# 《停指测试技术》实验报告书

学生姓名：严翔宇

学 号：31601371

班 级：软件工程1601

指导老师：杨枨

专 业：软件工程

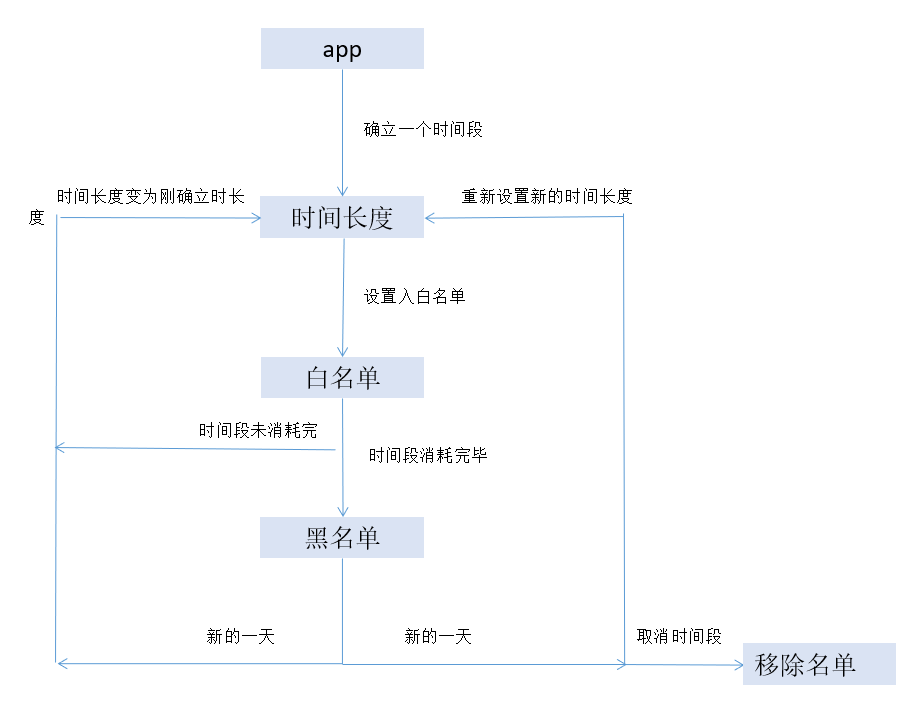
提交时间：2018年6月2日

# 白盒测试

1. 实验目的

通过程序的白盒测试，熟悉测试功能，对软件测试进行一定的了解，并对软件进行一定的测试。

1. 实验内容
2. 被测试功能：软件加锁。
3. 程序定义：设定时间段，当时间用完的时候，就会不断提醒到达时间，从而达到禁止使用禁止的app。
4. 测试环境：Android studio 自带模拟器
5. 实验步骤
6. 程序流程：



2．代码：

[java] view plain copy

/\*\*

\* <pre>

\* 倒计时单例工具类

\* </pre>

\*

\* @author hyj

\* @Date 2016-4-21 下午2:49:21

\*/

public class BTCCTimer extends CountDownTimer {

/\*\*

\* 倒计时总时长(s)

\*/

private final static long millisInFuture = 5 \* 1000;

private static BTCCTimer instance;

private Activity activity;

/\*\*

\* 倒计时是否结束

\*/

private static boolean isTimeFinish = false;

private BTCCTimer(long millisInFuture, long countDownInterval) {

super(millisInFuture, countDownInterval);

}

/\*\*

\* 获取一个倒计时实例

\*

\* @return

\*/

public static BTCCTimer getInstance() {

if (null == instance) {

synchronized (BTCCTimer.class) {

if (null == instance) {

instance = new BTCCTimer(millisInFuture, 1 \* 1000);

}

}

}

return instance;

}

@Override

public void onTick(long millisUntilFinished) {

BTCLogManager.logE("执行倒计时：" + (millisUntilFinished / 1000) + " "

+ millisUntilFinished);

if (millisUntilFinished <= 2 \* 1000) {

isTimeFinish = true;

return;

}

isTimeFinish = false;

