0. 单例模式

1. 单例模式实现方式?
    - 懒汉模式 - 单例对象在使用的时候被创建出来, 线程安全问题需要考虑

```
class Test
{
public:
    static Test* getInstance();
private:
    Test();
    Test(const Test& t);
    // 静态变量使用之前必须初始化
    static Test* m_test;
}
Test* Test::m_test = NULL;  // 初始化
// 第一种实现方式
Test::Test* getInstance()
{
    if(m_test == NULL)
    {
        m_test = new Test();
    }
    return m_test;
}
// 弊端: 有线程安全问题, 会创建多个对象, 每个线程创建一个
// 解决方案: 线程同步, 加锁

// 第2种实现方式
// 假设在c++中有一个mutex对象, lock, unlock
Test::Test* getInstance()
{
    mutex.lock();   // 加锁
    if(m_test == NULL)
    {
        m_test = new Test();
    }
    mutex.unlock(); // 解锁
    return m_test;
}
// 弊端: 效率很低, 每个线程得到单例对象是, 线性执行的
// 第3种实现方式
// 假设在c++中有一个mutex对象, lock, unlock
Test::Test* getInstance()
{
    if(m_test == NULL)
    {
        mutex.lock();   // 加锁
        if(m_test == NULL)
```

```
            {
                m_test = new Test();
            }
            mutex.unlock(); // 解锁
        }
        return m_test;
    }
    // 弊端：第一次获取单例对象的时候，线程是线性执行的，第二次以后是并行的
    // 第四种 ：要求编译器支持c++11
    class Test
    {
    public:
        static Test* getInstance();
    private:
        Test();
        Test(const Test& t);
    }
    Test::Test* getInstance()
    {
        static Test test;
        return &test;
    }
```

- 饿汉模式 - 单例对象在使用之前被创建出来

```
    class Test
    {
    public:
        static Test* getInstance()
        {
            return &m_test;
        }
    private:
        Test();
        Test(const Test& t);
        // 静态变量使用之前必须初始化
        // static Test* m_test;
         static Test m_test;
    }
    // Test* Test::m_test = new Test(); // 初始化
    Test Test::m_test;
```

2. 如何在单例类中存储数据?

```
// 实现了一个单例模式的类
// 存储用户名/密码/服务器的iP/端口
class Test
{
public:
    static Test* getInstance()
    {
        return &m_test;
    }
    // 设置数据
```

```
    void setUserName(QString name)
    {
        // 多线程-> 加锁
        m_user = name;
        // 解锁
    }
    // 获取数据
    QString getUserName()
    {
        return m_user;
    }
private:
    Test();
    Test(const Test& t);
    // 静态变量使用之前必须初始化
    // static Test* m_test;
     static Test m_test;
    // 定义变量 -> 属于唯一的单例对象
    QString m_user;
    QString m_passwd;
    QString m_ip;
    QString m_port;
    QString m_token;
}
// Test* Test::m_test = new Test(); // 初始化
Test Test::m_test;
```

在客户端登录的时候, 服务器回复给客户端的数据

```
// 成功
{
    "code": "000",
    "token": "xxx"
}
// 失败
{
    "code": "001",
    "token": "faild"
}
token -> 客户端成功连接了服务器, 服务器针对于客户端的个人信息生成了一个唯一的身份标识
    - 可以按照每个人的身份证号理解
    - 服务器将这个token发送给客户端
    - 客户端token的使用和保存:
        - 使用: 登录成功之后, 向服务器在发送任意请求都需要携带该token值
        - 保存方式: 放到单例对象中
    - 服务器端的使用和保存:
        - 使用: 接收客户端发送的token, 和服务器端保存的token进行认证
            - 认证成功: 合法客户端, 失败: 客户端非法
        - 保存: 服务器需要保存所有客户端的token
                - 数据库中
                - 配置文件 -> 效率低
                - redis中 -> 效率最高
(客户端信息+随机数)*des*md5*base64
```

1. QSS样式表

1.1 选择器类型

选择器	示例	说明
通用选择器	*	匹配所有部件 匹配当前窗口所有的子窗口
类型选择器	QWidget	匹配QWidget及其子类窗口的实例
类选择器	.QPushButton	匹配QPushButton的实例，但不包含子类。相当于*[class~=”QPushButton”]。
ID选择器	**QPushButton#okButton**	匹配所有objectName为okButton的QPushButton实例。
后代选择器	QDialog QPushButton	匹配属于QDialog后代（孩子，孙子等）的QPushButton所有实例。
子选择器	QDialog >QPushButton	匹配属于QDialog直接子类的QPushButton所有实例。

