2010 年全国大学生信息安全竞赛

作品报告

作品名称： 基于 Android 的手机信息安全保护系统

参赛学校： _____

学院/系： _____

指导教师： _____

组　　长： _____

组　　员： _____

通信地址： _____

电　　话： _____

电子邮箱： _____

提交日期： _____

目　　录

第一章 摘要

随着 iPhone 和 Android 的发布，越来越多的隐私信息被人们存储于手机中，包括各种 Web 账户的密码。然而一旦手机被盗或丢失，由于手机缺乏访问控制和认证机制，用户的隐私信息将毫无安全性可言。虽然网秦公司推出了手机防盗卫士，但它仅仅提供了备份通讯录的功能，无法从根本上保护用户的隐私。鉴于此，我们基于 Android 平台提出了一个成熟的解决方案，在 GAE 平台上架设服务端，通过短信指令的方式达到服务端远程控制手机客户端的功能，从而在手机客户端实现了对隐私信息的访问控制。一旦手机被盗或丢失，用户可以登录 Web 服务端来发送控制指令，实现远程锁定被盗手机，远程备份和删除手机中的隐私数据等功能。而小偷一旦尝试替换 SIM 卡，手机客户端就会自动锁机。定位追踪模块则通过实时地图的方式，不断记录被盗手机当前所处的地理位置，达到了可审计和取证的目的。

关键词： 手机信息安全；访问控制；可审计；远程控制；Android；GAE；

Abstract

With the iPhone and Android released, more and more private information is stored in the phone, including a variety of Web account password. But once the phone is stolen or lost, due to the lack of access control and mobile phone authentication, the user's private information will have no security at all. Although the company launched the mobile network Qin security guards, but it only provides a backup address book function, can fundamentally protect the privacy of users. In view of this, we based Android platform, made a mature solution in GAE platform on setting up the server, via SMS card way to the service-side remote control client function to mobile client-side implementation of access to private information control. Once the phone is stolen or lost, users can log Web server to send control commands to remote lock stolen mobile phones, remote backup and delete phone data privacy functions. Location-tracking module is the way real-time map, and constantly records the current geographical location of stolen mobile phones, achieves the purpose of audit and evidence.

Key Words: Mobile Security; Access Control; Audit; Remote Control; Android; GAE;

第二章 作品介绍

2.1 背景分析

手机，特别是智能手机的安全问题已经成为信息安全新课题。

据统计，从2005年到2009年，已知的手机操作系统和第三方程序漏洞已超过40个，涉及目前市面绝大部分流行的手机及智能手机，囊括Windows Mobile，Symbian，iPhone以及Android等多种主流手机操作系统。

除智能手机操作系统本身及第三方应用软件带来的安全漏洞外，用户信息丢失和泄漏更是严重威胁到用户个人信息的安全。McAfee 移动安全实验室在其 2009 年发布的《移动安全报告》中指出，2008 年全年，用户数据丢失在移动设备所受安全威胁中占到极大的比例，我们可以从图 2.1 中清楚地看到（数据来源：Informa Telecoms & Media ©2009 Informa UK Ltd）。

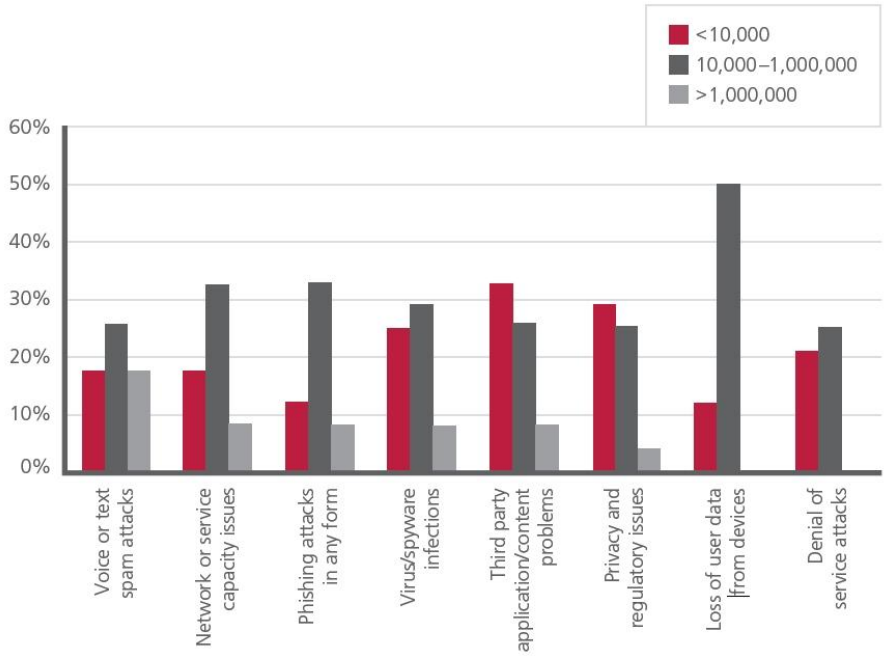

图 2.1　2009 年移动安全报告

2.2 研究成果

该系统分为手机客户端和Web服务端，客户端在Android平台上实现，而Web端则

直接架设在Google App Engine上（域名为http://androidmobilesec.appspot.com/）。通过手机客户端和Web服务端的共同作用，实现的功能有：

1) 远程锁定被盗手机，并报警给预先设定好的号码；

2) 手机SIM卡被替换后，则自动通知特定号码，并且锁机；

3) 远程备份、删除、恢复手机中的数据（如通讯录、短消息、通话记录等）；

4) 远程追踪手机中SIM卡，并在Web服务端留下详细的log记录；

5) 用户可以在Web服务端直接查看他的各项数据，并可以给手机客户端发送远程指令；

6）利用手机的GPS功能，对遗失或被盗手机进行追踪定位，在Web服务端留下历史记录，并利用Google Map直观显示位置信息。

2.3 应用前景

本系统较好地解决手机被盗后的信息安全问题，并且在Google App Engine的http://androidmobilesec.appspot.com/ 上做了一个可以被广大公网用户使用的Web服务端。对于要求内部使用的企业用户，还可以将这个Web服务端很方便地移植到公司的局域网络中，从而保护内部员工的手机信息安全。

第三章 实现方案

3.1 系统方案

3.1.1 总体方案设计

本系统采取手机客户端和 Web 服务端相结合的监控机制，两者相互合作来共同保障手机中信息的安全。系统的主要功能在手机客户端部分实现，而 Web 服务端起到了一种控制的功能，并且给用户提供了一个友好的用户界面来查看自己的手机数据。总体方案的框架设计图见图 3.1，主要描述了两者是 Web 服务端是如何控制手机客户端执行相应的操作的，即两者是如何协同工作的。

