历史版本

软件	文档	修改说明	修改人	修改时间
V0.1.1	V0.1.0	1.初始版本；	吴志涛	2017.03.19
	V0.1.1	1.修改缓存逻辑到中间件DLL；	吴志涛	2017.03.23
	V0.1.2	1.更名为接收器对接手册，并增加函数说明；	吴志涛	2017.03.25
	V0.1.3	1.增加发送数据指令返回说明和答题器信号强度；	吴志涛	2017.03.30
	V0.1.4	1.增加导入配置信息指令与导出配置信息指令；	吴志涛	2017.03.31
	V0.1.5	1.修改answer_start，剔除total这个Item； 2.支持抢答；	吴志涛	2017.04.05
	V0.1.6	1.修改get_device_info，增加配置信息；	吴志涛	2017.04.06
V0.1.2	V0.1.7	1.修改设置学号流程分2次返回，分为： [1].执行返回 {'fun':'set_student_id','result':'0'} [2].写数据返回 {'fun':'set_student_id','card_id':'1340735750','student_id':'1234'} 2.增加错误信息；	吴志涛	2017.04.20
	V0.1.8	1.增加下载程序指令；	吴志涛	2017.04.22
V0.1.3	V0.1.9	1.考勤设置指令； 2.答题器自检； 3.使用双引号指令风格；	吴志涛	2017.05.03
	V0.2.0	1.增加通用题型； 2.修改answer_start接口，增加举手和签到功能； 3.白名单采用FIFO存储逻辑，免除满需清空重新绑定的麻烦； 4.增加2个刷卡相关的API，用来获取刷卡时的卡片信息；	吴志涛	2017.05.09
	V0.2.1	1.从开始答题指令中分离，举手和签到功能； 2.增加答题器调试信息显示指令；	吴志涛	2017.05.22
	V0.2.2	1.增加软件版本的对应关系	吴志涛	2017.06.05
	V0.2.3	1.修改刷卡相关的数据返回格式 2.调整与PC间通讯的JSON的引号风格为单引号 3.增加bootloader指令细节 4.修改2.4G考勤设计的接口含义	吴志涛	2017.06.06

目录

1 文档说明

1.1 系统说明

在整个系统中由PC端通过串口发送指令控制接收器，接收器再通过13.56Mhz的天线和2.4G的天线与答题器完成数据交互。

1.2 写作目的

在答题器系统中接收器与DLL之间调用采用**JSON**格式来封装数据，本文讲解DLL与接收器之间的JSON格式的交互指令。

1.3 术语与缩写解释

白名单：已经绑定答题器的列表。
uID ：答题器与接收器的的设备ID。
绑定：答题器要与接收器通讯需要首先绑定一下，在绑定的过程中答题器与接收器会完成**设备ID（uID）**的交换，这个答题器与接收器才能相互识别。
信道：答题器与接收器通讯的的频段。
发送功率：答题器与接收器通讯的2.4G无线信号的发送功率。

2 指令函数说明

2.1 指令 函数集

答题器的 指令函数大致分为3类，答题相关的只需熟悉**绑定**和**答题**相关函数，**设置**相关的函数是在安装调试设备时需要关心的。

- 绑定相关函数

```
// 清除白名单
int clear_wl( );
// 开始绑定
int bind_start( );
// 获取绑定的卡片信息
char *get_card_info( )
// 停止绑定
int bind_stop( );
```

- 答题相关函数

```
// 开始答题1
int  answer_start( int is_quick_response, char *answer_str );
// 开始答题2
int  answer_start_with_raise_hand( int is_quick_response,
                                   int raise_hand, char *answer_str );
// 设置举手功能
int  raise_hand_set( int raise_hand );
// 设置签到功能
int  sign_in_set( int attendance );
// 获取答案
char *get_answer_list( );
// 停止作答
int answer_stop( );
```

- 设置相关函数

```
// 查询设备信息
char *get_device_info( );
// 写入学号
char *set_student_id( char *student_id_str );
// 获取设置学号卡片信息
char *get_student_id_info( );
// 设置信道
int set_channel( int tx_ch, int rx_ch );
// 设置发送功率
int set_tx_power( int tx_power );
// 设置2.4G考勤功能
int attendance_24g( int is_open, int pro_index );
// 下载程序
int bootloader(  char *file_path );
```

2.2 答题器绑定相关指令

2.2.1 清除白名单指令

【对应API函数声明】：

```
int clear_wl( );
```

【指令功能简介】：
此函数是清除接收器设备的白名单。
【DLL->接收器指令】：

```
{
  'fun': 'clear_wl'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'clear_wl',
    'result':    '0'
}
//  0 : 成功
// -1 : 失败
```