}

@Override

public void onFinish() {

if (isTimeFinish) {

BTCLogManager.logE("倒计时解锁，执行锁屏操作");

BaseActivity.CheckPass(activity);

isTimeFinish = false;

} else {

BTCLogManager.logE("倒计时被中断");

}

}

/\*\*

\* 显示解锁界面

\*/

public void showLock(Activity activity) {

if (isTimeFinish) {

BTCLogManager.logE("跳进解锁界面");

BaseActivity.CheckPass(activity);

isTimeFinish = false;

}

start(activity);

}

/\*\*

\* 开始倒计时

\*

\* @param activity

\*/

public void start(Activity activity) {

this.activity = activity;

stop();

if (activity instanceof CreateGesturePasswordActivity) {

BTCLogManager.logE("当前为九宫格解锁界面");

return;

}

instance.start();

}

/\*\*

\* 停止倒计时

\*/

public void stop() {

instance.cancel();

instance.onFinish();

}

}

/\*\*

\*

\* <pre>

\* 屏幕状态监听

\* </pre>

\*

\* @Date 2016-4-21 下午2:52:36

\*/

public class ScreenListener {

private Context mContext;

private ScreenBroadcastReceiver mScreenReceiver;

private ScreenStateListener mScreenStateListener;

public ScreenListener(Context context) {

this.mContext = context;

this.mScreenReceiver = new ScreenBroadcastReceiver();

}

/\*\*

\* 开始监听screen状态

\*

\* @param listener

\*/

public void begin(ScreenStateListener listener) {

this.mScreenStateListener = listener;

registerListener();

getScreenState();

}

/\*\*

\* 获取screen状态

\*/

private void getScreenState() {

if (null == mScreenStateListener) {

return;

}

PowerManager manager = (PowerManager) mContext

.getSystemService(Context.POWER\_SERVICE);

if (manager.isScreenOn()) {

mScreenStateListener.onScreenOn();

} else {

mScreenStateListener.onScreenOff();

}

}

/\*\*

\* 启动screen状态广播接收器

\*/

private void registerListener() {

IntentFilter filter = new IntentFilter();

filter.addAction(Intent.ACTION\_SCREEN\_ON);

filter.addAction(Intent.ACTION\_SCREEN\_OFF);

filter.addAction(Intent.ACTION\_USER\_PRESENT);

mContext.registerReceiver(mScreenReceiver, filter);

}

/\*\*

\* 停止screen状态监听

\*/

public void unregisterListener() {

mContext.unregisterReceiver(mScreenReceiver);

}

/\*\*

\*

\* <pre>

\* screen状态广播接收者

\* </pre>

\*

\* @author hyj

\* @Date 2016-4-20 上午9:36:00

\*/

private class ScreenBroadcastReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

switch (intent.getAction()) {

case Intent.ACTION\_SCREEN\_ON:// 开屏

mScreenStateListener.onScreenOn();

break;

case Intent.ACTION\_SCREEN\_OFF: // 锁屏

mScreenStateListener.onScreenOff();

break;

case Intent.ACTION\_USER\_PRESENT:// 解锁

mScreenStateListener.onUserPresent();

break;

}

}

}

/\*\*

\*

\* <pre>

\* 返回给调用者屏幕状态信息

\* </pre>

\*

\* @Date 2016-4-20 上午9:34:07

\*/

public interface ScreenStateListener {

/\*\*

\* 开屏

\*/

public void onScreenOn();

/\*\*

\* 锁屏

\*/

public void onScreenOff();

/\*\*

\* 解屏

\*/

public void onUserPresent();

}

}

public class BTCCMWActivityLifecycle implements ActivityLifecycleCallbacks {

/\*\*

\* 是否现在退出APP

\*/

public static boolean isExitAPP = false;

private int activityCount = 0;// 当前活动Activity的数量

private Activity preActivity;// 上一个展示界面

@Override

public void onActivityCreated(Activity activity, Bundle savedInstanceState) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivityCreated");

}

@Override

public void onActivityStarted(Activity activity) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivityStarted");

activityCount++;

}

@Override

public void onActivityResumed(Activity activity) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivityResumed");

preActivity = activity;

