**文档编号：sylincom\_JinHengTaiElectrombile\_1**

**云涌智成系统设计文档**

**拟制：马凯**

**时间：2015.04.15**

**北京中科晶上科技有限公司**

**业务系统集成组**

修改记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件编号 | 版本号 | 拟制人/  修改人 | 拟制/修改日期 | 更改理由 | 主要更改内容  （写要点即可） |
| sylincom\_JinHengTaiElectrombile\_1 | 1.0 | 马凯 | 2015/4/15 | 建立 |  |
| sylincom\_JinHengTaiElectrombile\_1 | 1.1 | 马凯 | 2015/12/5 | 添加 | 3.5.3添加离线地图内容 |
|  |  |  |  |  |  |
| 注1：每次更改归档文件（指归档到组内及研究室的文件）时，需填写此表。  注2：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。 | | | | | |

修改列表：

后续版本对之前版本所做修改逐条说明清楚。

本文档的程序或内容受版权法的保护，未经北京中科晶上科技有限公司的书面许可，不得擅自泄漏、拷贝或复制本文档资料的全部或部分。

引言 4

1. 项目概要 4

2. 编写目的 4

3. 项目背景 4

一、系统功能模块划分 4

二、系统架构设计 5

2.1．系统层次架构设计 5

2.2． 包结构及功能设计 5

三、系统详细设计 7

3.1 链接库及第三方类库准备 7

3.2 全局变量 8

3.3 网络请求 9

3.4 本地数据的存取 11

3.5 功能模块的实现 13

3.5.1 绑定 13

3.5.2 设置查询 16

3.5.3 跟踪定位 17

3.5.4 速度 18

3.5.5 电量 19

3.5.6 参数设置 21

3.5.7 恢复出厂设置 22

# 引言

## 项目概要

该项目是为金恒泰电动车开发的配套软件，使用该软件可以实现对与之绑定电动车的控制和信息获取。

## 编写目的

本文档定义XXX iOS端设计，包括系统架构设计、系统功能模块划分等。供系统编码人员、系统测试人员、系统维护人员、项目组负责人及相关管理人员使用。

编写此文档的目的在于：

* 为开发、测试、维护人员提供依据
* 为项目负责人控制系统设计开发的过程、安排工作进度提供依据

## 3. 项目背景

软件名称：云涌智成

软件开发者：北京中科晶上科技有限公司

开发工具：Xcode 6.3(6D570)

开发语言：Objective-C

开发硬件平台：MacBook Air (13-inch, Mid 2012)

开发软件平台：Mac OS X Yosemite 10.10.3

软件运行硬件平台：iPhone 4s及之后发行各版本iPhone

软件运行系统平台：iOS 7.0 +

# 系统功能模块划分

软件在实现与对应电动车的一对一绑定后，共提供六大功能：设置查询、定位追踪、速度、电量、参数设置和恢复出厂设置。见图1.1。



**图1.1 功能划分**

需要说明的是，定位追踪的子功能“安全范围设定”与参数设置中的子功能“非法位移距离设置”是属于一个功能两个接口。

# 二、系统架构设计

## 2.1．系统层次架构设计

本系统使用MVC架构实现，鉴于iOS SDK中特殊的UIViewController文件，View 与 Controller 在ViewController.h/ViewController.m 文件中集中实现。

Model 实现对与服务器数据交互类的封装。

## 2.2． 包结构及功能设计

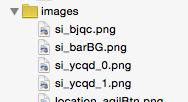
为方便对工程文件的管理，项目对实现不同功能的文件进行了分包管理。



**图2.1 第三方类库**

图2.1 是对工程引用的第三方类库的存储。

LoadingView类库实现加载指示器，BadiduMap类库引用了百度地图，ASIHttpRequest是一个非常高效易用的实现网络数据请求的类库。



**图2.2 包images**

包images存储工程用到的所有图片。



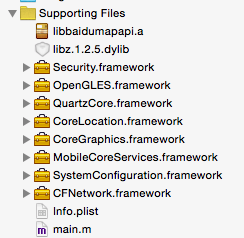
**图2.3 全局变量**

包全局变量存储工程中用到的存储全局变量的类。其中，PrefixHeader.pch是添加的系统文件，该文件中实现一些需要全局使用静态变量。AppInfo.h/AppInfo.m文件是通过单例模式实现的全局变量的存储。



