微信蓝牙外设协议

Project BlueShadow

V1.0.4

文档变更日志

版本	变更	日期
0．1	初稿	2013/10/9
0．2	针对常见问题，增加说明	2013/10/18
0．3	针对常见问题，增加说明。修改包头结构。	2013/11/5
0．4	第二期初稿 增加 17 条 protobuf 协议，废弃 1 条，修改 1 条	2013/11/26
0．5	暂时去掉二期的内容（17 条 protobuf），在一期的基础上增加三条协议，并修改 auth 协议。加入加密解密部分说明。	2014/2/12
0．6	修改 uuid 的规定。补充协议字段和错误码。补充加密部分细节。	2014/2/27
0．7	修改加密协议，增加例子。补充不加密的协议。补充错误码。	2014/3/4
0．7．1	修改 proto 里的 UserId，Challeange 字段。	2014/3/7
0．7．2	规定低功耗蓝牙广播包必须包含 MAC 地址。InitResp 增加可选字段以支持自动同步模式，修改解码失败错误码。	2014/4/3
0．7．3	修改 SendData， SendDataPush 两条协议，增加 type 字段，用来区分是发送给厂商服务器，还是发送给公众平台服务器。	2014/4/24
0．7．4	修改 ble 蓝牙 service uuid 的值。	2014/4/25
0．7．5	公众平台协议手环修改：增加 BaseResponse 字段。	2014/4/30
1．0．1	提升版本号为 1.0.1 Beta	2014/6/9

1.0.2	增加 Read Characteristics，以支持 ios 多 app 连接。 增加 html jsapi 支持。 公众平台协议手环修改：增加 rtc 时间支持。 废弃一些字段，修改协议名字为 SendData，RecvData 等。	2014/7/15
1.0.3	增加蓝牙扫描绑定相关规范。	2014/8/18
1.0.4	去掉蓝牙绑定方式，增加厂商服务器绑定方式。 增加蓝牙 Jsapi：扫描，连接，获取 ticket 等。 新增认证方式：使用 MAC 地址且不加密认证（允许不烧 deviceId，方便产商的产线流程）。	2014/12/17

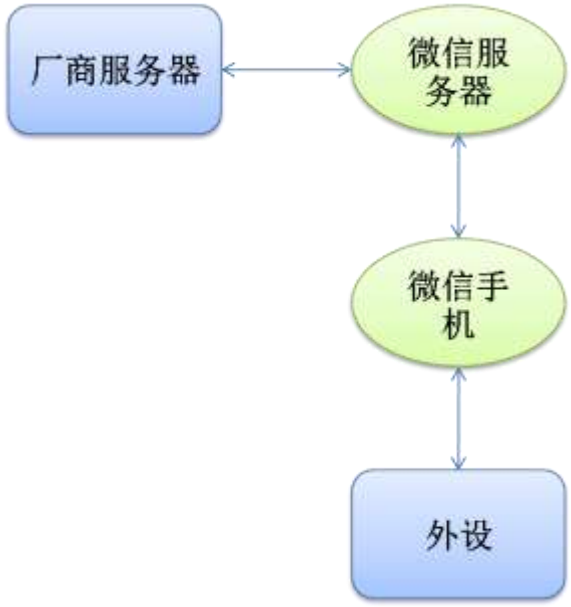
目录

概要

该文档规范了蓝牙设备和手机上的微信的通信协议

协议支持经典蓝牙和 4.0 BLE 蓝牙，目前支持 ios 和 andriod 两个系统，后续会扩展到其他系统。

整体架构



厂商服务器和外设，由厂商开发完成。

微信会提供服务器的接口以对接厂商的服务器，会提供手机的接口（如本文规定的蓝牙协议）以对接厂商的外设。

主要功能

该协议打通了设备和厂商服务器之间的数据链路，也就是支持将设备上的数据发送到厂商的

服务器上，也支持将厂商的数据发送到设备。

厂商的数据对于微信来说，是黑盒，微信不对设备数据做分析。

该协议也打通了设备和微信服务器之间的数据链路。

设备和微信服务器之间的数据格式由微信规定，例如登录，新消息通知等。

蓝牙 BLE 模拟成流

微信支持蓝牙 BLE。

微信规定了蓝牙 BLE 设备需要先模拟成流(即 stream 输入输出流)。经典蓝牙的 RFCOMM，就是一个流。流具有的特性有：

a. 可以传输无限长度的数据

b. 双工，读写可以并发，互不干扰。

显然，蓝牙 BLE 无法传输无限长度的数据，为了实现这个目的，需要定义一个规范。

蓝牙设备需暴露两个特征值（Characteristics）：Write 特征值，Indication 特征值。蓝牙设备从 Write 特征值接受数据，从 Indication 特征值发送数据。

Indication 特征值类型是 bytes。

这里我们约定，把一个特征值一次传输的数据，称为一帧（不同类型的特征值一次传输的数据长度是不一样的）。

注意：应用层上的数据包（例如 1k 大小），会分散成许多帧来传输。

蓝牙设备写过程：

1. 分帧：假设蓝牙手环上有 1k 数据，要发给手机微信。由于一个特征值长度有限（如 20 个字节），显然需要分多次才能传输完成。1k 数据，要分成 1024 字节/ 20 字节 =51 个帧。剩下的 4 个字节，不足一帧（20 个字节），需补齐为一帧并对剩下的 16 个字节赋 0。总共是 52 帧。
2. 发送第一个帧：把第一个帧的内容放入特征值里面。然后通知手机读取数据，通知有两种方式，Indication 和 notify，这里使用 Indication 方式，即带响应的通知。当通知完成的时候，可以认为手机已经读完数据。这就完成了发送第一个帧。
3. 按照 2 的步骤，依次发送剩下的帧。

