SVN 编译结果 RTX 自动实时通知方案的实现

By Long.Luo

Revision History

版本历史

Version 版本	Editing Note 修订历史	Reviser/Date 修订人/日期	Confirm/Date 确认/日期	Approval/ Date 批准/日期	Implement Date 实施/日期
1.0	创建	罗龙	08/18/2012		
2.0	修改完善	罗龙	08/21/2012		

Table Of Contents

一、概述

目前 12021/12015 项目深圳的编译及版本发布工作是由本人负责。但是由于没有编译结果实时通知机制，编译结果都需要进入编译服务器中去查看最新的编译 log 才能知道，十分不方便，而且编译出错了提交的同事也不知道。考虑到大家都安装了 RTX，通过调用 RTX 的接口将编译结果通过 RTX 消息实时发送给个人，从而大大提高编译出错时的解决效率。

本文对 **SVN 的编译流程、服务器架构、脚本、手动编译、编译出错处理**进行详细的介绍，作为经验传承。

1.1 SVN 编译流程

SVN 的编译流程图如下图 1.1 所示：

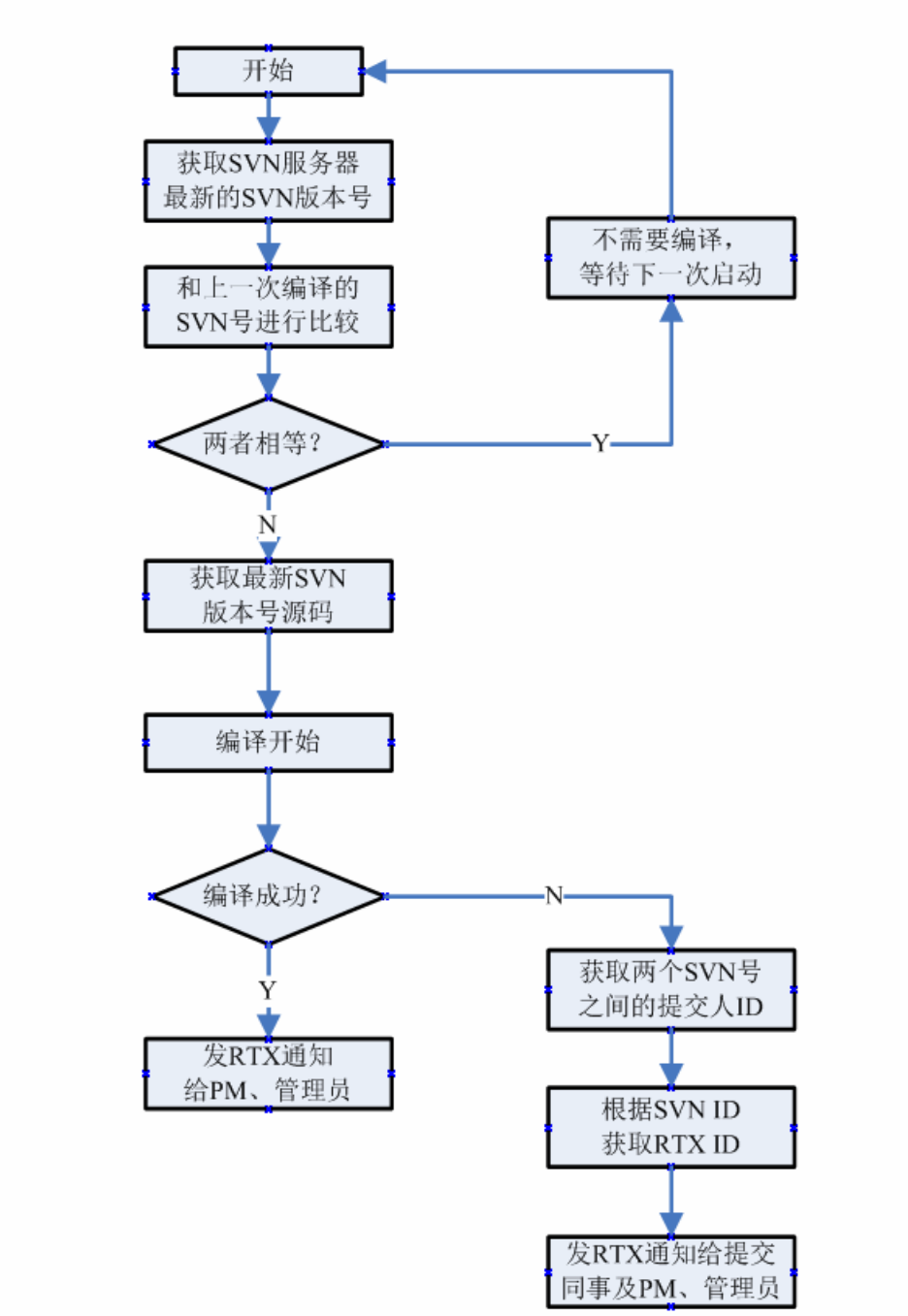


图 1.1 编译流程图

由于编译服务器一天编译次数比较多，RTX 通知过多会影响到大家的工作，因此只是在编译出错才会通知有提交的同事。没有消息就是好消息。**因此如果你没有收到编译错误通知，说明编译是 OK 的**。

二、编译架构

2.1 服务器目录结构

下面以 12021 编译服务器为例说明 SVN 编译服务器文件架构，见表 2.1 所示：

表 2.1 12021 编译服务目录结构及说明

12021 SVN 编译服务器目录结构	说明
12021/	
├── OUT	所有错误的记录会保存在编译服务器此目录下，会保留一个月
│?? ├── 12021.12015 编译错误汇总.xls	编译错误记录表
│?? ├── 2012-08-16-15_SVN4078_ERR	
│?? ├── 2012-08-17-10_SVN4177_ERR	
│?? │?? ├── oppo77_12021_android.log	编译错误 log
