Jstchat6

2025年6月20日 15:39

------ Ep15 加密 + 获取验证码按钮限时 + 输入框错误提示 + 隐藏和显示密码 + 注册成功页面跳转 + 页面跳转简化 -------

》》》》Grpc 在之前是我们用来获取验证码的服务，在这里UP又说 grpc 可以用于获取聊天服务的 IP 和 token，那么 Grpc 究竟有什么作用？（
》》》》或者 Grpc 在聊天服务中又能起到什么作用?
gRPC 是一个高性能的开源远程过程调用（RPC）框架，它由 Google 开发，主要用于在不同服务之间进行高效的通信。
因为其高效、低延迟、跨语言的特性，gRPC 可以被用于开发多种服务：

微服务架构中的服务间通信	在微服务架构中，各个微服务通常需要进行高效的通信。gRPC 非常适合这种场景，因其提供了低延迟、高性能的远程过程调用（RPC）机制。 示例服务：用户身份验证服务、订单处理服务、支付服务、日志服务、数据存储服务等。
实时通信系统	gRPC 支持双向流（Bi-directional streaming），非常适合需要低延迟和实时数据流的应用程序。 它可以用于构建实时通信服务，如视频通话、即时消息、在线游戏等。 示例服务：即时消息系统、视频会议系统、在线客服聊天机器人、游戏服务器等。
流媒体服务	gRPC 的流式数据传输功能特别适合构建流媒体服务。双向流和服务端流（server-side streaming）使得它可以实现高效的媒体传输和实时更新。 示例服务：视频流媒体平台（如 Netflix、YouTube）、音频流平台（如 Spotify）、直播平台等。
数据处理与分析服务	对于需要处理大量数据、进行高效计算和分析的服务，gRPC 提供了高效的跨服务数据传输机制，可以支持分布式数据处理框架、实时数据流处理等。 示例服务：大数据处理服务、实时数据分析服务、机器学习模型训练和推理服务等。
API 网关与服务代理	gRPC 可以用于开发 API 网关或服务代理层，用于协调和路由来自客户端的请求，帮助多个微服务聚合和负载均衡。 示例服务：API 网关服务，代理客户端请求，调用后端微服务，提供统一的接口和负载均衡。
后台管理服务	许多后台管理系统和管理员工具需要高效的 API 来处理大量的请求和数据。 gRPC 提供的高效数据交换机制使得这些服务能够快速响应和处理来自前端的请求。 示例服务：企业资源管理（ERP）系统、客户关系管理（CRM）系统、后台数据分析平台等。

》》》》什么是 MD5 协议？有什么作用?
MD5（Message Digest Algorithm 5）是一种广泛使用的哈希函数（哈希算法），用于生成固定长度的128位（16字节）散列值（通常以32个十六进制数字表示）。
它通常用于数据完整性校验和数字签名等应用，但它并不是一种加密协议。

特点：

哈希函数：	MD5 不是加密算法，它是哈希算法。哈希函数的目的是将任意长度的输入数据（如文件、消息）映射为固定长度的输出值（即散列值）。这个过程是单向的，也就是说，不能从散列值恢复出原始数据。
生成128位散列值：	MD5 输入的数据经过处理后生成一个128位的哈希值，通常以32个字符的十六进制数字表示。例如，输入的字符串 "hello" 经过 MD5 哈希处理后，输出为 "5d41402abc4b2a76b9719d911017c592"。
数据完整性：	MD5 常用于验证数据的完整性。通过计算文件或消息的 MD5 哈希值，并与预先计算的值进行比较，可以检查数据是否在传输或存储过程中发生了改变。
碰撞问题：	虽然 MD5 曾广泛使用，但它已经被证明存在严重的碰撞漏洞。也就是说，两个不同的输入数据可能产生相同的 MD5 散列值。这使得 MD5 在一些安全要求较高的场景中不再被推荐使用。

使用场景：

文件完整性校验：	通过比较文件的 MD5 哈希值来确认文件是否被篡改。
数字签名和证书：	在某些情况下，MD5 曾被用于数字签名和加密协议中（但现在通常被更安全的哈希函数如 SHA-256 取代）。
存储密码（已不推荐）：	过去，MD5 也被用来存储用户密码的哈希值，但由于其安全性问题（如容易遭受暴力破解和碰撞攻击），现在不再推荐用于密码存储。