2.2.2 开启绑定指令

【对应API函数声明】：

```
int bind_start( );
```

【指令功能简介】：
此函数的功能是开启接收的绑定功能，绑定的方法：将答题器靠近接收器的刷卡区刷卡，如果听到蜂鸣器叫一下则表示刷卡绑定成功。
注意：

1. 在此协议版本**中一个接收器最多可以绑定**120**个答题器，当接收器已经绑定120个答题器时如果继续绑定，就会将最先绑定的答题器替换掉，之后依次类推。
2. 开启一次绑定可以完成对所有答题器的绑定；
3. 绑定完成之后需要，需要停止绑定指令才能停止；
4. 绑定过程中，答题器如果有刷卡会立即上报绑定的答题器信息；
5. 答题器与接收器可以多次刷卡绑定；
6. 答题器与不同接收器绑定会冲掉之前的绑定信息，保留最近的依次绑定信息。

【DLL->接收器指令】：

```
{
  'fun': 'bind_start'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'bind_start',
    'result':    '0'
}
//  0 : 成功
// -1 : 失败
```

2.2.3 获取绑定信息

【对应API函数声明】：

```
char *get_card_info( );
```

【指令功能简介】：
此函数的功能是用来轮询接收器上报的答题器的绑定信息的。

【接收器->DLL指令】：

```
{
    'fun': 'update_card_info',
    'card_id': '1234567890',
    'replace_uid': '0987654321'
}
```

JSON格式说明：

- card_id: 表示当前刷卡的答题器设备ID。
- is_replace_uid: 当前白名单中是否有UID被替换

  > 1:表示有UID被替换
  > 0:表示没有UID被替换

- replace_uid: 表示被替换的UID

注意：当前的系统中接收器最多总支持与120个答题器绑定，当系统中已经有120个答题器时，继续绑定新的答题器，最先绑定的答题器将会被替换掉，此时返回 is_replace_uid = 1，表示有答题器会替换，被替换的答题器UID为replace_uid的值。

2.2.4 停止绑定指令

【对应API函数声明】：

```
int bind_stop( );
```

【指令功能简介】：
此函数的功能是停止接收器的绑定功能。
【DLL->接收器指令】：

```
{
  'fun': 'bind_stop'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'bind_stop',
    'result':    '0'
}
//  0 : 成功
// -1 : 失败
```

2.3 答题器作答相关指令

- 普通工作模式（推荐用法）
  这种方式每次作答时都需要重新发送题目，发送题目之后接收器一致处于监听状态，答题器之后才

能提交答案，直到接收器收到停止作答指令，接收器将拒绝接受答题器指令。如果要开始新的答题，需要重新发送题目。

- 简单工作模式
  这种工作方式比较简单，只用发送一次题目，上位机不会发送停止作答指令，这个样接收器一致处于监听状态，答题器之后随时都能提交答案，此时相当答题器与接收器没有交互过程，仅仅是由答题器发送答案到接收器。这个有上位机软件根据**答题器提交答案的时间**判断答案是对应的那个题目。

注意：普通工作方式可以切换题目，有下发停止作答指令，简单工作方式不发送停止作答指令，用发送停止作答指令。

2.3.1 发送题目指令

【对应API函数声明】：

```
// 开始答题1
int  answer_start( int is_quick_response, char *answer_str );
// 开始答题2
int  answer_start_with_raise_hand( int is_quick_response,
                                   int raise_hand, char *answer_str );
```

【指令功能简介】：
将题目信息发送到接收器，再广播给答题器。

- is_quick_response：表示是否为抢答题。
  > 1:表示为抢答题，在应用获取答案之前，在答题器提交的答案所有中，动态库只缓存第一次的提交结果。
  > 0:表示为非抢答题，在应用获取答案之前，在答题器提交的答案所有中，动态库只缓存最后一次的提交结果。

发送题目有2中风格，一种是带有设置举手功能的一种是不带这样设置

- raise_hand：表示是否允许举手。
  > 1:允许举手。
  > 0:不允许举手。

注意：接收器发送完一次题目需要4S，接收器能自动对白名单题目中的答题器重发题目，无需PC端轮询发送。如果PC端要强制重新发送，需要调 answer_start接口来停止之前题目的发送，再发送新的题目。

【DLL->接收器指令】：
下面以一个例子来说明题目的JSON格式：
例如下发送4道题目的参数如下:

题目	类型	类型编码	题号	题号编码	作答范围	作答范围编码
第1题	单选	s	1	1	A~D	A-D
第2题	多选	m	13	13	A~F	A-F
第3题	判断	j	24	24	对或错	
第4题	评分	d	27	27	1~5	1-5
第5题	通用	g	36	27	1~9、A~F 或 对错	

对应的JSON格式的数据如下：

```
// 带有设置举手功能风格函数对应指令
{
  'fun': 'answer_start',
  'time': '2017-02-15 17:41:07:137',
  'raise_hand': '1',
  'questions': [
    {'type': 's','id': '1','range': 'A-D'},
    {'type': 'm', 'id': '13','range': 'A-F'},
    {'type': 'j','id': '24','range': ''},
    {'type': 'd','id': '27','range': '1-5'},
    {'type': 'g','id': '36','range': ''}
  ]
}

// 不带设置举手功能风格函数对应指令
{
  'fun': 'answer_start',
  'time': '2017-02-15 17:41:07:137',
  'questions': [
    {'type': 's','id': '1','range': 'A-D'},
    {'type': 'm', 'id': '13','range': 'A-F'},
    {'type': 'j','id': '24','range': ''},
    {'type': 'd','id': '27','range': '1-5'},
    {'type': 'g','id': '36','range': ''}
  ]
}
```

