自动备份系统

项目介绍

该项目完成一个自动对指定目录下的文件进行备份的功能；

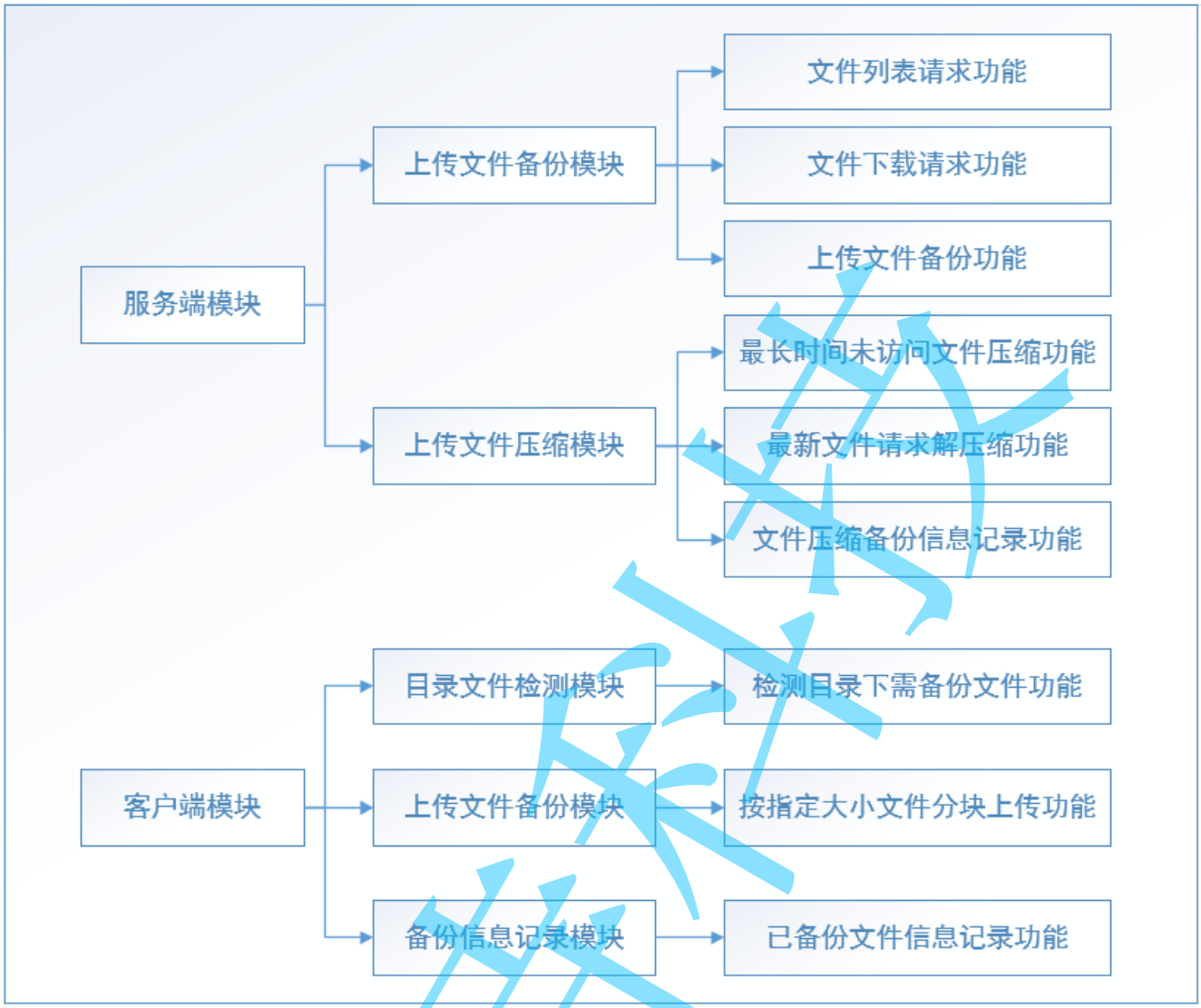
1. 客户端对指定目录进行监控，对每个文件生成etag信息，鉴别是否需要备份
2. 客户端将需要备份的文件基于http协议的PUT请求上传到服务器端
3. 服务端对于PUT上传的文件进行备份到指定目录下
4. 整个通信的过程使用SSL/TLS加密传输