1.2 QSS的使用步骤

```
// QSS是一个文件, 样式表文件(CSS文件)
//  - Qt样式表支持css2.0, 1.0 所有的语法, css3.0部分样式在qt中不支持
// 如何使用
/*
    1. 根据介绍的选择器对所有的控件样式设置, 写入qss文件中
    2. 在程序中读样式表文件, 得到一个字符串 -> 样式字符串
    3. 将读出的样式设置给QT的应用程序对象
    4. 在qt中有一个全局的应用程序指针qApp
    5. qApp->setStyleSheet("样式字符串");
    6. QFile读磁盘文件, 磁盘文件的编码格式必须是utf8
*/
```

1.3 登录窗口设置

```
/* 登录窗口设置背景图片 */
/* 登录窗口所有控件设置字体, 字体大小 */
/* 设置登录/注册/服务器设置窗口标题字体, 字体大小 */
/* 设置logo显示的图片 */
/* 设置窗口标题字体, 字体大小, 加粗 */
/* 没有账号马上注册按钮: 字体颜色和添加下划线 */
/* 登录/注册/OK按钮: 字体颜色, 宽度, 高度, 字体大小, 显示图片 */
/* 标题栏按钮: normal, hover, press三种状态切换 */
```

```
/* 按钮的默认状态 */
QPushButton#loginBtn
{
    color:  white;
    width: 200;
    height: 50;
    font-size: 30px;
    border-image: url(:/images/balckButton.png); /* 默认显示的图片 */
}
/* 按钮的悬停状态 */
QPushButton#loginBtn:hover
{
    border-image: url(:/images/balckButton1.png); /* 默认显示的图片 */
}
/* 按钮的按下状态 不是css中的标准状态, qt独有的 */
QPushButton#loginBtn:pressed
{
    border-image: url(:/images/balckButton2.png); /* 默认显示的图片 */
}
```

2. 客户端post方式上传数据

2.1 常用的四种方式

- application/x-www-form-urlencoded

```
# 请求行
POST http://www.example.com HTTP/1.1
# 请求头
Content-Type: application/x-www-form-urlencoded;charset=utf-8
# 空行
# 请求数据(向服务器提交的数据)
title=test&user=kevin&passwd=32222
```

- application/json

```
POST http://www.example.com HTTP/1.1
Content-Type: application/json;charset=utf-8
{"title":"test","sub":[1,2,3]}
```

- text/xml

```
POST http://www.example.com HTTP/1.1
Content-Type: text/xml
<?xml version="1.0" encoding="utf8"?>
<methodcall>
    <methodname color="red">examples.getStateName</methodname>
    <params>
        <value><i4>41</i4></value>
    </params>
</methodcall>
```

- multipart/form-data

  tool.oschina.net

```
POST http://www.example.com HTTP/1.1
Content-Type: multipart/form-data
# 发送的数据
------WebKitFormBoundaryPpL3BfPQ4cHShsBz \r\n
Content-Disposition: form-data; name="file"; filename="qw.png"; md5="xxxxxxxxxx"
Content-Type: image/png\r\n;
\r\n
.............文件内容.................
.............文件内容.................
------WebKitFormBoundaryPpL3BfPQ4cHShsBz
Content-Disposition: form-data; name="file"; filename="qw.png"; md5="xxxxxxxxxx"
Content-Type: image/png\r\n;
\r\n
.............文件内容.................
.............文件内容.................
------WebKitFormBoundaryPpL3BfPQ4cHShsBz--
```

3. 上传协议

> 文件上传的一般步骤:
>
> - 尝试秒传 -> 文件并没上传
>   - 给服务器发送的不是文件内容, 是文件的哈希值
>   - 在服务器端收到哈希值, 查询数据库
>   - 查到了 -> 秒传成功
>   - 没查到 -> 秒传失败, 需要进行一个真正的上传操作
> - 进行真正的上传
>   - 需要的时间长
>   - 上传有文件内容, 文件的哈希值
>   - 文件内容 -> 分布式文件系统
>   - 哈希值 -> 数据库

1. 秒传
   - 客户端

```
# url
http://127.0.0.1:80/md5
# post数据格式
{
    user:xxxx,
    token:xxxx,
    md5:xxx,
    fileName: xxx
}
```

- 服务器

```
location /md5
{
    # 转发数据
    fastcgi_pass localhost:10002;
    include fastcgi.conf;
}
```

文件已存在(秒传成功):	{"code":"005"}
秒传成功:	{"code":"006"}
秒传失败:	{"code":"007"}

fastCGI程序编写

```
int main()
{
    while(FCGI_Accept() >= 0)
    {
        // 1. 得到post数据的长度
        char* length = getenv("content-length");
        int len = atoi(length);
        // 2. 根据len将数据读到内存中, json对象字符串
        // 3. 解析json对象, user,md5, token, fileName
        // 4. token认证 , 查询redis/数据库
        //     -- 成功: 继续后续操作, 失败, 返回, 给客户端一个结果
        // 5. 打开数据库, 并查询md5是否存在
        //     -- 存在   {"code":"006"}
        //     -- 不存在  {"code":"007"}
    }
}
```