│?? │?? ├── oppo77_12021_android.log_err	
│?? │?? ├─ oppo77_12021_AppAssets_Overlay.log	
│?? │?? ├── oppo77_12021_check-dep.log	
│?? │?? ├── oppo77_12021_check-modem.log	
│?? │?? ├── oppo77_12021_codegen.log	
│?? │?? ├── oppo77_12021_custgen.log	
│?? │?? ├── oppo77_12021_drvgen.log	
│?? │?? ├── oppo77_12021_imejavaoptgen.log	
│?? │?? ├── oppo77_12021_javaoptgen.log	
│?? │?? ├── oppo77_12021_kernel.log	
│?? │?? ├── oppo77_12021_nandgen.log	
│?? │?? ├── oppo77_12021_preloader.log	
│?? │?? ├── oppo77_12021_ptgen.log	
│?? │?? ├── oppo77_12021_sign-modem.log	
│?? │?? ├── oppo77_12021_uboot.log	
│?? │?? └── oppo77_12021_update-api.log	
│?? │?? └── TempErrorLog.log	从此次编译错误 log 中提取出错信息并存放在这个文件中，会拷贝到发布服务器中
│?? ├── dailyBuild.log	历次编译所产生的 log 信息
│?? ├── TempErrorLog.log	存放的是此次编译从错误 log 中提取的错误信息，下次编译执行会删除此文件
│?? └── TotalErrorLog.log	存放的是历次编译从错误 log 中提取的错误信息
├── script	脚本文件夹
│?? ├── 12021_compile_daily.sh	正式版本编译发布脚本
│?? ├── 12021_compile_daily_temp.sh	临时版本编译发布脚本
│?? ├── clear.sh	清除脚本，一个月执行一次