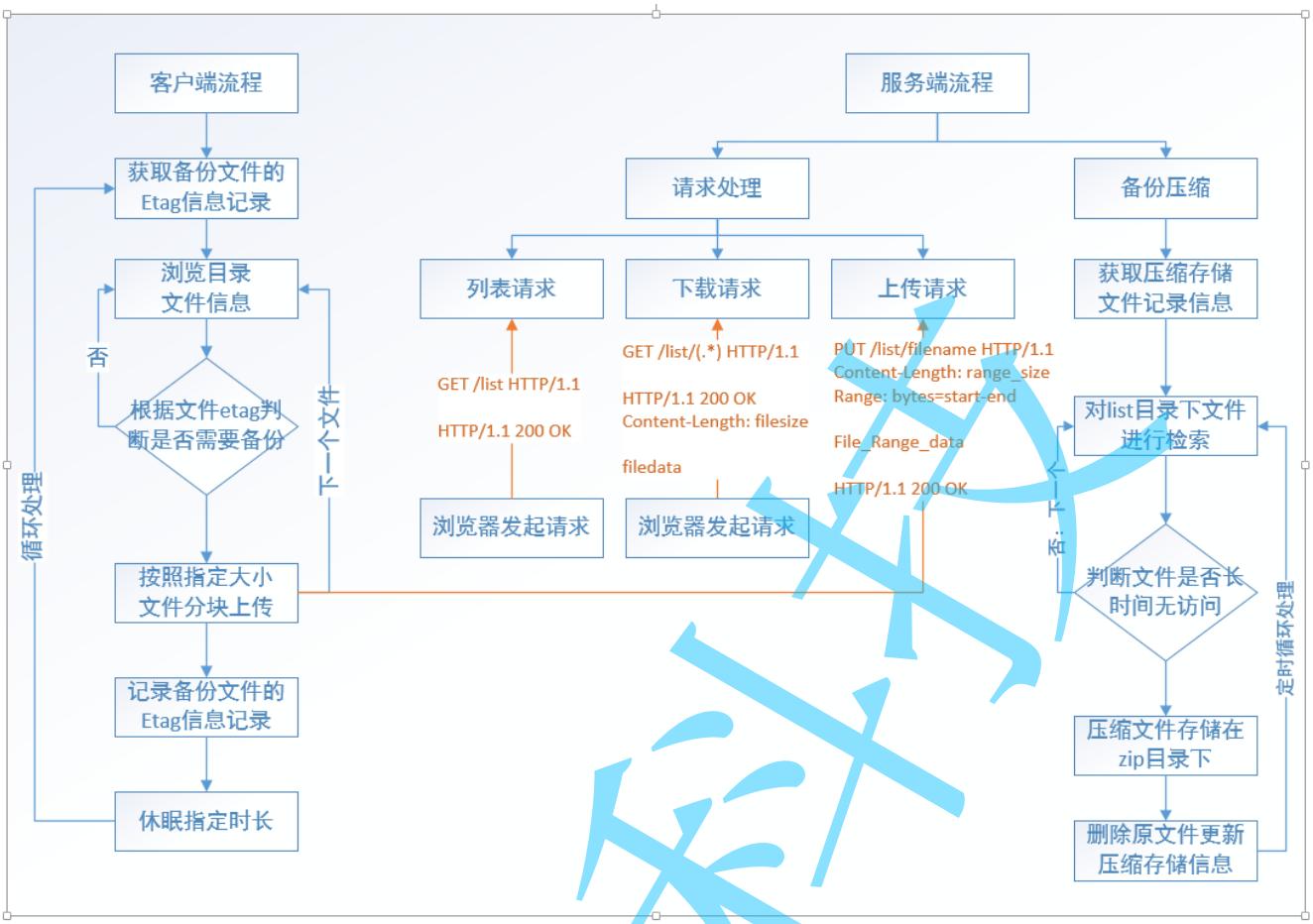
讲解课时：5~6课时

课程讲解流程

2. socket套接字编程（了解最基本线程池版本任务处理的tcp服务端程序）
3. http协议（了解最基本的http服务器中数据的接收处理响应流程，文件传输）
4. cpp-httplib的基本使用（能够搭建基本的http服务器，并创建客户端发起请求得到响应）
5. 项目流程框架介绍
6. 接口类封装介绍
7. 文件压缩存储的基本原理与目的
8. 加密传输（了解基本的加密传输流程原理）
    - 夫妻两人聊天例子
    - 某：我喜欢你；某：我也喜欢你 聊了三年发现对方不是他老婆
    - 某：我喜欢你；某：先向我证明你是我老婆 某：我就是你老婆；某：那我也喜欢你 最后发现对方虽然说是自己老婆，但是其实不是
    - 某：我喜欢你；某：先向我证明你是我老婆 某：我就是你老婆，这是我的身份证；某：那我也喜欢你； 最后因为聊天太过肉麻，双方被网监部门抓捕
    - 聊天过程进行加密 非对称加密/对称加密：在通信时先使用非对称加密算法进行身份认证以及对称加密算法协商；协商完毕后，双方使用协商好的对称加密算法进行数据传输（非对称加密传输效率较低）
    - 非对称加密：
        - 采用RSA加密算法生成一对密钥（包含公钥和私钥）
        - 使用密钥生成证书（包含认证信息及公钥...）（PS：证书有可能被劫持，因此需要权威机构颁发）
        - 通信链接建立成功后将证书发送给对方，证明自己是自己，并且提取出公钥对传输的数据进行加密
        - 对方收到数据后通过私钥进行解密；在传输的数据中包含了对称加密算法协商

整体流程框架


服务端模块
上传文件备份模块
文件列表请求功能
文件下载请求功能
上传文件备份功能
上传文件压缩模块
最长时间未访问文件压缩功能
最新文件请求解压缩功能
文件压缩备份信息记录功能
客户端模块
目录文件检测模块
检测目录下需备份文件功能
上传文件备份模块
按指定大小文件分块上传功能
备份信息记录模块
已备份文件信息记录功能



服务端设计

实现功能：

1. 提供解析基于https协议的put请求，将文件数据进行备份
2. 提供浏览器能够查看服务器上文件信息功能
3. 提供浏览器能够下载服务器上文件功能
4. 提供对后台长时间无访问文件的压缩存储功能

httplib库实现

主要接口功能设计：

```
/*客户端请求处理模块*/
/*下载请求：
    请求：GET /list/ HTTP/1.1              /(list/){0,1}
    响应：HTTP/1.1 200 OK
  列表请求：
    请求：GET /list/filename HTTP/1.1      /list/(.*)
    响应：
        HTTP/1.1 200 OK

        file data
  上传请求：
    请求：
        PUT /list/filename HTTP/1.1    /list/(.*)
```

```