蓝牙设备读过程：

当蓝牙设备发现读特征值收到数据的时候，就接收数据，并追加到设备的 buf 里。

注意：蓝牙设备必须等微信 app 订阅了 Characteristics 之后，才能 indicate 数据，否者会造成设备发送数据丢失的问题。

协议

1 前提

厂商需先开通微信公众平台的硬件号功能。

厂商需先在公众平台上注册设备（具体 api 见服务器端的微信公众平台的硬件功能文档）。

2 设备和广播

为了和微信能够通信，设备的广播包需符合以下的格式：

低功耗蓝牙（android&ios）设备需要广播：

a. 微信规定的 service uuid（具体见附录）。

b. 厂商自定义字段里，包含 MAC 地址（具体见附录）。

c. 包含指定的 Characteristics（具体见附录）。

Andriod 经典蓝牙设备需要广播：

a. 暴露一个指定 uuid 的 rfcomm 服务（具体见附录）。

b. 厂商自定义字段里，包含 MAC 地址（具体见附录）。

Ios 经典蓝牙需要通过 mfi 认证，并且 SerialNumber 需为 MAC 地址（字符串形式）。

设备可分为两种：

可确认设备：如有按钮，或者可以双击，或者有可检测翻转等传感器，可与人交互的设备。大部分设备属于可确认设备。

无法确认的设备：即没有按钮等传感器，无法与人交互的设备。这种设备很少。

目前的市面上设备，有很多是要设备做出确认之后（如长按某个按钮，或者靠近手机），才会进行设备和用户的绑定。

相应的，广播包有两种：

普通包：我们把设备正常情况下无时无刻广播的包成称为普通包

确认包：当用户进行设备确认时（如双击手环，或者按按钮），广播的特殊的包称为确认包。微信规定了微信认识的普通包和确认包的格式 :通过 manufature data 来区别（具体见附录）。

3 绑定

用户绑定设备有两种方式：扫码绑定，厂商服务器绑定

这两种方式都需要先向微信公众平台注册授权设备（具体 api 参见公众平台文档）。

扫码绑定

用户通过扫描设备二维码绑定设备（获取二维码的方法见公众平台文档）。

用户场景：用户打开扫一扫界面，扫码设备二维码，出现公众号页面。用户点击关注，进行绑定设备。

扫码绑定并不需要设备在身边。

厂商服务器绑定

厂商可通过微信公众平台提供的接口，绑定厂商指定的用户和设备。

该功能给厂商比较大的自由，可自定义绑定的流程和界面（通过 html5）。

该功能目前需要通过本协议的 jsapi 来实现。

绑定的用户场景和技术细节详见本文档 jsapi 一章。

4 扫描和连接

进入特定界面后(如具有硬件功能的公众号界面)，微信会开始扫描设备。当设备广播符合微信定义的广播包时，则微信可以扫描到设备。

扫描到设备后，微信会连接设备（Ios 经典蓝牙需要用户手动在设置界面里面连接上设备）。

当连上设备之后，微信和设备之间开始传输数据。数据的具体格式见下面章节。

5 流

经典蓝牙使用 RFCOMM 通信（是个流），蓝牙 BLE 也模拟成流。

6 包

流之上运载的是一个紧接着一个的业务逻辑的数据包。

数据包的发送方和接受方：设备<->厂商服务器，或者设备<->微信服务器。

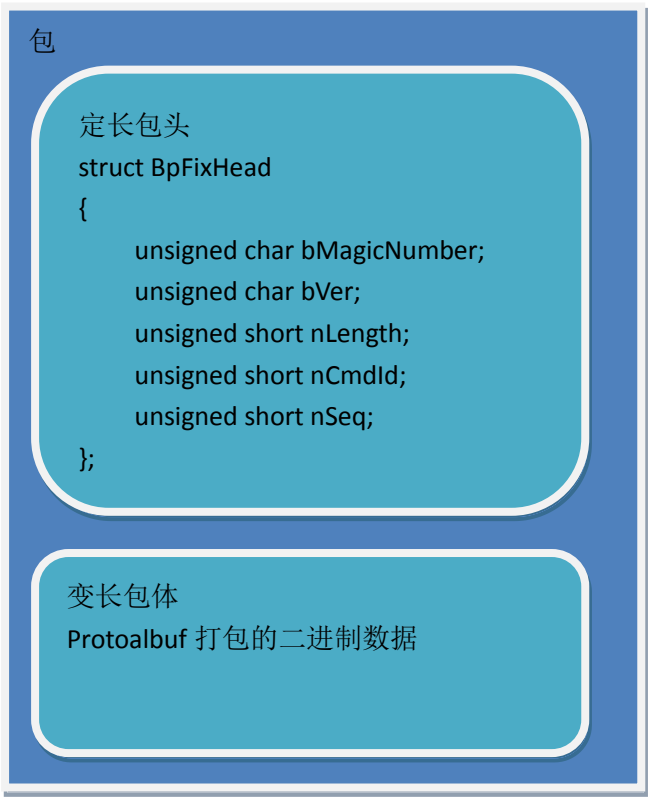
把设备->厂商服务器/微信服务器的请求称为 Req，回包称为 Resp。一个请求，对应着一个回包。

把厂商服务器/微信服务器->设备的请求称为 PushReq，没有回包（即没有 PushResp

这样的包)。

7 包结构

由定长包头和变长包体组成。



包的二进制例子见附录。

8 定长包头

字段	类型	说明
bMagicNumber	unsigned char	填 0xFE

bVer	unsigned char	包格式版本号，填 1
nLength	unsigned short	为包头+包体的长度
nCmdId	unsigned short	命令号，如 ECI_req_auth，ECI_resp_sendDataToManufacturerSvr 等
nSeq	unsigned short	递增。 一个 Req 对应一个 Resp ,并且它们的 nSeq 相同，并且永不为 0。 Push 的 nSeq 永远为 0；

9 变长包体

为 Protobuf（protobuf 的介绍见附录）打包的结构。例如 AuthReq。

一个包体里面只有一个 Req，或者一个 Resp，或者一个 PushReq。

每个 Req/Resp/PushReq 都有对应的 EmCmdId，例如 AuthReq 的命令号为 ECI_req_auth。

具体的定义见附录。

10 加密的身份验证

微信支持身份验证和加密，以使得厂商的数据更加安全。

加密算法选用 aes 128 位，并使用 cbc 模式，pkcs7 填充。初始向量为密钥。

具体验证和加密的步骤如下：

a. 设备需要烧一个 Key（128 位）到硬件上，微信服务器也要记录下这个 Key。一个设备（deiveId+deviceType 唯一确定一个设备），对应一个 Key。

   key 要保护好(类似用户密码 ,银行卡账号密码),千万不要印刷出来或者打印出来。

   另外有一个 MD5(deviceType+deviceId)，如果设备算 MD5 很麻烦的话，建议先算好，直接烧进设备里面。该 MD5 类似于用户名。

b. 对于设备来说，手机和服务器可看成一个黑盒。设备和手机服务器之间，通过 Auth 命令，使用 key，最终，把 sessionKey 下发给到设备。

   Auth 的步骤为：

   设备发送 AuthReq，里面的字段有 MD5(deviceType+deviceId)，另一个字段为 AES 加密的一段 buffer（具体字段细节在附录描叙）。