?? ├── find_str_argv.pl	从错误 log 中寻找出错的行数并输出到 TempErrorLog.log 文件中的 perl 脚本
?? ├── log	
?? ?? └── 12021_info.log	存放上次编译的 SVN 版本号
?? ├── rtx_send_msg.py	发送 RTX 消息的 python 脚本
?? ├── svn_commit_history_temp.map	存放的这次编译和上次编译之间所有提交者的 SVN id 和版本号映射表，在下一次执行脚本之后会清除掉
?? ├── svn_commit_history_total.map	所有编译的提交者的 SVN id 和版本号
?? ├── svn_id_2_rtx_id.map	SVN ID 与 RTX ID 的映射表
├── SRC	存放的从 SVN 服务器获得的代码
?? ├── CheckSoftwareInfo_temp.java	
?? ├── development	
?? ?? ├── abi	
?? ?? ├── auto_sync_android.log	
?? ?? ├── auto_sync_kernel.log	
?? ?? ├── auto_sync_oppo.log	
?? ?? ├── bionic	
?? ?? ├── bootable	
?? ?? ├── build	
?? ?? ├── checkenv.log	
?? ?? ├── cloneProject	
?? ?? ├── cts	
?? ?? ├── dalvik	
?? ?? ├── development	
?? ?? ├── device	
?? ?? ├── docs	
?? ?? ├── external	
?? ?? ├── frameworks	
?? ?? ├── hardware	
?? ?? ├── kernel	
?? ?? ├── libcore	
?? ?? ├── Makefile	
?? ?? ├── makeMtk	
?? ?? ├── makeMtk.ini	
?? ?? ├── mbldenv.sh	
?? ?? ├── mediatek	
?? ?? ├── mk -> makeMtk	
?? ?? ├── ndk	
?? ?? ├── oppo	
?? ?? ├── out	
?? ?? ├── packages	
?? ?? ├── pack_bootimage.sh	
?? ?? ├── pack_systemimage.sh	

│?? │?? ├── prebuilt	
│?? │?? ├── report	
│?? │?? ├── sdk	
│?? │?? ├── system	
│?? │?? └── vendor	
│?? ├── DeviceInfoSettings_temp1.java	
│?? └── ProjectConfig_temp.mk	

三、脚本详细说明

目前一天中 6, 9, 12, 15, 18, 21 这 6 个时间会各编译一个正式版本，打包烧写 image 包及 out 目录包并拷贝到发布服务器；而 7:15，8:15，10:15, 11:15，13:15，14:15，16:15，17:15，19:15, 20:15, 22:15，23:15 这几个时刻会编译临时版本，不过临时版本只是用来发现提交错误，并不会打包 image 包及 out 目录包。

3.1 编译脚本

目前编译脚本有 2 个，分别为 12021_compile_daily.sh（编译正式版本）与 12021_compile_daily_temp.sh（编译临时版本）。正式版本比临时版本只是多了打包和发布到发布服务器的 2 个过程，其他一致。

编译脚本是通过 Linux 下的 crontab 来进行周期启动的，**如果编译正常进行的话，正式版本编译耗时 58 分钟左右，临时版本耗时 42 分钟左右。**

3.1.1 Shell 编译脚本

项目的编译脚本是使用 Linux Shell 编写的，图 1.1 已经给出了编译执行的流程图，其中几个关键部分的源码如下所示：

- **获取 SVN 提交者 ID**

使用 svn info 命令之后使用 awk 得到提交者 ID 信息。

```
fn_get_svn_committer_id()
{
    echo "  fn_get_svn_committer_id..."

    SVN_VERSION_TEMP=${SVN_VERSION_OLD}
    echo "SVN_VERSION_TEMP=${SVN_VERSION_TEMP}"

    while [ ${SVN_VERSION_TEMP} -le ${SVN_VERSION_NOW} ]
    do
        SVN_COMMITTER=`svn info ${BASE_URL} -r ${SVN_VERSION_TEMP} | awk '/Last.Changed.Author:/{print $4}'`

        echo "${SVN_VERSION_TEMP} ${SVN_COMMITTER} " >> ${SVN_COMMIT_HISTORY_TEMP}
        echo "${SVN_VERSION_TEMP} ${SVN_COMMITTER} " >> ${SVN_COMMIT_HISTORY_TOTAL}

        SVN_VERSION_TEMP=`expr ${SVN_VERSION_TEMP} + 1`
        EXIT=$?
        check_exit
    done
}
```