**图2.4 网络请求**

网络请求包存储实现网络请求的封装类。其中，MKURL.h/MKURL.m类存储数据接口。MKHttpRequest.h/MKHttpRequest.m实现网络数据请求的逻辑处理。



**图2.5 Supporting Files**

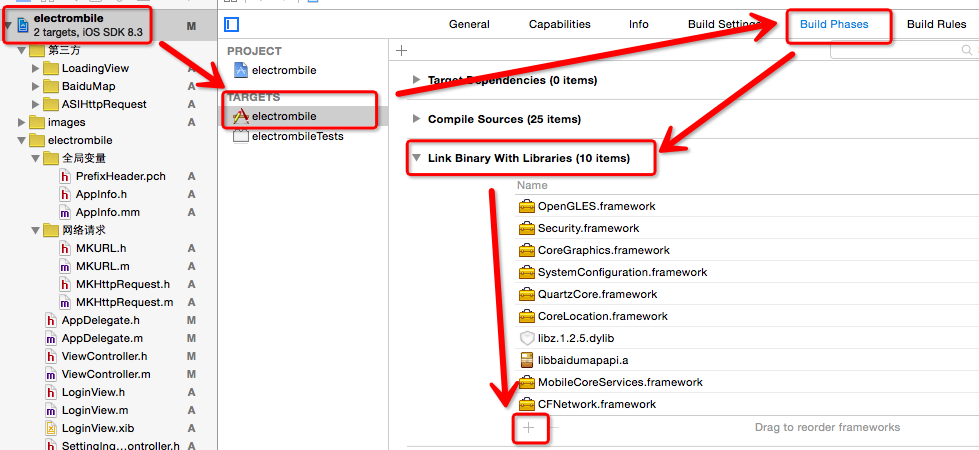
Supporting Files包存储引入的对工程运行起支撑作用的iOS SDK框架和链接库。其中，info.plist文件保存的工程的一些基本设置。

# 三、系统详细设计

## 3.1 链接库及第三方类库准备

新建工程初始引入的SDK 框架是基本的包含基本控件的UIKit，使用第三方类库或实现其他功能还需要因为其他的框架。参考图2.5。

添加过程如下：



**图3.1 添加框架**

本工程用到了三个第三方类库：LoadingView、BadiduMap、ASIHttpRequest。

LoadingView实现网络数据请求时指示器的指示功能。

BaiduMap是引用了百度地图API。

ASIHttpRequest封装了数据请求的一套高效易用的类库，极大的提高了开发效率。

相关类库可在官方网站或其他互联网网站上下载。

## 全局变量

**表3.1 PrefixHeader.pch 部分**

|  |
| --- |
| #define NSNotificationNameBind @"bind"  #define NSNotificationNameBindState @"send\_bind"  #define NSNotificationNameSend\_SFState @"send\_sf"  #define NSNotificationNameSend\_CFState @"send\_cf"  #define NSNotificationNameSend\_YCQDState @"send\_ycqd"  #define NSNotificationNameSend\_BJOState @"send\_bjo"  #define NSNotificationNameSend\_BJCState @"send\_bjc"  #define NSNotificationNameSend\_FORMATState @"send\_format"  #define NSNotificationNameSend\_ZDTXOState @"send\_zdtxo"  #define NSNotificationNameSend\_ZDTXCState @"send\_zdtxc"  #define NSNotificationNameSend\_AQJLState @"send\_aqjl"  #define NSNotificationNameSend\_DCTXOState @"send\_dctxo"  #define NSNotificationNameSend\_DCTXCState @"send\_dctxc"  #define NSNotificationNameSend\_YLTJState @"send\_yltj"  #define NSNotificationNameGPS @"gps"  #define NSNotificationNameSpeed @"speed"  #define NSNotificationNamePower @"power"  //#define SendAPI @"10.21.0.122:8080/MinaTCPServer"  //#define InfoStateAPI @"10.21.0.122:8080/GPSAndroidDataModel"  #define SendAPI @"159.226.40.218:8080/MinaTCPServer"  #define InfoStateAPI @"159.226.40.218:8080/GPSAndroidDataModel"  //发送指令后等待获取状态的时间  #define WAITTIME 1  //电动车最高车速  #define MAXSPEED 60 |