可选择替换为： SHA-256 或者 SHA-3

》》》》示例（这里我选择使用 QT 封装好的 MD5 哈希函数进行加密操作）

```
10
11  QString MD5Encrypt(const QString &input)
12  {
13      QByteArray byteArray = input.toUtf8();      // 将输入字符串转换为字节数组
14      // 使用 MD5 进行加密
15      QByteArray hash = QCryptographicHash::hash(byteArray, QCryptographicHash::Md5);
16      // 返回十六进制的字符串类型加密结果
17      return QString(hash.toHex());
18  }
19
```

我们将会将输入框中输入的密码：hello 转换为十六进制下的：5d41402abc4b2a76b9719d911017c592，并将其存入数据库中。
如此一来，如果有人盗取了数据库的数据，也没办法破译用户的关键信息。（其实对于能够盗取数据库信息的人，MD5 应该是防不住他的 哈哈）

使用流程：
当我们需要核实用户填入的密码是否正确时，我们将用户在登录时填入的密码进行同样的哈希转化，然后与数据库中存放的内容进行对比，以此来判断用户输入的密码是否正确。

---------- 分割线----------

》》》》在 TimerBtn 构造函数的定义中，为什么需要显示调用其父类的构造函数 QPushButton()？子类不是会默认调用父类的构造函数吗？

首先需要明确的是，在调用子类构造函数时，**子类构造函数的确会默认调用其父类构造函数，但调用的是默认构造函数（无参构造函数）**，所以如果父类（基类）没有构造函数时，就必须在子类的构造函数中显式的调用父类含参的构造函数。
比如这里，QPushButton 的构造函数必须要填入一个参数，用于初始化。由于 QPushButton 并没有默认构造函数，因此在子类 TimerBtn 中，我们就需要显式的调用一下 QPushButton（父类）的构造函数。

```
4
5   TimerBtn::TimerBtn(QWidget *parent):QPushButton(parent),_counter(10)
6   {
7       _timer = new QTimer(this);
8
9       connect(_timer, &QTimer::timeout, [this](){
10          _counter--;
11          if(_counter <= 0){
12              _timer->stop();
13              _counter = 10;
14              this->setText("获取");
15              this->setEnabled(true);
16              return;
17          }
```

》》》》QTime 类型的变量 m_Time 有什么作用？怎么使用？
m_timer 是一个 QTimer 对象，用来定时触发倒计时操作。它会定期发出 timeout() 信号，告诉应用程序每隔 1 秒执行一次操作。