【接收器->DLL指令】：

```
{
    'fun': 'answer_start',
    'result':    '0'
}
// 0  : 成功
// -1 : 当硬件忙，无法装载发送。
// -2 : 题目长度太长，无法装载发送。
```

2.3.2 举手设置指令

【对应API函数声明】：

```
int  raise_hand_set( int raise_hand );
```

【指令功能简介】：

修改当前题目的举手设置功能，例如在开始作答的指令中是不允许作答的，但是在后面的想将其设置为允许作答的状态，级就可以用这条这令来修改设置。

- raise_hand：表示是否允许举手。
  > 1:允许举手开始。
  > 0:允许举手停止。

【DLL->接收器指令】：

```
{
  'fun': 'raise_hand_set',
  'raise_hand': '1'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'raise_hand_set',
    'result':    '0'
}
// 0  : 成功
// -1 : 发送失败。
```

2.3.3 签到设置指令

【对应API函数声明】：

```
int  sign_in_set( int attendance );
```

【指令功能简介】：
签到指令是开始上课之后，老师发送一个签到指令，之后学生便可以通过，使用答题器来完成签到。

- attendance：表示是否允许举手。
  > 1:签到开始
  > 0:签到结束

【DLL->接收器指令】：

```
{
  'fun': 'sign_in_set',
  'attendance': '1'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'sign_in_set',
    'result':    '0'
}
// 0  : 成功
// -1 : 发送失败。
```

2.3.4 获取答案指令

【对应API函数声明】：

```
char * get_answer_list( );
```

【指令功能简介】：

获取答题器提交的答案，上位机发送完题目之后，通过轮询的方式来获取答题器提交的答案。

【接收器->DLL指令】：

注意：由于接受器内部RAM空间的限制，无法存放太多的数据，一旦有答题器发送数据上来，接收器会立即发送给上位机。

```
{
  'fun': 'update_answer_list',
  'card_id': '0983584512',
  'rssi': '-51',
  'update_time': '2017-02-15 17:41:07:237',
  'raise_hand': '1',
  'attendance': '1',
  'answers': [
    {'type': 's','id': '1','answer': 'A'},
    {'type': 'm','id': '13','answer': 'BC'},
    {'type': 'j','id': '24','answer': 'false'},
    {'type': 'd','id': '27','answer': '3'},
    {'type': 'g','id': '36','range': 'ACF'}
  ]
}
```

JSON格式说明：

- card_id: 表示答题器的设备ID。
- rssi: 接收到答题器数据时的信号强度，**使用时要保证信号强度大于-90。**
- update_time: 表示答题器提交答案的时间。
- raise_hand：当前学生是否按了举手按键。
- attendance：当前学生是否按了签到按键。
- answers: 详细的答案信息

> type: 题目类型(question type)。
>
> > s ：表示单选题，单选最大范围限制在A~F；
> > m：表示多选题，多选最大范围限制在A~F；
> > j：表示判断题；
> > d：表示数字评分，评分最大范围限制在0~9；
> > g：表示通用题型，支持作答范围允许为单选、多选、判断和数字；
>
> id: 题号(question id):**题号范围1~99**。
> answer: 当前题目提交的答案。

【DLL->接收器指令】：

```
{
    'fun': 'update_answer_list',
    'result':    '0'
}
//  0 : 成功
// -1 : 失败
```

2.3.5 停止作答指令

【对应API函数声明】：

```
int answer_stop();
```

【指令功能简介】：
此函数的功能是停止作答功能，调用该函数将停止接收器的接收功能，并清除答题器上面显示的题目信息。
【DLL->接收器指令】：

```
{
  'fun': 'answer_stop'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'answer_stop',
    'result':    '0'
}
//  0 : 成功
// -1 : 失败
```

---

3 设置指令

3.1 理论说明

设置指令是配置答题器参数的指令，主要操作是答题器上面的一块标签芯片，设置参数就是往这块标签中写入数据，答题器开机的时候会读取这个里面的设这参数，来调整自己的通讯参数。

3.2 答题器设置学号相关指令

3.2.1 设置学号指令

【对应API函数声明】：

```
char *set_student_id( char *student_id_str );
```

【指令功能简介】：
此函数完成将学生的学号信息写入到答题器。我们给学号分配的空间是10个byte。如果是字节，这个学号字符串需是纯数字，长度最大20位。
注意：此指令有2次返回，发送指令时会根据当前硬件状态是否忙，和参数合理性检查返回是否合理，返回是否执行成功；当答题器完成刷卡写入学号时，会将答题器的设备ID和写入的学号读回。
【DLL->接收器指令】：

```
{
  'fun': 'set_student_id',
  'student_id': 'xxxxxxxxx'
}
```

