集成构建优化之导入第三方应用集成

1 背景

在手表的构建中，目前有两个平台，分别是mips和arm，并且由于构建及历史遗留的原因，导致会签的app会提交到5个不同的git库中，包括中兴的dev，master & DR分支以及inwatch的dev & master。

那么，如果类似平衡球应用的更新就会出现需要同时提交到5个不同的分支上面，同时因为需要考虑两个处理器架构，常常就是顾此失彼，漏洞百出，严重影响了集成的效率。

以平衡球V16版本的更新为例，所有的分支得到正确更新之前，出现了6次的失误，包括合入错误的so库，漏掉so库等，表现在平台上面就出现了启动平衡球的时候出现无法正常刷出画面以及crash的情况。

而如果随着目标机型的增加，集成的工程量会成倍地扩大，带来的构建成功率将会变得非常底，因此急需要一个便捷、高效的集成方式来应对当前混乱的局面。

2 原因分析

2.1 当前第三方应用的集成模式

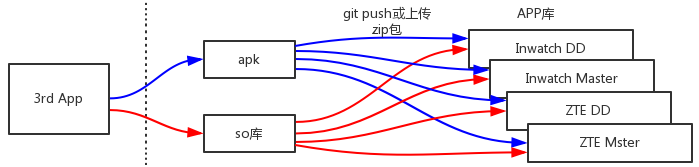
当前第三方应用的集成模式主要有两种：

1. 直接在对应的git库中建立对应的目录，将该版本的apk拷贝到该目录下面，然后更改Android.mk的属性，再手动将整个目录push到对应的git库中。
2. 基于rci集成平台，在系统管理的集成配置中，选择对应的机型，然后在“合入应用”中新增加一列用来告知该应用需要会签合入。接着，启动机型流程，通过手动上传zip包来完成apk的合入。

而对于apk使用的so库，则也有两种方式：

1. 直接在目录下面的Android.mk中进行关联配置；
2. 使用PRODUCT\_COPY\_FILE实现so直接拷贝；

综合起来，当前的第三方应用构建模式框图如下：



2.2 当前模式存在的不足

当前的第三方应用的集成模式存在以下几点关键障碍，验证影响集成效率：

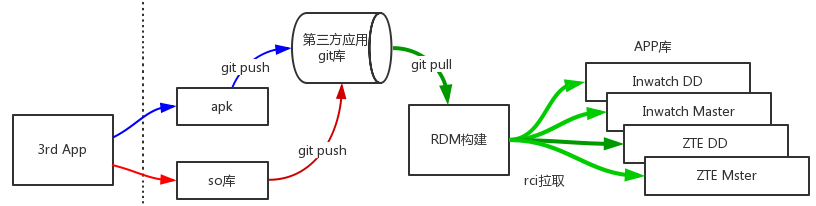
1. 应用导入到新机型中步骤繁琐，包括Android.mk的编辑，上传apk，上传so库，so库合入方式等；
2. 更新工作步骤较多，包括更新Android.mk，上传apk，更新so库等，增加出错的概率；
3. 需要更新到多个分支。重复性的劳动随着机型的增加而成倍加重，并由于引发的漏传，架构错误，版本不匹配等问题层出不穷，效率低下；
4. Apk合入及so库的集成方式多样，导致了不同分支的集成方式存在差异，给更新过程增加了难度。

因此，急需一个规范、简洁、可方便操作的第三方应用集成方式，把集成员从重复的劳动中解放出来。

3 解决方案

针对以上分析，结合目前rci，rdm平台已有的功能，提出了一种通过rdm构建导入第三方应用，并通过rci平台完成集成的构建方式，大大降低了第三方应用合入到新机型的工作步骤以及应用本身版本更新的工作量，提高了工作效率，降低了出错的概率。

该构建方式框架图如下：



同时，辅助编译结束之前的so库检查机制，通过自动比较合入到system/lib下面so库与apk中包含的so库的md5码，进一步验证so库和app应用的匹配，从集成的角度保证了版本集成的稳定性。

整个解决方案的关键在于如何实现第三方应用的rdm构建。为此，我编写了一个脚本：wsd\_build\_qrom\_app\_3rd.sh用来在rdm上面实现第三方app的构建，存放路径在：<http://tc-svn.tencent.com/mqq/mqq_wsrdplat_rep/NANJI_ROM_proj/trunk/QROM/dailybuild>。同时，为了能够匹配该脚本，需要在rdm的自定义参数中，完成如下变量的定义：