表3.1 贴出了文件PrefixHeader.pch中声明的部分静态变量。PrefixHeader.pch是系统文件，在此文件中声明的静态变量可在整个工程中当做全局静态变量使用。

其中以字符串“NSNotificationName”开头的变量是程序中会用的通告名称。SendAPI和InfoStateAPI是网络接口的一部分。WAITTIME是部分网络请求刷新数据的频率。MAXSPEED是速度模块用到的最大速度值，留有此接口方便修改。

类文件AppInfo.h/AppInfo.m是用单例模式实现的对全局变量的存储。说明见下表3.2：

**表3.2 AppInfo.h部分及说明**

|  |
| --- |
| //用户登录与否  @property (nonatomic, assign) BOOL isLogin;  //电动车硬件ID  @property (nonatomic, strong) NSString \*hardwareSim;  //用户手机号  @property (nonatomic, strong) NSString \*phoneSim;  //登录密码  @property (nonatomic, strong) NSString \*password;  //设防打开与否  @property (nonatomic, assign) BOOL SF\_OPEN;  //报警器开与关  @property (nonatomic, assign) BOOL BJQ\_OPEN;  //电动车位置纬度  @property (nonatomic, assign) float lat;  //电动车位置经度  @property (nonatomic, assign) float lon;  //电量报警开关与否  @property (nonatomic, assign) BOOL DLBJ\_OPEN;  //震动报警开关与否  @property (nonatomic, assign) BOOL ZDBJ\_OPEN;  //安全距离  @property (nonatomic, assign) int AQJL; |

## 网络请求

网络请求被封装在两个类中：MKURL和MKHttpRequest。

其中，MKURL.h/MKURL.m存储网络接口，说明见表3.3

**表3.2 AppInfo.h部分及说明**

|  |
| --- |
| //移动终端与硬件绑定接口  + (NSURL \*)bindURL;  //移动终端指令上传接口  + (NSURL \*)sendURL;  //GPS数据获取接口  + (NSURL \*)gpsURL;  //速度信息获取接口  + (NSURL \*)speedURL;  //电量信息获取接口  + (NSURL \*)powerURL;  //获取状态接口  + (NSURL \*)stateURL; |

MKHttpRequest.h/MKHttpRequest.m文件封装了网络数据请求的发起和接收逻辑。

**表3.3 MKHttpRequest.h/MKHttpRequest.m部分及说明**

|  |
| --- |
| **MKHttpRequest.h** |
| //移动终端与硬件绑定接口  + (void)httpRequestBindWithValue:(NSString \*)value notificationName:(NSString \*)notificationName;  //移动终端指令上传接口  + (void)httpRequestSendWithValue:(NSString \*)value;// notificationName:(NSString \*)notificationName;  //GPS数据获取接口  + (void)httpRequestGpsWithImsi:(NSString \*)imsi hardware\_id:(NSString \*)hardware\_id notificationName:(NSString \*)notificationName;  //速度信息获取接口  + (void)httpRequestSpeedWithImsi:(NSString \*)imsi hardware\_id:(NSString \*)hardware\_id notificationName:(NSString \*)notificationName;  //电量信息获取接口  + (void)httpRequestPowerWithImsi:(NSString \*)imsi hardware\_id:(NSString \*)hardware\_id notificationName:(NSString \*)notificationName;  //获取状态  + (void)httpRequestStateWithImsi:(NSString \*)imsi hardware\_id:(NSString \*)hardware\_id head:(NSString \*)head notificationName:(NSString \*)notificationName; |
| **MKHttpRequest.m** |
| //GPS数据获取接口  + (void)httpRequestGpsWithImsi:(NSString \*)imsi hardware\_id:(NSString \*)hardware\_id notificationName:(NSString \*)notificationName{    //声明并发送请求  ASIFormDataRequest \*request = [[ASIFormDataRequest alloc] initWithURL:[MKURL gpsURL]];  [request setPostValue:imsi forKey:@"imsi"];  [request setPostValue:hardware\_id forKey:@"hardware\_id"];  [request setTimeOutSeconds:5];  [request startAsynchronous];  //获取返回数据通过通告的方式发送  [request setCompletionBlock:^{  NSData \*data = [request responseData];  [[NSNotificationCenter defaultCenter] postNotificationName:notificationName object:data];  }];  //若获取的数据为空，则通过公告发送的数据也为空。  [request setFailedBlock:^{  [[NSNotificationCenter defaultCenter] postNotificationName:notificationName object:nil];  }];  } |