   微信发送回包 AuthResp，里面的字段有用 AES 加密的含有 sessionkey 的一段 buffer（具体字段细节在附录描叙）。设备解密这段 buffer 可得到 SessionKey。

c. 设备和手机之间，就通过 sessionKey 来加解密包。

AuthReq， AuthResp 之后的所有命令，例如 InitReq，InitResponse，SendDataRequest，SendDataResponse，RecvDataPush 等都需要加解密。

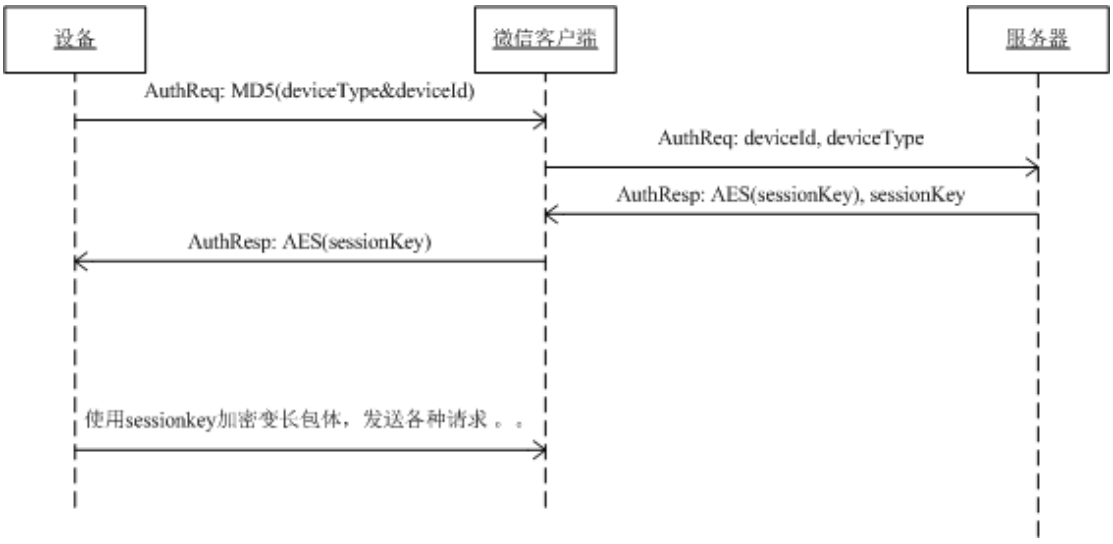
d. 加密只针对变长包体，不需要加密定长包头。

e. 为了提高安全性，设备可再进一步的验证手机是否可信。

设备在 InitReq 里填入 Challeange（一个 4 字节的随机数 random4），手机将在 InitResp 里面返回 ChalleangeAnswer（一个 4 字节的数，值为 crc32(random4)）。

设备验证 ChalleangeAnswer 是否正确。如果是，说明手机是可信的。

身份验证和加解密大体的流程图为：



另外有一些细节：

a. 当 sessionKey 过期，手机对设备的所有请求都返回错误码 EEC_sessionTimeout，要求设备重做一次 auth。

b. 当发现解密设备的包失败时，返回 EEC_decode。这种情况下，通常是 sessionkey 过期，或者是设备程序有问题。

11 不加密的身份验证

微信客户端同时支持不加密的方式，以降低设备厂商的接入难度，加快开发产品的进程。不加密的方式下，设备和微信之间的通信的安全级别，将与设备和系统原生的蓝牙 api 之间的通信的安全级别一样。为了确保厂商设备和服务器的数据安全，**厂商需自行实现安全机制**。

不加密有两种方式：

1. MD5 认证：使用 MD5(deviceType+deviceId)和微信认证，如果设备算 MD5 很麻烦的话，建议先算好，直接烧进设备里面。该 MD5 类似于用户名。这种方法要求产线上对不同的设备需要烧写不同的 MD5。
2. MAC 地址认证：使用 MAC 地址和微信认证。Mac 地址类似于用户名。这种方法不需要产线上对设备烧写额外的东西(MAC 地址在原来的产线上本来就是要烧进设备的)。该方法减低了产线的难度，但安全性不如 MD5 认证。

步骤如下：

将 AuthReq 里的 AesSign 字段付空。

如果是用 MD5 登录，则赋值 Md5DeviceTypeAndDeviceId 字段，且 AuthMethod=EAM_md5，如果是 MAC 地址登录，则赋值 MacAddress 字段，且 AuthMethod=EAM_macNoEncrypt。

发送 AuthReq。

收到 AuthResp 之后，忽略掉 AesSessionKey 字段。

将 InitReq 里的 Challeange 字段随便付一个值。发送 InitReq。

收到 InitResp，忽略掉 ChalleangeAnswer 字段。

随后的包都不需要加解密。

12 会话约定

a. 设备连上微信之后，需要发送 AuthReq，等收到成功的回包之后，接着还要发送 InitReq，并收到成功的回包之后，才能正常发送数据。如果设备没有 auth，手机对设备的所有请求都返回错误码 EEC_needAuth。

b. 当出现解包异常的时候，直接断开连接。

c. Push 类的包 seq 永远为 0。Req 类和 Resp 类的包的 seq 永不为 0。

d. 服务器可随时发送 Push 包。

e. 厂商服务器发送的 Push 包（注意 Push 包是没有回包的，即没有 PushResp），如果需要设备的回包，需要由厂商自己实现。

   具体方法如：厂商发送 RecvDataPush 给设备，设备收到 push 后，向厂商服务器发送一个 SendDataRequest。这时厂商服务器可知道设备收到了 push，并且可以从 Req 里取得设备的回应数据。