【接收器->DLL指令】：执行结果返回

```
{
  'fun': 'set_student_id',
  'result': '0'
}
//  0 : 成功
// -1 :设备忙
// -2 :参数非法
```

3.2.2 获取设置学号卡片信息指令

【对应API函数声明】：

```
char *get_student_id_info( );
```

【指令功能简介】：
接收器接收到绑定的学号信息之后，一直等待卡进入磁场，一旦答题器进入磁场接收器就会将信息写入，写入成功之后会立刻返回绑定的答题器的ID和学生ID。
由于绑定的时间不可预知，获取这个绑定信息需要PC端一直轮询来获取结果。

【接收器->DLL指令】：写入学号结果返回

```
{
  'fun': 'set_student_id',
  'card_id': '1234567890',
  'replace_uid': '0987654321',
  'student_id': 'xxxxxxxxx1'
}
```

JSON格式说明：

- card_id：写入学号信息的答题器的uID。
- is_replace_uid: 当前白名单中是否有UID被替换

  > 1:表示有UID被替换
  > 0:表示没有UID被替换

- replace_uid: 表示被替换的UID
- student_id：写入之后读回的学生ID，10进制学生ID的字符串，长度最大20位。

说明:答题器的 开启绑定 与 写入学号 的关系：

- **写入学号**与**开启绑定**都是占用13.56Mhz的刷卡的硬件资源，所以无法在开启绑定的情况下执行写入学号指令，系统只允许在某一时刻只有一条指令占用此硬件。
- 如果使用接收器给**没有绑定**自己的答题器写入学号，则在写入学号的过程中**自动完成**绑定。
- 如果在本机上已经绑定的答题器，写入学号，则**原绑定信息不变**。

3.3 答题器设置通讯参数相关指令

答题器在与接收器通讯的过程中有一些关键参数有可能需要修改，比如信道和发送功率等，设置的这些参数对通讯是否成功至关重要。

注意：

1. 这些参数是在答题器贴近接收器刷卡区的过程中写入到答题器的。

2. 修改这些参数必须完成刷卡才能生效，建议在完成绑定之前先设定这些参数。

3. 系统通讯关键参数出厂时已经写入了默认值，不修改可以直接使用。

3.3.1 设置信道指令

【对应API函数声明】：

```
int set_channel( int tx_ch, int rx_ch );
```

【指令功能简介】：

这条函数是用来设置答题器与接收的通选信道的，有的时候为了避免干扰，需要修改答题器与接收器的信道，**发送与接收的信道是不能相同。**

- tx_ch：答题器的发送信道，范围1~11。
- rx_ch：答题器的接收信道，范围1~11。
  注意：此处我们推荐几组信道设置

组别	发送信道 TX_CH	接收信道RX_CH
第1组	2	4
第2组	1	3
第3组	5	7
第4组	6	8
第5组	9	11

1. 信道设置上不要将发送与接收设置为2个相邻的信道，通讯质量将更好。
2. 临近使用的接收器如果信道设置为非重叠，通讯质量将更好。

【DLL->接收器指令】：

```
{
  'fun': 'set_channel',
  'tx_ch': '2',
  'rx_ch': '4'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'set_channel',
    'result':    '0'
}
//  0 : 成功
// -1 :失败
```

3.3.2 设置发送功率指令

【对应API函数声明】：

```
int set_tx_power( int tx_power );
```

【指令功能简介】：
此函数是用来设置答题器的发送功率，我们答题器默认是最大功率发送的，有时候这个功率是比较大的，当应该场景不需要这么大的功率时，我们可以使用这个接口修改发送功率。答题器的发送功率，范围1~5，分别表示对应的不同档位大发送功率，1当最小，5档最大。
注意：答题器默认出厂的设置为5档，用户如果要修改设置的档位，请根据句答题器提交答案中的RSSI的值来设置，保证使用过程中最角落的答题器提交的答案的RSSI大于-90即可。

【DLL->接收器指令】：

```
{
  'fun': 'set_tx_power',
  'tx_power': '5'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'set_tx_power',
    'result':    '0'
}
//  0 : 成功
// -1 :失败
```

3.3.3 查询设备信息指令

【对应API函数声明】：

```
char *get_device_info( );
```

【指令功能简介】：
此函数用来查询接收器的设备信息，包括设备ID，软件版本号，公司信息和接收器的配置信息。
【DLL->接收器指令】：

```
{
  'fun': 'get_device_info'
}
```

【接收器->DLL指令】：

```
{
  'fun': 'get_device_info',
  'device_id': '3768114656',
  'software_version': 'v0.1.2',
  'hardware_version': 'ZL-RP551-MAIN-F',
  'company': 'zkxltech',
  'tx_ch': '2',
  'rx_ch': '4',
  'tx_power': '4',
  'attendance_status': 'off',
  'list': [
    {'upos': '0','uid': '4020554507'},
    {'upos': '1','uid': '0091540235'}
  ]
}
```

