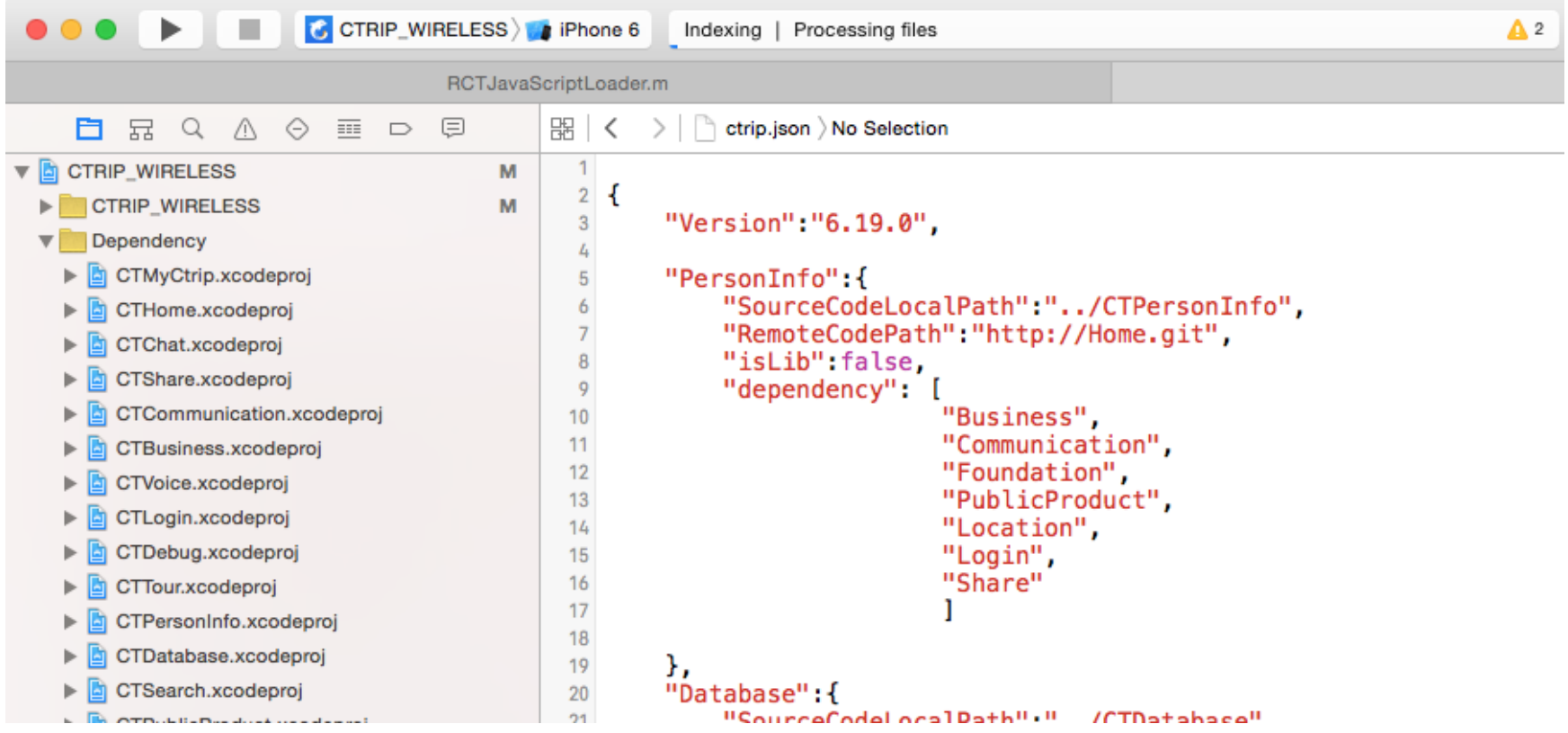
- 伴随业务不断扩张，开发团队不断扩大，工程规模也不断变大
- 这个过程中，工程效率不可避免的遇到瓶颈

工程解耦-整体



- CtripWireless_Business, 公共model，通讯层，DB，公共数据存储
- CtripWireless_Sender, 业务逻辑层，数据部分
- CtripWireless, View层，各个业务界面, 包括通用UI组件