- **根据获取的 SVN ID 得到 RTX ID**

获得 SVN ID 之后，根据 ID 名称和数据库文件里面找到对应的 RTX ID。

```
fn_get_rtx_id_by_svn_id()
{
    echo "  fn_get_rtx_id_by_svn_id..."

    SVN_COMMITTER_ID_ARRAY=(`cat ${SVN_COMMIT_HISTORY_TEMP}`)
    SVN_COMMITTER_ID_ARRAY_LENGTH=${#SVN_COMMITTER_ID_ARRAY[@]}

    for ((i=0; i<${SVN_COMMITTER_ID_ARRAY_LENGTH}; i++))
    do
        j=`expr ${i} + 1`

        SVN_COMMITTER_ID=${SVN_COMMITTER_ID_ARRAY[${j}]%/*}

        for ((x=0; x<${RTX_ID_MAP_ARRAY_LENGTH}; x++))
        do
            y=`expr ${x} + 1`
            RTX_NAME=${RTX_ID_MAP_ARRAY[${x}]%' '*}

            if [[ "${RTX_NAME}" == "${SVN_COMMITTER_ID}" ]]; then
                RTX_ID=${RTX_ID_MAP_ARRAY[${y}]%' '*}
                RTX_NOTIFY_RECEIVER="${RTX_NOTIFY_RECEIVER},${RTX_ID}"
            fi

            x=`expr ${x} + 1`
        done

        i=`expr ${i} + 1`
    done

    echo "RTX_NOTIFY_RECEIVER=${RTX_NOTIFY_RECEIVER}"
}
```

- **发送 RTX 消息**

调用 rtx_send_msg.py 脚本实现将相关消息发送给相关同事的目的。

```
fn_send_rtx_notify()
{
    echo "  fn_send_rtx_notify...: ${1}"

    echo "cd ${SH_PATH}"
    cd "${SH_PATH}"

    RTX_NOTIFY=${1}
    echo ">>> RTX_NOTIFY=${RTX_NOTIFY}"

    echo "./rtx_send_msg.py -s ${RTX_SDK_SERVER} -u ${RTX_NOTIFY_RECEIVER} -m ${RTX_NOTIFY}"
    ./rtx_send_msg.py -s ${RTX_SDK_SERVER} -u ${RTX_NOTIFY_RECEIVER} -m ${RTX_NOTIFY}

    check_exit
}
```

3.2 RTX 通知脚本

RTX 通知脚本是利用 RTX 提供的接口而实现在 Linux 平台下也可以将编译结果发送到安装了 RTX 的同事，从而提高编译错误的解决效率。

RTX 的接口是提供了一个名为 SendNotify.cgi 的 CGI 接口， 在使用这个接口之前，需要让信息管理部的同事将你的 IP 加入 RTX 服务器安装目录下的 SDKProperty.xml 文件的信任列表中，如下图所示：

```
<?xml version="1.0"?>
<Property>
<APIClient>
    <IPLimit Enabled="1">
        <IP>127.0.0.1</IP>
        <IP>192.168.3.182</IP>
        <IP>192.168.3.183</IP>
        <IP>192.168.3.184</IP>
        <IP>192.168.144.3</IP>
    </IPLimit>
</APIClient>
<sdkhttp>
    <IPLimit Enabled="1">
        <IP>127.0.0.1</IP>
        <IP>192.168.1.153</IP>
        <IP>192.168.3.182</IP>
        <IP>192.168.3.183</IP>
        <IP>192.168.3.184</IP>
        <IP>192.168.144.3</IP>
    </IPLimit>
</sdkhttp>
</Property>
```