JSON格式说明：

- device_id: 表示接收器的设备ID。
- software_version: 接收器当前的软件版本。
- hardware_version:接收器当前的硬件版本。
- company：公司的缩写。
- tx_ch:绑定过程中需要写入的答题器发送数据的信道。
- rx_ch:绑定过程中需要写入的答题器接收数据的信道。
- tx_power:绑定过程中需要写入的答题器发送功率的档位。
- attendance_status:2.4G考勤的状态，默认状态是关闭

  > off : 关闭
  > on : 开启

- list:表示于此接收器绑定的的答题器表。

  > upos:绑定过程中，接收器分配给答题器的快速索引的序号，**重复绑定，分配的序号不变**。
  > uid:绑定过程中，接收器读取的答题器的设备ID。

3.3.4 2.4G考勤设置指令

接收器支持直接使用串口指令开启或者关闭2.4G考勤功能，定义下面指令：
【对应API函数声明】：

```
int attendance_24g( int is_open, int pro_index )
```

【指令功能简介】：
此函数是开启或者关闭答题器2.4G考勤功能。

- is_open：是否开启2.4G考勤功能 ；
- tx_ch：考勤所使用的信道，信道的范围为1~127；

【DLL->接收器指令】：

```
{
  'fun': 'attendance_24g',
  'attendance_status': '1',
  'attendance_pro_index': '0'
}
```

JSON格式说明：

- attendance_status:2.4G考勤功能设置的状态。

  > 1:表示开启
  > 0:表示关闭

- attendance_pro_index:2.4G考勤功能设置的协议的索引号。

  > 协议的索引号范围:0~12。

协议索引号与协议对照表

协议索引号	协议名
0	我司
1	江西移动
2	重庆移动
3	内蒙移动
4	广西移动(初稿)
5	广西移动(定稿)
6	贵州移动
7	甘肃移动
8	山西移动_天波
9	山西移动_鑫诺
10	山西移动_统一协议
11	安徽移动
12	四川移动

【接收器->DLL指令】：

```
{
  'fun': 'attendance_24g',
  'result': '0'
}
//  0 : 成功
// -1 :失败
```

3.3.5 答题器调试信息设置指令

【指令功能简介】：
这条指令是用来维护设备是通过刷卡开启或者关闭答题器查看设备信息的功能
【DLL->接收器指令】：

```
{
  'fun': 'dtq_debug_set',
  'open_debug': '1'
}
```

JSON格式说明：

- open_debug:是否允许答题器组合件查看设备信息的。
  > 1:表示开启
  > 0:表示关闭

【接收器->DLL指令】：

```
{
  'fun': 'dtq_debug_set',
  'result': '0'
}
//  0 : 成功
// -1 :失败
```

---

4 其他指令

这些指令并不影响答题器的通讯，主要是一些辅助作用。

4.1 查看配置信息指令

在跟换接收器的时候，我们需要重新配置设置参数，且需要重新绑定，为了减少这些操作，将信息存放到本地，此时需导出这些指令，将信息存放到本地，定义下面指令：
【对应API函数声明】：
无
【指令功能简介】：
查看所有通讯先关配置数据。

【DLL->接收器指令】：

```
{  'fun': 'check_config'}
```

【接收器->DLL指令】：

```
{
  'fun': 'check_config',
  'addr':'3768114656',
  'tx_ch': '2',
  'rx_ch': '6',
  'tx_power': '5',
  'list': [
    {  'upos': '0',  'uid': '0983584512' },
    {  'upos': '1',  'uid': '1180852736' }
  ]
}
```

JSON格式说明：

- tx_ch:绑定过程中需要写入的答题器发送数据的信道。
- rx_ch:绑定过程中需要写入的答题器接收数据的信道。
- tx_power:绑定过程中需要写入的答题器发送功率的档位。
- list:表示于此接收器绑定的的答题器表。
  - upos:绑定过程中，接收器分配给答题器的快速索引的序号，**重复绑定，分配的序号不变。**
  - uid:绑定过程中，接收器读取的答题器的设备ID。

【配置文件说明】：
此处采用**windows的配置文件格式**来存放导出的**配置信息。**
文件名的命名规则
格式：DTQ_RP551CPU+设备ID.ini
文件名需要有接收器的ID信息，这样能保证一台电脑上能接入多个接收器，且直接通过指令查询到设备ID之后就能直接定位文件则对应的设备文件信息如下：

```
DTQ_RP551CPU_3768114656.ini
```

4.2 导入配置信息指令

在跟换接收器的时候，我们需要重新配置设置参数，且需要重新绑定，为了减少这些操作，将信息存放到本地，此时需导入这些指令，将信息直接导入到设备，定义下面指令：
【对应API函数声明】：
无
【指令功能简介】：
导入通讯相关配置数据到接收器。
【DLL->接收器指令】：

```
{
  'fun': 'import_config',
  'addr':'3768114656',
  'tx_ch': '2',
  'rx_ch': '6',
  'tx_power': '5',
  'list': [
    {  'upos': '0',  'uid': '0983584512' },
    {  'upos': '1',  'uid': '1180852736' }
  ]
}
```