工程解耦-子工程



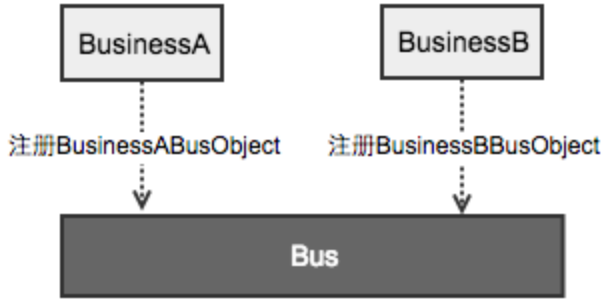
- 按照业务拆分成多个工程
- 抽取公共工程CTBusiness
- 业务工程之间不能相互依赖
- 业务之间有接口调用如何解决？

工程解耦-子工程-资源解耦

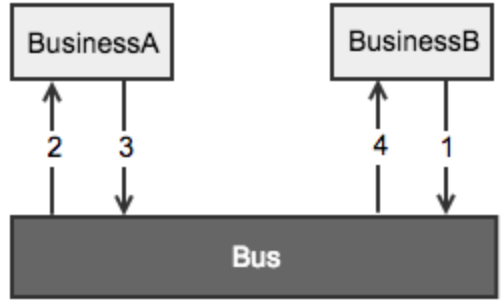
- 子工程独立打包资源bundle，修改代码从bundle中获取资源
- build时，runscript将多个子工程的资源合并到一起

工程解耦-总线

总线设计



1. 注册BusObject到Bus



2. 通过Bus转发跨模块调用

- 通过公共工程CTBusiness中的类，分发业务之间的调用
- 运行时解耦合，尽可能减少编译报错
- 支持数据总线，URL总线，同步/异步

工程解耦-总线实现(1)

总线(Bus)设计

```
@interface CTBus : NSObject
//启动App，反射获取各业务BusObject，注册
+ (void)register:(CTBusObject *)busObj;

//同步数据调用，支持可变长参数
+ (id)callData:(NSString *)buinessName param:(NSObject *)param,

//异步数据调用，支持可变长参数
+ (void)asyncCallData:(NSString *)bizName
               result:(AsyncCallResult)result
                param:(NSObject *)paramDict,... NS_REQUIRES_NIL

//同步URL调用
+ (id)callURL:(NSURL *)url;

//异步URL调用，不建议使用
+ (void)asyncCallURL:(NSURL *)url result:(AsyncCallResult)resul
@end
```

工程解耦-总线实现(2)

业务团队使用

```
//同步调用
id ret = [Bus callData:@"businessA/doJob1" param:obj1, obj2, ni

//异部调用
[Bus asyncCallData:@"businessA/doJob2" result:^(id retObj, NSEr
        //TODO:result is Here
    } param:obj1, obj2, obj3, nil];

//URL总线调用
id ret = [Bus callURL:@"ctrip://businessA/doJob2" param:obj1, c
```

- bizName定义hostName/path, 类似URL规则
- bizName会分发给各个业务的BusObject处理

工程解耦-总线实现(3)

BusObject基类设计，业务团队需Override

```
@interface BusObject : NSObject
- (id)initWithHost:(NSString *)host;

 //同步处理任务，需要子类重载
- (id)doDataJob:(NSString *)businessName
         params:(NSArray *)params;

//异步处理任务，需要子类重载
- (void)doAsyncDataJob:(NSString *)businessName
                params:(NSArray *)params
           resultBlock:(AsyncCallResult)result;

//同步根据URL处理任务，需要子类重载
- (id)doURLJob:(NSURL *)url;

//异步根据URL处理任务，需要子类重载(不建议使用)
- (void)doAsyncURLJob:(NSURL *)url resultBlock:(AsyncCallResult
@end
```

工程解耦-bundle化

- 各个业务工程之后，不通过source code依赖，都依赖中间产物bundle
- iOS中间产物为静态库.a, Android为aar/apk(依赖于我们开发的动态加载框架)
- 公共CTBusiness必须精简稳定，否则底层改动，会影响到上层开发稳定性