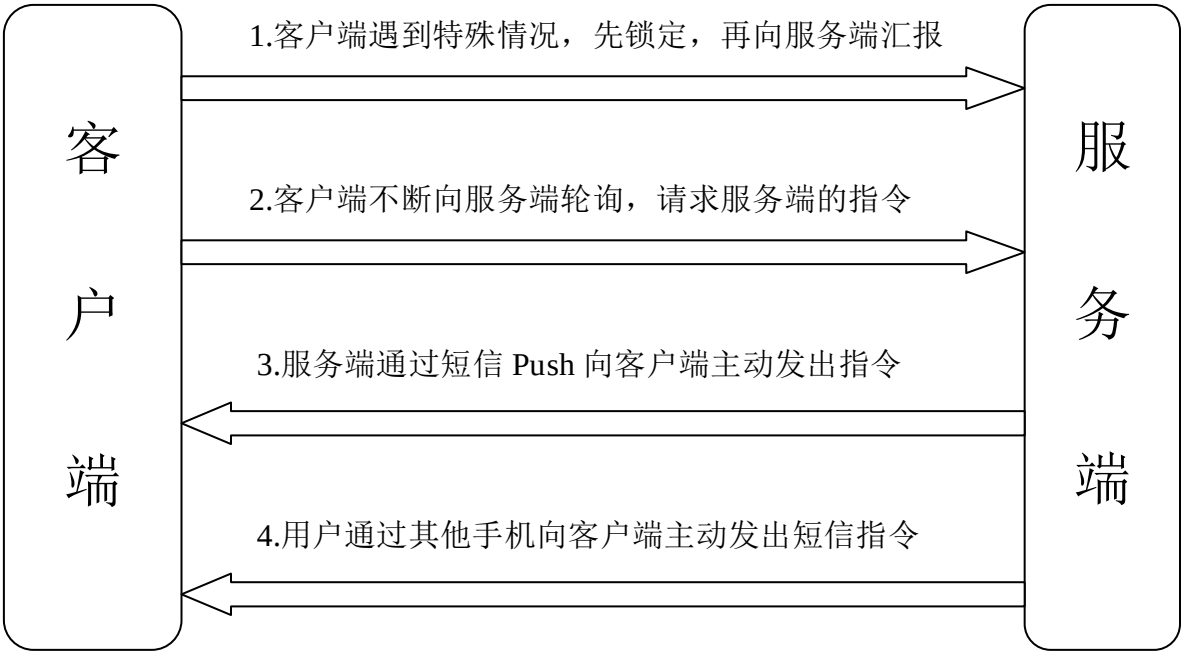

图 3.1 系统总体框架设计图

3.1.2 手机客户端框架设计

手机客户端的框架设计如图 3.2：

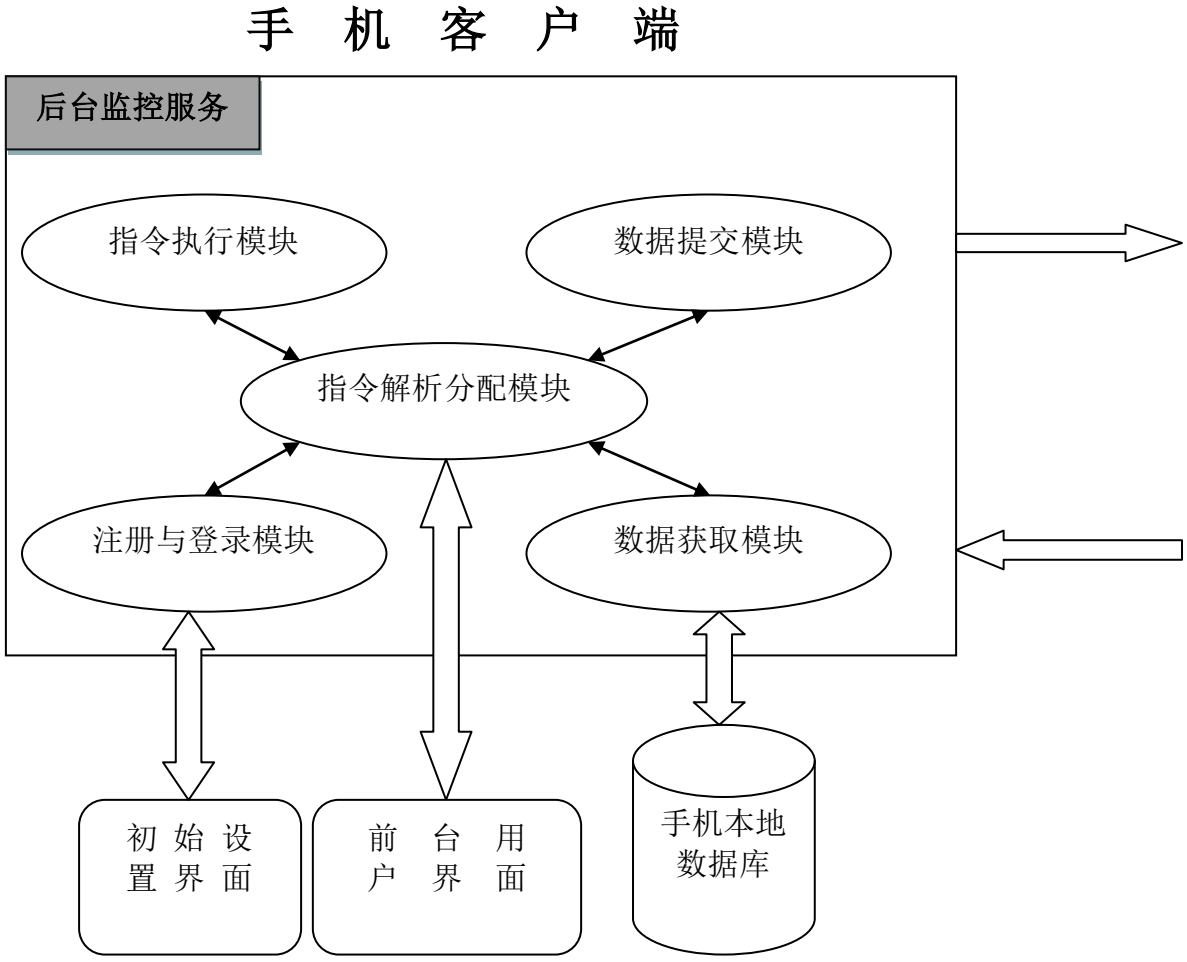

图 3.2　手机客户端框架设计图

3.1.3　Web 服务端框架设计

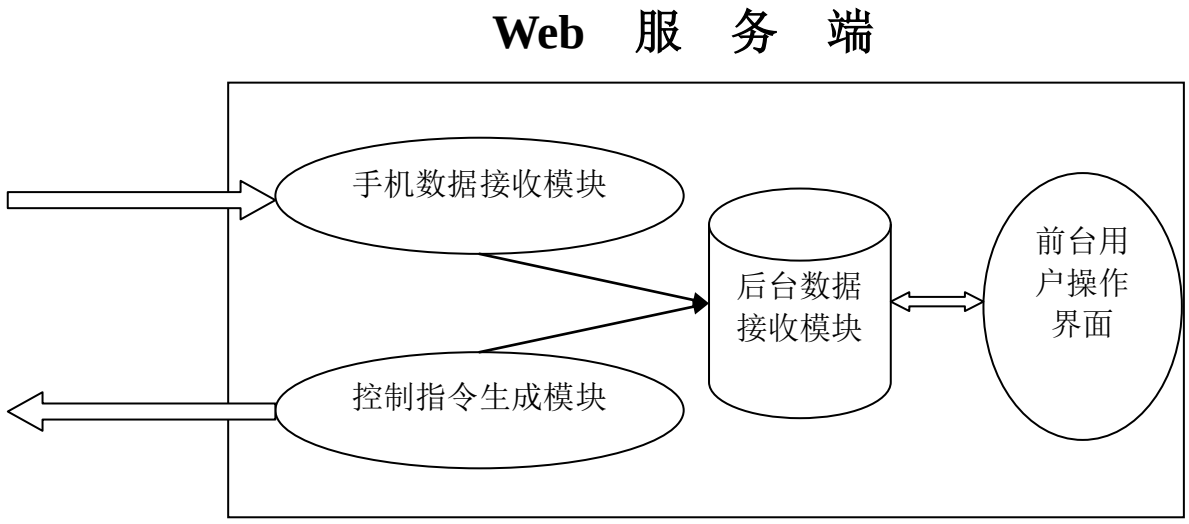

图 3.3　Web 服务端框架设计图

3.2 系统所用技术

3.2.1 客户端：基于 Android 系统

现在手机平台很多，有Windows、Symbian、BlackBerry和新推出的Android系统。鉴于Android系统的开源特性和Google的强大推动力，我们最终选择了Android平台作为我们手机客户端的研发平台。

