蓝牙4.0

什么是蓝牙

* 蓝牙，是一种支持设备短距离通信的无线电技术，主要用于相关外设的无线信息交换，利用蓝牙技术，能够有效的简化移动终端的之间的通信，蓝牙支持点对点，点对多点的通信，与wifi等通信一样，工作在2.4GHz下，随着移动终端的火热，物联网也依靠着移动终端飞速发展

蓝牙的版本

* 蓝牙分为2.0和4.0版本
* 他们有什么区别呢?
* 2.0是最早的蓝牙版本需要进行蓝牙配对后才可以连接，功率消耗比较大
* 4.0是2012年最新蓝牙版本，更省电，3毫秒延迟，超长连接距离，最远100米，建议30米内有效距离

* 安卓上面大部分蓝牙依然是2.0版本，只有少部分支持了4.0版本

* iOS上4s之前的版本只能支持2.0版本，在4s包括4s之后，都可以支持蓝牙4.0

iOS蓝牙2.0

iOS在蓝牙方面一直管控的比较严格，在2.0的版本，需要通过苹果官方的MFi认证，MFi认证在苹果官方认证后，苹果会提供相应的芯片给厂商，才能够用于生产，这个MFi认证的在苹果官方是收费的，需要9万美元一个产品

蓝牙4.0

* 蓝牙4.0至少需要设备iphoen在4s以上，ipad在2以上，并且要求系统版本不低于iOS5.1才可以，使用coreBluetooth框架

ZCPeripheralManager 周边设备

* 名词说明
* 周边设备可以理解为数据发送
* CBPeripheralManager 周边设备管理类
* CBMutableCharacteristic 可变服务特性
* 主要过程
* 1、创建一个CBMutableCharacteristic特性 加入到CBMutableService的characteristics的服务上
* 2、把服务加到管理CBPeripheralManager
* 3、开始发送广播

初始代码

* 创建管理类
* self.peripheralManager=[[CBPeripheralManager alloc]initWithDelegate:self queue:nil];
* 检测状态代理
* //检测状态
* -(void)peripheralManagerDidUpdateState:(CBPeripheralManager *)peripheral{

* }

在检测状态函数内

```
//检测状态
-(void)peripheralManagerDidUpdateState:(CBPeripheralManager *)peripheral
{
    if (peripheral.state!=CBPeripheralManagerStatePoweredOn) {
        return;
    }
    //启动service
  //启动可变服务特性properties：Notify允许没有回答的服务特性，向中心设备发送数据，permissions：read通讯属性为只读
  self.transferCharacteristic=[[CBMutableCharacteristic alloc]initWithType:[CBUUID
UUIDWithString:TRANSFER_CHARACTERISTIC_UUID] properties:CBCharacteristicPropertyNotify value:nil
permissions:CBAttributePermissionsReadable];
    //创建服务primary 是首次还是第二次
  CBMutableService*transferService=[[CBMutableService alloc]initWithType:[CBUUID
UUIDWithString:TRANSFER_SERVICE_UUID] primary:YES];
    //把特性加到服务上
  transferService.characteristics=@[self.transferCharacteristic];
    //把服务加到管理上
    [self.peripheralManager addService:transferService];

    //发送广播，标示是TRANSFER_SERVICE_UUID为对方观察接收的值，2边要对应上

     [self.peripheralManager startAdvertising:@{CBAdvertisementDataServiceUUIDsKey:@[[CBUUID
UUIDWithString:TRANSFER_SERVICE_UUID]]}];
}
```

连接代码

```
//订阅成功
-(void)peripheralManager:(CBPeripheralManager *)peripheral central:(CBCentral *)central didSubscribeToCharacteristic:(CBCharacteristic *)characteristic
{
    NSLog(@"订阅成功");
}
//当中央设备结束订阅时候调用
-(void)peripheralManager:(CBPeripheralManager *)peripheral central:(CBCentral *)central didUnsubscribeFromCharacteristic:(CBCharacteristic *)characteristic
{

}
```