工程解耦-bundle化

- bundle存储在maven仓库
- 3个工具，开发工具+build工具+发布平台
- 开发工具需更新到各个业务最新的bundle
- 开发工具需一键切换bundle和source code
- build工具需处理好bundle之间的build依赖

工程解耦-bundle化

bundle配置表

```
{
    "Version":"6.19.0",
    "PersonInfo":{
        "SourceCodeLocalPath":"../CTPersonInfo",
        "RemoteCodePath":"http://Home.git",
        "isLib":false,
        "dependency": [
                        "Business",
                        "Communication",
                        "Foundation",
                        "Login"
                        ]
    }
}
```

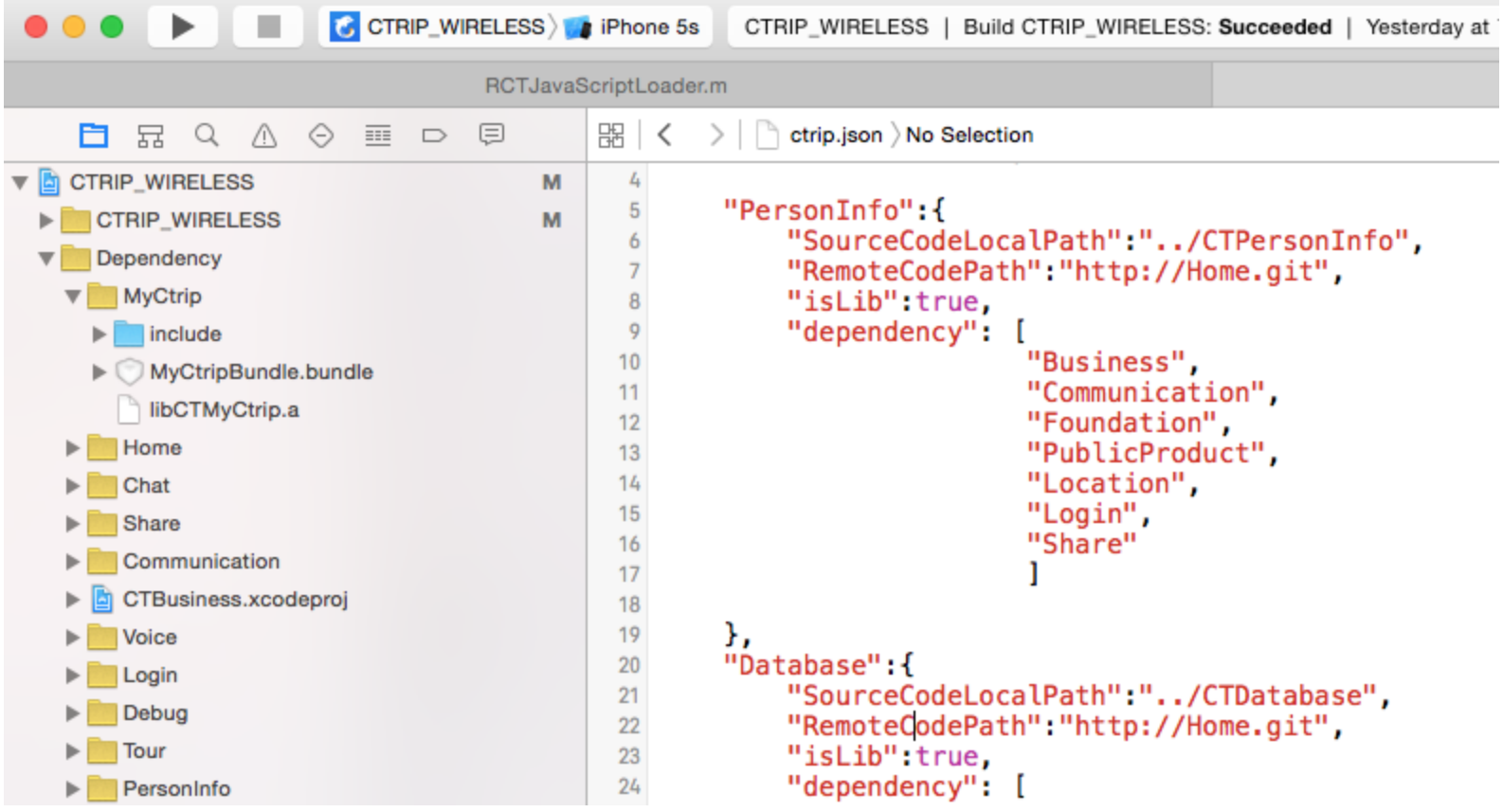
工程解耦-流程优化

流程优化

- 开发人员开发完毕，到打包平台打bundle包
- 测试人员自行到打包平台测试包，验证自己业务的bundle
- 测试验证通过后，将验证通过的bundle发布到RC环境
- 集成阶段，从RC环境拉取最新的bundle打集成包
- 发布到集成环境的bundle，需要自动打包确认该bundle是能提交的

工程解耦

方案实施完成后



工程解耦

方案实施完成后

- 集成打包从原先的10min减少到2min，Mac Pro设备上
- Android 集成打包时间从6min，减少到1.5min，普通linux虚拟机；
- 开发不需要build其它业务的source code，1min之内能build完成，SSD，iMac设备