与其他手机操作系统相比，Android平台具有几个无可比拟的优点：

- 开放性。不同的厂商可以根据自己的需求对平台进行定制和扩展，并且使用这个平台无需任何授权和许可费用。
- 所有应用程序是平等的。所有的应用程序都运行在一个核心引擎上面，即一个提供了一系列用于应用程序和硬件资源间通信的API的虚拟机。抛开此核心引擎，系统核心应用和第三方应用是完全平等的。
- 应用程序间无界限。Android打破了应用程序间的界限，开发人员可以把Web上的数据与本地结合起来。应用程序还可以声明它们的功能可以供其他应用程序使用。
- Android采用了软件堆层（Software Stack）的架构，从软件分层的角度看，Android平台由应用程序、应用程序框架、Android运行时、库以及Linux内核共5部分构成，如图3.4所示。

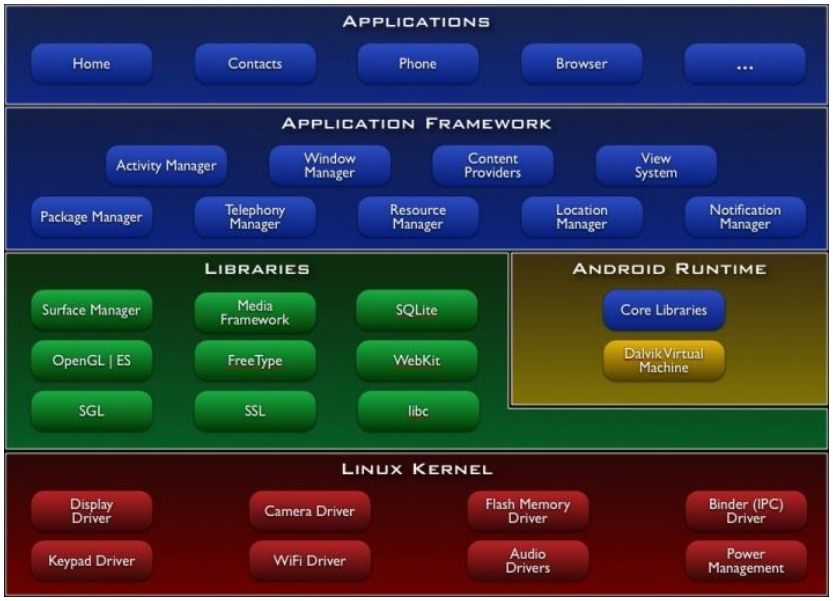

图 3.4　Andoid 系统架构图

Android平台默认包含了一系的核心营应用程序，这些程序均由Java语言写成。应用程序框架层是进行Android开发的基础，开发人员可以完全访问核心应用程序所使用的API框架，同时Android平台在设计时就考虑了组件的重用。Android运行时包括核心库和Dalvik虚拟机两部分，核心库提供Java程序所要调用的功能方法和Android核心库，Dalvik虚拟机是专门为移动设备设计的一种基于寄存器的Java虚拟机。系统库提供Android方法库，是应用程序框架的支撑。Android平台中的操作系统采用了Linux2.6版的内核，它为我们在软件和硬件层建立了一个抽象层，使得应用程序开发人员无需关心硬件细节，从而专注于软件开发。

3.2.2 服务端：基于 GAE 平台

由于服务端具有公网 IP，才能被手机客户端访问到。限于条件限制，我们无法搞到公网 IP 和 Web 服务器。幸好，Google 于 2008 年 4 月发布的 Google App Engine 解决了我们这个困扰。GAE 使用了云计算技术，给开发者提供了虚拟服务器、一定的带宽和 CPU 负载，最重要的是它给每个应用都提供了一个域名。比如，本系统采用的域名是 http://androidmobilesec.appspot.com/ 采用 Python 作为编程语言。

1) 支持的编程语言

当前，Google App Engine 支持的编程语言是 Python 和 Java。

2) 支持的模板框架

支持 Django、WebOb、PyYAML 的有限版本。当前，我们就采用 Django 作为前台框架，GAE 自带的 webapp 作为后台框架。

3) 数据存储 GQL

GAE 的数据存储使用一个与 SQL 类似的语法叫做“GQL”。GAE 限制每次数据存储请求最多返回 1000 行数据。如果一个应用程序需要每次操作返回更多的记录，可以使用自己的客户端软件或者一个 Ajax 页面来按查询顺序对无限的行进行操作。不像关系数据库，GAE 的数据存储 API 是不关联的。

3.3 手机客户端

3.3.1 前台用户界面模块

前台用户界面模块包含三个部分：

1) 刚安装本客户端时的初始注册界面，见图 3.5；

2) 用户在手机客户端上进行日常操作的界面，见图 3.6；

3) 手机被远程锁定时的锁定界面，见图 3.7；

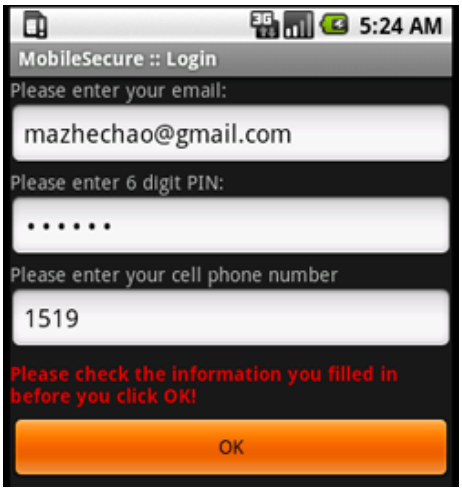

图 3.5　用户初次运行软件时需要注册

在图 3.5 这种界面中，进行一些初始化设置，提示用户输入他要绑定的 Email 和 Password。然后，用户注册模块（存在于后台监控服务中）将 Password 进行 Hash，将该用户的 Email 和散列后的 Password 传到 Web 服务端。如果注册失败，则返回注册失败的提示信息，并提示用户重新输入。

图 3.6　客户端主界面

图 3.7　锁定界面

3.3.2 用户注册与认证模块

1) 用户初始化注册时需要唯一性认证

用户刚装上本手机客户端时需要进行一些初始化注册，如图 3.5 所示。当用户点击【确定】按钮时，将启动 sendLoginData()方法，从而将 Email 和散列后的 Password 传到 Web 服务端。

Web 服务端接收到该数据后，将首先检查数据库中是否已经存在该 Email，如果存在，则向客户端返回注册失败的提示信息，并提示用户重新输入。如果不存在，则向后台数据库插入一条记录，并且向客户端返回一个 KEY，这将为作为数据提交模块的一个识别码。这将在【3.4.2 后台数据存储模块】进行详细阐述。

注册成功后，客户端需要在本地手机数据库中保存 Email 和散列后的 Password，方便以后认证用户是否合法。同时，对于服务端返回的 KEY，客户端也必须将它保存到本地数据库中，并且在以后每次提交数据的时候加上这个 KEY 作为该用户的唯一标识。

KEY 是 Web 服务端根据所插记录的 ID 号生成的标识码，它在整个 Web 数据库中都是唯一标识的，所以用它可以很方便地区分开不同的手机用户。如图 3.8，用户 clzqwdy@gmail.com 的 KEY 值为【agludXB0ZXItY25yDQsSBUxvZ2luGIH6AQw】，经过 GAE 自定的编码格式，可以将这个 KEY 值转化为全局唯一的 ID 号。

