题目

城市天气查询

前言

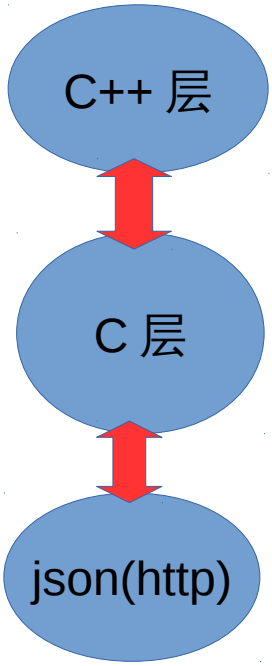
天气一直是人们关注的焦点，

系统设计思路

整体我只需要实现两层，姑且称为(C++层)，(C 层)，服务器及以下由 api 提供方实现了。

如果按互联网协议栈来看，我是只需要实现应用层， 传输层以及以下由 api 提供方实现，虽然也不全是，因为提供方是用 http 协议，然后还用 json 作为格式， 所以这么看其实很繁杂，如果 api 提供的是更加底层一点的数据，看起来就没那么别扭了。

Big Picture



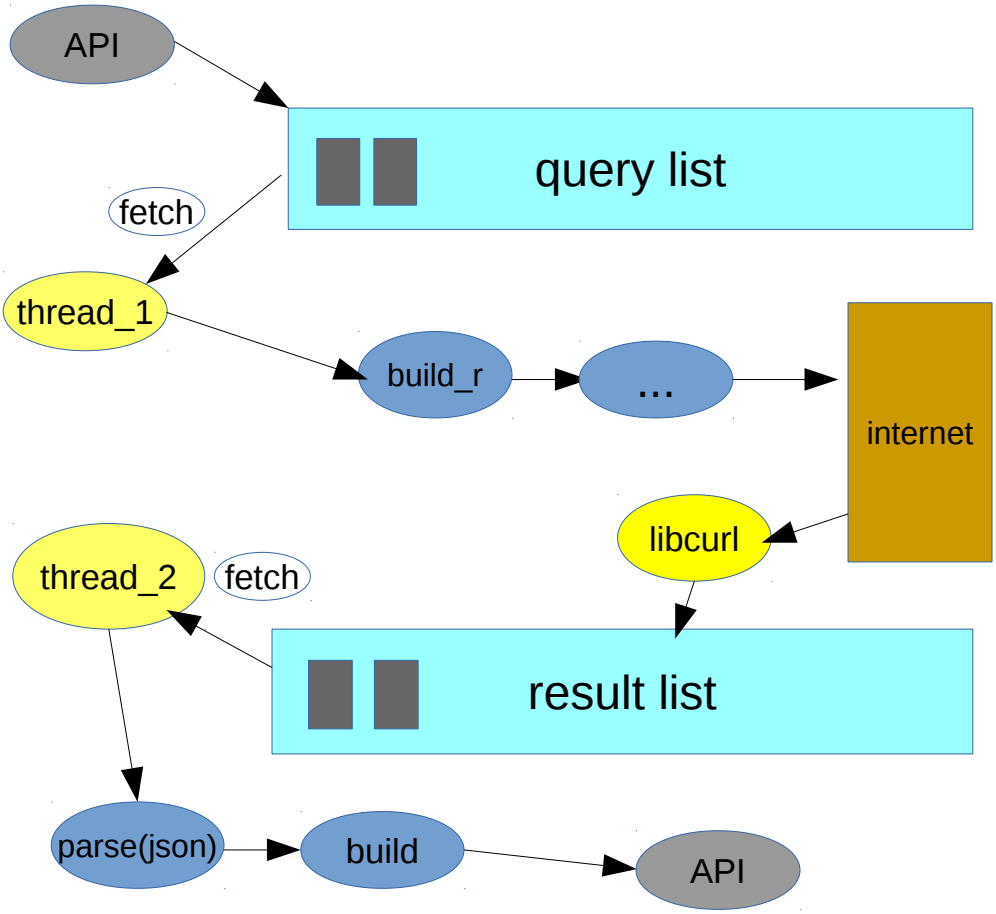
业务层(C 实现)就是完成：

1. 展示层发送查找的数据，由其完成处理(build_request)，使其满足 api 调用的规范，具体就是添加 api 的服务器 url, 并完成一些细化的要求，比如语言。总之这一部分就是字符串处理为主。然后调用下层接口。
2. 由网络而来的数据(json)格式，我们需要 parse，这里由 json.org 提供一个 json_parser, 然后再紧接着，几个处理，完成展示层需要的天气数据。调用接口，通知上层。

展示层(C++实现) 就是完成：

1. 美观的界面
2. 能对下层发送的事件进行处理并展示
3. 主要完成一些信号和槽的设计，搜索框 切换天气图标...