13 时序

时序图请见附录。

14 Read Characteristics

为了支持同一手机上一个 app 连接了设备后，微信还能搜素并连接到设备的情况，设备需要在微信的 service 下面，暴露一个 read character，内容为 6 字节的 MAC 地址。当 ios 上的其他 app 连接上设备时，设备不会再广播，微信会读取该特征值，以确定是否要连接该设备。

15 其他

目前规定了一些基础的协议，更多的协议等待补充。

微信的 Protobuf 协议

1 概述

变长包体部分，使用的是 Protobuf 定义的协议。Protobuf 文件见附录。

里面规定了设备和微信客户端之间的命令。

2 命令列表

名称	描叙
Auth	登录
Init	初始化
SendData	设备发送数据给厂商或微信公众平台或微信客户端。 当 type 为空或者等于 0 时，表示发送给厂商服务器。 当 type 为 10001 时 ,表示发送给微信客户端 html5 设备会话界面。 当 type 为其他时，表示发送给公众平台服务器。具体的定义请看 附录：微信公众平台 proto 文件。举个例子，type 等于 1 时，表示手环数据。
RecvDataPush	厂商或微信客户端或微信公众平台发送数据给设备 当 type 为空或者等于 0 时，表示厂商发送设备。 当 type 为 10001 时 ,表示收到微信客户端 html5 设备会话界面的数据。 当 type 为其他时，表示公众平台发送给设备。具体的定义请看 附录：微信公众平台 proto 文件。举个例子，type 等于 1 时，表示手环数据。
SwitchViewPush	微信客户端进入退出界面的通知
SwitchBackgroudPush	微信客户端进入退出后台的通知

3 错误码

Proto 里的错误码

EEC_system	微信客户端一般的错误
EEC_needAuth	设备未登录。需要登录。
EEC_sessionTimeout	sessionKey 超时。需要重新登录。
EEC_decode	微信客户端解 proto 失败。可能是设备端打包代码有 bug。
EEC_deviceIsBlock	微信客户端一段时间之内禁止设备的请求。通常是设备某些异常行为引起，如短时间多次登录，大量发送数据等。
EEC_serviceUnAvalibleInBackground	ios 处于后台模式，无法正常服务
EEC_deviceProtoVersionNeedUpdate	设备的 proto 版本过老，需要更新
EEC_phoneProtoVersionNeedUpdate	微信客户端的 proto 版本过老，需要更新
EEC_maxReqInQueue	设备发送了多个请求，并且没有收到回包。微信客户端请求队列拥塞。
EEC_userExitWxAccount	用户退出微信帐号。
为正数时	具体见微信公众平台。

固定包头里的错误码：

固定包头里的错误码放在 cmdid 字段里。当设备收到这样的错误码后，可以通过 seq 查出是那个命令失败。

目前只有一个错误码。

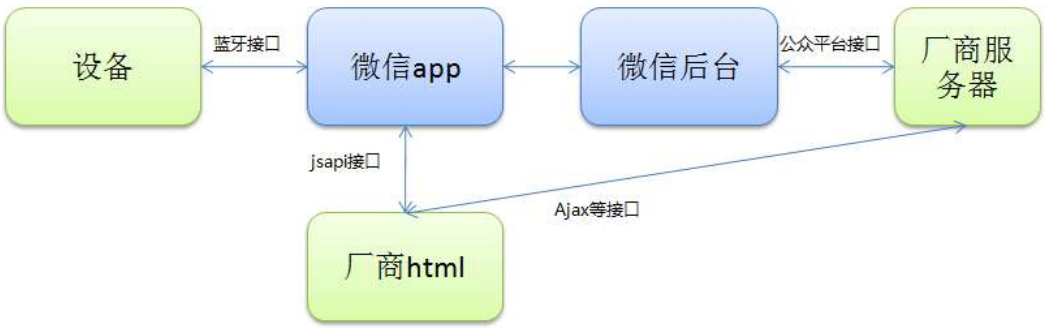
ECI_err_decode	微信客户端解密包体失败。 通常是因为 sessionKey 过时，需要重新做一次 auth，也可能是设备加密代码有 bug。

4 JSAPI

概述

微信 jsapi 提供给厂商在网页通过 javascript 对设备操作的接口。

例如扫描设备，连接设备，收发数据，绑定设备等。



Html 通过 Jsapi 可以和设备本地收发数据（即 Html 发送给微信客户端，微信客户端发给设备），无需通过服务器中转，所以速度较快。

实时性要求高的蓝牙设备（如遥控汽车）可采用 jsapi 收发数据。

此外，扫描，绑定，连接等接口提供给厂家更多的自由发挥的空间。

用户场景

举例如：

用户打开微信，进入硬件公众号，点击菜单，进入厂家自定义的 html 界面。

用户点击 html 界面上的扫描按钮，开始扫描周围设备。当扫到多个设备的时候，html 界面显示一个设备列表。

用户点击某个设备(该设备未绑定),进行绑定。html 界面显示绑定成功 ,并开始连接设备。

Html 界面显示连接设备成功。

用户点击 Html 界面上的开始同步按钮。界面显示进度条，再显示同步成功。

接口

具体细节见《微信硬件 JSAPI 介绍文档》。

附录

1 ProtoBuf 协议介绍

ProtoBuf 是 google 提供的一套开源的软件协议。它主要作用是把 c/c++的 struct 打包成为二进制数据，或者把二进制数据解包成 c/c++的 struct。

具体使用过程为：

1. 定义 proto 文件
2. 通过工具把.proto 文件编译成.h, .c 文件（里面包含 struct 和函数）
3 调用.h 文件里的封包解包函数

官方网站为：

https://developers.google.com/protocol-buffers/docs/overview

http://code.google.com/p/protobuf/

http://code.google.com/p/protobuf-c/

其他相关例子和工具见附件。

2 蓝牙硬件一些规定

名称	值
ServiceUUID	0xFEE7（该 uuid 经蓝牙官方授权）
Write Characteristics UUID	0xFEC7
Indicate Characteristics UUID	0xFEC8
Read Characteristics UUID	0xFEC9

Andriod RFCOMM UUID	e5b152ed-6b46-09e9-4678-665e9a972cbc

Ios 经典蓝牙的 iap 层的 SerialNumber 必须赋值，且等于 MAC 地址（字符串形式）。

普通广播包：manufature specific data 需以 MAC 地址（6 字节）结尾。并且 manufature specific data 长度需大于等于 8 字节（最前两个字节为 company id，没有的话随便填）。