// BTCCTimer.getInstance().showLock();

BTCCTimer.getInstance().start(preActivity);

}

@Override

public void onActivityPaused(Activity activity) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivityPaused");

}

@Override

public void onActivityStopped(Activity activity) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivityStopped：" + preActivity);

activityCount--;

if (activityCount <= 0 && !isExitAPP) {

BTCLogManager.logI("所有Actvity停止...");

BTCCTimer.getInstance().start(preActivity);

if (isBackground(activity)) {

IPhoneDialogUtil.showToast(activity, "随心用正在后台运行");

GlobalInfo.homeFlag = true;

ArgumentBaseActivity.isNoNeedPwd = false;

BTCCMWApplication.isSettingPassword = false;

}

} else {

BTCCTimer.getInstance().start(preActivity);

}

}

/\*\*

\* 判断当前应用程序处于前台还是后台

\*/

public static boolean isApplicationBroughtToBackground(final Context context) {

ActivityManager am = (ActivityManager) context

.getSystemService(Context.ACTIVITY\_SERVICE);

List<RunningTaskInfo> tasks = am.getRunningTasks(1);

if (!tasks.isEmpty()) {

ComponentName topActivity = tasks.get(0).topActivity;

if (!topActivity.getPackageName().equals(context.getPackageName())) {

return true;

}

}

return false;

}

/\*\*

\* 判断应用是处于后台

\*

\* @param context

\* @return

\*/

public static boolean isBackground(Context context) {

ActivityManager activityManager = (ActivityManager) context

.getSystemService(Context.ACTIVITY\_SERVICE);

List<RunningAppProcessInfo> appProcesses = activityManager

.getRunningAppProcesses();

for (RunningAppProcessInfo appProcess : appProcesses) {

if (appProcess.processName.equals(context.getPackageName())) {

String msg = "importance = " + appProcess.importance;

msg += "\nbackground = 400;empty = 500;foreground = 100;";

msg += "gone = 1000; perceptible = 130;service = 300;visible = 200";

BTCLogManager.logI(msg);

if (appProcess.importance != RunningAppProcessInfo.IMPORTANCE\_FOREGROUND) {

BTCLogManager.logI("检测到在后台运行");

return true;

} else {

BTCLogManager.logI("还在前台1");

return false;

}

}

}

BTCLogManager.logI("还在前台2");

return false;

}

@Override

public void onActivitySaveInstanceState(Activity activity, Bundle outState) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivitySaveInstanceState");

}

@Override

public void onActivityDestroyed(Activity activity) {

BTCLogManager.logI("运行类名：" + activity.getClass().getSimpleName()

+ "：onActivityDestroyed");

}

}

registerActivityLifecycleCallbacks(new BTCCMWActivityLifecycle());

// 屏幕状态监听

ScreenListener.ScreenStateListener screenListener = new ScreenListener.ScreenStateListener() {

@Override

public void onScreenOn() {

}

@Override

public void onScreenOff() {

BTCLogManager.logE("屏幕锁定");

GlobalInfo.homeFlag = true;

ArgumentBaseActivity.isNoNeedPwd = false;

BTCCMWApplication.isSettingPassword = false;

}

@Override

public void onUserPresent() {

}

};

ScreenListener mScreenListener = new ScreenListener(this);

mScreenListener.begin(screenListener);

@Override

public boolean dispatchTouchEvent(MotionEvent ev) {

//循环遍历注册过时间的Fragment对象

for (MyOnTouchListener listener : onTouchListeners) {

BTCLogManager.logI("循环Fragment：" + ev.getSource());

listener.onTouch(ev);

}

BTCLogManager.logI("dispatchTouchEvent action：" + ev.getAction());

switch (ev.getAction()) {

case MotionEvent.ACTION\_DOWN:

BTCLogManager.logI("dispatchTouchEvent 按…………");

BTCCTimer.getInstance().stop();

break;

case MotionEvent.ACTION\_UP:

BTCLogManager.logI("dispatchTouchEvent 松…………");

BTCCTimer.getInstance().start(this);

break;

}

/\*\*

\* 回调接口

\* @author zhaoxin5

\*

\*/

public interface MyTouchListener

{

public void onTouchEvent(MotionEvent event);