为了使用户界面反应迅速，是不可能让其睡眠等待，所以使用线程，让底层完成数据的处理之后，发送事件通知，这时候界面才响应并作一定的处理。这样的界面用户体验更好。



thread_1:

当上层调用 send_query() 即发送查询请求，只需要把请求的数据放在 query_list，便可以立即返回，这是**没有阻塞**的，所以用户界面不会卡顿，当然完全可以让用户界面直接用一个线程完成全部操作(但是数据传递会有一点问题？)，我这是其中一种办法。

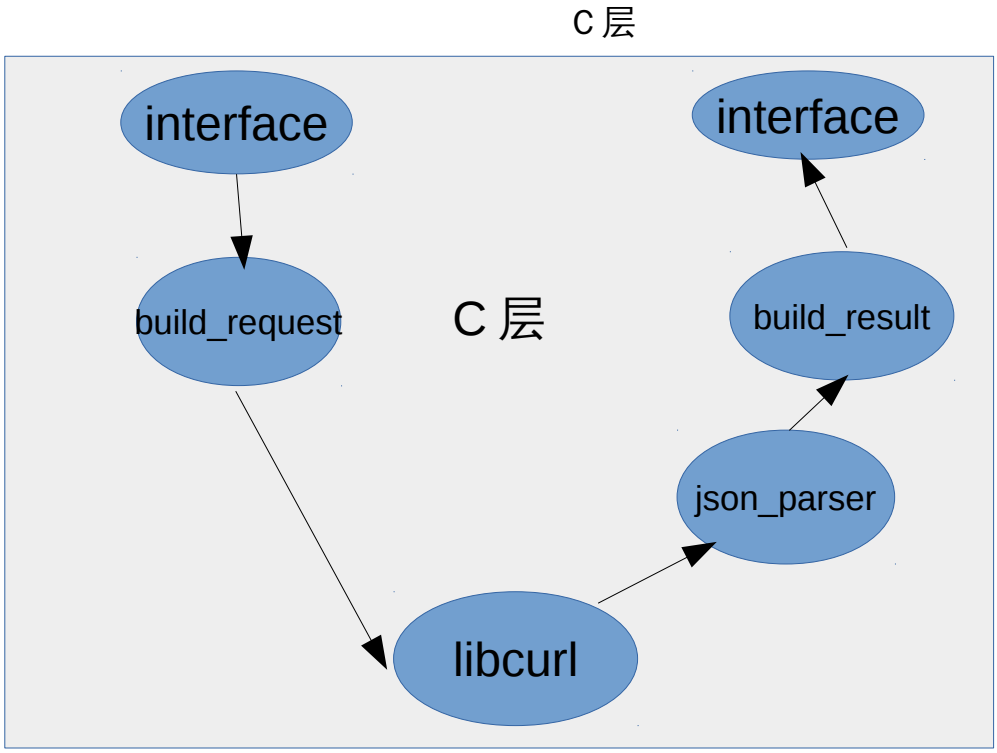
build: 就是完成数据的处理，接着就可以发送到 Internet 了。当 **libcurl** 完成数据发送并接受到服务器的应答，直接把数据放回到 result_list. 这时候线程的工作完成，继续查看 query_list 处理上

层的数据。

thread_2:

这个线程很容易想到，线程 1 最后面干的事情就是把网络来到的 raw 数据放在的 result_list，线程 2 就是把数据 cook，然后通知上层，其中涉及到 js 数据的解析。

系统设计过程



libcurl: 库，实现 http 协议，对指定的 url 进行 post, get, put...等等操作，是通往传输层的接口。

json_parser: 实现对 json 格式的文件解析，使之变为 C 语言中的数据。

build_result: 处理后的 json，在这里进行组装，变成展示层需要的数据结构。

build_request: 把展示层的数据进行处理，使之能被远程的 server 处理。

传递的数据结构：

```
struct query {
        unsigned int flag;
        struct list_head q_node;
         char * url;
        size_t size;
        void * data;
        void * arg;
};
```