之后我们可以通过 HTTP 的 GET/POST 方法将编译结果发出去，也可以在浏览器输入如下内容实现发送消息的功能：

http://RTX 服务器地址:服务器端口/sendnotify.cgi?title=通知标题&receiver=接受者 ID&msg=消息内容

3.2.1 脚本源码

RTX 通知脚本是使用 python 写的，源代码如下图 3.1 所示：

```python
#! /usr/bin/env python
#
#   History:
#       long.luo created it, 2012/08/10.
#

import urllib,sys,optparse,sys

p=optparse.OptionParser("%prog -f Contacts -m Message",version='1.0',prog="rtx_send_msg.py",epilog='Copyright(c) long.luo')
p.add_option('-f','--filename',dest='Filename',help='filename Contains Contacts')
p.add_option('-m','--message',dest='Message',help='Message Sent To Contacts')
p.add_option('-s','--server',dest='server',help='RTX SDK SERVER')
p.add_option('-u','--userlist',dest='userlist',help='UserList split by ",",do not use with -f')

if len(sys.argv)==1:
        p.print_help()
        sys.exit(1)

#
opt,args=p.parse_args()

if not opt.Message or     (opt.Filename and opt.userlist)  or not opt.server or not (opt.Filename or opt.userlist) :
        p.print_help()
        sys.exit(1)

def transcode(stotrans):
        return stotrans.decode('utf-8').encode('gb2312')

if opt.Filename:
        try:
            reciver=','.join(open(opt.Filename).read().splitlines())
        except IOError,e:
            print e
            sys.exit(1)
else:
        reciver=opt.userlist

server=opt.server
Title=transcode("12021 Compile Infomation:")

if sys.platform == 'win32':
        reciver=reciver
        MESSAGE=opt.Message+' '+' '.join(args)
else:
        reciver=transcode(reciver)
        MESSAGE=transcode(opt.Message+' '+' '.join(args))

urllib.urlopen('http://'+server+'/sendnotify.cgi?msg='+MESSAGE+'&receiver='+reciver+'&title='+Title)

```

图 3.1 RTX 通知脚本源码

3.2.2 使用规范

send_rtx_notify.py 可以通过下列操作来实现：

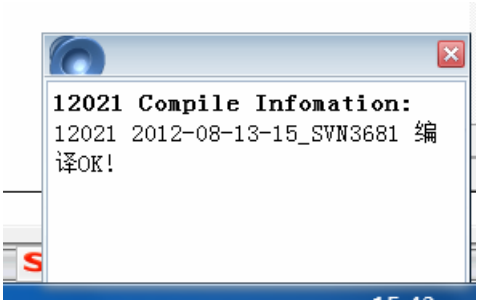
```
/* 使用示范 */
./send_rtx_notify.py -s RTX 服务器地址 -u 要通知者的 ID -m 消息内容
/* 如果要发送到多个通知者，必须在用户 ID 之间添加半角状态的“,”来分割。 */
```

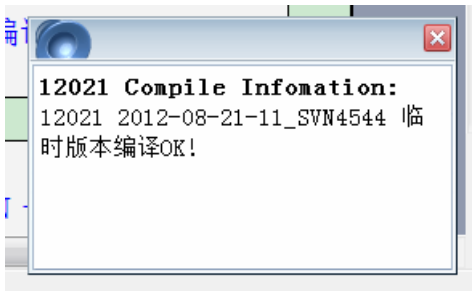
3.2.3 编译结果通知规范

编译成功：发送RTX消息给管理员及PM；

```
正式版本格式如下：
    项目名 编译日期-编译时间_编译 SVN 号 编译 OK!
```

```
临时版本格式如下：
    项目名 编译日期-编译时间_编译 SVN 号 临时版本编译 OK!
```





编译失败：除发送RTX消息给管理员及PM，另外通知此次编译和上次编译之间有提交SVN号的同事。

正式版本格式如下：
项目名 编译日期-编译时间_编译 SVN 号 编译失败! 上次编译 SVN 号~当前编译 SVN 号 提交人请修正!

正式版本格式如下：
项目名 编译日期-编译时间_编译 SVN 号 **临时版本**编译失败! 上次编译 SVN 号~当前编译 SVN 号 提交人请修正!