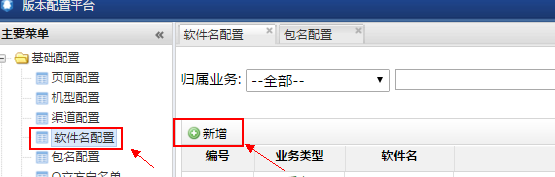
* SR\_GIT\_ADDRESS --- 对应app存放的git地址库
* SR\_GIT\_BRANCH --- 对应app存放的git地址库分支
* APP\_NAME ---对应app的应用名称

4 具体实施

4.1 新应用导入步骤

这里面我们以平衡球Labyrinth为例

1. 在v.rom.wsd.com中配置软件名和包名
2. 软件名配置
3. 选择新增



1. 填写信息



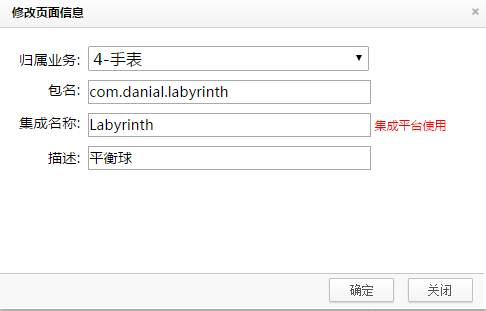
注意：软件名的命名规则为ADR+\*\*\*+TWS（其中最后一个字段TWS是根据归属业务来定的）

具体可以参考已有的软件配置信息。

1. 包名配置
2. 选择新增



1. 填写信息



1. 上传app到git库上面

目前，已经为类似平衡球的第三方app创建了一个git库地址：

<http://git.code.oa.com/wear/3rd_app.git>

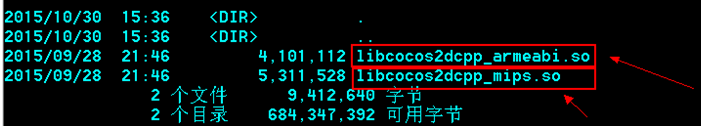
当拿到一个新版本的apk之后，需要根据app命名规则对取到的apk进行重新命名，比如：Labyrinth\_1.0.150929\_16\_release\_internal.apk

命名组合方式：应用名\_主版本号.次版本号.日期\_编译号\_release\_internal.apk

同时，也需要准备so库zip包，规范命名类似为：Labyrinth\_1.0.150929\_16\_so.zip

命名组合方式为：应用名\_主版本号.次版本号.日期\_编译号\_so.zip

因为目前rci会签时候“上传zip”会将文件夹略去，这样给以文件夹进行区分芯片架构的so库没有办法拉取进行自动会签。因此，暂时对于so的命名做了规范，libname\_ICArch.so。比如，



1. rdm构建

基于rdm 3.0的app构建可以参考文章：

<http://km.oa.com/group/22595/articles/show/240381>

这里面不具体对步骤进行说明。只是对于配置参数进行说明，还是以平衡球Labyrinth为例：

1. 工程路径及编译脚本

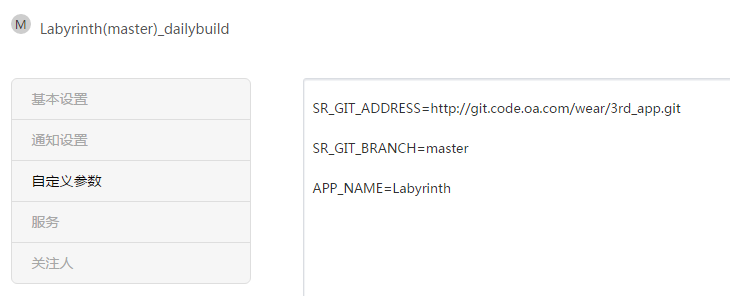
工程路径请选择：<http://tc-svn.tencent.com/mqq/mqq_wsrdplat_rep/NANJI_ROM_proj/trunk/QROM/dailybuild>