可以这么说，struct query 是真正连接两层的核心，API 就是利用这个数据结构实现。当上层想要发送一个查询数据的时候，比如查询广州的天气，那么就会在搜索框里按下，guangzhou, 这时候在展示层会新建几个类，但是最重要的一点就是会新建一个 struct query，在 url 这里，填入"guangzhou", 然后就调用 API 把 struct query 这个结构放在 query_list, 接下来由线程完成接下来的全部操作，界面只需要等下层的事件到来再处理即可。

Flag:

```
#define QUERY_FAIL  (1UL)
#define QUERY_SUCCESS (1UL << 1)
#define QUERY_UPDATE (1UL << 2)
```

QUERY_SUCCESS: 成功， 意味着上层可以成功解析数据
QUERY_FAIL： 此次查询是失败的，需要上层重新处理
QUERY_UPDATE: 当界面按下刷新按钮，会设置这个标志

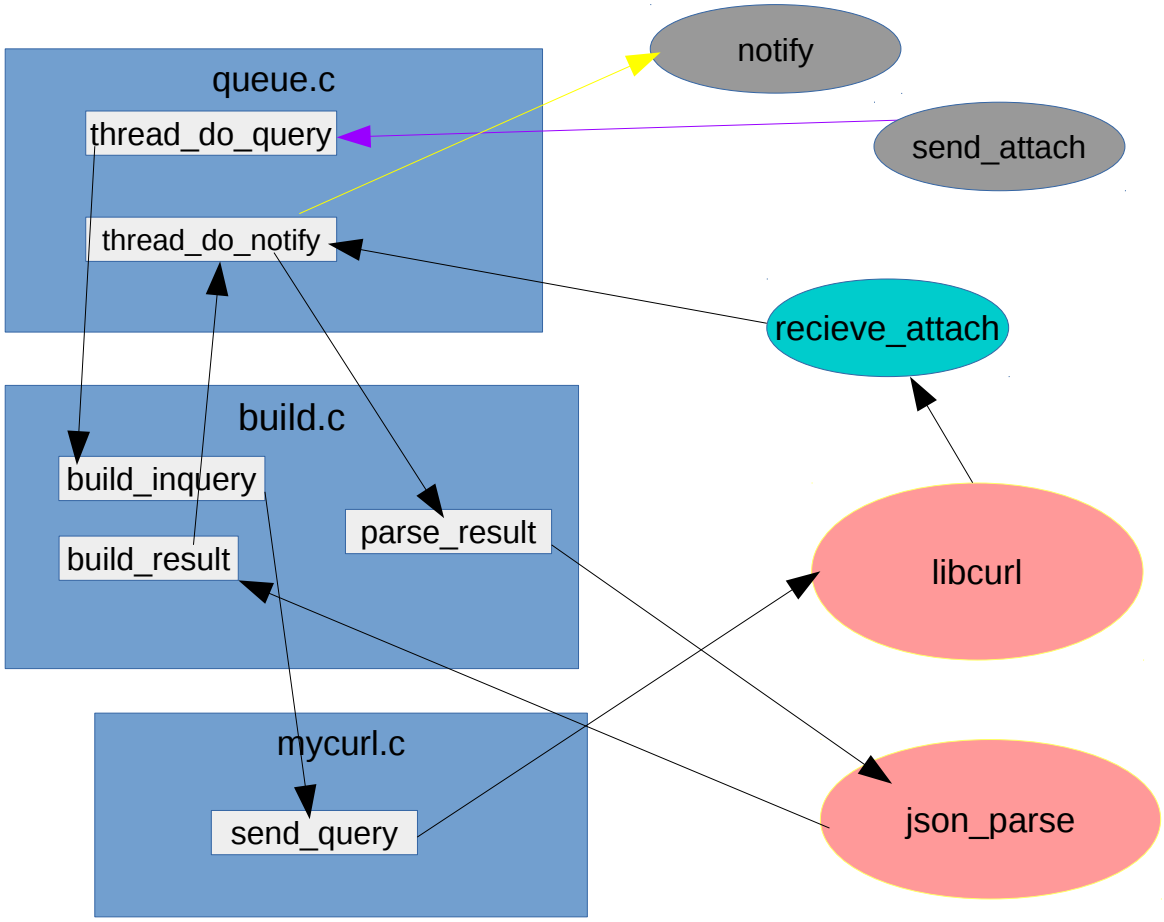
q_node: list_head:
链表连接的结构，加上它就是为了挂在链表上（从 linux 内核引入）