- 打包平台开放给所有业务团队使用，打包数量比较之前增加200%
- 集成效率提升，发布日发布时间从先前平均晚上12点，提前到6:30

AppSize

- 主要争对iOS
- AppStore有规定，如果AppSize大于100MB，不允许再移动网络下下载
- 随着业务不断增加，size很快就在100MB附近徘徊
- size怎么减？

AppSize

方案

- 开发size计算工具
- 各业务设置size指标
- 全局性的减少size方案

AppSize

size计算工具

- AppStore会对提交的ipa包可执行程序加壳，加壳比例基本稳定
- 计算公式：
  - 假设自己打的ipa包(ipa包实际格式为zip)size为B，其中可执行程序size为B1，资源后size为B2，即B＝B1+B2;
  - 上线之后预估Size为A，则A＝B1*2.14+B2
  - 计算时候，删除一个bundle，打包一次，2次比较，即可计算出当前bundle的size
- 以上述计算方式，开发脚本，自动计算所有业务bundle大小

AppSize

各业务设置size指标

- 以100MB为临界值，预留buffer(比如10MB)，按照业务规模，分配给不同业务
- 惩罚机制，量化size价格

AppSize

- 编译选型
  - 打包选择cpu架构 只选择armv7+arm64(必须)，armv7用于兼容老设备
  - Strip style设置，不要将符号表打进Release包的exe
  - 子工程deploy target version调整，7.0vs6.0的xib优化
- 图片
  - png也需要压缩
  - jpg质量建议0.8
  - 使用svg矢量图替换存色图片(iOS,svg->font)，成本偏高
- 代码
  - 减少冗余代码
  - 部分业务转用H5 Hybrid或者React Native
  - Tips: 如果有使用Marsony这个框架，将头文件中的category实现放到单独.m文件