        Range: bytes=range_start-range_end
        Content-Length: range_size

        file range data
    响应: HTTP/1.1 200 OK
*/
class CloudBackupServer
{
    private:
        std::string _pri_key_file;
        std::string _cert_file;
    public:
        CloudServer(const std::string &cert, const std::string &key);
        /*httplib文件下载请求处理功能实现接口*/
        static void FileDownload(const httplib::Request &req, httplib::Response &rsp);
        /*httplib文件列表请求处理功能实现接口*/
        static void GetFileList(const httplib::Request &req, httplib::Response &rsp);
        /*httplib文件上传请求处理功能实现接口*/
        static void PutFileBackup(const httplib::Request &req, httplib::Response &rsp);
};
```

```
/*文件压缩存储处理模块*/
class FileBackup
{
    private:
        std::unordered_map<std::string, std::string> _backup_list;
        pthread_rwlock_t _rwlock;
    public:
        /*服务端每次启动从指定备份信息文件获取压缩存储的文件信息*/
        bool GetRecored();
        /*服务端定时将压缩存储的文件信息记录到文件中*/
        bool SetRecored();
        /*服务端定时对上传的文件进行判断是否时不常访问文件，进行压缩存储，并删除源文件*/
        bool GZipBackupFile();
        /*向外提供获取备份文件列表信息*/
        bool GetFileList(std::vector<std::string> &list);
        /*向外提供读取文件数据功能（直接读取文件/先解压文件数据再读取）*/
        bool ReadFileBody(const std::string &file, std::string &body);
        /*向外提供向指定文件的指定位置写入文件数据*/
        bool WriteFileBody(const std::string &file, const std::string &body, int64_t off)
};
```