3.3 提取出错信息脚本

find_str_argv.pl 可以实现从 log 中提取出错的信息的功能，是用 Perl 脚本写的，使用规范如下：

```
/* 传递的参数是某个搜索路径 */
perl find_str_agrv.pl ${路径}
```

```
/* 在 Shell 编译脚本中的使用 */
# 会将出错的 log 文件及出错信息输出到后面指定的文件中
perl find_str_argv.pl ${1} >> ${BUILD_ERROR_LOG_TEMP}
perl find_str_argv.pl ${1} >> ${BUILD_ERROR_LOG_TOTAL}

输出格式如下：
文件名 ： 出错误的行信息
```

具体源代码如下图 3.2 所示：

```perl
#!/usr/bin/perl
#
#       History:
#    long.luo, created it, 2012/08/16
#

use warnings;

#print "ARGV: @ARGV\n";

if(@ARGV > 0){
    my $searchDir = $ARGV[0];

    if(scalar(@ARGV) > 1 && -d $ARGV[0]){
        $searchDir=$ARGV[0];
    }

    findstr($searchDir);
}
else{
    print("usage:dir.\n");
}


sub findstr{
    my $file = $_[0];

    if(-f $file){
        my $FILE;
        open($FILE, $file);

        while(<$FILE>){
            if (/^make.*:.*\[(.*)\].*$/){
                print("$file : $_");
            }
        }

    }elsif(-d $file){
        my $DIR;
        my $dir;
        opendir($DIR, $file);

        while($dir = readdir($DIR)){
            if($dir ne ".." && $dir ne "."){
                findstr("$file/$dir");
            }
        }

        closedir($DIR);
    }
}

```

图 3.2 提取错误信息脚本源码

四、手动编译步骤

当我们需要手动编译时，需要按照如下步骤进行操作（以 12021 编译服务器为例）：

1. 在 CRT 上执行 crontab -e, 在相关的行前面输入 # 将其注释掉，然后按 ESC 键，再输入 wq，即关闭了 crontab. 因为如果不关闭的话，会和手动执行的脚本产生冲突，造成错误。

```
21 #
22 # m h  dom mon dow   command
23 #00 6,9,12,15,18,21 * * * /work/12021/script/12021_compile_daily.sh >> /work/12021/OUT/dailyBuild.log 2>&1 &
24 #15 7,8,10,11,13,14,16,17,19,20,22,23 * * * /work/12021/script/12021_compile_daily_temp.sh >> /work/12021/OUT
   /dailyBuild.log 2>&1 &
25 30 2 * * * /work/12021/script/clear.sh
```

```
23 #00 6,9,12,15,18,21 * *
24 #15 7,8,10,11,13,14,16,
   /dailyBuild.log 2>&1 &
25 30 2 * * * /work/12021/
:wq
```

2. 执行 cd /work/12021/script 命令， 进入 script 目录下，在命令行输入：

```
root@ubuntu:/work/12021/script# bash -x 12021_compile_daily.sh
```

(说明：也可以不加 -x 选项，此选项是一个 debug 选项，可以看到所有语句执行情况。)
或者

```
root@ubuntu:/work/12021/script# source 12021_compile_daily.sh
```

然后脚本就会开始执行了。

3. 假如在执行的过程中，弹出了 XXX Don't Need Rebuild. 的提示，说明此次编译的版本号和上次编译的版本号一致，因此不需要编译。因此，你需要修改 log/12021_info.log 文件，将文件中的数值改小一点，比如将 3998 改成 3995，但切记不要改的太小。因为脚本执行有一个流程就是取文件中的 svn 号和当前最新的 svn 号之间所有提交者的 svn id 并查找对应的 RTX id，假如人数过多的话，这个过程非常耗时。

4. 如果一切正常，那么恭喜你!!! 如果提示编译失败，请看下一章出错处理。

五、编译出错处理

承接上一章，由于很多原因都可能会导致失败，不过我总结一般原因分为以下几种：

1. SVN 服务器的问题：svn 取当前版本号提示错误，svn: no such version xxx，导致没有获得准确的版本号，在 svn export 那一步就会出错；或者 svn export 时提示上述错误，都会导致获取代码不准确，导致后续编译出错。另外一个问题就是在取代码中途出错，在上周经常出这个问题，不过后来信息管理部解决了这个问题。