## 本地数据的存取

有关用户的数据保存在本地磁盘，因为数据量不大且都是简单的基本变量，因此采用类NSUserDefaults实现存取。

在文件AppDelegate.m中getNativeInfo方法实现本地数据的取出。

**表3.3 方法 getNativeInfo**

|  |
| --- |
| - (void)getNatvieInfo {    NSUserDefaults \*user = [NSUserDefaults standardUserDefaults];    [AppInfo appInfo].isLogin = [user boolForKey:@"isLogin"];  if ([AppInfo appInfo].isLogin) {  [AppInfo appInfo].phoneSim = [user stringForKey:@"phoneSim"];  [AppInfo appInfo].hardwareSim = [user stringForKey:@"hardwareSim"];  [AppInfo appInfo].password = [user stringForKey:@"password"];  [AppInfo appInfo].SF\_OPEN = [user boolForKey:@"SF\_OPEN"];  [AppInfo appInfo].BJQ\_OPEN = [user boolForKey:@"BJQ\_OPEN"];  [AppInfo appInfo].lat = [user floatForKey:@"lat"];  [AppInfo appInfo].lon = [user floatForKey:@"lon"];  [AppInfo appInfo].DLBJ\_OPEN = [user boolForKey:@"DLBJ\_OPEN"];  [AppInfo appInfo].ZDBJ\_OPEN = [user boolForKey:@"ZDBJ\_OPEN"];  [AppInfo appInfo].AQJL = (int)[user integerForKey:@"AQJL"];  }  } |

每次进行网络请求成功后，对本地保存的数据进行更新。

## 功能模块的实现

### 3.5.1 绑定



**图3.2 绑定界面**

LoginView是UIImageView的子类，界面主要由LoginView.xib文件实现。此视图添加了图3.2所示的必要控件，添加在ViewController视图之上，并由ViewController实现LoginView的代理方法：- (void)loginConnect;此方法是用户点击“连接”按钮时的触发函数。

该方法在进行必要的数据合法性验证之后通过MKHttpRequest发送指令：

|  |
| --- |
| NSString \*value = [NSString stringWithFormat:@"ZC#%@#%@#%@", [AppInfo appInfo].password, [AppInfo appInfo].phoneSim, [AppInfo appInfo].hardwareSim];  [MKHttpRequest httpRequestBindWithValue:value notificationName:NSNotificationNameBind]; |

发送指令时，在类中添加一个观察者Observer，对名为NSNotificationNameBind的通告进行监控，触发方法是：- (void)notificationBind:(NSNotification \*)noti

|  |
| --- |
| [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(notificationBind:) name:NSNotificationNameBind object:nil]; |

notificationBind:方法的实现见下表：

**表3.4 notificationBind:方法**

|  |
| --- |
| - (void)notificationBind:(NSNotification \*)noti {    [[NSNotificationCenter defaultCenter] removeObserver:self name:NSNotificationNameBind object:nil];    NSData \*data = [noti object];    if (data) {  NSDictionary \*dic = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingMutableLeaves error:nil];  NSLog(@"bind dic = %@", dic);    NSString \*state = [dic objectForKey:@"state"];  if ([state isEqualToString:@"success"]) {  [self performSelector:@selector(getBindState)];  }else if([state isEqualToString:@"error"]) {    NSLog(@"密码错误");    [self showMessage:@"绑定失败!" withSuccess:NO];  return;  }  }  } |

在该方法中对返回发送的绑定指令返回进行分析，如果state是successs说明指令发送成功，至于指令绑定成功与否，还需要再次进行数据请求查询指令执行结果。因此，还要在调用getBindState方法发送请求。同时，添加一个观察者Observer，在收到的返回中判断指令执行结果。