客户端设计

实现功能：

1. 提供监控目录的功能，能够获取目录下文件信息，鉴别文件是否需要备份
2. 备份文件，基于HTTPS协议PUT请求，实现文件多线程分块上传功能
3. 文件的信息记录，便于文件是否需要备份的鉴别

实现流程

1. 查看当前目录下是否有backup.list文件，获取文件中的备份文件etag（mtime/size）信息
2. 打开指定目录，浏览目录内的文件信息；获取文件etag信息与记录信息进行比对，鉴别是否需要备份
3. 对需要备份的文件，创建https协议的put请求头信息；并且获取文件数据进行上传
4. 根据响应的状态，判断是否成功备份，更新文件名称对应的etag信息

新文件创建：当文件名在文件信息表中查找不到的情况 旧文件修改：当文件的etag与信息表中对应的的etag不同

文件信息表格式

```
文件名          文件etag  (验证是否修改)
filename1    mtime-filesize
filename2    mtime-filesize
filename3    mtime-filesize
...
```

主要接口功能设计：

```
/*云备份客户端信息*/
namespace bf = boost::filesystem;
class CloudBackupClient
{
    public:
        Start();
    private:
        CloudClient(const std::string &path, int wait_time = 3);
        /*文件上传备份*/
        bool FileBackup(const bf::path &file);
        /*目录下文件检测*/
        bool BackupDirListen(const bf::path &dir);
    private:
        std::string _listen_dir;
        int _wait_time;
        BackupRecored _recored;
};
```

```
/*分块上传文件信息模块*/
class RangeBackup
{
    private:
        bf::path
    public:
        bool _backup_statu;//判断分块是否上传成功
    public:
        RangeBackup();
        /*设置分块信息*/
        void SetRange(const bf::path &file, int64_t start, int64_t end);
        /*进行文件上传*/
        void Backup();
};
```

```
class BackupRecored
{
    private:
        std::unordered_map<std::string, std::string> _backup_list;
    public:
        /*插入文件备份信息*/
        bool InsertRecored(std::string &file, std::string &etag);
        /*判断文件名对应etag是否是最新*/
        bool NewestRecored(std::string &file, std::string &etag);
        /*记录文件备份信息到指定文件中*/
        bool SetRecored();
        /*从指定文件中获取文件备份信息*/
        bool GetRecored();
};
```

其它接口使用

httplib基本使用:

```
/*test.cpp*/
#include "httplib.h"

static void HelloWorld(const httplib::Request &req, httplib::Response &rsp) {
    rsp.set_content("<html>hello world</html>", "text/html");
    return;
}

int main() {
    httplib::Server srv;
    srv.set_base_dir("./www");
    srv.Get("/", HelloWorld);
    srv.listen("0.0.0.0", 9000);
}
/*g++ -std=c++0x test.cpp -o test -lpthread -lboost_filesystem -lboost_system*/
```

zlib基本使用:

```
/*test.cpp*/
#include <iostream>
#include <fstream>
#include <string>
#include <zlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <boost/algorithm/string.hpp>
#include <boost/filesystem.hpp>

bool Compress(const std::string &file)
{
    std::string zip = file + ".gz";
```

```
    int f_fd = open(file.c_str(), O_RDONLY);
    if (f_fd < 0) {
        return false;
    }
    gzFile z_fp = gzopen(zip.c_str(), "wb");

    char buf[1024] = {0};
    int ret = 0;
    while((ret = read(f_fd, buf, 1024)) > 0) {
        gzwrite(z_fp, buf, ret);
    }
    close(f_fd);
    gzclose(z_fp);
    return true;
}
bool UnCompress(const std::string &file)
{
    boost::filesystem::path path(file);
    std::string name = path.replace_extension("").string();

    gzFile zf = gzopen(file.c_str(), "rb");
    int fd = open(name.c_str(), O_CREAT|O_WRONLY|O_TRUNC, 0664);

    char buf[1024] = {0};
    int ret = 0;
    while((ret = gzread(zf, buf, 1024)) > 0) {
        write(fd, buf, ret);
    }
    close(fd);
    gzclose(zf);
    return true;
}
int main()
{
    std::string file = "./test.text";
    std::string zip = "./test.text.zip";
    //Compress(file);
    UnCompress(zip);
    return 0;
}
/*g++ test.cpp -lboost_filesystem -lboost_system -lz*/
```