JSON格式说明：

- tx_ch:绑定过程中需要写入的答题器发送数据的信道。
- rx_ch:绑定过程中需要写入的答题器接收数据的信道。
- tx_power:绑定过程中需要写入的答题器发送功率的档位。
- list:表示于此接收器绑定的的答题器表。
  - upos:绑定过程中，接收器分配给答题器的快速索引的序号，**重复绑定，分配的序号不变。**
  - uid:绑定过程中，接收器读取的答题器的设备ID。

【接收器->DLL指令】：

```
{
    'fun': 'import_config',
    'result':    '0'
}
//  0 : 成功
// -1 :失败
```

关于设备信息的导入与导出的说明：
将一个设备信息的导出，再导入另一个设备，就能实现对一个设备的**克隆**，这个克隆的设备就可以无需任何设置，能像之前的设备一样工作。

4.3 bootloader指令

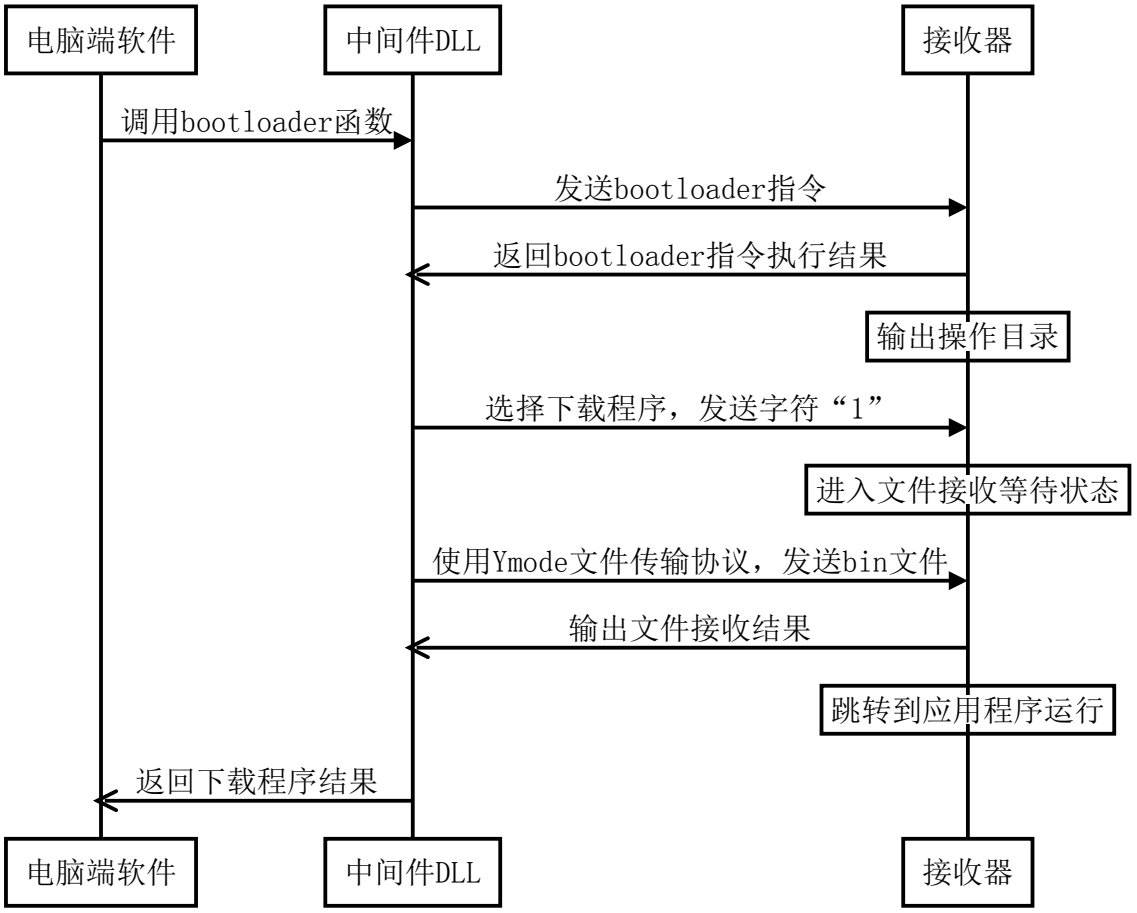
接收器支持直接使用串口更新程序的功能，定义下面指令：
【对应API函数声明】：

```
int bootloader( char *file_path );
```

【指令功能简介】：
此指令是完成接收器的升级，此处只需要给出文件的路径。

- file_path ：表示下载的bin文件的路径。

升级的过程比较复杂，设计到的指令比较多，下面给出流程图：



bootloader 指令格式如下：

【DLL->接收器指令】：

```
{
    'fun': 'bootloader'
}
```

【接收器->DLL指令】：

```
{
    'fun': 'bootloader',
    'result':    '0'
}
//  0 : 成功
// -1 :失败
```

4.4 答题器自检指令

接收器支持直接使用串口指令开启答题器自检功能，定义下面指令：

【DLL->接收器指令】：

```
{
  'fun': 'dtq_self_inspection'
}
```

【接收器->DLL指令】：

开启指令执行返回：

```
{
  'fun': 'dtq_self_inspection',
  'result': '0'
}
//  0 : 指令发送成功
// -1 :失败
```