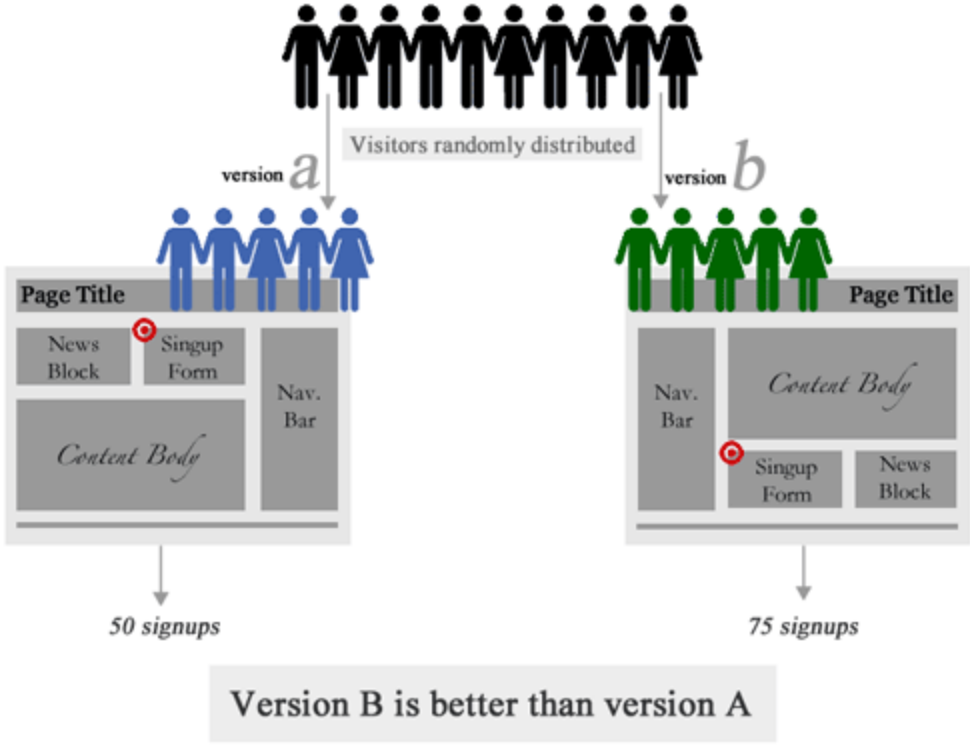
开发-日志系统

- 成熟App必备组件，用户数据分析依赖于此
- 用户行为日志
- 性能数据日志
- 自动埋点日志
- trace/metrics (number value)
- 独立日志系统，和App服务分开
- 日志记录SDK

ABTest

- 作为PM，领导说，首页上查询按钮颜色由灰色换为蓝色，如果不认同，怎么破？
- 开发人员，对于某个核心功能实现方案做交大的调整，但又怕风险，怎么办？
- ABTest，搞定。

ABTest



随机的选取一定比例的用户，访问不同的实验内容，根据转化率比例判断哪个实验更好。

ABTest

模型定义

```
public static class ABTestModel{
        public String expCode =    "";
        public String beginTime =    "";
        public String endTime =    "";
        public String expVersion  = "";
        public JSONObject expAttrs = null;
}
```

- App启动，从服务端拉取所有实验列表
- 调用某个API，如果有做实验，根据实验号，获取实验版本，进入对应逻辑
- 用户行为日志记录用户进入/开启了当前的某个实验

ABTest

后端需提供功能

- API，提供用户查询当前所参与的实验列表
- 实验管理系统，增加，暂停，删除实验
- 分流，切换任意比例的用户，进入某个实验
- 实验结果大数据分析，给出实验某个版本是否显著的结论(订单转换率，停留时差，以及实验所属KPI)

开发-crash

- crash是在App使用过程中非正常终止
- 不同于web开发，在App上是一种正常表现
- crash数据需要监控，收集
- 推荐使用腾讯bugly，免费，功能强大，反馈问题响应及时

开发-crash

- 计算公式
  - crash用户数/使用用户数
  - crash次数/app打开次数<包括前后台切换>
- 合理范围
  - bugly计算采用第一种方式
  - 业界平均0.8%以内

开发-Crash问题解决

- Crash callstack不能定位到问题的，分析用户共同属性
  - 分析crash所在的线程
  - 查看此时其他系统日志
  - 分析机型，系统版本，网络等特征信息
  - 查询用户行为日志
- 最容易解决的是空指针、数组越界问题
- 最难解决的是野指针、多线程同步、过度释放等问题
  - 分析野指针调用的函数，核查调用该函数的对象，保护性修复
  - 逻辑上分析多线程并发，极端情况下的异常，保护性修复
- 系统层的偶发crash
  - Crash callstack全部是系统API
  - 如果数量大，查询日志，分析用户行为，找共同属性
  - 考虑其它方案，规避该API调用