编译脚本为：wsd\_build\_qrom\_app\_3rd.sh



1. 自定义参数

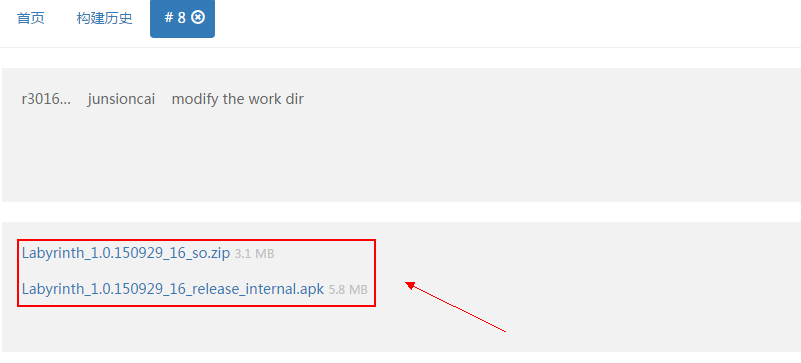
完成自定义参数的配置：



C.启动RDM构建



再次确认构建的结果是否满足预期：

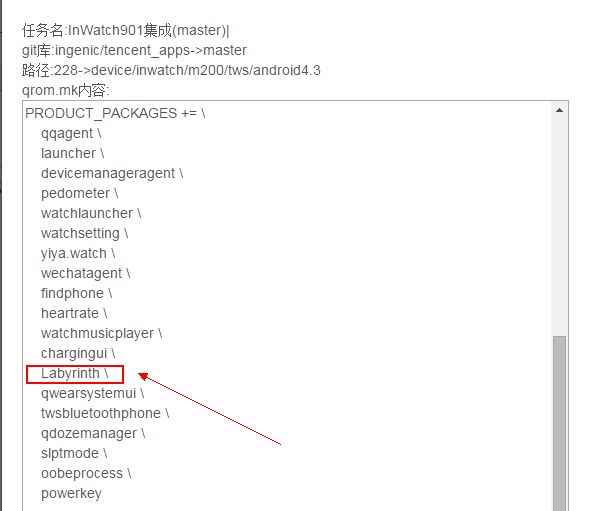


1. 合入应用

在rci平台的“系统管理”的“集成配置”中选择一个合入机型，在“合入应用”中增加平衡球的选项：



然后在qrom.mk中将Labyrinth添加到qrom.mk中。



这样整个过程就已经完成导入。在机型会签流程开始后，就可以进行会签工作。

4.2 新版本导入步骤

我们还是以平衡球Labyrinth为例。

1. 上传app到git库上面

目前，已经为类似平衡球的第三方app创建了一个git库地址：

<http://git.code.oa.com/wear/3rd_app.git>

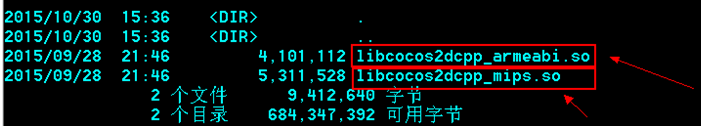
当拿到一个新版本的apk之后，需要根据app命名规则对取到的apk进行重新命名，比如：Labyrinth\_1.0.151102\_17\_release\_internal.apk

命名组合方式：应用名\_主版本号.次版本号.日期\_编译号\_release\_internal.apk

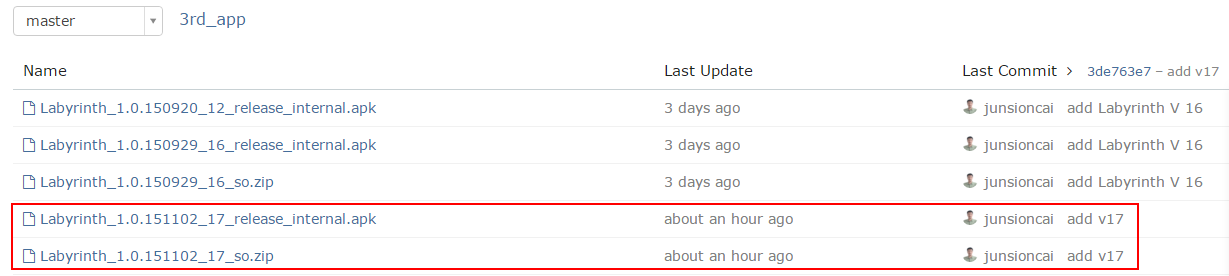
同时，也需要准备so库zip包，规范命名类似为：Labyrinth\_1.0.151002\_17\_so.zip

命名组合方式为：应用名\_主版本号.次版本号.日期\_编译号\_so.zip

因为目前rci会签时候“上传zip”会将文件夹略去，这样给以文件夹进行区分芯片架构的so库没有办法拉取进行自动会签。因此，暂时对于so的命名做了规范，libname\_ICArch.so。比如，