发送数据代码

```
-(void)sendDataClick{

    if (sendingEOM) {
        //第三个参数代表指定与我们的订阅的中心设备发送，返回一个布尔值，代表发送成功
    BOOL didSend=[self.peripheralManager updateValue:[BluetoothEnd dataUsingEncoding:NSUTF8StringEncoding] forCharacteristic:self.transferCharacteristic
onSubscribedCentrals:nil];

        if (didSend) {
            //全部发送完成
     sendingEOM=NO;
            NSLog(@"发送完成");
            self.BlockResult(1);
        }
        //如果没有发送，我们就要退出并且等待
    //peripheralManagerIsReadyToUpdateSubscribers 来再一次调用sendData来发送数据
    return;
     }
     //如果没有正在发送BluetoothEnd，就是在发送数据
  self.sendData=[self.message dataUsingEncoding:NSUTF8StringEncoding];
    //判断是否还有剩下的数据
  if (self.sendDataIndex>=self.sendData.length) {
        //没有数据，退出即可
    return;
     }
     //如果有数据没有发送完就发送它，除非回调失败或者我们发送完
  BOOL didSend=YES;
    while (didSend) {
        //发送下一块数据，计算出数据有多大
    NSInteger amountToSend=self.sendData.length-self.sendDataIndex;
        if (amountToSend>NOTIFY_MTU) {
            //如果剩余的数据还是大于20字节，那么我最多传送20字节
     amountToSend=NOTIFY_MTU;
        }
        //切出我想要发送的数据 +sendDataIndex就是从多少字节开始向后切多少
    NSData*chunk=[NSData dataWithBytes:self.sendData.bytes+self.sendDataIndex length:amountToSend];
        //发送
    didSend=[self.peripheralManager updateValue:chunk forCharacteristic:self.transferCharacteristic onSubscribedCentrals:nil];
        //如果没发送成功，等待回调发送
    if (!didSend) {
            return;
        }else{
            self.sendDataIndex+=amountToSend;
            //判断是否发送完
     if (self.sendDataIndex>=self.sendData.length) {
                //发送完成，就开始发送结束标示bluetoothEND
       sendingEOM=YES;
                [self performSelector:@selector(sendDataClick) withObject:nil afterDelay:0.1];
//                //发送
            }
        }

    }
}
```

ZCCentralManager中央设备

* 名词说明
* 中央设备也叫做接收类，负责接收消息

初始化

* `self.centralManager = [[CBCentralManager alloc] initWithDelegate:self queue:nil];`

开启检测服务

```
//检测中央设备状态
-(void)centralManagerDidUpdateState:(CBCentralManager *)central
{

    if (central.state!=CBCentralManagerStatePoweredOn) {
        //如果蓝牙关闭，那么无法开启检测，直接返回
    NSLog(@"蓝牙关闭");
        return;
    }
    //开启检测
    [self scan];

}
-(void)scan{
    [self.centralManager scanForPeripheralsWithServices:@[[CBUUID
UUIDWithString:TRANSFER_SERVICE_UUID]]

options:@{ CBCentralManagerScanOptionAllowDuplicatesKey : @YES }];
    //options中的意思是否允许中央设备多次收到曾经监听到的设备的消息，这样来监听外围
设备联接的信号强度，以决定是否增大广播强度，为YES时会多耗电

}
```

发现周边设备

```
//当外围设备广播同样的UUID信号被发现时，这个代理函数被调用。这时我们要监测RSSI即Received Signal Strength Indication接收的信号强度指示，确保足够近，我们才连接它
- (void)centralManager:(CBCentralManager *)central didDiscoverPeripheral:(CBPeripheral *)peripheral advertisementData:(NSDictionary *)advertisementData RSSI:(NSNumber *)RSSI
{


    NSLog(@"Discovered %@ at %@", peripheral.name, RSSI);

    // 判断是不是我们监听到的外围设备
  if (self.discovedPeripheral != peripheral) {
        self.discovedPeripheral = peripheral;
        [self.centralManager connectPeripheral:peripheral options:nil];;

        NSLog(@"Connecting to peripheral %@", peripheral);
       // [self performSelector:@selector(xx) withObject:nil afterDelay:0.1];
    }
}
```

连接周边设备

```
//连接上外围设备后我们就要找到外围设备的服务特性
-(void)centralManager:(CBCentralManager *)central
didConnectPeripheral:(CBPeripheral *)peripheral
{

//连接完成后，就停止检测
    [self.centralManager stopScan];

    [self.data setLength:0];
    //确保我们收到的外围设备连接后的回调代理函数
  peripheral.delegate=self;
    //让外围设备找到与我们发送的UUID所匹配的服务
    [peripheral discoverServices:@[[CBUUID
UUIDWithString:TRANSFER_SERVICE_UUID]]];

}
```

找到我们想要的特性

```
//相当于对方的账号
-(void)peripheral:(CBPeripheral *)peripheral
didDiscoverServices:(NSError *)error{

    if (error) {
        NSLog(@"Errordiscover:%@",error.localizedDescription);
        [self clearUp];
        return;
    }
    //找到我们想要的特性
  //遍历外围设备
  for (CBService*server in peripheral.services) {
        [peripheral discoverCharacteristics:@[[CBUUID
UUIDWithString:TRANSFER_CHARACTERISTIC_UUID]]
forService:server];
    }

}
```

订阅消息

```
//当发现传送服务特性后我们要订阅他 来告诉外围设备我们想要这个特性所持有的数据
-(void)peripheral:(CBPeripheral *)peripheral didDiscoverCharacteristicsForService:(CBService *)service error:(NSError *)error
{
    if (error) {
        NSLog(@"error  %@",[error localizedDescription]);
        [self clearUp];
        return;
    }
    //检查特性
  for (CBCharacteristic*characteristic in service.characteristics) {
        if ([characteristic.UUID isEqual:[CBUUID UUIDWithString:TRANSFER_CHARACTERISTIC_UUID]]) {
            //有来自外围的特性，找到了，就订阅他
       // 如果第一个参数是yes的话，就是允许代理方法peripheral:didUpdateValueForCharacteristic:error: 来监听 第二个参数 特性是否发生变化
            [peripheral setNotifyValue:YES forCharacteristic:characteristic];
          //完成后，等待数据传进来
      NSLog(@"订阅成功");

        }
    }

}
```