url: char *
第一次查找的时候，上层传递就是城市的名字，查找一次之后存放的就是完整 url 了，下次刷新(通过设置 flag 位)的时候就不要经过 build_url 这一繁杂的过程。

data: char *
在 libcurl 处理之后，data 指向的就是 json 数据，经过 parse 之后，指向就是 json_value, 经过 build 之后，data 指向的就是可以被展示层使用的数据，所以在不同时刻，这个指向的内存都不一样。

arg : void *
这个存放的是 malloc 出这个数据结构的类地址，以后应该要废除，因为它只在发送事件的时候用到了。而 query 的地址本来就可以确定这个类了。

C 层函数(从紫色开始到黄线结束)



中间还涉及两条链表，已经包含在线程中，不画出了，send/recieve_attach，就是对其操作。然后 send_attach 和 notify 就是两层的 API。

这些函数大了部分传递的参数就是 struct query*，操作起来非常方便。

```
typedef struct {
        unsigned int length;
        char * ptr;
}string;
```

```
struct location {
        string id;
        string name;
        string country;
        string timezone;
        string time_offset;
};

struct weather {
        string text;
        string code;
        string temperature;
};

struct result {
        struct location * locate;           /* location */
        struct weather * now;                   /* weather now */
        #ifdef __SUGGESTION__
        struct suggest * sugs;              /*      suggestion if exist*/
        #endif
};
```

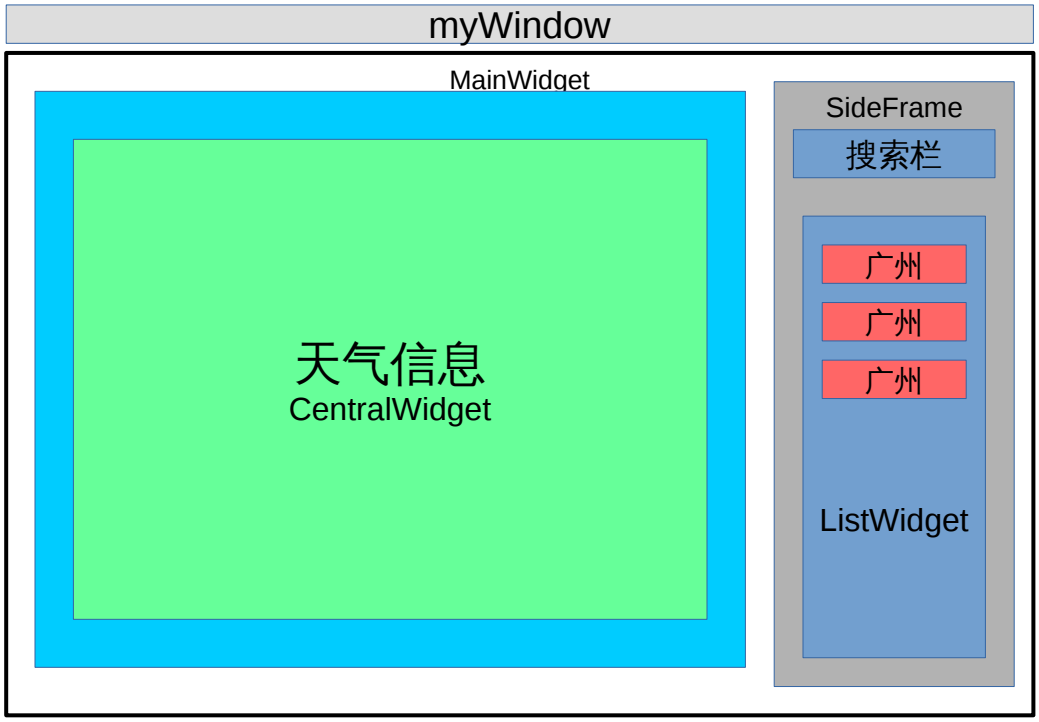
以上是经过 build_result 函数处理之后的数据结构，供上层使用。struct result 的地址最后存在了 query→data。

C++层

这一层是用 Qt 的库实现，因为它的跨平台，今后在嵌入式也有可能会有点用，所以趁机试一试。实现起来很简单，就是信号和槽的设计，还有就是布局，还有每次数据到来的时候发送一个事件，使得数据能够被及时的处理。

class:

1. Mywindow　　自定义的框，最大化最小化，刷新等按钮，不过程序已经设置成固定大小了
2. MainWidget　　存放天气展示框，和菜单栏的类，主要是和 MyWindow 协调
3. SideFrame　　菜单栏，里面有搜索栏和选择展示的城市的列表
4. SearchBox　　就是搜索框的类
5. ListWidget　　城市列表的类，显示目前搜索的类
6. CentralWiget　　每一个城市都会自带的这个类，存放天气信息
7. ReusItItem　　相当于 C 层的 struct query 是大部分信号传递的参数用来协调各窗口的互动
8. QweatherEvent　事件类，当数据到达的时候发送的 Event

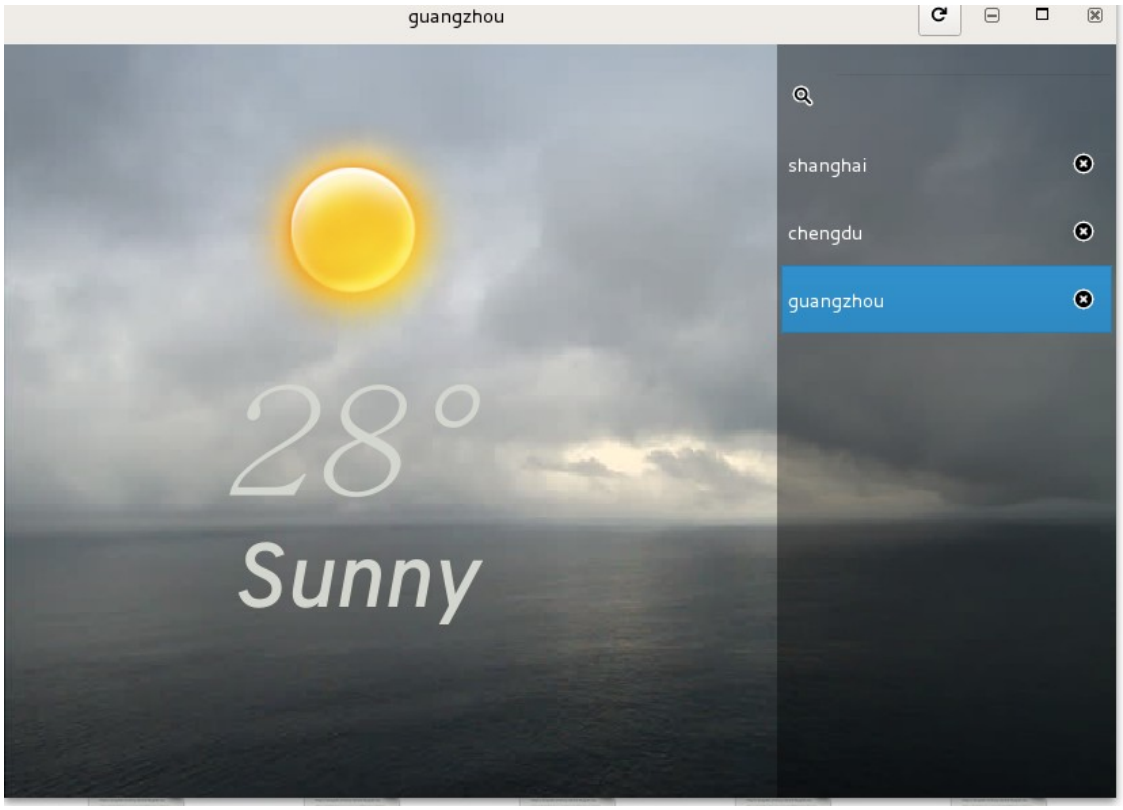


一些信号和槽：

比如搜索栏里面输入的时候，会出现清空的一个按钮，当按下搜索键，清空，并把数据传给 ListWidiget 由它新建一个 ResultItem， 接着新建一个 CentralWIdiget，它又会新建一个 Struct Query, 这是 C 层的最重要的数据结构，再调用下一层的 API，把数据送给 C 层做处理。然后数据回到的时候，发送事件给 CentralWIdget 做数据的展示。

最后是美化，这也是最麻烦的一部分，因为 Qt 的排版太玄学了，处理这一部分花了大部分的事件，而信号和槽的处理以及事件函数的重写并不难。

最后的效果



最后还有一个地方不好处理，就是 C++ 和 C 的编译，本身并无问题，但是 Qt 本身的代码编写的简单性决定它后面有庞大的库和自动生成的代码，之间依赖性非常麻烦。好在它支持 extern “C” 语句，并且是利用自动化生成的 Makefile，而且也不会太复杂，但是很长，不过决定性的代码就那几句，加上去 C 的文件，再增加所需要的库就 OK.