2. 上传
    - 客户端

```
# url
http://127.0.0.1:80/upload
# post数据格式
------WebKitFormBoundary88asdgewtgewx\r\n
Content-Disposition: form-data; user="mike"; filename="xxx.jpg"; md5="xxxx";
size=10240
Content-Type: image/jpg
真正的文件内容
------WebKitFormBoundary88asdgewtgewx--
```

- Qt中如何组织上述post数据块

```
// 组织数据块 - > QHttpPart
QHttpPart::QHttpPart();
// 设置数据头
void QHttpPart::setHeader(QNetworkRequest::KnownHeaders header, const QVariant
&value);
    - header:
        - QNetworkRequest::ContentDispositionHeader
        - QNetworkRequest::ContentTypeHeader
    - value:
        "form-data; 自定义的数据, 格式 xxx=xxx, 中间以;间隔"
// 适合传递少量的数据
void QHttpPart::setBody(const QByteArray &body);
    - body: 传递的数据串
// 传递大文件
void QHttpPart::setBodyDevice(QIODevice *device);
    - 使用参数device, 打开一个磁盘文件

// QHttpMulitPart
QHttpMultiPart::QHttpMultiPart(ContentType contentType, QObject *parent =
Q_NULLPTR);
    - 参数contentType: QHttpMultiPart::FormDataType
// 调用该函数会自动添加分界线  -> 使用频率高的函数
void QHttpMultiPart::append(const QHttpPart &httpPart);
// 查看添加的分界线的值
QByteArray QHttpMultiPart::boundary() const;
// 自己设置分界线, 一般不需要自己设置
void QHttpMultiPart::setBoundary(const QByteArray &boundary);

// 使用post方式发送数据
QNetworkReply *QNetworkAccessManager::post(const QNetworkRequest &request,
QHttpMultiPart *multiPart);
```

- 服务器

```
location /upload
{
    # 转发数据
    fastcgi_pass localhost:10003;
    include fastcgi.conf;
}
```

```
// fastCGI程序
int main()
{
    // 1. 获取数据长度 1024000
    // 2. 循环读post数据内容
}
```

成功	{"code":"008"}
失败	{"code":"009"}

- 服务器端fastCGI 部分 代码

```
// 取出 Content-Disposition 中的键值对的值，并得到文件内容，并将内容写入文件
int recv_save_file(char *user, char *filename, char *md5, long *p_size)
{
    int ret = 0;
    char *file_buf = NULL;
    char *begin = NULL;
    char *p, *q, *k;

    char content_text[512] = {0}; //文件头部信息
    char boundary[512] = {0};     //分界线信息

    //==========> 开辟存放文件的 内存 <===========
    file_buf = (char *)malloc(4096);
    if (file_buf == NULL)
    {
        return -1;
    }

    //从标准输入(web服务器)读取内容
    int len = fread(file_buf, 1, 4096, stdin);
    if(len == 0)
    {
        ret = -1;
        free(file_buf);
        return ret;
    }

    //===========> 开始处理前端发送过来的post数据格式 <============
    begin = file_buf;    //内存起点
    p = begin;

    /*
       ------WebKitFormBoundary88asdgewtgewx\r\n
       Content-Disposition: form-data; user="mike"; filename="xxx.jpg";
md5="xxxx"; size=10240\r\n
       Content-Type: application/octet-stream\r\n
       ------WebKitFormBoundary88asdgewtgewx--
    */
```

```
38
39      //get boundary 得到分界线, ------WebKitFormBoundary88asdgewtgewx
40      p = strstr(begin, "\r\n");
41      if (p == NULL)
42      {
43          ret = -1;
44          free(file_buf);
45          return ret;
46      }
47
48      //拷贝分界线
49      strncpy(boundary, begin, p-begin);
50      boundary[p-begin] = '\0';   //字符串结束符
51      p += 2; //\r\n
52      //已经处理了p-begin的长度
53      len -= (p-begin);
54      //get content text head
55      begin = p;
56
57      //Content-Disposition: form-data; user="mike"; filename="xxx.jpg";
    md5="xxxx"; size=10240\r\n
58      p = strstr(begin, "\r\n");
59      if(p == NULL)
60      {
61          ret = -1;
62          free(file_buf);
63          return ret;
64      }
65      strncpy(content_text, begin, p-begin);
66      content_text[p-begin] = '\0';
67
68      p += 2;//\r\n
69      len -= (p-begin);
70
71      //=========================================获取文件上传者
72      //Content-Disposition: form-data; user="mike"; filename="xxx.jpg";
    md5="xxxx"; size=10240\r\n
73      q = begin;
74      q = strstr(begin, "user=");
75      q += strlen("user=");
76      q++;    //跳过第一个"
77      k = strchr(q, '"');
78      strncpy(user, q, k-q);  //拷贝用户名
79      user[k-q] = '\0';
80
81      //=========================================获取文件名字
82      //"; filename="xxx.jpg"; md5="xxxx"; size=10240\r\n
83      begin = k;
84      q = begin;
85      q = strstr(begin, "filename=");
86      q += strlen("filename=");
87      q++;    //跳过第一个"
88      k = strchr(q, '"');
```

```
89        strncpy(filename, q, k-q);  //拷贝文件名
90        filename[k-q] = '\0';
91
92        //=======================================获取文件MD5码
93        //"; md5="xxxx"; size=10240\r\n
94        begin = k;
95        q = begin;
96        q = strstr(begin, "md5=");
97        q += strlen("md5=");
98        q++;    //跳过第一个"
99        k = strchr(q, '"');
100       strncpy(md5, q, k-q);   //拷贝文件名
101       md5[k-q] = '\0';
102
103       //=======================================获取文件大小
104       //"; size=10240\r\n
105       begin = k;
106       q = begin;
107       q = strstr(begin, "size=");
108       q += strlen("size=");
109       k = strstr(q, "\r\n");
110       char tmp[256] = {0};
111       strncpy(tmp, q, k-q);   //内容
112       tmp[k-q] = '\0';
113       *p_size = strtol(tmp, NULL, 10); //字符串转long
114
115       begin = p;
116       p = strstr(begin, "\r\n");
117       p += 2; //\r\n
118       len -= (p-begin);
119
120       //下面才是文件的真正内容
121       /*
122          ------WebKitFormBoundary88asdgewtgewx\r\n
123          Content-Disposition: form-data; user="mike"; filename="xxx.jpg";
    md5="xxxx"; size=10240\r\n
124          Content-Type: application/octet-stream\r\n
125          真正的文件内容\r\n
126          ------WebKitFormBoundary88asdgewtgewx--
127       */
128       // begin指向正文首地址
129       begin = p;
130
131       // 将文件内容抠出来
132       // 文件内容写如本地磁盘文件
133
134       free(file_buf);
135       return ret;
136   }
```