接收数据

```
//这个函数类似网络请求时候只需收到数据的那个函数
-(void)peripheral:(CBPeripheral *)peripheral didUpdateValueForCharacteristic:(CBCharacteristic
*)characteristic error:(NSError *)error
{
    if (error) {
        NSLog(@"error~~%@",error.localizedDescription);
        return;
    }
    //characteristic.value 是特性中所包含的数据
  NSString*stringFromData=[[NSString alloc]initWithData:characteristic.value
encoding:NSUTF8StringEncoding];
    NSLog(@"%@",stringFromData);

    if ([stringFromData isEqualToString:BluetoothEnd]) {
        //完成发送，调用代理进行传递self.data
     NSString*str=[[NSString alloc]initWithData:self.data encoding:NSUTF8StringEncoding];
        //取消订阅
        [peripheral setNotifyValue:NO forCharacteristic:characteristic];
        [self.centralManager cancelPeripheralConnection:peripheral];
        self.blockValue(str);

    }else{
    //数据没有传递完成，继续传递数据
        [self.data appendData:characteristic.value];

    }

}
```

连接状态的变化

```
//外围设备让我们知道，我们订阅和取消订阅是否发生
-(void)peripheral:(CBPeripheral *)peripheral didUpdateNotificationStateForCharacteristic:(CBCharacteristic *)characteristic error:(NSError *)error
{
    if (error) {
        NSLog(@"error  %@",error.localizedDescription);
    }
    //如果不是我们要特性就退出
  if (![characteristic.UUID isEqual:[CBUUID UUIDWithString:TRANSFER_CHARACTERISTIC_UUID]]) {
        return;
    }

    if (characteristic.isNotifying) {
        NSLog(@"外围特性通知开始");
    }else{
        NSLog(@"外围设备特性通知结束，也就是用户要下线或者离开%@",characteristic);
        //断开连接
        [self.centralManager cancelPeripheralConnection:peripheral];

    }
}
-(void)centralManager:(CBCentralManager *)central didDisconnectPeripheral:(CBPeripheral *)peripheral error:(NSError *)error
{
    NSLog(@"disconnected");
    self.discovedPeripheral=nil;
    [self scan];
}
```

其他相关函数

```
//连接失败时的处理
-(void)centralManager:(CBCentralManager *)central didFailToConnectPeripheral:(CBPeripheral *)peripheral error:(NSError
*)error
{
    NSLog(@"Failed to connect to %@~~~%@",peripheral,[error localizedDescription]);
    [self clearUp];

}

-(void)clearUp{
    if (![self.discovedPeripheral isConnected]) {
        return;
    }

    if (self.discovedPeripheral.services!=nil) {
        for (CBService*server in self.discovedPeripheral.services) {

            if (server.characteristics!=nil) {
                for (CBCharacteristic*chatacter in server.characteristics) {

                    if ([chatacter.UUID isEqual:[CBUUID UUIDWithString:TRANSFER_CHARACTERISTIC_UUID]]) {

                        //查看是否订阅了
          if (chatacter.isNotifying) {
                            //如果订阅了。取消订阅
                            [self.discovedPeripheral setNotifyValue:NO forCharacteristic:chatacter];
                            return;
                        }

                    }

                }
            }
        }
    }
    //如果我们连接了，但是没有订阅，就断开连接即可
    [self.centralManager cancelPeripheralConnection:self.discovedPeripheral];

}
```

总结

以上是整体蓝牙相关代码，但是如果要掌握太多了，并且我们使用相关功能，并不是要做到很懂，而是要做到会用，能够实现通讯后，处理相关数据，所以我封装了该蓝牙，简化了所有的方法，仅需几行代码即可完成接收数据和发送数据，请查看代码调用

代码调用

1、添加头文件#import “ZCCentralManager.h” #import "ZCPeripheralManager.h"

1、发送消息类的实例化

```
manager1=[[ZCPeripheralManager alloc]initWithBlock:^(int x) {
        NSLog(@"发送完成");
    }];
```

2、接收消息类的实例化

```
 ZCCentralManager*manager=[[ZCCentralManager alloc]initWithBlock:^(NSString *str) {
        NSLog(@"接收到得信息%@",str);
        UIAlertView*al=[[UIAlertView alloc]initWithTitle:@"11" message:str delegate:self cancelButtonTitle:@"22" otherButtonTitles: nil];
        [al show];

    }];
```

3、发送消息

```
 manager1.message=@"哈哈哈哈";

    manager1.sendDataIndex=0;
    [manager1 sendDataClick];
```