vs2017 openssl及boost库使用

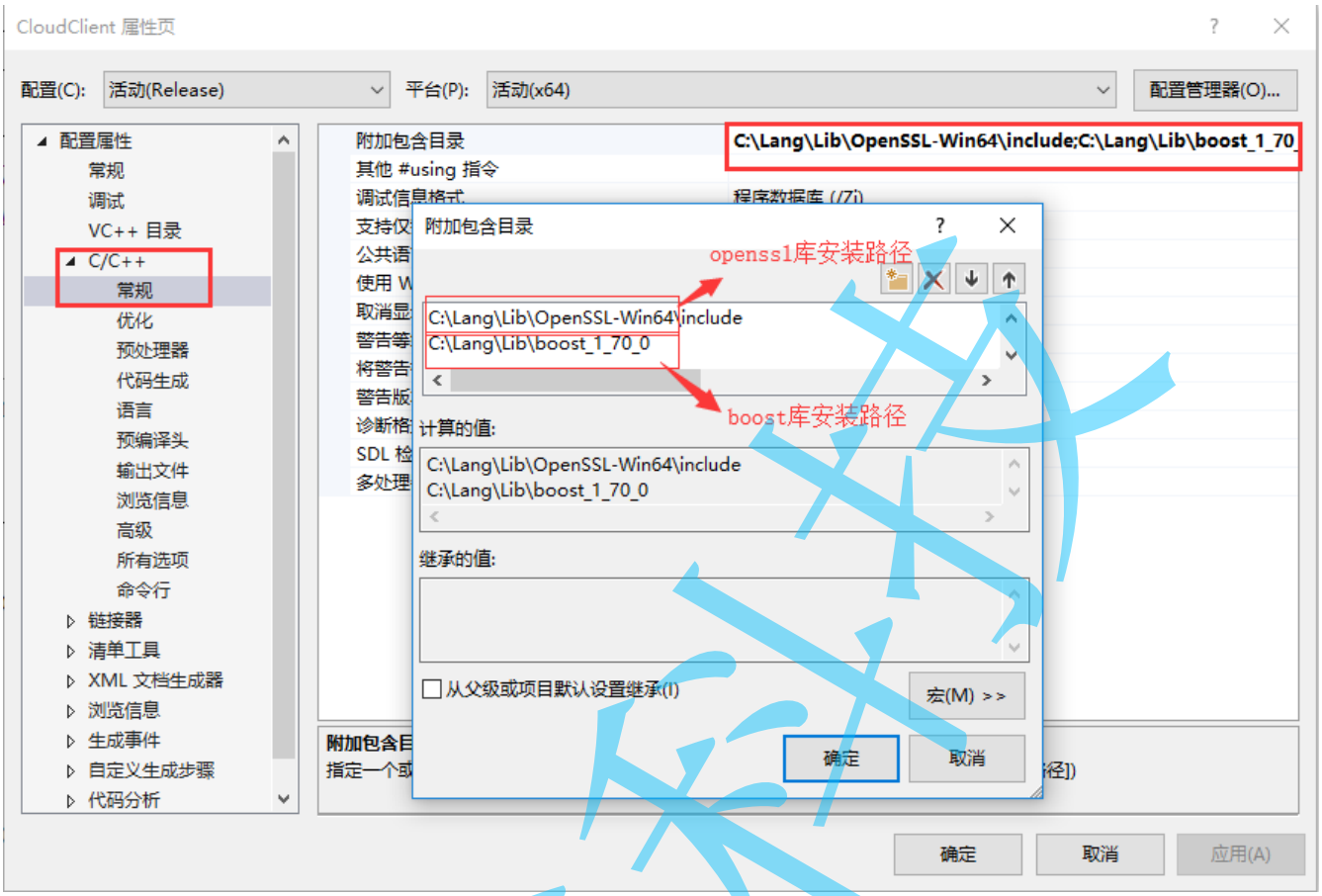
boost库下载路径：boost_1_70_0-msvc-14.1-64.exe