信号强度返回：

```
{
  'fun': 'dtq_self_inspection',
  'card_id': '0091540235',
  'dtq_rssi': '-44',
  'jsq_rssi': '-44'
}
```

JSON格式说明：

- card_id:当前测试的答题器的设备ID
- dtq_rssi:答题器发送数据的信号强度
- jsq_rssi:接收器发送数据的信号强度

附录1错误返回

1 json语法错误

接收器收到指令如果不满足json语法，将会报错
格式如下：
【接收器->DLL指令】：

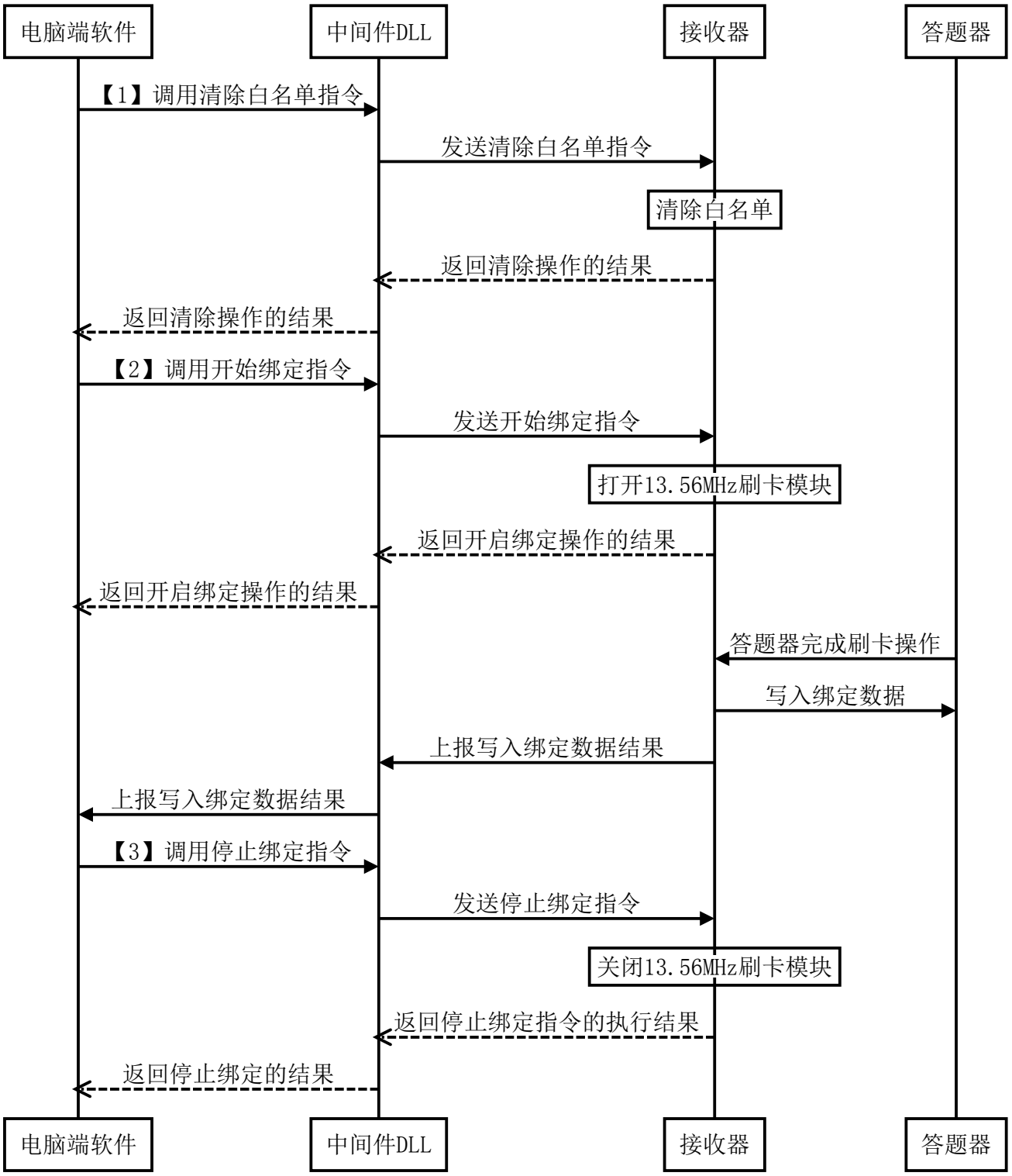
```
{
  'fun': 'Error',
  'description': 'json syntax error!'
}
```