图 3.8 一个 Entity KEY 示例

上述客户端的整个注册认证过程在 Android 平台的具体编码实现，就如下面的 sendLoginData()所示：

```
protected void sendLoginData() {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(loginURL);
```

```
    List<NameValuePair> nvPair = new ArrayList<NameValuePair>();
    nvPair.add(new BasicNameValuePair("email",
field_email.getText().toString()));
    nvPair.add(new BasicNameValuePair("pswd",
field_pswd.getText().toString()));
    nvPair.add(new BasicNameValuePair("sim",
field_phonenum.getText().toString()));

    try {
        // 连接与数据上传
        ……
    }
}
```

2) 后台监控服务提交数据到 Web 服务端时需要 KEY 认证

后台监控服务 GuardService 在提交用户的隐私数据到 Web 服务端时，也要进行认证，即需要加上 KEY，从而让服务端区分出哪些是正常的数据包，并且区分出各个数据库对应到哪个用户。

我们的客户端与 Web 服务端数据通信的格式是基于 XML 的，比如，备份通话记录到 Web 服务端的数据格式如下：

```
<?xml version='1.0' encoding='UTF-8'?>
<calllog nid='agludXB0ZXItY25yDQsSBUxvZ2luGIH6AQw'>
<entity>
<name>Wudaoyuan</name>
<number>13770927023</number>
<type>Incoming</type>
<date>2010-03-27 13:43:30</date>
<duration>12</duration>
</entity>
<entity>
```

<name>Test</name>

<number>15555218135</number>

<type>Incoming</type>

<date>2010-03-27 13:42:05</date>

<duration>22</duration>

</entity>

</calllog>

其中，<calllog nid='agludXB0ZXItY25yDQsSBUxvZ2luGIH6AQw'>中的 nid 部分表示的就是某个用户的 KEY 值。那么，后面的两条<entity>记录就代表的是该用户的两条通话记录。

3) 开机时检查 SIM 卡是否为用户原先注册的

为了防止用户手机丢失或被盗后，盗窃人员更换 SIM 卡对手机进行操作，从而保护用户手机中隐私数据的安全，需要在开机时对 SIM 卡进行检查。注册时，程序会记录下用户的当前的 SIM 卡号 strSimNum 记录下来，用于今后启动时的比对。当当前 SIM 卡号 currentSimNum 和 strSimNum 匹配时，通过验证，可以继续操作。当否则启动锁定服务 LockService，将用户手机锁定。

```
String currentSimNum;   //当前 SIM 的卡号
String strSimNum;       //注册时的 SIM 卡号
boolean isLogined;      //是否注册过
boolean isLocked;       //是否被锁定

Intent iLockService = new Intent(this,LockService.class);
SharedPreferences sharedData = getSharedPreferences(LoginActivity.PREF,
0);
strSimNum = sharedData.getString(LoginActivity.PREF_SIMNUM, "");
isLogined = sharedData.getBoolean(LoginActivity.PREF_LOGIN, false);
isLocked  = sharedData.getBoolean(LoginActivity.PREF_LOCK, false);
if(isLogined){          //SIMCheck，当且仅当注册过才检查
   currentSimNum = getSimNum();
   if(currentSimNum.equals(strSimNum)){ //当且仅当 SIM 卡号正确
```

```
        if(LockActivity.isLocked){
            Intent              unlockIntent              =              new
Intent("com.nupt.stitp.action.UNLOCK");
            unlockIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            unlockIntent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
            startActivity(unlockIntent);
            LockActivity.isLocked = false;
        }
    }
    else{                  //其他情况跳到锁定
        ……
    }
}
if(isLocked){
    startService(iLockService);
}
else{
    stopService(iLockService);
}
```

3.3.3 隐私数据获取模块

该模块就是要将手机中的隐私数据提取出来，方便其他模块进行调用。本系统中，目前将用户的隐私数据定义为通讯录（Contact）、短消息（Sms）、通话记录（CallLog）和地理位置（Location）。于是，问题就转化为如何提取 Android 系统中的通讯录、短消息和和通话记录了。

提取 Contact 和 CallLog 相对比较容易，因为 Android 已经提供了相应的接口，即为它们提供了相应的 Content Providers。Android 平台的数据默认都是私有的，一个应用的数据只能它自己访问，如果它想要使自己的数据被别的应用访问到，它可以为这些数据提供一个接口，即 Content Provider。

我们只需定义自己的 Content Resolver，通过它来访问相应的 Content Provider，两者对接，即可提取出 Content Provider 中所提供的数据了。

1) 获取通讯录（Contact）

首先，我们定义一个自己的 Content Resolver，用它来作为我们访问数据的接口：

```
ContentResolver cr = getContentResolver();
```

接着，我们使用 ContentResolver 类中的 query 来查询通讯录的 URI，得到数据结果集 curContact：

```
Uri allPeople = People.CONTENT_URI;
Cursor curContact = cr.query(allPeople, null, null, null, null);
```

最后，我们循环遍历这个数据结果集 curContact，读出手机中的所有通讯录，并生成相应的 XML 数据格式，如下面的 getContactData(Cursor cur)方法所示：

```
private void getContactData(Cursor cur){
   if (cur.moveToFirst()) {
      String name;
      String phoneNumber;
      int nameColumn = cur.getColumnIndex(People.NAME);
      int phoneColumn = cur.getColumnIndex(People.NUMBER);

      do {
         // Get the field values
         name = cur.getString(nameColumn);
         phoneNumber = cur.getString(phoneColumn);
         // produce XML data
         backupXML +=
                "<entity>" +
                "<name>" + name + "</name>" +
                "<phone>" + phoneNumber +"</phone>" +
                "</entity>";
      } while (cur.moveToNext());
      // add </contact> to 'backupXML'
```

```
        backupXML += "</contact>";
    }
}
```

2) 获取通话记录（CallLog）

CallLog 的获取跟 Contact 很相似，同样，我们定义自己的 Content Resolver，并对 android.provider.CallLog 这个 Content Provider 进行查询，返回出我们想要的数据集：

```
Uri allCalls = Calls.CONTENT_URI;
Cursor curCallLog = cr.query(allCalls, null, null, null, Calls.DATE + "
DESC");
getCallLogData(curCallLog);
```

在 getCallLogData(Cursor cur)中，由于不全是字符串类型的，我们还需要进行一定的类型转化：

```
private void getCallLogData(Cursor cur){
    if (cur.moveToFirst()) {
        String strNumber, strDate, strType, strName, strDuration;
        long nDate;
        int nType;

        // Retrieve the column-indexes:
        // phoneNumber, date, type, name and duration
        // all return String type
        int numberColumn = cur.getColumnIndex(Calls.NUMBER);
        int dateColumn = cur.getColumnIndex(Calls.DATE);
        int typeColumn = cur.getColumnIndex(Calls.TYPE);
        int nameColumn = cur.getColumnIndex(Calls.CACHED_NAME);
        int durationColumn = cur.getColumnIndex(Calls.DURATION);

        do {
            // number
            strNumber = cur.getString(numberColumn);
```

```
            // date
            nDate = cur.getLong(dateColumn);
            strDate = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date(nDate));
            // type
            nType = cur.getInt(typeColumn);
            switch (nType)
            {
            case Calls.INCOMING_TYPE:
                strType = "Incoming";
                break;
            case Calls.MISSED_TYPE:
                strType = "Missed";
                break;
            case Calls.OUTGOING_TYPE:
                strType = "Outgoing";
                break;
            default:
                strType = "Error";
                break;
            }
            // name
            strName = cur.getString(nameColumn);
            // duration
            strDuration = cur.getString(durationColumn);
            // produce call log XML data
            calllogXML +=
                   "<entity>" +
                   "<name>" + strName + "</name>" +
                   "<number>" + strNumber +"</number>" +
```

```
                "<type>" + strType + "</type>" +
                "<date>" + strDate + "</date>" +
                "<duration>" + strDuration + "</duration>" +
                "</entity>";
        } while (cur.moveToNext());
        calllogXML += "</calllog>";
        Log.i(TAG, calllogXML);
    }
}
```