确认广播包：manufature specific data 需以下面格式结尾，

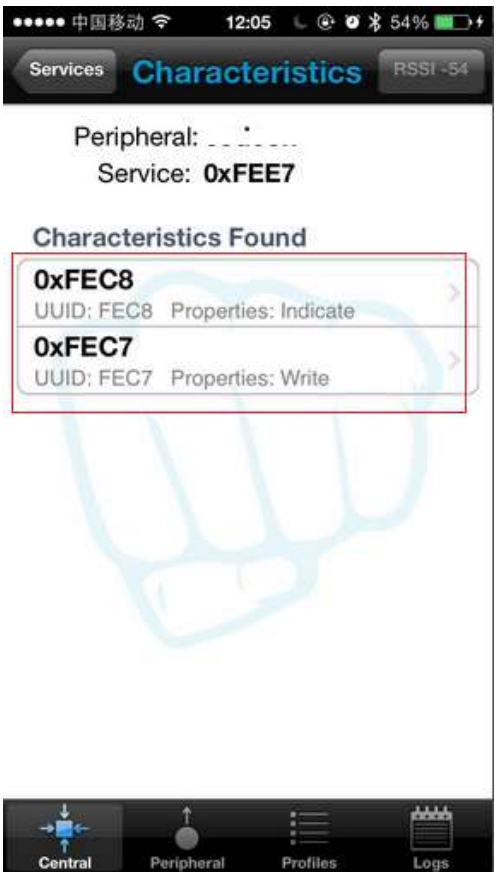
0xfe 0x01 0x01 + MAC 地址（6 字节）。并且 manufature specific data 长度需大于等于 8 字节（最前两个字节为 company id，没有的话随便填）。

3 Ios BLE 设备的截图

LightBlue 是一个 Iphone 上的 app。这是设备的截屏。请下载 App 并检查： 1. serviceId 是否正确 2. characteristics 是否正确（包括 uuid 和属性） 3. 是否在 manufature data 最尾部带上了 MAC 地址。


●●●●● 中国移动
11:55
49%
About
LightBlue
Scanning for Peripherals...
UUID:
FE91E28A-1A5A-2272-775C-524B42A21BDA
Local Name:
Service: Unknown (<fee7>)
Service: Battery
Manufacturer Data: <11313247 0a333237 36370047
0ed87b9 2e3d90>
ble的广播包里需带上mac地址
Central mode lets you connect to peripherals around y...
Central
Peripheral
Profiles
Logs


●●●●● 中国移动
12:05
53%
Peripherals
Services
RSSI -57
Peripheral:
Clone Peripheral
Services Found
0xFEE7
UUID: FEE7
Device Information
UUID: 180A
Battery Service
UUID: 180F
Central
Peripheral
Profiles
Logs



4 包的二进制例子

Req 接受两种格式：20 字节对齐补零格式，不对齐格式。以 AuthReq 为例子（不加密）：

1. 20 字节对齐补零（ble 特征值有些芯片只能发送固定 20 个字节，20 字节剩下的部分补零）

   fe（***MagicNumber***）01（***版本号***）00 3b（***总长度***） 2711（***命令号***）0001（***Seq***）（***变长包体***：0a 00 12 10 b4 3f 12 04 2a 02 e0 1c 2b dd 7d 02 90 62 13 a3 18 80 80 04 20 01 28 01 32 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 62 03 41 4d 33)（***补齐***：00）

   注意：这里的包长度为 003b(即 59 字节)，由于 20 字节对齐，导致总长为 60 字节。

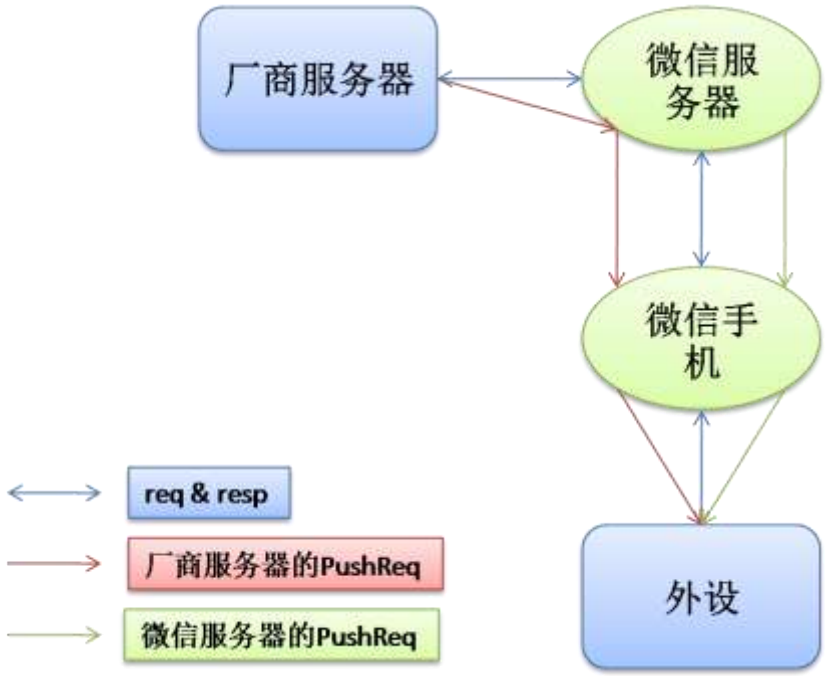
2. 不对齐（经典蓝牙没有对齐问题，某些 ble 芯片也没有对齐问题）

fe（*MagicNumber*）01（*版本号*）00 3b（*总长度*） 2711（*命令号*） 0001（*Seq*）（*变长包体*：0a 00 12 10 b4 3f 12 04 2a 02 e0 1c 2b dd 7d 02 90 62 13 a3 18 80 80 04 20 01 28 01 32 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 62 03 41 4d 33）

