Weex源码分析系列（七）之Weex SDK可优化细节思考

1、前言

在上一篇文章中我们介绍了Weex SDK源码中可借鉴的细节，那么现在的Weex SDK已经是最优的吗？作为技术RD，我们心中一定要有敬畏：艺无止境，学习的过程中逐渐反思，寻找最优解。那么我们今天就来说说Weex SDK中有哪些可以优化的细节。

2、反射获取方法

大家知道对于Weex来说，JS引擎与Native的交互本质上就是方法的调用，最终调用的时候必然是反射无疑，但是方法的获取也是反射这一点存在很大优化空间。

先回忆下我们使用Module组件的方法：

- 继承WXModule；
- 编写函数体；
- 给函数体打上注解@JSMethod；

而这些被打上注解的函数则可以拿来与Js进行交互，我们回忆下Module注册的代码，分别有Native的注册和Js的注册，注册有一步是获取Module组件中打上注解的方法：

```
@Override
public String[] getMethods() {
  if (mMethodMap == null) {
    generateMethodMap();
  }
  Set<String> keys = mMethodMap.keySet();
  return keys.toArray(new String[keys.size()]);
}

private void generateMethodMap() {
  if(WXEnvironment.isApkDebugable()) {
```

```
      WXLogUtils.d(TAG, "extractMethodNames:" + mClazz.getSimpleName());
    }
    HashMap<String, Invoker> methodMap = new HashMap<>();
    try {
      for (Method method : mClazz.getMethods()) {// 拿到方法
        // iterates all the annotations available in the method
        for (Annotation anno : method.getDeclaredAnnotations()) {// 方法是
否被打上注解
          if (anno != null) {
            if(anno instanceof JSMethod) {
              JSMethod methodAnnotation = (JSMethod) anno;
              String name =
JSMethod.NOT_SET.equals(methodAnnotation.alias())?
method.getName():methodAnnotation.alias();
              methodMap.put(name, new MethodInvoker(method,
methodAnnotation.uiThread()));// 封装成MethodInvoker对象
              break;
            }else if(anno instanceof WXModuleAnno) {
              WXModuleAnno methodAnnotation = (WXModuleAnno)anno;
              methodMap.put(method.getName(), new
MethodInvoker(method,methodAnnotation.runOnUIThread()));
              break;
            }
          }
        }
      }
    } catch (Throwable e) {
      WXLogUtils.e("[WXModuleManager] extractMethodNames:", e);
    }
    mMethodMap = methodMap;
  }
```

可以看到，Module注册的过程必定是相对耗时的，而Module越多时间也越长，应用启动阶段注册的话尤为明显。而对于另一个组件Component基本也是一样的，只不过多了个注解@Component：

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Component {
  boolean lazyload() default true;
}
```

区别在于：对于Native注册，会有懒加载的判断，不过效果一般，因为Js端注册前已经生成一次了：

```
  @Override
  public void loadIfNonLazy() {
    Annotation[] annotations = mClz.getDeclaredAnnotations();
    for (Annotation annotation :
      annotations) {
      if (annotation instanceof Component){
        // 懒加载则暂时跨过这步
        if(!((Component) annotation).lazyload() && mMethodInvokers ==
null){
          generate();
        }
        return;
      }
    }
  }
```

那么反射消耗性能加上耗时的缺点应该如何解决呢？这里提供两条解决途径供参考：

- **对Weex进行异步初始化，绝大多数应用使用Weex不会是整个App都由Weex完成，那么只要能保证Weex的初始化在使用之前完成即可；这点很容易做到毕竟App都有闪屏的时间可以利用；**
- **参考EventBus不同版本的改进，我们也可以将Weex的运行时注解改为编译时注解，这样就将在运行时的反射工作挪换到编译时，这种方式显然更好，也不需要再进行异步初始化；**

3、适配的问题

对于Weex，它默认的将显示的宽度设置为750px作为适配的标准。

```
  @Deprecated
  public static float getRealPxByWidth(float pxValue) {
     return getRealPxByWidth(pxValue,750);
  }

  public static float getRealPxByWidth(float pxValue,int customViewport) {
    if (Float.isNaN(pxValue)) {
      return pxValue;
    }
    if (mUseWebPx) {
      return (float) Math.rint(pxValue);
    } else {
      float realPx = (pxValue * getScreenWidth() / customViewport);
```

```
        return realPx > 0.005 && realPx < 1 ? 1 : (float) Math.rint(realPx);
      }
    }
```

那提一个开发中的细节问题：我怎么知道需要在Vue的布局代码中写多少px呢？如果UI同学给出的不是750标准就需要自己使用公尺去计算。

解决思路：

- **规范使用统一的页面适配保证比如出图按照750或者720（修改Weex适配的代码）；**
- **修改Weex的webpack-loader，还是使用类如Android原生dp一样的布局单位（需要前端同学配合写个工具）；**

4、基础能力不够好

这个之前在Module的源码解析中其实提到过，比如网络请求的能力，不管是线程池的使用还是对缓存、协议等的支持都很一般。类如别的一些基础能力比如Stroage等模块也是一样。

不过这条属于苛责，之前也总结过对于Weex来说我们实际上只需要保存其Js引擎与Native的交互能力即可，别的都属于Weex为了吸引开发者而做的简略能力。现实的要求是把这些代码去掉，自己桥接原生原有的能力，缩减Apk的方法数。

5、其它

- **CSS的支持度不够完善，一些效果在Web上的显示和Native上不一样，这要求调试尽量在Native；**
- **关于文档，不可否认Weex的思路和SDK代码都非常优秀，但是文档就相对一般了，有些属于错误的；有些属于过于简单，比如对网络请求，没有Post请求的示例，对Component的自定义，没有ViewGroup的示例，有点避重就轻的嫌疑；**

欢迎持续关注Weex源码分析项目：Weex-Analysis-Project