3) 获取短消息（Sms）

由于 Android 系统没为短消息提供相应的 Content Provider，所以获取该数据相对有些复杂。我们使用 content://sms/inbox 这个 URI 来作为我们获取 Sms 的接口：

```
Uri allSmsInbox = Uri.parse("content://sms/inbox");
Cursor curSms = cr.query(allSmsInbox,
        new String[] { "address", "person", "date", "body" },
        null,  null,  "date DESC");
getSmsData(curSms);
```

由于返回的数据集，并不包含【发信人】这个属性中的字符串，而是 Contact 中相应的 ID 号，所以还需要在 android.provider.Contacts 中进行一次联合查询：

```
private void getSmsData(Cursor cur) {
    if (cur.moveToFirst()) {
        long nDate;
        long nName;
        String strBody, strName, strNumber, strDate;
        ContentResolver oneCR = getContentResolver();
        do {
            // get Number from "address"
            strNumber = cur.getString(0);            // 手机号
            // get Name from "person"
            nName = cur.getLong(1);
```

```
          /*   get query in People   */
          oneName = ContentUris.withAppendedId(People.CONTENT_URI,
nName);
          // Then query for this specific record
          Cursor curName = oneCR.query( oneName,
                 new String[] { People.NAME },
                 null, null, null);
          if (curName.moveToFirst()) {
             strName = curName.getString(0);
          } else {
             strName = "Unknown";
          }
          // get Date from "date"
          nDate = cur.getLong(2);
          strDate = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date(nDate));
          // get Body from "body"
          strBody = cur.getString(3);
          // produce sms XML data
          smsXML +=
             "<entity>" +
             "<name>" + strName + "</name>" +
             "<number>" + strNumber +"</number>" +
             "<date>" + strDate + "</date>" +
             "<body>" + strBody + "</body>" +
             "</entity>";
       } while (cur.moveToNext());
       smsXML += "</sms>";
       Log.i(TAG, smsXML);
    }
```

```
}
```

4) 获取地理位置（Location）

Android 系统提供了 Location 类，可以方便地利用手机的 GPS 功能获取手机所处地的地理位置信息。

我们首先定义了 LocationManager mgr 来获得系统的位置服务，并定义 double latitude 和 double longitude 来保存获取的经纬度。best 用于保存系统当前可调用的最佳的信息提供者。

```
private LocationManager mgr;
private String best;
private double latitude;
private double longitude;
```

运行该模块时，首先启动系统的位置服务，然后利用当前最佳的信息提供者，调用 dumpLocation( )方法获取地理位置信息。

```
mgr = (LocationManager)getSystemService(LOCATION_SERVICE);
dumpProviders();
Criteria criteria = new Criteria();
best = mgr.getBestProvider(criteria, true);
Location location = mgr.getLastKnownLocation(best);
dumpLocation(location);
snedGPSData(latitude,longitude);
```

dumpLocation( )方法具体实现如下。

```
private void dumpLocation(Location loc){
    if(loc == null){
    }
    else{
        latitude = loc.getLatitude();     //获取纬度
        longitude = loc.getLongitude();   //获取经度
    }
}
```

3.3.4 隐私数据提交模块

数据提交，就是将带有 KEY 的 XML 数据包发送到相应的 Web 接收端，用不同的网址 strURL 来标识。

我们定义了 sendData(String strXML, String strURL)这个方法来将 XML 数据包 strXML 发送到指定的网址 strURL。

```
// send XML data to server
private void sendData(String strXML, String strURL) {
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(strURL);
    try {
        StringEntity entityXML;
        entityXML = new StringEntity(strXML, "utf-8");
        entityXML.setContentEncoding("utf-8");
        entityXML.setContentType("text/xml");
        httpPost.setEntity(entityXML);
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    try {
        // Execute HTTP Post Request
        HttpResponse response = httpClient.execute(httpPost);
        // if 201, return and also log the response
        if (response.getStatusLine().getStatusCode() ==
HttpStatus.SC_CREATED) {
            // 释放连接
            httpClient.getConnectionManager().shutdown();
        }
    } catch (ClientProtocolException e) {
        e.printStackTrace();
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

对于位置信息，由于传输数据量少，所以采用键值对的方式直接传输，由snedGPSData( )方法实现。

```
protected boolean snedGPSData(double latitude,double longitude){
    HttpClient httpClient = new DefaultHttpClient();
    HttpPost httpPost = new HttpPost(gpsURL);
    List<NameValuePair> nvPair = new ArrayList<NameValuePair>();
    nvPair.add(new                    BasicNameValuePair("latitude",
Double.toString(latitude)));
    nvPair.add(new                    BasicNameValuePair("longitude",
Double.toString(longitude)));
    nvPair.add(new BasicNameValuePair("nid",strKey));
    try {
        //连接与数据上传
        ……
    }
}
```

3.3.5 指令获取与解析模块

目前服务端远程控制手机就是通过短信 Push 的方式，客户端收到特定的短信指令，则进行解析，然后做相应的操作！

我们定义了一个 Receiver，并且向系统进行注册，来接收短信到达的事件：

```
<receiver android:name=".SMSReceiver" android:enabled="true">
        <intent-filter>
                <action
android:name="android.provider.Telephony.SMS_RECEIVED" />
```

```
        </intent-filter>
</receiver>
```

一旦有指令短信发来时，就会被我们的 SMSReceiver 这个 BroadcastReceiver 类接收到，然后首先提取出短信的具体内容：

```
Bundle bundle = intent.getExtras();
SmsMessage[] smsMsgs = null;

if (bundle != null) {
    Object[] pdus = (Object[]) bundle.get("pdus");
    smsMsgs = new SmsMessage[pdus.length];

    for (int i=0; i<smsMsgs.length; i++){
        smsMsgs[i] = SmsMessage.createFromPdu( (byte[])pdus[i] );
    }
    Intent myIntent = new Intent(context, MobileSecService.class);
```

接着，判断这条短信是不是我们的指令短信，并且提取出指令的内容和相应的 ID 号：

```
//根据的是消息的内容，判断是否有我们要处理的指令
for (SmsMessage message : smsMsgs) {
    String strMsg = message.getMessageBody();

    if ( strMsg.startsWith("ms:--") && strMsg.endsWith("--") ) {
        if (DEBUG)      Log.v(TAG, "It is our cmd!");

        String regexSplit = "--";
        String[] strMsgs = strMsg.split(regexSplit);
        String strCmd = strMsgs[1];                  // "b1"
        String strID = strMsgs[2];                   // "66001"

        myIntent.putExtra("ID", strID);
```

最后，根据指令内容，分派我们的后台监控服务 MobileSecService 执行相应的操作。下面仅以备份数据的指令分派做示例，其他功能的指令分派类似，不再赘述。