以 AuthResp 为例子：

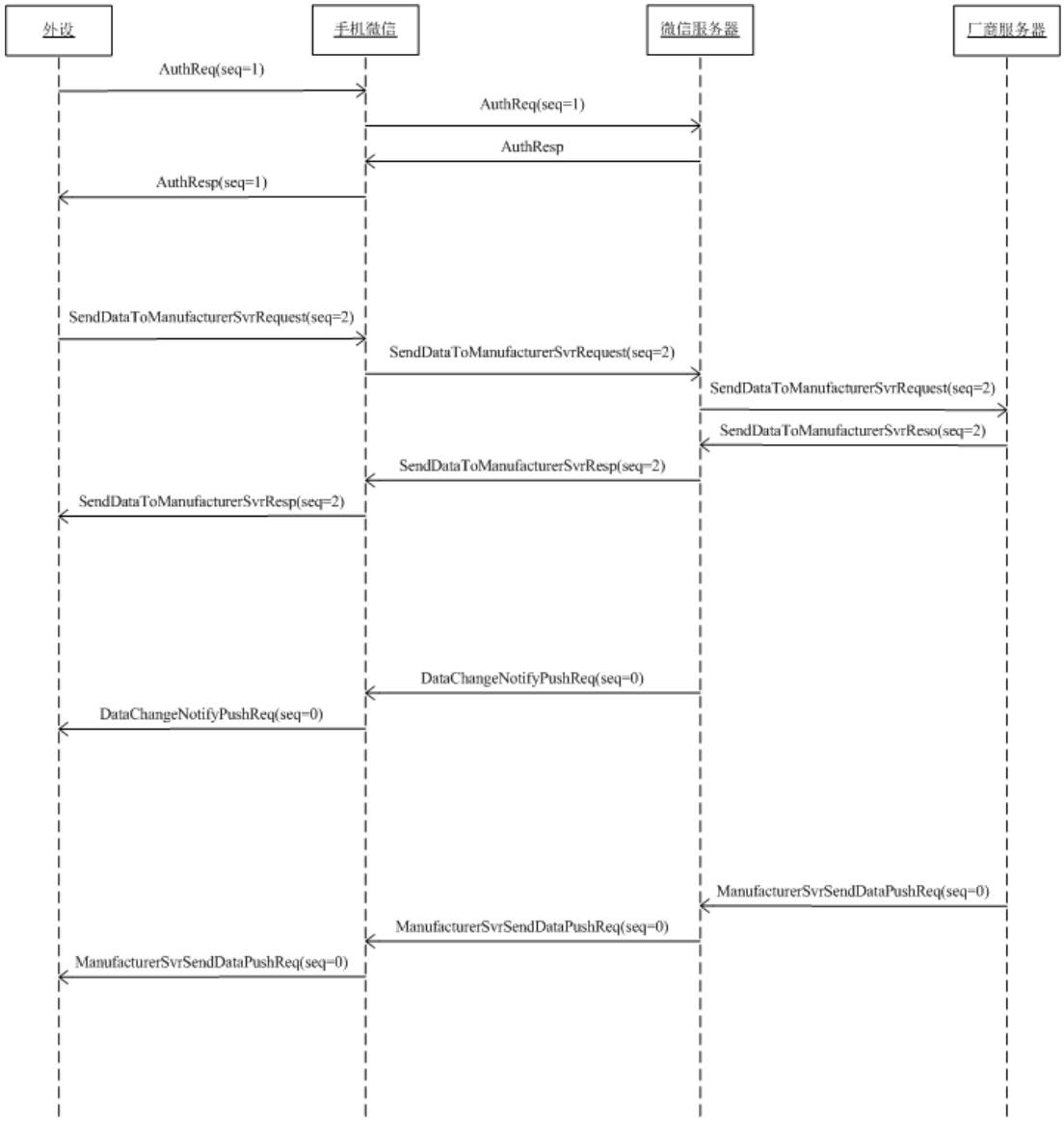
fe（*MagicNumber*）01（*版本号*）000e（*总长度*） 4e21（*命令号*）0001（*Seq*）（*包体*：0a020800 1200）