在本系统中，对于所有发送指令的的网络交互都有两个过程：



**图3.3 指令执行过程**

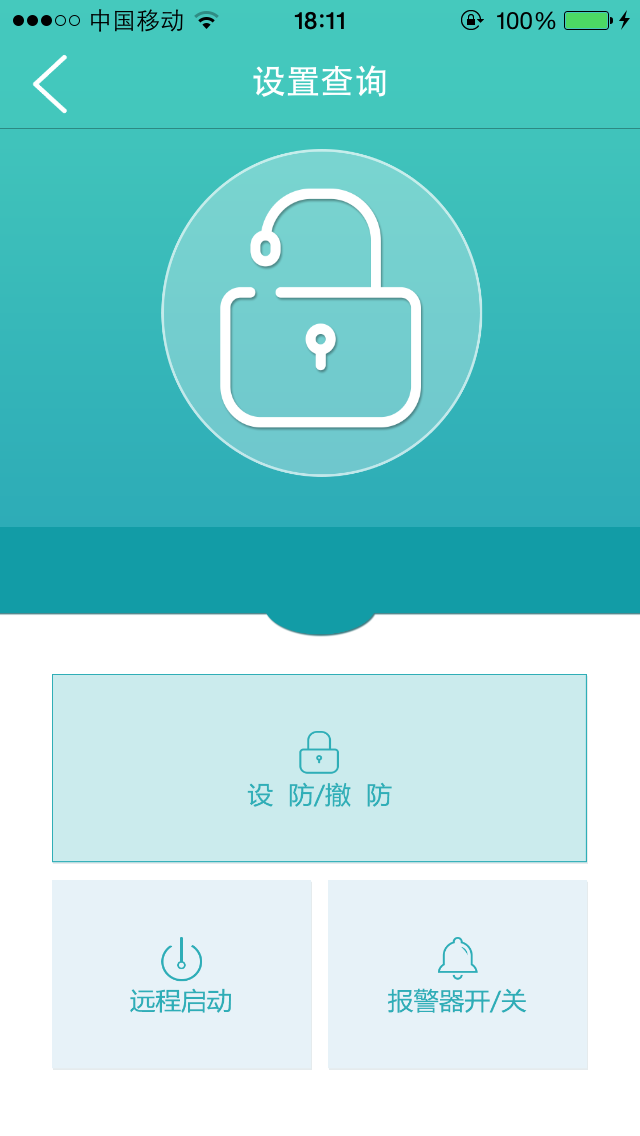
在下文3.5.2设置查询中三个子功能、跟踪定位的安全距离设定功能、参数设置的三个子功能以及恢复出厂设置的功能都遵循此流程。

如果绑定成功，LoginView就会消失，显示程序的主界面。



**图3.4 主界面**

### 3.5.2 设置查询



**图3.5 设置查询**

设置查询的界面主要由SettingInquireViewController.xib文件实现。

在发送指令并等待执行结果的过程中，使用第三方类库LoadingView实现指示器指示功能。需要注意的时，在等待的过程中，界面上的所有按钮应禁止工作，在接收到反馈后，恢复使用。由两个方法实现：

**表3.5 加载和停止指示器指示方法**

|  |
| --- |
| #pragma mark - start / stop LoadingView  - (void)startLoadingView {    [loadingView setAlpha:1];  [loadingView startAnimating];    [self.sfcfShowBtn setEnabled:NO];  [self.sfcfBtn setEnabled:NO];  [self.bjqkgShowBtn setEnabled:NO];  [self.bjqkgBtn setEnabled:NO];  [self.ycqdShowBtn setEnabled:NO];  [self.ycqdBtn setEnabled:NO];  }  - (void)stopLoadingView {    [loadingView setAlpha:0];  [loadingView stopAnimating];    [self.sfcfShowBtn setEnabled:YES];  [self.sfcfBtn setEnabled:YES];  [self.bjqkgShowBtn setEnabled:YES];  [self.bjqkgBtn setEnabled:YES];  [self.ycqdShowBtn setEnabled:YES];  [self.ycqdBtn setEnabled:YES];  } |