```
//------------------------------
// backup data
//------------------------------
// backup Contacts
if ( strCmd.equals("b0") ) {
    if (DEBUG) Log.i(TAG, "ready to backup contact!");
    myIntent.putExtra("cmd", 0);
    context.startService(myIntent);
}
// backup Sms
if ( strCmd.equals("b1") ) {
    if (DEBUG) Log.i(TAG, "ready to backup sms!");
    myIntent.putExtra("cmd", 1);
    context.startService(myIntent);
}
// backup CallLogs
if ( strCmd.equals("b2") ) {
    if (DEBUG) Log.i(TAG, "ready to backup calllog!");
    myIntent.putExtra("cmd", 2);
    context.startService(myIntent);
}
```

3.3.6 后台监控服务模块

该模块把以上几个功能模块整合到一起，作为后台服务运行在 Android 系统里，出现相应的情况就去执行相应的操作。

既然要时刻保护手机信息的安全，就必须要求该服务开机时就能自启动，从而监管手机从开机到关机的整个生命周期。

Android 系统中有个 intent 的概念，an abstract description of an operation to be performed，是用来描述将要被执行的一组操作。而 Android 开机这个事件就被描述为这样一个 intent：android.intent.action.BOOT_COMPLETED。

所以，我们只要用 BroadcastReceiver 来接收这个 intent，并在 onReceive (Context context, Intent intent)方法中启动我们的后台服务，这样就达到了开机自启动的效果。

首先，我们需要在 AndroidManifest.xml 中定义我们想要接收 android.intent.action.BOOT_COMPLETED 这个 intent：

```
<receiver android:name=".StartupIntentReceiver" android:enabled="true">
   <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
      <category android:name="android.intent.category.HOME" />
   </intent-filter>
</receiver>
```

并且，执行这个操作是需要一定权限的，我们在 AndroidManifest.xml 中定义这个权限，代表有权来接收这个 intent：

```
<uses-permission android:name="android.permission.
                              RECEIVE_BOOT_COMPLETED" />
```

最后，我们定义一个继承 StartupIntentReceiver 的类来开启我们的后台监控服务：

```
public class StartupIntentReceiver extends BroadcastReceiver {
       @Override
       public void onReceive(Context context, Intent intent) {
       Intent mBootIntent = new Intent(context,MobileSecure.class);
              mBootIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
              context.startActivity(mBootIntent);
       }
}
```

3.4 Web 服务端

3.4.1 前台用户操作页面

该模块的基本思想是：从后台数据库中读取待显示的信息，然后动态生成页面，显示给用户。并且，给用户提供一些控制接口，从而远程控制手机执行一定的操作！

该模块共有八个前台显示页面，其对应关系如下：

主页：见图 3.9

锁定：让用户锁定或解锁手机，见图 3.10

备份：让用户选择是否要备份信息，见图 3.11

删除：让用户选择是否要删除信息，见图 3.12

追踪：让用户追踪手机的地理位置 ，见图 3.13

通讯录：显示用户手机中的 Contact，见图 3.14

短消息：显示用户手机中的 Sms，见图 3.15

通话记录:显示用户手机中的 CallLog，见图 3.16

图 3.9　主页

图 3.10 锁定

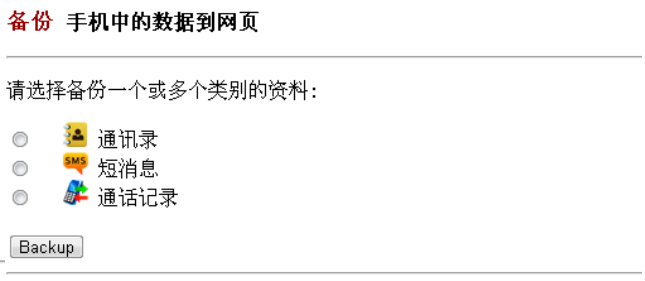

图 3.11 备份

图 3.12 删除

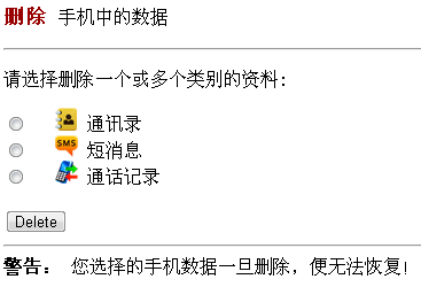

图 3.14 通讯录

图 3.13 追踪地理位置

加载您的短消息到网页

你的短消息记录为：

> >

发信人	手机号	日期	消息内容
Wuzhigang	13914333770	2010-03-27 17:15:58	Hi,I am wu. Just for test!
Wudaoyuan	13770927023	2010-03-27 14:22:45	Hello, I am Wudaoyuan!

图 3.15　短消息

加载您的通话记录到网页

你的通话记录为：

通话人	手机号	呼叫类型	呼叫时间	通话时间(s)
Wudaoyuan	13770927023	Incoming	2010-03-27 13:43:30	12
Test	15555218135	Incoming	2010-03-27 13:42:05	22

图 3.16　通话记录

我们使用 Django 模板系统，使得前台 HTML 页面和后台逻辑能够分离得很好。比如，我们的主页中，只要将 username 和 url 传过去，就能使用户处于不同的状态下来访问主页得到的效果是不一样的。如下就是我们的主页后台 Python 代码，当用户处于未登录状态时，能够自动跳转到 Google 验证页面：

```
class MainPage(webapp.RequestHandler):
   def get(self):
      user = users.get_current_user()
      if user:
         username = user.nickname()
         url = users.create_logout_url(self.request.uri)
         # vals in templates
         template_values = {
            'username': username,
            'url': url,
         }
```

```
            path = os.path.join( os.path.dirname(__file__),
                                 'templates/index.html')
            self.response.headers['Content-Type'] = 'text/html;
                                 charset=GBK'
            self.response.out.write( template.render(path,
                                 template_values) )
        else:
            url = users.create_login_url(self.request.uri)
            self.redirect(url)
```

3.4.2 后台数据存储模块

Web 服务端的关键在于各种数据表的设计，因为这将直接决定着后台逻辑的复杂度。整个 Web 服务端共有 7 张数据表，它们的属性和作用如下：

1) Login 表存储客户端传过来的 email 和 password 的哈希值，它的 KEY 作为其他表的外键。

```
class Login(db.Model):
    email = db.StringProperty()
    pswd = db.StringProperty()
```

2) Phone 表存储手机当前的 SIM 卡号，以及它的锁定状态。

```
class Phone(db.Model):
    nid = db.ReferenceProperty(Login)        # KEY Properties
    num = db.StringProperty()                # 当前 SIM 卡号
    lock = db.StringProperty()               # 是否锁定
```

3) Contact 表存储手机中的所有通讯录，包括名字和手机号。

```
class Contact(db.Model):                     # backup table
    nid = db.ReferenceProperty(Login)        # KEY Properties
    name = db.StringProperty()               # every contact's name
    phone = db.StringProperty()              # every contact's phone
```

4) CallLog 表存储手机中的所有通话记录，包括通话时间等属性。

```
class CallLog(db.Model):
    nid = db.ReferenceProperty(Login)          # KEY Properties
    name = db.StringProperty()                 # person called
    number = db.StringProperty()
    type = db.StringProperty()                 # 是主叫还是被叫
    date = db.StringProperty()
    duration = db.StringProperty()             # 通话持续时间
```

5) Sms 表中存储手机中的所有短消息。

```
class Sms(db.Model):
    nid = db.ReferenceProperty(Login)
    name = db.StringProperty()                 # <name>
    number = db.StringProperty()               # <number>
    date = db.StringProperty()                 # <date>
    body = db.StringProperty()                 # <body>
```

6) Location 表中定义用户追踪手机的地理位置历史（经度和纬度）