5 包的数据流图



6 包的时序图

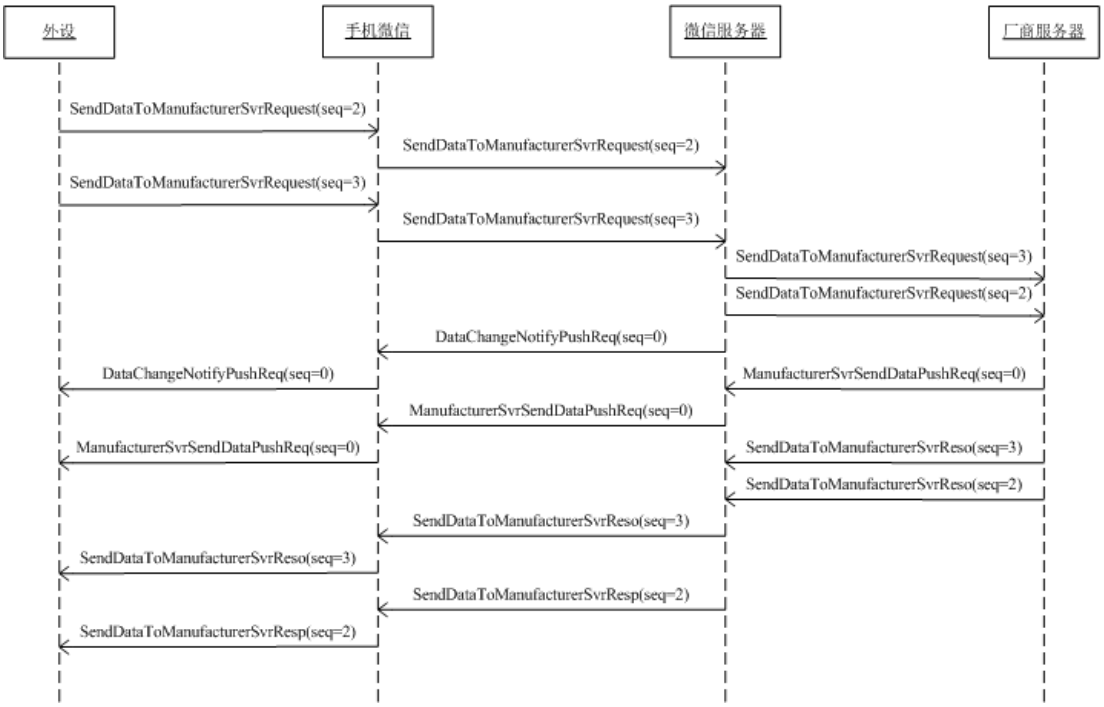
最简单的时序图：



Push 包插入的时序图

多 Req 包并且有 push 包插入的时序图：

注意：多个 Req 包时，到达厂商服务器的先后顺序不能保证。同样多 Resp 包的情况下，到达外设的顺序也不能保证。



7 加解密字段

AuthReq 里的 AesSign 字段 = AES（Ran | Seq | CRC32（DeviceId | Ran | Seq））

a) 设备生产一个随机数 Ran（4 字节），注意：随机数应尽量随机

b) 若设备有能力获得当前的时间戳，则使用时间戳作为 Seq （ 4 字节 ），否则，使用一个从 0 开始递增的序列号，保证每次 auth 的 Seq 递增

c) 将 Ran 和 Seq 拼接到 DeviceId（不含 DeviceType）后得到二进制串 1（变长）

d) 计算二进制串 1 的 CRC32（4 字节 ）

e) 将 Ran、Seq、CRC 值按顺序拼接得到 12 字节的二进制串 2，使用设备 key 对二进制传 2 进行 AES 加密，得到 16 字节的 AesSign

AuthResp 里的 AesSessionKey 字段 = AES (SessionKey）

设备 auth 成功后将取得一个 32 字节的 AesSessionKey 字段，使用设备 key 做 AES 解密后可得到 16 字节的明文内容.该内容为 sessionKey。

InitReq 里的 Challeange 字段为随机数(4 字节，记为 Random3)

InitResp 里的 ChalleangeAnswer 字段为 CRC32(Random3)。

上述身份认证和加密的方式为 0x1 版本，对应 AuthReq 里面的 AuthMethod 字段。该版本未来可能升级。

8 Md5DeviceTypeAndDeviceId 的例子

Md5DeviceTypeAndDeviceId 为 16 字节二进制数据。

假设 deviceType 为：gh_d53f87f298e5，deviceId 为：test_device。将

deviceType+deviceId 拼起来，不算\0：

md5(gh_d53f87f298e5test_device,32) = 0x26cdd942b8ee68b022cc53bba16c7039

9 CBC 例子

Key 为字符串 “3141592653589793”

Iv 和 Key 相同。

数据：

<len:15> <data:0x6c656e6774685f6f665f31355f625f>

CBC（pkcs7）结果：

<len:16> <data:0x3154f6e6c796d521398e060a5b1fb1b9>

数据:

<len:16> <data:0x6c656e6774685f6f665f31365f625f5f>

CBC（pkcs7）结果：

<len:32> <data:0x4b6b8f1257e8d62f0ddfaea0122af4124414f4ff8fc86f348700581625d346f1>

数据:

<len:32> <data:0x6c656e6774685f6f665f33325f625f5f31323334353637386162636465666768>

CBC（pkcs7） 结果:

<len:48><data:0x817692fdba867c913f7c717b2da336acc6dad854b2f9ff5ac849291d86ba86dcc7
7f586770ad2c7298f00f2a881393bb>