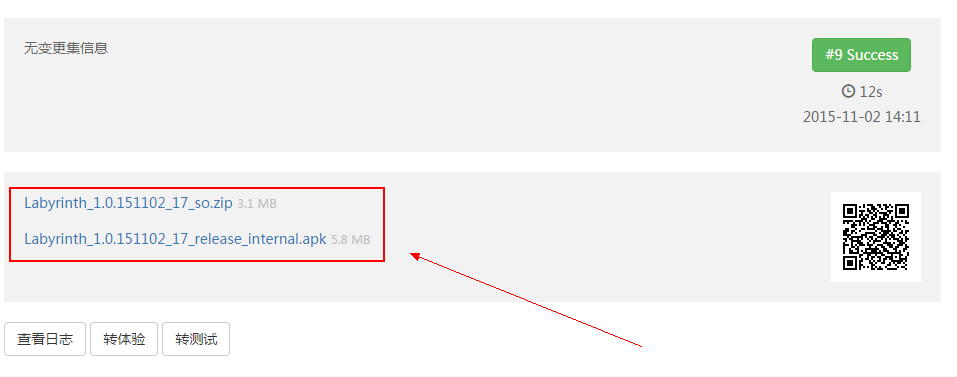
提交完成之后，可以在对应的git库中看到相应的提交文件：



1. 启动RDM构建



再次确认构建的结果是否满足预期：



4.3 so校验机制

与此同时，也提供了一个最后校验的机制，可以检验合入的apk和对应的so库是否可以匹配。

当然需要进行额外的配置，针对于反编译机型，以zte为例需要在机型目录的config.sh中添加如下的代码：

|  |
| --- |
| NEED\_APP\_SO\_CHECK=true  CHECK\_APP\_NAME=Labyrinth  CHECK\_SO\_NAME=libcocos2dcpp  IC\_ARCH=armeabi  APP\_OUT\_DIR=priv-app  export NEED\_APP\_SO\_CHECK  export CHECK\_APP\_NAME  export CHECK\_SO\_NAME  export IC\_ARCH  export APP\_OUT\_DIR |

启动编译之后，如果校验成功的话，就可以在编译日志中看到如下的信息：

|  |
| --- |
| +++ md5sum /home/wsd/QROM\_RE\_BUILD\_WORKSPACE/DQROM/out/P802W10-5.1.1/release/SYSTEM/priv-app/lib/armeabi/libcocos2dcpp.so  +++ awk '{print $1}'  ++ app\_so\_md5=139c67be8184fe851aadafa701c97ff0  +++ md5sum /home/wsd/QROM\_RE\_BUILD\_WORKSPACE/DQROM/out/P802W10-5.1.1/release/SYSTEM/lib/libcocos2dcpp.so  +++ awk '{print $1}'  ++ lib\_so\_md5=139c67be8184fe851aadafa701c97ff0  ++ echo 'libcocos2dcpp.so in Labyrinth.apk md5 value :139c67be8184fe851aadafa701c97ff0'  libcocos2dcpp.so in Labyrinth.apk md5 value :139c67be8184fe851aadafa701c97ff0  ++ echo 'libcocos2dcpp.so in system/lib md5 value :139c67be8184fe851aadafa701c97ff0'  libcocos2dcpp.so in system/lib md5 value :139c67be8184fe851aadafa701c97ff0  ++ [[ 139c67be8184fe851aadafa701c97ff0 == 139c67be8184fe851aadafa701c97ff0 ]]  ++ echo 'Same libcocos2dcpp.so of Labyrinth.apk'  Same libcocos2dcpp.so of Labyrinth.apk |

而对应源码机型的校验，比如inwatch，我们就把这些参数的设定放在rdm构建自定义参数中。



5 优化总结

5.1 效果分析

以目前inwatch & zte两个机型，5个构建分支来看，根据上文第4节中的步骤来统计：

对应新应用导入的步骤，优化前每个分支都需要进行“上传app到git库”及“合入应用”两个步骤，这样就是10个步骤，这整个过程加上最后的验证确认，大概需要8个小时时间。

而优化后，“配置软件名和包名”，“rdm构建”以及“上传app到git库”都指需要一次完成，“合入应用”需要按机型分开配置。这样算下来，所需要的总共步骤为8个，所需要时间估计为2个小时

对于版本更新的情况，优化后的效果更加明显。只需要执行“上传app到git库”及启动rdm构建就可以了，需要时间最多不超过0.5小时。而原先的方案，则需要考虑到不同分支的配合情况。完成整个过程的验证，至少需要4个小时时间。

同时优化前的各个分支是独立的，那么对应分支的修改理论上面都存在出错的可能性。而新方式所有的分支都基于同一个修改。通过一个构建就可以完成版本的验证。同时，由于添加了so库的校验机制，至少保证可能的出错在编译阶段就可以暴露出来，提高了构建版本的成功率。

数据总结如下：

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 优化前 | 优化后 |
| 新app | 步骤 | 10 | 8 |
| 耗时 | 8小时 | 2小时 |
| 版本更新 | 步骤 | 4 | 2 |
| 耗时 | 4小时 | 0.5小时 |

而如果考虑到导入新机型后，原来集成方式的工作量将成倍的增加，就可以明显看到优化的效果。

5.2 遗留工作

由于目前rci在拉取zip包的时候，会将目录略去，这样就导致基于目录区分架构的so库会出现不匹配的情况。目前暂时的解决方案是手动在so库上加上芯片架构的标识。

后续这部分得到改善的话，上文中“上传app到git库”中的步骤将更加简单。