```
 class Location(db.Model):
    nid = db.ReferenceProperty(Login)          # KEY Properties
    longitude = db.StringProperty()            #经度
    latitude = db.StringProperty()             #纬度
    date = db.DateTimeProperty(auto_now_add=True) #追踪时间
```

7) Cmd 表中定义服务端要向手机客户端下达的指令。

```
class Cmd(db.Model):
    message = db.StringProperty()              # 包含命令的字符串
   nid = db.ReferenceProperty(Login)       # KEY Properties
```

3.4.3 手机数据接收模块

该模块从 Android 客户端接收提交过来的数据，保存到数据库中，并返回给客户

端成功 or 失败。共有 6 个 Handler 来处理，其与 URL 的对应关系如下：

- ('/rpc/lock', LockHandler)，接收锁机状态；
- ('/rpc/backup', BackUpHandler)，接收通讯录信息；
- ('/rpc/calllog', CallLogHandler)，接收通话记录信息；
- ('/rpc/sms', SmsHandler)，接收短消息；
- ('/rpc/login', LoginHandler)，接收用户 Email 和 Password；
- ('/trace/location',LocationHandler),接收 longitude 和 latitude;

以提取 CallLog 为例，一个典型的处理过程如下。

1) 首先提取出 HTTP 数据包中的 XML 信息：

```
class CallLogHandler(webapp.RequestHandler):
    def post(self):
        strXML = self.request.body
        docXML = minidom.parseString(strXML)
```

2) 然后从 XML 中提取出 KEY：

```
        key = db.Key( docXML.getElementsByTagName('calllog')[0].
                                    getAttribute('nid') )
```

3) 接着，遍历解析整个 XML 文档，循环提取出 CallLog：

```
        nodeList = docXML.getElementsByTagName('entity')
        dataList = []
        for node in nodeList:
            calllog = CallLog()
            calllog.nid = key
            calllog.name = node.firstChild.firstChild.data
            calllog.number = node.childNodes[1].firstChild.data
            calllog.type = node.childNodes[2].firstChild.data
            calllog.date = node.childNodes[3].firstChild.data
            calllog.duration = node.lastChild.firstChild.data
            dataList.append(calllog)
        db.put(dataList)
        # also response data to client
```

```
        self.response.set_status(201)
```

3.4.4 控制指令生成和发送模块

首先，有必要对控制指令的格式做一些解释。在 Cmd 表中，属性 message 代表要下达的指令。格式如下：

```
# backup: 'b';
# delete: 'd';
# 0: contact, 1: sms, 2: calllog
```

所以，如果是要备份 Sms 的话，则命控制指令为：'b1'。

Lock、Backup、Wipe 和 Trace 页面所提交的表单请求都提交到 http://androidmobilesec.appspot.com/wipe/action/cmd.do 下面就看看 CmdAction 的具体处理过程：

```
class CmdAction(webapp.RequestHandler):
    def get(self):
        user = users.get_current_user()
        if not user:
            url = users.create_login_url(self.request.uri)
            self.redirect(url)
        else:
            email = user.email()
            queryLogin = db.GqlQuery("SELECT __key__ FROM Login WHERE email
= :1", email)
            # 最多从数据存储区中抓取 1 个结果
            key = queryLogin.get()
            self.response.out.write(key)
            strQuery = self.request.query_string
            # 从 url 中提取出参数, add 'delete'
            requestParams = CmdAction.getRequestParams(strQuery,
['contact', 'sms', 'calllog', 'backup', 'delete'])
```

```
          if requestParams.has_key("backup"):
              strCmd = 'b'
              if requestParams.has_key("contact"):
                  strCmd += '0'
              if requestParams.has_key("sms"):
                  strCmd += '1'
              if requestParams.has_key("calllog"):
                  strCmd += '2'
              cmd = Cmd()
              cmd.nid = key
              cmd.put()

self.redirect('http://androidmobilesec.appspot.com/backup')
          elif requestParams.has_key("delete"):
              strCmd = 'd'
              if requestParams.has_key("contact"):
                  strCmd += '0'
              if requestParams.has_key("sms"):
                  strCmd += '1'
              if requestParams.has_key("calllog"):
                  strCmd += '2'
              cmd = Cmd()
              cmd.message = strCmd
              cmd.nid = key
              cmd.put()

self.redirect('http://androidmobilesec.appspot.com/wipe')
          else:
```

指令生成完成后则调用移动官方 Fetion API 自动发送到目的手机:

```
        url = "http://sms.api.bz/fetion.php?username=发出手机&password=飞信登陆密码&sendto=目的手机&message="+指令
        urlfetch.fetch(url)
```

3.5 关键技术细节和重要思想

3.5.1Web Service

相对于没有手机客户端和 Web 服务端概念的同类产品来说，Web Service 技术是本系统使用的一项重要的技术。

Web Service 是由开发者发布的完成其特定需求的在线应用服务，用户能够通过 Internet 来访问并使用这项在线服务。用户无需受到硬件平台的限制，也无需下载安装，只要有网络，有浏览器即可使用服务，这大大降低了服务的使用门槛。这符合当今基于云端技术的 SaaS（Software as a service），即软件即服务的思想，也是软件开发的最新趋势。

在本系统中，用户可以方便地在 Web 页面上使用我们提供的安全服务，操作简单，自主性强。每种安全服务仅需轻点两三次鼠标，即可由服务器和手机客户端自动完成。没有 Web 服务端的产品则需要通过人工服务或者要求用户经历多步操作等较为复杂的方式进行，这大大降低了软件的易用性，或者提高了服务的运营维护成本。

本系统的 Web Service 主要体现在 XML-RPC 技术上。XML-RPC 的全称是 XML Remote Procedure Call，即 XML 远程方法调用。这种远程过程调用使用 http 作为传输协议，XML 作为传送信息的编码格式。XML-RPC 的定义尽可能的保持了简单，但同时能够传送、处理、返回复杂的数据结构。本系统绝大多数的数据传输均采用了 XML-RPC 方式，这使得数据保持良好的格式，在可以传输结构复杂的数据的同时，却不影响数据的提取和使用。

3.5.2 客户端的唯一性

如何保证用户在 Web 端使用的服务就作用于用户注册的手机，而不受其他因素的干扰，即如何保证客户端的唯一性呢？下面两套机制保证了这一点。

首先，用户选择相应的服务后，在 Web 端产生的指令是以短信的形式发送到用

户的手机上的，这就保证了指令发送到了用户注册的手机上并且进行相应操作，而非别人的手机上。这里的唯一性是由运营商保证的。

其次，用户注册时服务端将返回给客户端一个全局唯一的 KEY（在 3.3.2 节的第一部分有详细阐述），即服务端授予客户端访问的权限。这个 KEY 是由 GAE 自动产生，并参与到数据上传和 Web 端的数据查询中，与数字证书类似，可以实现抗否认性。这里的唯一性是由 KEY 的全局唯一性保证的。

这两套机制保证用户注册后，客户端和 Web 端绑定在一起，从而保证了客户端的唯一性。

3.5.3 加密传输

虽然本系统的数据备份功能可以在一定程度上保护手机内用户隐私数据的安全，但在数据传输过程中却可能出现问题，比如数据中途被窃或丢失等安全性和完整性问题。

为此，我们采用了 SSL 加密传输技术。SSL（Secure Sockets Layer）即安全套接层，是为网络通信提供安全及数据完整性的一种安全协议。SSL 在传输层对网络连接进行加密，从而确保了数据的安全性。

SSL 协议可分为两层： SSL 记录协议：它建立在可靠的传输协议（如 TCP）之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。 SSL 握手协议：它建立在 SSL 记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。

SSL 协议提供的服务主要有：认证用户和服务器，确保数据发送到正确的客户机和服务器；加密数据以防止数据中途被窃取；维护数据的完整性，确保数据在传输过程中不被改变。SSL 提供的这些服务能够很好地解决数据传输过程中可能产生的问题。

当我们采用 SSL 方式传输数据时，我们的域名以及各个数据接收地址的协议就要由 http 变成 https。简单地讲是 http 的安全版，即 http 下加入 SSL 层。

3.5.4 锁定功能对手机屏幕和键盘的处理

我们采用了通过全屏界面屏蔽触屏和键盘虚拟码屏蔽键盘结合的方式处理锁定

功能。

全屏界面上无可点击的内容即可达到屏蔽触屏的效果。屏蔽键盘的难点在于屏蔽 Back 键和 Home 键。Back 键可以结束当前应用，Home 键可以导航至桌面。

我们创建一个监听接口，用于捕获按键所触发的消息，当其虚拟码与 Back 键和 Home 键的虚拟码匹配时，重新启动锁定界面，即可达到所要的效果。

在实测中，由于 Android 系统对 Home 键做了特殊处理，类似于 Windows 中的 Ctrl+Alt+Delete 组合键，拥有较高的级别，因此对 Home 键的屏蔽存在一定的缺陷。

第四章 性能测试

4.1 测试环境

4.1.1 手机客户端

型号：HTC　TATTOO (G4)　　CPU：Qualcomm MSM7225 528MHz

操作系统：Android 1.6　　Kernel 版本：2.6.29-g6561203

Android SDK 版本：Android SDK 1.6　　GPS：内置 GPS 天线

4.1.2 Web 服务端

Python 版本：Python 2.5　　Google Map API 版本：v2

4.2 功能模块测试

4.2.1 注册功能模块

清空使用历史遗留痕迹，对注册功能模块进行测试。若注册成功，返回“你已经成功注册，现在跳到主界面!”的字样，并跳转至主界面，如图 4.1 所示。同时，服务端也会显示注册信息，如图 4.2 所示。

图 4.1 注册成功图

4.2 Web 端的注册信息

4.2.2 信息保护模块

手机隐私信息保护功能模块主要有锁定手机，备份和删除通讯录、通话记录和短消息的功能。这里以备份通话记录为例进行测试。

如图 4.3 所示，手机中现有通话记录 2 条。

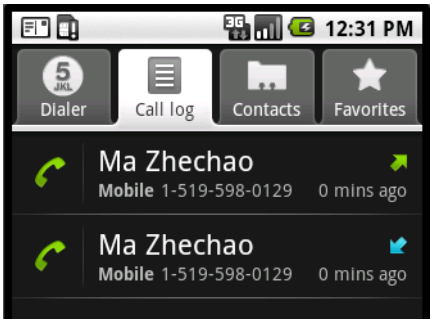

图 4.3 测试用通话记录

在 Web 段端进行备份操作后，待显示指令已完成后，如图 4.4 所示，可见手机中的通话记录已经备份到服务端，如图 4.5 所示。

图 4.4 备份指令已经执行完成

加载您的 通话记录 到网页

您的通话记录为：

通话人	手机号	呼叫类型	呼叫时间	通话时间(s)
Ma Zhechao	15195980129	Outgoing	2010-06-28 12:29:52	7
Ma Zhechao	15195980129	Incoming	2010-06-28 12:29:25	6

图 4.5 Web 端显示已备份的通话记录

4.2.3 定位追踪模块

在点击追踪并等待指令完成之后，可以查看到当前手机的地理位置，这里调用了 Google Map 来直观地显示手机的位置，如图 4.6 所示。

图 4.6 追踪到的手机的当前位置

4.3 测试结果

经过测试，本系统各项功能完整，运行正常。在数据传输一块还需进一步优化，以避免网络的不稳定对本系统造成大的影响。

第五章 创新性

本系统基于 Android 平台和 Google App Engine，较好地解决了日益严重的手机信息安全问题。比较市场上的保护手机信息安全的软件（如网秦公司的防盗卫士），本系统有以下几点优势：

1）现有的防盗卫士只有手机客户端，并没有 Web 服务端，于是它们对手机的各种远程操作都需要采用短信方式来进行数据传输，而短信能够传输的数据是很有限的（如不能备份智能手机中的文件），而且通过短信方式发送的数据格式也是很差的。而本系统则加入了 Web 服务端，手机客户端能够直接跟 Web 服务端建立 TCP 连接（且通过 SSL 加密），能够传输大批量的数据，且数据能够以友好的 Web 界面显示出来。

2）现有的防盗卫士需要用户记住大量拗口的短信控制指令，而本系统则提供了方便易用的 Web 控制界面，方便用户对手机信息进行远程控制。

3）现有的防盗卫士仅提供单一的用户隐私数据备份功能，如仅提供手机通讯录的备份，不能全方位地满足用户的安全需求。由于我们的产品在数据传输上采用了 TCP 连接传输的方式，因此可以提供多种数据备份功能，如手机通讯录，短信，通话记录等。而程序的良好的扩展性便于今后实现更多的功能，如手机内重要文件的备份等。

4）现有的防盗卫士大多只是在 Symbian 和 Windows Mobile 平台实现了，而我们这个手机客户端则采用了开源的 Android 平台，紧跟市场脚步，符合技术发展趋势。

5）现有的防盗卫士不具备手机追踪定位功能，而我们的系统可以实现手机追踪定位。

6）手机客户端和 Web 服务端均实现了 SSL 加密传输，对于保护用户的隐私数据有着积极的作用。

第六章 总结

6.1 报告内容总结

本报告先从总体的背景和设计思路入手，进而给出系统总体方案，然后描述了本系统所采用的技术，接着详述了各个模块的实现。最后，总结了一下作品的创新点。

6.2 已完成的主要工作

目前已经完成的主要工作有：

1）手机客户端：前台用户界面，用户注册与认证模块，SIM 卡开机检查模块，隐私数据获取、提交、删除模块，指令获取与解析模块，后台监控服务模块；

2）Web 服务端：前台用户操作页面，后台数据存储模块，手机数据接收模块，控制指令生成模块，定位追踪模块；

3）手机客户端和 Web 服务端的数据交互模块完成。

6.3 下一步的研发方向

我们将继续从以下几个方面对当前系统进行开发与研究：

1）完善数据传输模块，使数据传输更流畅，以应对较为恶劣的网络环境；

2）完善客户端功能，使用户可以在客户端上进行相应的操作；

3）利用Android操作系统开源的特点，在客户端做一些系统级的安全保护工作 ；

4）手机客户端在更多的手机平台上进行实现。

参考文献

[1] Android Developers, http://androidappdocs.appspot.com/index.html, 2008

[2] 姚尚朗 / 靳岩，《*Google Android开发入门与实战*》，人民邮电出版社，2009年7月1日

[3] 余志龙，《*Google Android SDK开发范例大全*》，人民邮电出版社，2009年

[4] 盖索林，《*Google Android开发入门指南*》，人民邮电出版社，2009年11月1日

[5] Wesley J.Chun，*Python Core Programming*，2006

[6] John Goerzen，*Foundations of Python Network Programming*，June 3, 2007

[7] Google App Engine - Google Code, http://code.google.com/appengine/, 2008

[8] Rick Rogers / John Lombardo / Zigurd Mednieks / Blake Meike，*Android Application Development*，May 26, 2009

[9] Frank Ableson / Charlie Collins / Robi Sen / Robert Cooper，*Unlocking Android*，January 28, 2009