- 使用fastCGI管理器启动fastCGI程序

```
spawn-fcgi -a IP地址 -p 端口 -f ./fastcgi程序
    - 提示启动失败
        - ldd fastCGI程序
```

4. Http上传下载进度

```
// QNetworkReply - 信号
void QNetworkReply::downloadProgress(qint64 bytesReceived, qint64 bytesTotal)
    - bytesReceived: 已经接收的字节数
    - bytesTotal: 要接收的总字节数
void QNetworkReply::uploadProgress(qint64 bytesSent, qint64 bytesTotal)
    - bytesSent: 已经发送的字节数
    - bytesTotal: 要发送的总字节数
```

5. 上传大文件Nginx设置

1. 413 错误

   > 服务器提示：413 Request Entity Too Large 的解决方法
   >
   > - 原因: 上传文件太大, 请求实体太长了
   > - 解决方案:
   >   - 在配置文件nginx.conf中添加: **client_max_body_size 10M**
   >   - 10M: 用户指定的大小

2. 设置的位置:

   在http{ }中设置：client_max_body_size 20m;

   - 所有的server中的所有的location都起作用
   - 在server{ }中设置：client_max_body_size 20m;
     - 对当前server的所有的location生效
   - 在location{ }中设置：client_max_body_size 20m;
     - 只对当前location生效

3. 三者的区别是：
   - http{} 中控制着所有nginx收到的 http 请求。
   - 报文大小限制设置在server {} 中，控制该server收到的请求报文大小
   - 如果配置在location中，则报文大小限制，只对匹配了location 路由规则的请求生效。

6. Qt中的哈希运算

1. 哈希算法 - QCryptographicHash

```
// 构造哈希对象
```

```
QCryptographicHash(Algorithm method);
// 添加数据
// c格式添加数据
void QCryptographicHash::addData(const char *data, int length);
// qt中的常用方法
void QCryptographicHash::addData(const QByteArray &data);
// 适合操作大文件
bool QCryptographicHash::addData(QIODevice *device); // QFile
    - 使用device打开一文件, 在addData进行文件的读操作
// 计算结果
QByteArray QCryptographicHash::result() const;
// 一般适合,哈希值都是使用16进制格式的数字串来表示
QByteArray QByteArray::toHex() const;

[static] QByteArray QCryptographicHash::hash(const QByteArray &data, Algorithm
method);
    - 参数data: 要运算的数据
    - 参数method: 使用的哈希算法
    - 返回值: 得到的哈希值
```