2. 代码本身的问题：某几个版本号提交的代码编译不过导致编译失败，大部分编译失败是因为这个原因。

3. 脚本执行过程中由于各种原因导致编译出错，比如两个脚本同时执行，或者上一个脚本没执行完毕，这个脚本就开始执行同样会导致出错。

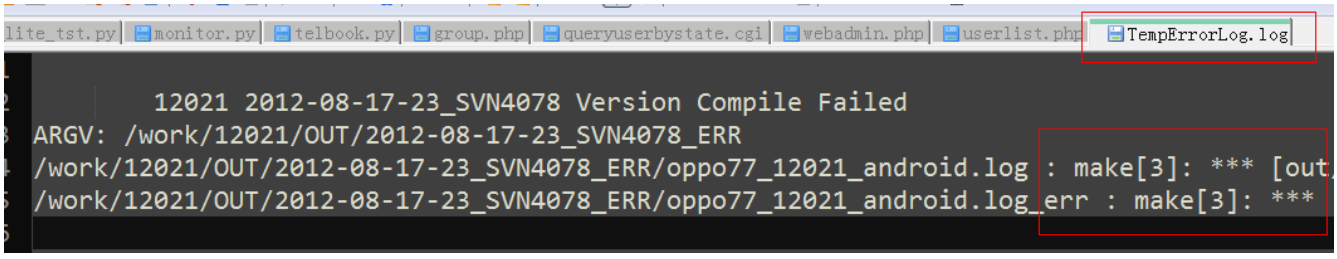
当编译失败之后，我们首先要去做的是查看编译产生的 log，位于/work/12021/OUT 目录下的 dailyBuild.log， 最新的编译 log 在该文件末尾，分析 log 看是在哪一步执行出错。

1. 如果 log 中提示：

```
2012/08/16 15:25:05 building android...
            BSBSBSBSLOG: out/target/product/oppo77_12021_android.log
                         out/target/product/oppo77_12021_android.log_err
                    ==> [FAIL]  2012/08/16 15:28:01
make[2]: *** [android] Error 1
make[1]: *** [remake] Error 2
make: *** [remakeall] Error 1
**********Build source takes 14 minutes 13 seconds
Compile Failed!
cp: omitting directory `out/target/product/oppo77_12021_assetsOverlay'
copy error to shenzhen
```

那说明是代码提交有问题，请去 OUT 目录相应版本的文件夹下分析出错 log，找出谁提交的代码所致。

你可以首先查看 OUT 目录下的 TempErrorLog.log 文件，初步找到出错模块。在仔细去相关 log 中找出错原因。



2. 如果 log 中提示：

```
SVN_VERSION_OLD=1884
svn: No such revision 1904
SVN_VERSION_NOW=
/work/12021/script/12021_compile_daily_temp.sh: line 67: [: =: unary operator expected
rm /work/12021/SRC/development
export source from SVN...
/work/12021/SRC
Thu Jul 26 13:15:28 CST 2012
svn: Syntax error in revision argument 'http://192.168.3.240/svn/oppo_swdp/ics2/development'
Thu Jul 26 13:15:28 CST 2012
```

说明 SVN 服务器有问题，没有取到最新的 svn 版本号，导致后续出错，请修改 log/12021_info.log 文件，文件会变成空值，请添加一个版本号，再重新编译。

3. 如果提示：

```
rm /work/12021/SRC/development
export source from SVN...
/work/12021/SRC
Fri Aug 17 12:01:06 CST 2012
svn: No such revision 4198
Fri Aug 17 12:01:06 CST 2012
**********SVN export code takes 0 minutes 0 seconds
```

说明在 svn export 代码时出现问题，可以重新编译。如果是临时版本的话，可以暂时不用管，因为目前提示 svn: no such revision 的错误，初步怀疑可能原因是深圳 svn 镜像服务器在同步长安服务器导致。具体原因还需要和信息管理部同事一起进行分析。

六、结语

本文档对 12021/12015 的 SVN 编译服务器的 RTX 通知方案的实现进行了总结，希望能对大家有所帮助。这个 RTX 通知方案也可以移植到其他项目的编译方案中去，以提高敏捷软件开发的效率。