}

/\*

\* 保存MyTouchListener接口的列表

\*/

private ArrayList<MyTouchListener> myTouchListeners = new ArrayList<MainActivity.MyTouchListener>();

/\*\*

\* 提供给Fragment通过getActivity()方法来注册自己的触摸事件的方法

\* @param listener

\*/

public void registerMyTouchListener(MyTouchListener listener)

{

myTouchListeners.add( listener );

}

/\*\*

\* 提供给Fragment通过getActivity()方法来取消注册自己的触摸事件的方法

\* @param listener

\*/

public void unRegisterMyTouchListener(MyTouchListener listener)

{

myTouchListeners.remove( listener );

}

/\*\*

\* 分发触摸事件给所有注册了MyTouchListener的接口

\*/

@Override

public boolean dispatchTouchEvent(MotionEvent ev) {

// TODO Auto-generated method stub

for (MyTouchListener listener : myTouchListeners) {

listener.onTouchEvent(ev);

}

return super.dispatchTouchEvent(ev);

}

/\*\*

\* Fragment中，注册

\* 接收MainActivity的Touch回调的对象

\* 重写其中的onTouchEvent函数，并进行该Fragment的逻辑处理

\*/

private MainActivity.MyTouchListener mTouchListener = new MyTouchListener() {

@Override

public void onTouchEvent(MotionEvent event) {

// TODO Auto-generated method stub

if(event.getAction() == MotionEvent.ACTION\_UP){

//逻辑处理

}

}

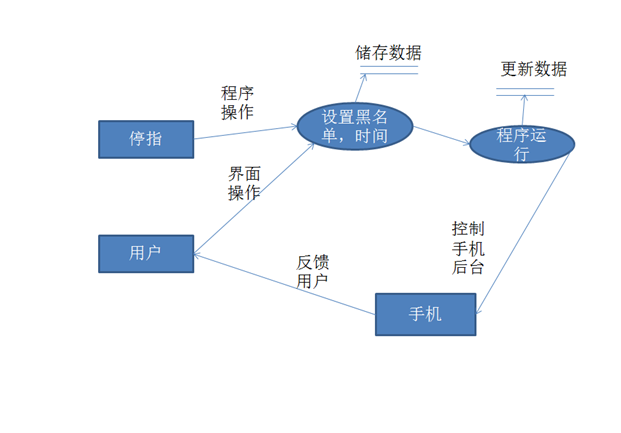
};

//在该Fragment的构造函数中注册mTouchListener的回调

((MainActivity)this.getActivity()).registerMyTouchListener(mTouchListener);

图片：

3．流图：



4．程序基本路径：

（1）.抖音 0分钟 预期输出:（抖音上锁）

（2）浏览器 1分钟 预期输出：（使用1分钟后，浏览器上锁）

（3）游戏 2分钟 预期输出：（使用2分钟后，游戏上锁）

# 黑盒测试

1. 实验目的

黑盒测试是一种常用的软件测试方法，在知道输入输出之间的关系或是程序功能的情况下来测试程序功能的正确性。

1. 实验内容
2. 被测试的功能和白盒一样
3. 测试环境一样
4. 测试方法
   1. 等价划分和测试用例设计
      1. 等价划分：
         1. 输入条件：需要设定的app
         2. 有效等价类：app名字和设定时间
         3. 无效等价类：输入app名字不存在，设定时间超过最大值或是小于最低值。
      2. 测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 用例 | App名字和时间 | 预期输出 | 有效/无效等价类 |
| 1 | 抖音 0分钟 | 抖音上锁 | 有效 |
| 2 | 浏览器 10分钟 | 10分钟内可以使用，达到时间后会上锁 | 有效 |
| 3 | 王者荣耀 25小时 | 超出时间设定上限 | 无效 |
| 4 | 微信 -5小时 | 低于时间最小值 | 无效 |
| 5 | 浏览器 24小时 | 当天可以一直使用 | 有效 |

图片：

1. 实验总结

在实验中，测试完毕后都达到要求则无问题，若到不到要求与预期不符合则需要修改。