指令发送的逻辑在3.5.2中已表述。

### 3.5.3 跟踪定位



**图3.6 跟踪定位**

跟踪定位界面由类LocatinViewController实现。

在类初始化时，首先获取一次GPS位置数据，经过处理在百度地图上加载电动车位置图标，调用的方法是requestGps：

**表3.6 请求GPS信息方法**

|  |
| --- |
| [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(notificationGps:) name:NSNotificationNameGPS object:nil];    [MKHttpRequest httpRequestGpsWithImsi:[AppInfo appInfo].phoneSim hardware\_id:[AppInfo appInfo].hardwareSim notificationName:NSNotificationNameGPS]; |

请求GPS信息的逻辑是发送请求，等待反馈。没有获取发送指令状态的步骤，因为，获取信息不考虑指令执行成功与否。下文中，请求速度和电量的逻辑相同。

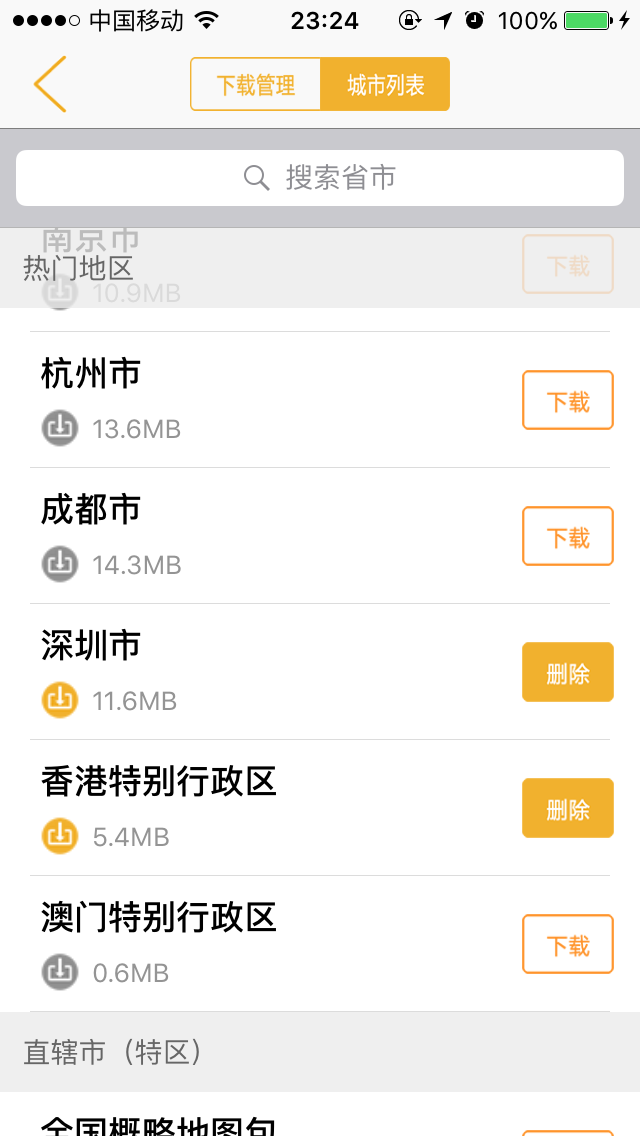
设定安全距离的视图如下：



**图3.7 设定安全距离**

此视图由AqjlView.xib文件实现，并在用户点击“安全范围设定”按钮时添加在定位追踪界面。并实现AqjlView的两个按钮代理方法。

离线地图由类OfflineMapViewController.h/m实现。共有两个切换界面，一个是下载管理，另一个是城市列表。在城市列表中，可以搜索选择响应的城市离线包进行下载，在下载管理页可以对下载的离线地图包进行管理。

（a） （b）

**图3.8 下载管理、城市列表**

类要实现代理BMKOfflineMapDelegate的方法：

- (void)onGetOfflineMapState:(int)type withState:(int)state.

不同的type代表不同的离线包操作，使用最多的是TYPE\_OFFLINE\_UPDATE，在下载地图过程中，该回调函数会不停的返回下载进度，在此使用方法[downloadManageTableView reloadData];[cityListTableView reloadData];更新界面显示。