openssl库下载路径：Win64OpenSSL-1_1_1c.exe

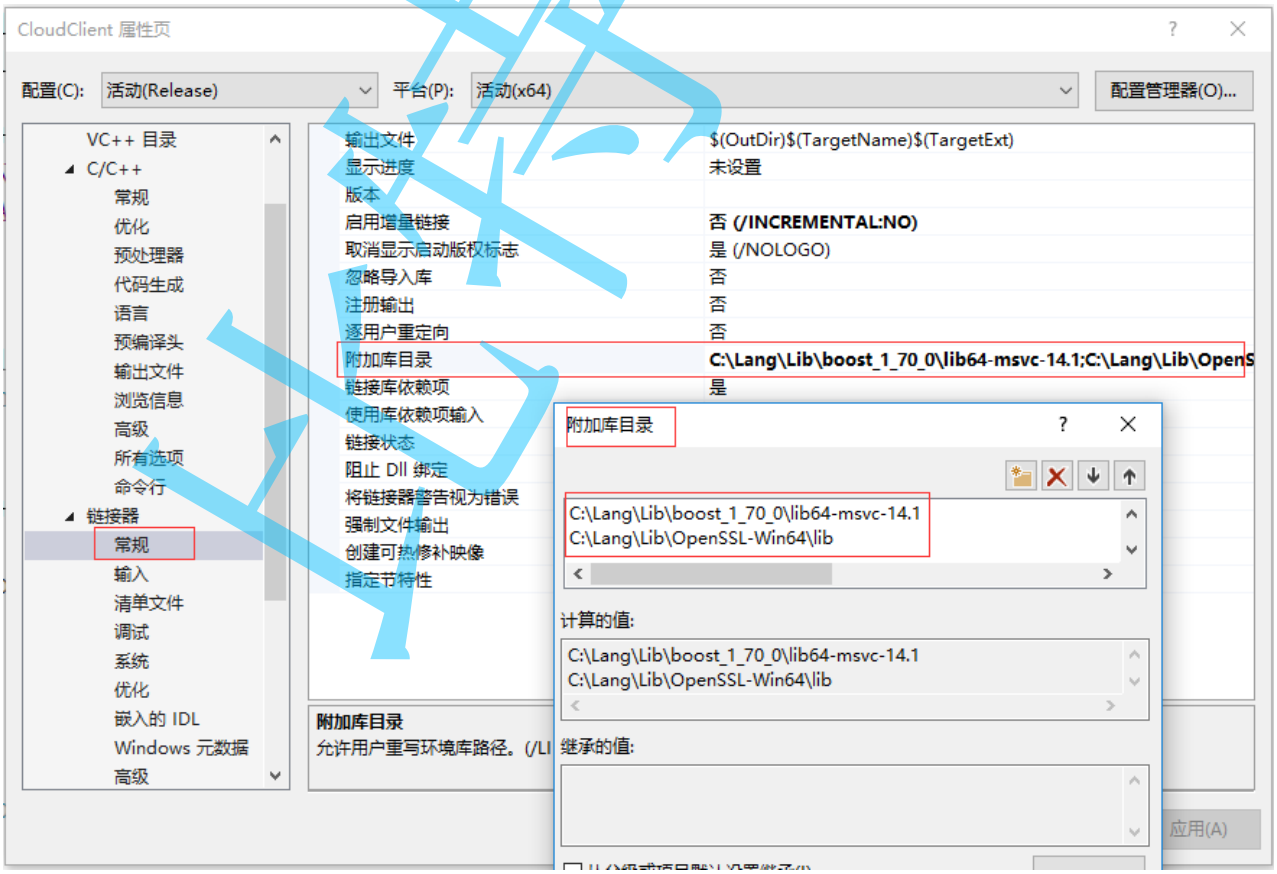
下载之后选定安装目录->傻瓜式安装即可

vs2017项目属性设置：

1. 头文件包含目录设置



2. 库文件包含目录设置



3. 编译选项设置