2 未识别指令

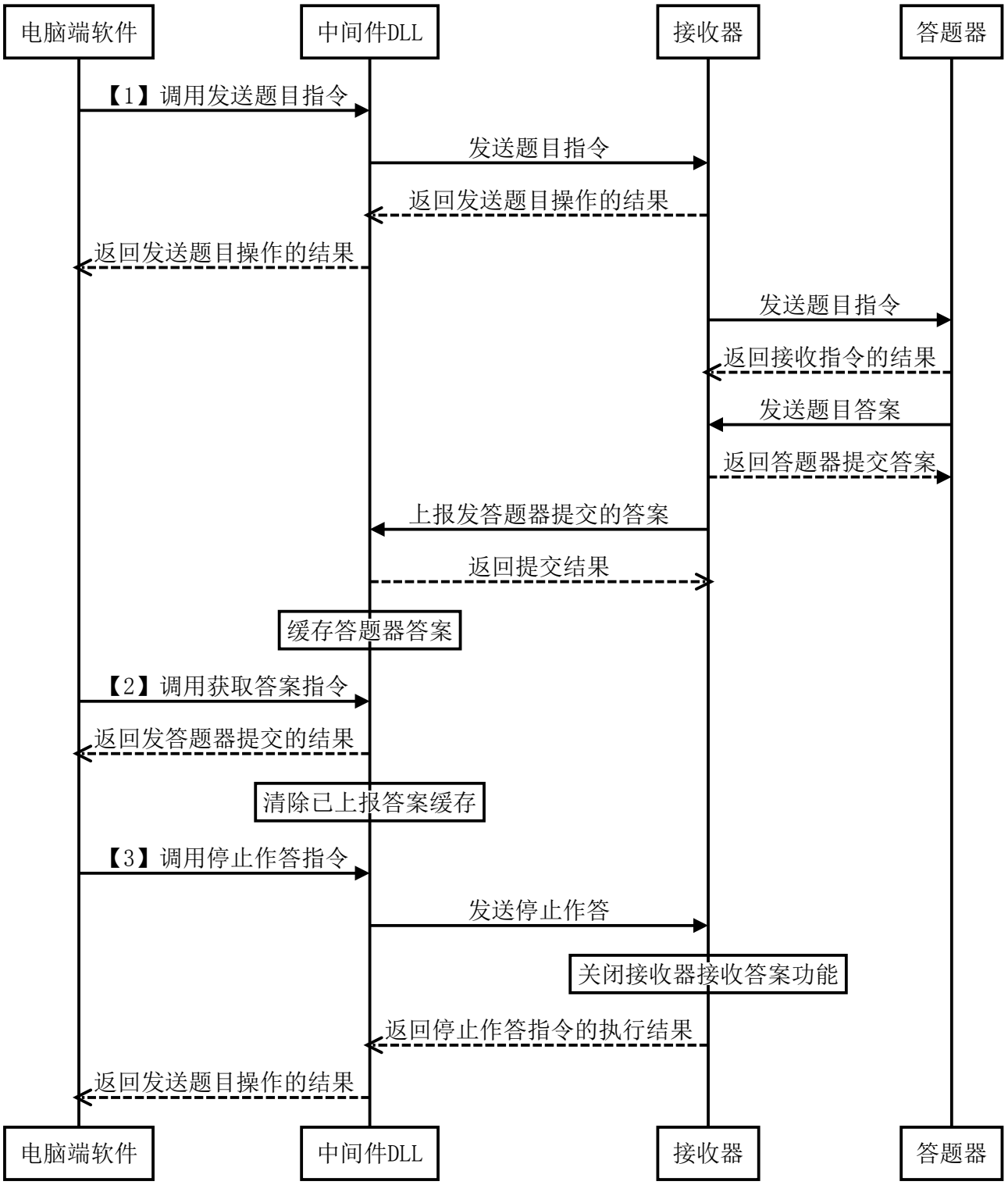
接收器收到指令符合json语法，但是字指令的关键字出错，且这条指令未定义，会报错。
格式如下：
【接收器->DLL指令】：

```
{
  'fun': 'Error',
  'description': 'unknow cmd!'
}
```

附录2 绑定过程时序图


电脑端软件
中间件DLL
接收器
答题器
【1】调用清除白名单指令
发送清除白名单指令
清除白名单
返回清除操作的结果
返回清除操作的结果
【2】调用开始绑定指令
发送开始绑定指令
打开13.56MHz刷卡模块
返回开启绑定操作的结果
返回开启绑定操作的结果
答题器完成刷卡操作
写入绑定数据
上报写入绑定数据结果
上报写入绑定数据结果
【3】调用停止绑定指令
发送停止绑定指令
关闭13.56MHz刷卡模块
返回停止绑定指令的执行结果
返回停止绑定的结果
电脑端软件
中间件DLL
接收器
答题器

附录3 答题过程时序图


电脑端软件
中间件DLL
接收器
答题器
【1】调用发送题目指令
发送题目指令
返回发送题目操作的结果
返回发送题目操作的结果
发送题目指令
返回接收指令的结果
发送题目答案
返回答题器提交答案
上报发答题器提交的答案
返回提交结果
缓存答题器答案
【2】调用获取答案指令
返回发答题器提交的结果
清除已上报答案缓存
【3】调用停止作答指令
发送停止作答
关闭接收器接收答案功能
返回停止作答指令的执行结果
返回发送题目操作的结果
电脑端软件
中间件DLL
接收器
答题器