|  |
| --- |
| **手机APP软件自动化测试研究报告总结** |
|  |
|  |
|  |
| |  | | --- | | **一、引言**  手机软件的自动化测试一直困扰着手机软件测试从业人员，本文将最近的一些研究新发现及具体思路作详尽阐述，希望能给予大家更多的参考萌发新的思路。  通过长期的手工测试得出如下可以以自动化测试来解决的问题：  1. 压力测试：一些连续不断的操作，比如反复切换歌曲播放及联网操作等；  2. 极限临界测试：一些极限条件的构造（创建多个列表）及输入字符个数等；  3. 兼容及中断：比如在播放或下载歌曲的时候来电话或者信息；  4. 基本功能回归测试：这样大大的节约了时间和人力成本。  对于以上的测试很多也是可以通过手工来完成，但部分测试采用手工测试是不可靠的，比如最近发现一个Bug（在联网的一瞬间如果来一个信息等中断操作出现死机），类似这种Bug出现条件非常苛刻和临界的情况在手工测试中是很难发现和构造这种测试环境的，即使发现了在很大程度上也属于一种偶然，同时给开发人员定位这个问题也带来了很大的困难。  面对诸多因素，我们不得不重视手机软件的自动化测试研究。其实如果掌握了一些自动化测试要领，从简单入手，逐步实现和突破，相信一定能够解决手机软件自动化测试的难题。  **二、自动化测试原理**  http://www.uml.org.cn/Test/images/54765_1264663217tZRf.jpg  【自动化测试原理图】  1. Test Agent  Test Agent为嵌入在手机软件系统中的一个测试代理模块，解决PC端与手机端交互处理及互联消息通讯问题，这是区别于其他桌面软件自动化测试的关键点，也是嵌入式软件自动化测试的主要特征之一。通过串口或蓝牙设备与PC端中的Test Tool建立通讯，其具备的主要功能如下：  1) 接收Test Tool发送的消息并向手机端软件系统分发消息及任务  2) 监控手机端软件运行情况并根据相应的约束反馈给PC端的Test Tool  3) 被测软件的功能（接口）封装及消息响应  2. Test Tool  Test Tool自动化测试工具在PC端用于测试控制及测试操作实体，与Test Agent对应，该工具与常规的自动化测试软件一样，其具备的主要功能如下：  1) 向手机端Test Agent发送可识别的消息及任务  2) 接收来自手机端Test Agent的反馈结果  3) 对来自手机端Test Agent的反馈进行测试业务的处理  4) 将测试业务的处理结果呈现给测试人员  **三、测试业务**  1. 主动式测试  Test Tool主动式测试是根据我们的测试需求比如（压力、性能、极限）在Test Tool中编写测试脚本控制手机端软件进行测试，或者构造一些手工很难实现的测试场景，通过运行脚本向Test Agent发送消息及任务，Test Agent再向被测软件分发消息及任务，并将结果原路返回给Test Tool，Test Tool再通过数据处理分析得出测试结果。关键点：发送和分发消息、接收及处理反馈结果（结果判断）。  2. 回归式测试  基本功能的回归测试最为简单的方法就是录制和回放机制，通过运行录制的测试脚本达到按照先前的操作顺序、步骤、输入数据等再次测试被测软件以此达到回归测试的目的。  1) 录制：就是在执行手工测试时将手工测试的任何操作及返回结果（预期正确的结果）通过Test Agent在Test Tool中保存下来，并进行分析处理形成一个可执行的脚本。录制的关键点：按键或触屏消息、坐标、响应结果（GUI界面）。  2) 回放：与录制相对应，运行录制时产生的脚本，与主动式测试方式不同的是回归式测试是事先要录制脚本，通过录制脚本来代替人工编写脚本。回放关键点：发送和分发消息、接收及处理反馈结果（结果判断）。  **四、关键技术**  1. 消息传送机制  利用手机Modem中提供的AT Command通过串口向手机端建立命令消息通讯，目前手机厂商提供了常用的AT Command，基本满足普通的自动化测试需求，另外厂商还提供了用户自定义AT Command的功能，当标准的AT Command不能满足自动化测试需求时，我们可以利用自定义AT Command来实现我们自动化测试中所需要的消息通讯。如下为MTK平台上实现自定义AT Command的关键样例代码：  view plain copy to clipboard print? //customer\_at\_command.c  #include "kal\_non\_specific\_general\_types.h"  #include "stdio.h"  #include "string.h"   #define CUSTOM\_SYMBOL '^' // '+' and '/' and ' \ 'is NOT allowed  #define MAX\_UART\_LEN 128   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* customer command  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define PLAY "play"  #define STOP "stop"   kal\_uint8 custom\_get\_atcmd\_symbol(void);  void custom\_command\_hdlr(char \*full\_cmd\_string);  extern void rmmi\_write\_to\_uart(kal\_uint8 \* buffer, kal\_uint16 length, kal\_bool stuff);   /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* FUNCTION  \* custom\_command\_hdlr()  \* DESCRIPTION  \* This function should parse the custom AT command and do correspondent action.  \* Customer should maintain and modify the code.  \* PARAMETERS  \* kal\_uint8 \* cmd\_string  \* RETURNS  \* none  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void custom\_command\_hdlr(char \*full\_cmd\_string)  {  char buffer[MAX\_UART\_LEN];  char cmd\_name[15];  kal\_uint8 index = 3; // we start parsing index after the CUSTOM\_SYMBOL  kal\_uint8 tmp\_idx = 0;   while ((full\_cmd\_string[index] != '=' ) && //might be TEST command or EXE command  (full\_cmd\_string[index] != '?' ) && // might be READ command  (full\_cmd\_string[index] != 13 )) //carriage return  {  cmd\_name[tmp\_idx] = full\_cmd\_string[index] ;  tmp\_idx ++;  index ++;  }  cmd\_name[tmp\_idx] = '\0' ;   /\* just a very basic example : customer can implement their own \*/  if (strcmp(cmd\_name, PLAY) == 0)  {  /\* BEGIN: do the following parsing and correspondent action \*/  /\* \*/  /\* \*/  /\* \*/  /\* END: do the following parsing and correspondent action \*/  /\* generate final result code: "OK" \*/  //Todo 实现消息分发或功能调用  sprintf(buffer, "Hello Play");  printf("%s\n", "Hello Play");  rmmi\_write\_to\_uart((kal\_uint8\*)buffer, (kal\_uint16)strlen(buffer), KAL\_TRUE);  }  else if (strcmp(cmd\_name, STOP) == 0)  {  /\* BEGIN: do the following parsing and correspondent action \*/  /\* \*/  /\* \*/  /\* \*/  /\* END: do the following parsing and correspondent action \*/  /\* generate final result code: "OK" \*/  //Todo 实现消息分发或功能调用  sprintf(buffer, "Hello Stop");  printf("%s\n", "Hello Stop");  rmmi\_write\_to\_uart((kal\_uint8\*)buffer, (kal\_uint16)strlen(buffer), KAL\_TRUE);  }  else  {  /\* unrecognized command \*/  /\* generate final result code: "ERROR" \*/  sprintf(buffer, "ERROR");  printf("%s\n", "ERROR");  rmmi\_write\_to\_uart((kal\_uint8\*)buffer, (kal\_uint16)strlen(buffer), KAL\_TRUE);  }   return;  }   kal\_uint8 custom\_get\_atcmd\_symbol(void)  {  return (CUSTOM\_SYMBOL);  }   　 2. 图像识别  图像识别主要通过抓取LCD屏幕显示图像进行智能识别来模拟测试工程师的双眼辨识文字或图像信息，以此判断测试结果。主要涉及图像的获取和对比分析，智能识别是一个比较专业的研究领域，更进一步的研究需要进行调研，目前我们可以考虑是否能够通过第三方工具来实现，比如借助目前已经成熟的测试工具QTP等。对于图像获取在手机平台上应该具备这样的接口，或者自行开发这个接口。  3. 录制回放  录制的信息及相应的实现方式如下：  1) 按键消息：由Test Agent捕获该消息并同步给PC端的Test Tool  2) 笔点消息：由Test Agent捕获该消息并同步给PC端的Test Tool  3) 坐标：由Test Agent捕获该坐标信息并同步给PC端的Test Tool  4) 响应结果（GUI界面回放的预期结果）：通过图像抓取接口抓取图像并同步给PC端的Test Tool（如果做到极致的话在PC端所呈现的GUI界面与实际手机GUI界面同步一致，等同于PC机上的显示为手机GUI的一个镜像）  5) 时钟同步：操作步骤的时间点、操作的先后顺序、输出结果响应时间  6) 录制脚本组装：Test Tool将所有的录制信息进行处理并组装成一套可运行的测试脚本，要求运行该脚本后能够与录制时的操作完全一样，并能将回放时的实际结果与预期结果进行比较从而得出执行结果。  7) 回放：主要是运行组装好的测试脚本，将回放时的实际结果与预期结果进行比较从而得出执行结果，关键点还是图像识别。  4. 测试管理  1) 脚本语言：可以选用Python、TCL作为脚本语言，用一些开源的工具进行脚本文件的管理维护。  2) 测试数据：需要建立一个数据仓库管理数据，比如录制时产生的消息、坐标、GUI图像信息等。  3) 测试结果管理：主要涉及测试报告的呈现及保存。  **五、 辅助工具及设备**  1. 串口线  2. 超级终端（PC机自带）或者手机厂商提供的配套工具  3. 脚本编写、调试、维护及管理工具（开源的比较多） | |