开发-hotfix

- App版本发布流程繁杂，时间长,不像web站点可以随时发布
- 线上App发生大量crash时，下发hotfix可以减少线上故障修复成本
- hotfix实现方案
  - iOS推荐使用jspatch，轻量+稳定
  - Android推荐 https://github.com/dodola/RocooFix
- hotfix发布
  - hotfix脚本传输过程中需要加密
  - hotfix脚本需要签名，加载时，需要校验签名
  - jspatch替换正在执行的函数会导致偶发crash

开发-动态化

- 动态下发可执行程序给App
- 互联网产品迭代频率快，插件化更新发布意义不大
- 插件化在iOS上行不通，受签名限制
- Android插件化可以提高开发，运行效率

开发-Android 动态化

why

- 解决主dex 65535方法数限制
- dex多时，multi-dex主线程dex优化，启动黑屏，容易ANR
- 协同开发效率

开发-Android 动态化

原理

-重载系统资源ID查找

-系统加载打包出的dex pathclassloader/dexclassloader

-修改aapt资源打包工具

开发-Android 插件化-资源加载

都是通过AssetManager类和Resources类来访问的。获取它们的方法位于Context类中。

```
Context.java

/** Return an AssetManager instance for your application's pack
public abstract AssetManager getAssets();

/** Return a Resources instance for your application's package.
public abstract Resources getResources();
```

开发-Android 插件化-资源加载

1.具体的实现在**ContextImpl**类中，我们需要重写这两个抽象方法。
2. 后续**Context**各子类包括**Activity**、**Service**等组件就都可以读取到资源

ContextImpl.java

```
private final Resources mResources;

@Override
public AssetManager getAssets() {
    return getResources().getAssets();
}

@Override
public Resources getResources() {
    return mResources;
}
```

开发-Android 插件化-资源加载

AssetManager有一个隐藏的方法addAssetPath，可以为添加新的资源路径。

```
/**
* Add an additional set of assets to the asset manager.  This c
* either a directory or ZIP file.  Not for use by applications.
* the cookie of the added asset, or 0 on failure.
* {@hide}
*/
public final int addAssetPath(String path) {
    synchronized (this) {
        int res = addAssetPathNative(path);
        makeStringBlocks(mStringBlocks);
        return res;
    }
}
```

开发-Android 插件化-资源打包

修改aapt工具，保证各业务APK资源ID不冲突

```
//android.jar中的资源，其PackageID为0x01
public static final int cancel = 0x01040000;

//用户app中的资源，PackageID总是0x7F
public static final int zip_code = 0x7f090f2e;
```

开发-Android 插件化-类加载

1. patchclassloader，添加我们dex路径即可
2. dex优化，参考multidex实现
3. dex优化 android 5.0 耗时长

```
private static void install(ClassLoader loader, List<File> addi
     File optimizedDirectory)
             throws IllegalArgumentException, IllegalAccessExce
             NoSuchFieldException, InvocationTargetException, N
    /* The patched class loader is expected to be a descendant
    * dalvik.system.BaseDexClassLoader. We modify its
    * dalvik.system.DexPathList pathList field to append additi
    * file entries.
    */
    Field pathListField = findField(loader, "pathList");
    Object dexPathList = pathListField.get(loader);
    expandFieldArray(dexPathList, "dexElements", makeDexElement
         new ArrayList<File>(additionalClassPathEntries), optim
}
```

测试

- UI自动化测试
  - case维护成本偏高
  - 自动化做回归测试比较合适
  - 不建议小团队做UI自动化测试
- 单元测试
  - 核心代码需要单元测试，提升稳定性
- 专项测试
  - 耗电、流量、启动时间、稳定性
  - 可以和monkey测试同步进行

发布

- 所有的发布需要可回滚
- 灰度发布
  - Android选择部分渠道、部分用户灰度升级
  - 后台关注升级上来的用户转化率、crash率等
  - iOS灰度发布效果不佳，testflight不好用，可以考虑部分越狱渠道灰度发布
- 渠道包&动态打包
  - 只支持Android，iOS需要重新打包
  - apk包的/meta/inf/目录可以存放渠道文件，修改/添加时APK包签名不变
- 定制版本管理
  - 独立git分支，不进主版本

Thanks & QA