### 3.5.4 速度

速度界面主要由SpeedViewController.xib文件实现。



**图3.8 速度**

每WAITTIME秒从服务器刷新一次速度信息，对获取的速度信息的处理见见方法- (void)handleWithSpeed :(float)speed。

### 3.5.5 电量

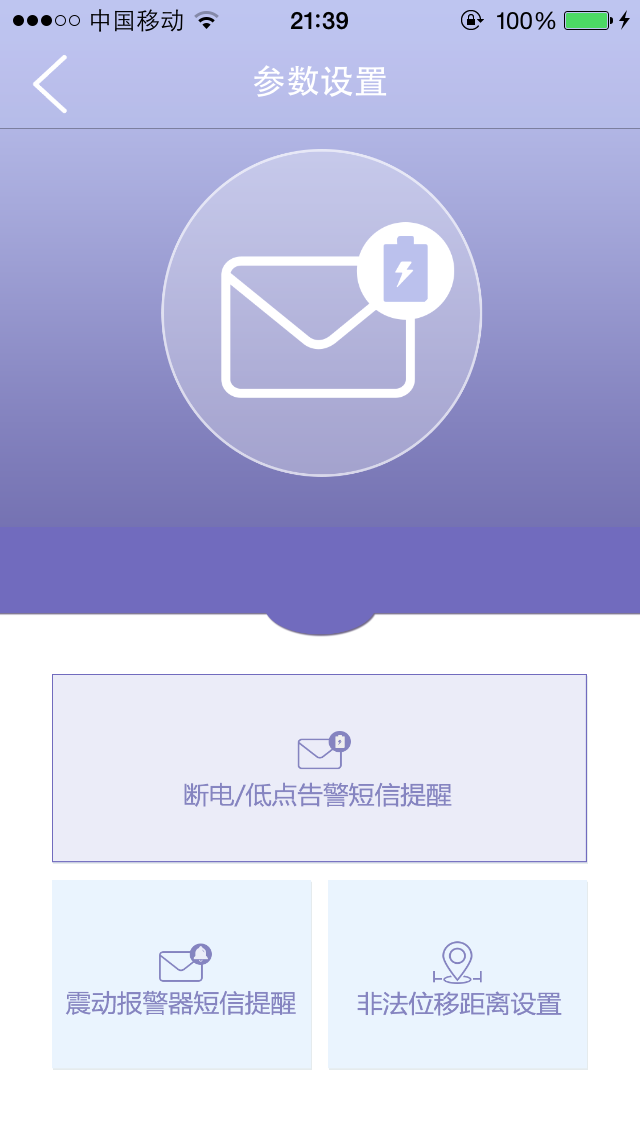
电量界面主要由文件PowerViewController.xib文件实现。



**图3.9 电量**

类初始化时，获取一次电量信息。

### 3.5.6 参数设置



**图3.10 参数设置**

设置查询的界面主要由RefSettingViewController.xib文件实现。

实现逻辑与3.5.2设置查询相同，此处不再赘述。

### 3.5.7 恢复出厂设置



**图3.11 恢复出厂设置**

恢复出厂设置主要由FormatViewController.xib实现。而弹出视图由FormatView.xib实现并加载。当然，在FormatViewController.m中实现FormatView类的代理方法- (void)formatOK 和 - (void)formatCancel。

如果出厂设置恢复成功，调用方法- (void)formatSuccess，在此方法中，要对本地保存的所有数据进行初始化，并退回到绑定界面。见下表：

**表3.7 方法formatSuccess**

|  |
| --- |
| - (void)formatSuccess {    self.formatLabel.text = @"恢复出厂设置成功！";  [self showMessage:@"恢复成功!"];    [AppInfo appInfo].isLogin = NO;      NSUserDefaults \*user = [NSUserDefaults standardUserDefaults];  [user setBool:[AppInfo appInfo].isLogin forKey:@"isLogin"];  [user setObject:@"" forKey:@"phoneSim"];  [user setObject:@"" forKey:@"hardwareSim"];  [user setObject:@"" forKey:@"password"];  [user setBool:NO forKey:@"SF\_OPEN"];  [user setBool:NO forKey:@"BJQ\_OPEN"];  [user setFloat:0 forKey:@"lat"];  [user setFloat:0 forKey:@"lon"];  [user setBool:NO forKey:@"DLBJ\_OPEN"];  [user setBool:NO forKey:@"ZDBJ\_OPEN"];  [user setInteger:0 forKey:@"AQJL"];    [user synchronize];  if ([user synchronize]) {  [self performSelector:@selector(backFormat) withObject:nil afterDelay:2];  }  } |