10 CRC32 例子

多项式为：0xedb88320L

字符串"test_device_ios"的 crc32 为：0x02e312f3。

11 微信蓝牙外设 proto 文件

```
// proto version: 1.0.4
package MmBp;


enum  EmCmdId
{
    ECI_none = 0;

    // req： 蓝牙设备 -> 微信/厂商服务器
    ECI_req_auth = 10001;                       // 登录
    ECI_req_sendData = 10002;                   // 蓝牙设备发送数据给微信或厂商
    ECI_req_init = 10003;                       // 初始化

    //  resp：微信/厂商服务器 -> 蓝牙设备
    ECI_resp_auth = 20001;
    ECI_resp_sendData = 20002;
    ECI_resp_init = 20003;

    // push：微信/厂商服务器 -> 蓝牙设备
    ECI_push_recvData = 30001;                  // 微信或厂商发送数据给蓝牙设备
    ECI_push_switchView = 30002;                // 进入/退出界面
    ECI_push_switchBackgroud = 30003;           // 切换后台

    ECI_err_decode = 29999;                     // 解密失败的错误码。注意：这不是 cmdid。为节省固定包头大小，这种特殊的错误码放在包头的 cmdid 字段。
}

enum EmErrorCode
{
    EEC_system = -1;            // 通用的错误
    EEC_needAuth = -2;          // 设备未登录
    EEC_sessionTimeout = -3;    // session 超时，需要重新登录
    EEC_decode = -4;            // proto 解码失败
```

```
    EEC_deviceIsBlock = -5;      // 设备出现异常，导致被微信临时性禁止登录
    EEC_serviceUnAvalibleInBackground = -6; // ios 处于后台模式，无法正常服务
    EEC_deviceProtoVersionNeedUpdate = -7;  // 设备的 proto 版本过老，需要更新
    EEC_phoneProtoVersionNeedUpdate = -8;   // 微信客户端的 proto 版本过老，需要更新
    EEC_maxReqInQueue = -9;          // 设备发送了多个请求，并且没有收到回包。微信客户端请求队列拥塞。
    EEC_userExitWxAccount = -10;     // 用户退出微信帐号。
}

message BaseRequest {
}

message BaseResponse {
    required int32 ErrCode = 1;
    optional string ErrMsg = 2;
}

message BasePush {
}


// req, resp =======================================

enum EmAuthMethod
{
    EAM_md5 = 1;                // 设备通过 Md5DeviceTypeAndDeviceId，来通过微信 app 的认证。1. 如果是用 aes 加密，注意设置 AesSign 有值。 2. 如果是没有加密，注意设置 AesSign 为空或者长度为零。
    EAM_macNoEncrypt = 2;       // 设备通过 mac 地址字段，且没有加密，来通过微信 app 的认证。
}


// 登录 ----------------------------------------
message AuthRequest {
    required BaseRequest BaseRequest = 1;
    optional bytes Md5DeviceTypeAndDeviceId = 2;     // deviceType 加 deviceId 的 md5，16 字节的二进制数据
    required int32 ProtoVersion = 3;                 // 设备支持的本 proto 文件的版本号，第一个字节表示最小版本，第二个字节表示小版本，第三字节表示大版本。版本号为 1.0.0 的话，应该填:0x010000; 1.2.3 的话，填成 0x010203。
    required int32 AuthProto = 4;                    // 填 1
    required EmAuthMethod AuthMethod = 5;            // 验证和加密的方法，见 EmAuthMethod
    optional bytes AesSign = 6;                      // 具体生成方法见文档
    optional bytes MacAddress = 7;                   // mac 地址，6 位。当设备没有烧 deviceId 的时候，可使用该 mac 地址字段来通过微信 app 的认证
```

```
    optional string TimeZone = 10;                  // 废弃
    optional string Language = 11;                  // 废弃
    optional string DeviceName = 12;                // 废弃
}

message AuthResponse {
    required BaseResponse BaseResponse = 1;
    required bytes AesSessionKey = 2;
}

// 初始化 ------------------------------------------
enum EmInitRespFieldFilter {
    EIRFF_userNickName      = 0x1;
    EIRFF_platformType      = 0x2;
    EIRFF_model             = 0x4;
    EIRFF_os                = 0x8;
    EIRFF_time              = 0x10;
    EIRFF_timeZone          = 0x20;
    EIRFF_timeString        = 0x40;
}

// 微信连接上设备时，处于什么情景
enum EmInitScence {
    EIS_deviceChat = 1;     // 聊天
    EIS_autoSync = 2;       // 自动同步
}

message InitRequest {
    required BaseRequest BaseRequest = 1;
    optional bytes RespFieldFilter = 2;     // 当一个 bit 被设置就表示要 resp 的某个字段：见
EmInitRespFieldFilter。
    optional bytes Challenge = 3;           // 设备用来验证手机是否安全。为设备随机生成的四个
字节。
}

enum EmPlatformType {
    EPT_ios = 1;
    EPT_andriod = 2;
    EPT_wp = 3;
    EPT_s60v3 = 4;
    EPT_s60v5 = 5;
    EPT_s40 = 6;
    EPT_bb = 7;
```

```
}

message InitResponse {
    required BaseResponse BaseResponse = 1;
    required uint32 UserIdHigh = 2;          // 微信用户 Id 高 32 位
    required uint32 UserIdLow = 3;           // 微信用户 Id 低 32 位
    optional uint32 ChalleangeAnswer = 4;    // 手机回复设备的挑战。为设备生成的字节的 crc32。
    optional EmInitScence InitScence = 5;    // 微信连接上设备时，处于什么情景。如果该字段为空，表示处于 EIS_deviceChat 下。
    optional uint32 AutoSyncMaxDurationSecond = 6;  // 自动同步最多持续多长，微信就会关闭连接。0xffffffff 表示无限长。

    optional string UserNickName = 11;       // 微信用户昵称
    optional EmPlatformType PlatformType = 12;       // 手机平台
    optional string Model = 13;              // 手机硬件型号
    optional string Os = 14;                 // 手机 os 版本
    optional int32 Time = 15;                // 手机当前时间
    optional int32 TimeZone = 16;            // 手机当前时区
    optional string TimeString = 17;         // 手机当前时间，格式如 201402281005285，具体字段意义为 2014（年）02（2 月）28（28 号）10（点）05（分钟）28（秒）5（星期五）。星期一为 1，星期天为 7。
}

// 设备发送数据给微信或厂商 ----------------------------
// 设备数据类型
enum EmDeviceDataType {
    EDDT_manufatureSvr = 0;                  // 厂商自定义数据
    EDDT_wxWristBand = 1;                    // 微信公众平台手环数据
    EDDT_wxDeviceHtmlChatView = 10001;       // 微信客户端设备 html5 会话界面数据
}

message SendDataRequest {
    required BaseRequest BaseRequest = 1;
    required bytes Data = 2;
    optional EmDeviceDataType Type = 3;      // 数据类型(如厂商自定义数据，或公众平台规定的手环数据，或微信客户端设备 html5 会话界面数据等）。不填，或者等于 0 的时候，表示设备发送厂商自定义数据到厂商服务器。
}

message SendDataResponse {
    required BaseResponse BaseResponse = 1;
    optional bytes Data = 2;
}
```

```
// push ================================================


// 微信或厂商发送数据给蓝牙设备 --------------------------
message RecvDataPush {
    required BasePush BasePush = 1;
    required bytes Data = 2;
    optional EmDeviceDataType Type = 3;     // 数据类型(如厂商自定义数据，或公众平台规定的手环数据，或微信客户端设备 html5 会话界面数据等）。不填，或者等于 0 的时候，表示设备收到厂商自定义数据。
}

// 微信客户端进入退出界面的通知 --------------------------
enum EmSwitchViewOp
{
    ESVO_enter = 1;
    ESVO_exit = 2;
}

enum EmViewId
{
    EVI_deviceChatView = 1;     // 微信客户端设备号会话界面
    EVI_deviceChatHtmlView = 2; // 微信客户端设备号 Html5 会话界面。注意：只有当 H5 界面主动和设备连接上之后，才会发送 push。
}

message SwitchViewPush
{
    required BasePush BasePush = 1;
    required EmSwitchViewOp SwitchViewOp = 2;   // 进入或者退出 View
    required EmViewId ViewId = 3;               // view 的 id
}

enum EmSwitchBackgroundOp
{
    ESBO_enterBackground = 1;       // 进入后台
    ESBO_enterForground = 2;        // 进入前台
    ESBO_sleep = 3;                 // 后台休眠
}

// 微信客户端进入退出后台的通知 ----------------------------
message SwitchBackgroudPush
{
```

```
    required BasePush BasePush = 1;
    required EmSwitchBackgroundOp SwitchBackgroundOp = 2;
}
```

12 微信公众平台 proto 文件

```
// type=1, 手环类
package mmbp_open;

message BaseResponse
{
     optional int32 errcode = 1;
     optional string errmsg = 2;
}


message WristbandRequest
{
     message StepDataItem
     {
          optional uint32 Step = 1;
          optional uint32 Timestamp = 2;
          optional uint32 TimeStampRtcYear = 3;
          optional uint32 TimeStampRtcMonth = 4;
          optional uint32 TimeStampRtcDay = 5;
          optional uint32 TimeStampRtcHour = 6;
          optional uint32 TimeStampRtcMinute = 7;
          optional uint32 TimeStampRtcSecond = 8;
     }

     repeated StepDataItem StepData = 1;
     optional bytes ExtData = 2;

}

message WristBandResponse
{
     optional BaseResponse BaseResp = 1;
}


message WristBandPush
```

{

}