1. IReceive对像

一、接收对像类说明：

１、**IReceive::IReceive(const char \*cmdstr, void (\*cb)(unsigned char \*dat, unsigned char len))**

item的组成

a\可以由构造函数传入，

b\也可以由：void IReceive::setCmdStr(const char \*cmdstr)

c\item 仅仅是一个指针

d\作用：是用来作为关键字，用于匹配（比如led），比如从远程接收回来的

字符串，含用此关键字，说认为此字符串，是合适此对像的。

当接收回来的字符串:

(本地地址：２个字节）－（远程地址：２个字节）－（包索引号１个字节）－**（匹配字最多１６个字节）**－（变量n个字节）)

，首先要看看收到字符串是否有led,然后

接收回来的字符串led + 变量部分

变量部分：on:表示让led打开

　off:表示让led关闭

2、当收到对像敏感的字符串后，就会调用回调函数。

这回调函数由外面定义，从构造函数传入

\* \*/

**２、void IReceive::setCmdStr(const char \*cmdstr)**

设置关键字，也说是给item赋值

**３、unsigned char IReceive::getCmdLen(void)**

获取item的长度

**4、unsigned char Ireceive::cmpAction(**

**unsigned char \*dat,**

**unsigned char len,**

**unsigned char \*var)**

dat表示所有从远程接收回来字节的集合

len表示上面字节长度

var表示动作字符串

返回的结果：如果为１表示，接收回来的动作和指定的动作是一样的

是判断，dat字符串中，是否有var字符串。如有的话返回１，否则返回０

返回１：表示，找到对应的字符串了。

返回０，表示，没有找到对应的字符串。

关键字相同的（比如，fan,还有很多子类对像，比如：speed,stop, 等等）。

5、void IReceive::msg\_handler(unsigned char \*dat, unsigned char len)

这个函数，在外部调用，当收到远程的字符串的函数时，就会调用这个对应的函数了。

这个函数将会调用用户处理函数，也就是构造函数传过来的cb.

6、boolean IReceive::isNewPackage(unsigned char \*dat)

根据dat就是远程传过来的完整包了，所有信息都在里面.

什么是新包？

初始值有一个index变量为０，当收到一个包时，会把包的第４个

字节的值和index比较，如果相同，**表示是重发的包**，如果不同，表示是

新包，这时把值赋值给index，下次再收到包时，就会变成重发的包了。

为什么要加入这个机制？

主要是因为，远程主动发送命令包的时候，本地也收到了，也执行了相关动作，

但是，本地发送应答包，远程在一定时间没有收到，然后，远程认为本地没有收到包。

所以，就会再发一个带相同索引值的包，从而导致本地又重复执行相同的动作，所以就

加入了这个机制。

比较是不是新包，只有在接收方会用，索引的记数在远程主动方去完成。

7、void Ireceive::clearAckBuf()

这个函数是把应答包的缓冲区清０，**目前缓冲区的最大长度为３２**。

8、void IReceive::saveAckBuf(unsigned char \*buf, unsigned char len)

这个函数是保存应答缓冲区。

9、unsigned char \*Ireceive::getAckBuf()

得到应答缓冲区的指针

１０、unsigned char Ireceive::getAckBufLen()

得到应答缓冲区的长度

二、如何使用接收对像：

**建立IReceive对像，当有数据收到的时候，会回调 cb\_fan。**

**第一步、**

static void cb\_fan(unsigned char \*dat, unsigned char len);

IReceive irec\_fan("fan", cb\_fan);

fan是关键字，当这个对像接收到fan开头的字符串后，就认为是配对的。

完整的是这样：

fanspeed：其中，speed表示变量部分

fanstop：其中，stop表示变量部分

...

**第二步、**

把从远方收到的数据，派发给IReceive对像处理

void u2::receive\_listener(unsigned char \*data, unsigned char len)

...

if (m\_init == 1) {

isender.msg\_handler(data, len);

#ifdef LED\_FUNC

irec.msg\_handler(data, len);

#endif

**#ifdef FAN\_FUNC**

**irec\_fan.msg\_handler(data, len);**

**#endif**

}

...

**第三步、定义这个函数**

#ifdef FAN\_FUNC

/\*

\*接收到从远程过来的风扇控制命令，然后根据变量判断，是开灯还是关灯，

并执行相关动作。

\*/

static void cb\_fan(unsigned char \*dat, unsigned char len)

{

**if(irec\_fan.isNewPackage(dat)) {**

if(irec\_fan.cmpAction(dat, len , (unsigned char \*)"speed")) {

speed\_press\_task.startDelayed();

myu2->m\_comm->send("fanspeed:ok", 11);

irec\_fan.saveAckBuf((unsigned char \*)"fanspeed:ok", 11);

} else if (irec\_fan.cmpAction(dat, len , (unsigned char \*)"off")) {

off\_press\_task.startDelayed();

myu2->m\_comm->send("fanoff:ok", 9);

irec\_fan.saveAckBuf((unsigned char \*)"fanoff:ok", 9);

}

} else {

myu2->m\_comm->send((const char \*)irec\_fan.getAckBuf(), irec\_fan.getAckBufLen());

}

}

#endif

1. DelayRun对像：

**一、使用 以下面的例子说明：**

/\*风扇调速的任务\*/

DelayRun **speed\_release\_task**(500, release\_speed\_key);

DelayRun **speed\_press\_task**(10, press\_speed\_key, &speed\_release\_task);

**二、构造函数说明：**

构造１，

DelayRun speed\_release\_task(500, release\_speed\_key);

第一参数表示，延时数单位为毫秒

第二参数为回调函数，延时指定时间后，执行这函数

构造２，

DelayRun speed\_press\_task(10, press\_speed\_key, &speed\_release\_task);

第一参数表示，延时数单位为毫秒

第二参数为回调函数，延时指定时间后，执行这函数

第三个参数表示，执行完第二个参数回调函数后，执行第二个DelayRun对像.

**三、回调函数的参数：**

static boolean press\_speed\_key**(Task\* task);**

**四、使用这个类要用到的头文件：**

**#include <DelayRun.h>**

**#include <Task.h>**

上面的情况就是：

10mS-> **press\_speed\_key**->500mS-> **release\_speed\_key->结束**