Qt 使用的是其自带的 qmake 工具，是在 gcc 之后增加的自己的东西，我也没有详细了解过。但是底层用肯定还是 gcc， 看编译的输出就知道了。完成和 C 层的混合编译，需要改变一下它自动生成的 Makefile。

```
/* Makefile */
CC            = gcc
CXX           = g++
DEFINES       = -DQT_DEPRECATED_WARNINGS -DQT_QML_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB
-DQT_CORE_LIB -DDEBUG
CFLAGS        = -pipe -g -Wall -W -D_REENTRANT -fPIC $(DEFINES)
CXXFLAGS      = -pipe -g -std=gnu++11  -W -D_REENTRANT -fPIC $(DEFINES)
INCPATH       = -I../myweather/include -I../myweather -I.
-I../../../trance/myaps/Qt5.9.1/5.9.1/gcc_64/include
-I../../../trance/myaps/Qt5.9.1/5.9.1/gcc_64/include/QtWidgets
```

-I../../../trance/myaps/Qt5.9.1/5.9.1/gcc_64/include/QtGui
-I../../../trance/myaps/Qt5.9.1/5.9.1/gcc_64/include/QtCore -I. -isystem /usr/include/libdrm
-I../../../trance/myaps/Qt5.9.1/5.9.1/gcc_64/mkspecs/linux-g++
QMAKE = /home/trance/trance/myaps/Qt5.9.1/5.9.1/gcc_64/bin/qmake
…
LIBS = $(SUBLIBS) -L/home/trance/trance/myaps/Qt5.9.1/5.9.1/gcc_64/lib -lQt5Widgets -lQt5Gui -lQt5Core -lGL **-lpthread -lcurl -lm**

后面还有很长，关键就是增加的粗体，增加 include 的路径，C 层依赖的库。define DEBUG 是为了调试 C 层的代码，这是在写的时候设定的。

系统测试情况

编译的时候先 define DEBUG, 那么函数调用的轨迹会全部打印，就很容易调试了。

程序会在一种情况崩溃，就是当输入的城市名字不合法的时候，因为不管到底查询是否成功，都必须发送事件到上层，所以这里只需要加上事件判断查询成功与否就好了，不成功就弹窗，更改查询的数据，然后重新查询就 ok（懒，还没写）。但是应该在更前预防，就是在搜索框输入的时候，就应该不允许发送错误的查询，（不然就是浪费系统资源）这就要求搜索框必须有全部城市的缓存，还能顺便提供搜索下拉框的提示，这是最好的更改方案，但是到底全部城市的数据怎么存，以及查询的时候应该提示，搜索的时候到底以什么方式查，还没有想好，所以没有实现。

其他情况正常。

系统优点及改进

优点

固定双线程，而不是每次查询抛出一个线程，因为不断的创建销毁是需要占用一定资源的。

Struct query 的设计，使得一层的所有函数，只处理，传递一个数据结构，无论是在维护的的角度还是读的角度，都很不错。还有 C++层的 ResultItem 也是基于此。

改进

上下两层应该提供的是注册函数和注销函数的接口，而不是直接声明之后调用。**

libcurl(http 库)的结构应该使用全局变量，然后加上锁机制，不过这里 libcurl 的文档并没有说清楚它的详细机制，所以不知道该如何修改。（目前是每次发送都采用新建一个传输句柄，开销估计也不小，但是这里还涉及到线程安全的问题，而文档里面说的也不详细)

取消 struct query 的 arg 这一域，这是多余的。

如果以后增加出行，传衣等建议(或者 PM2.5)查询，还得增加 flag 位的设计，那么事件也得增加函数，不过修改起来都不会复杂。关键问题我使用的 API 免费接口是很有限的信息。

参考文献

libcurl	https://curl.haxx.se/libcurl/c/
json_parser	https://github.com/udp/json-parser
json	json.org
《Qt Creator 快速入门》	霍亚飞编著
...	