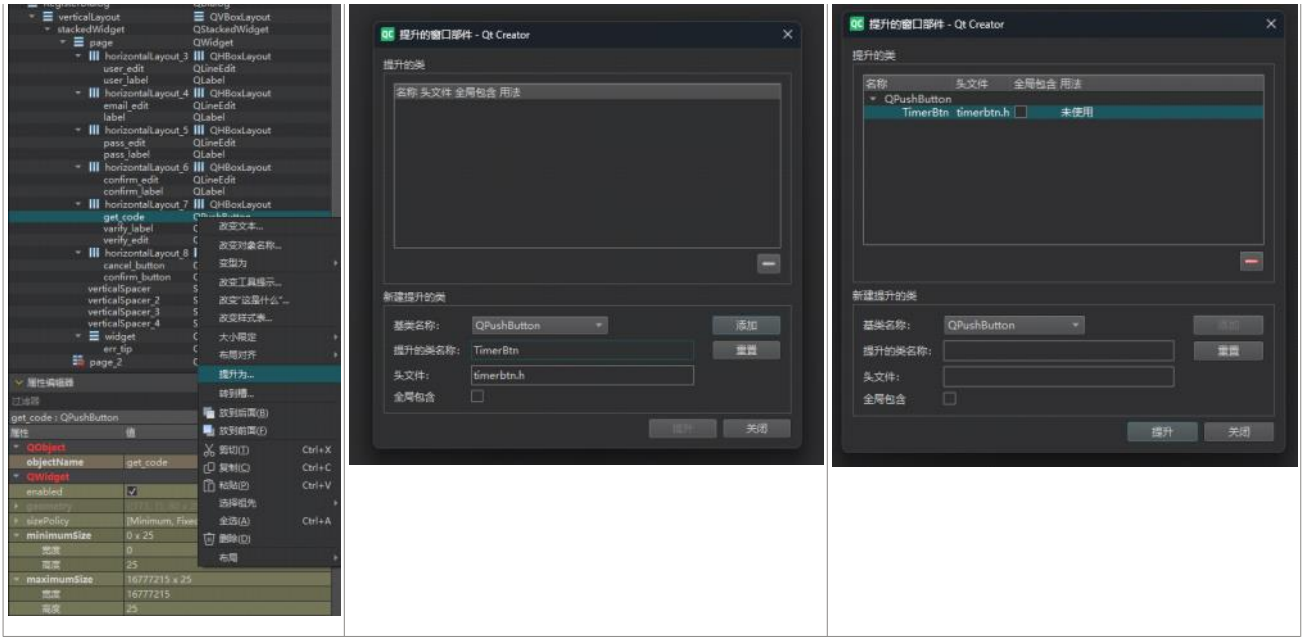
- 在构造函数中，_timer = new QTimer(this) 被设置为一个定时器。
- connect(_timer, &QTimer::timeout, [this](){ ... }) 连接了定时器的 timeout() 信号和一个槽函数。每当定时器超时（即每秒钟触发一次时），这个槽函数会执行，其中会减少 _counter 的值，并更新按钮的文本，直到 _counter 为 0 时停止定时器。
- 在 mouseReleaseEvent 中，调用 _timer->start(1000) 启动定时器，设置定时器每 1000 毫秒（即每秒）触发一次 timeout 信号，开始倒计时。

》》》》QPushButton 类中的成员函数： mouseReleaseEvent() 函数的功能是？
这是一个事件函数，会在按钮检测到“鼠标释放”时触发。（mouseReleaseEvent() 是 Qwidget 的成员函数（虚函数））

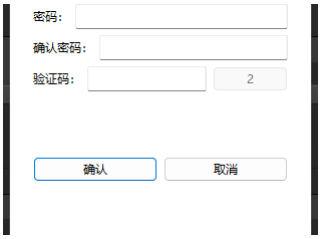
```
618 protected:
619     // Event handlers
620     bool event(QEvent *event) override;
621     virtual void mousePressEvent(QMouseEvent *event);
622     virtual void mouseReleaseEvent(QMouseEvent *event);
623     virtual void mouseDoubleClickEvent(QMouseEvent *event);
624     virtual void mouseMoveEvent(QMouseEvent *event);
625 #if QT_CONFIG(wheelevent)
```

如何被触发？
设立一个可以被鼠标点击的按钮，当用户点击该按钮之后，按钮会被“按压”，然后“释放”，这时mouseReleaseEvent() 被触发，程序会运行 mouseReleaseEvent() 中的逻辑代码。

》》》》不要忘记升级按钮！！！

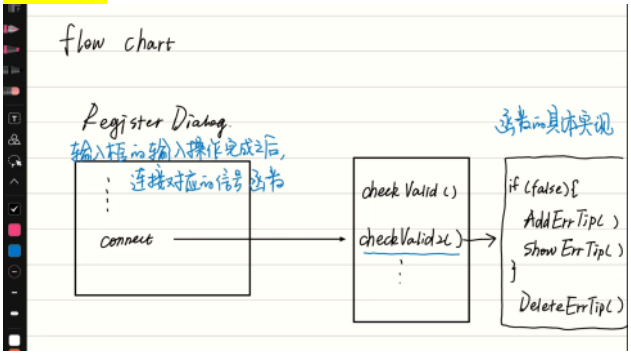


成功实现：



---------- 分割线 ----------

》》》》流程图



》》》》Map 中的数据会根据 key 值进行自动排序，于元素插入时的顺序无关

》》》》正则表达式的理解：

正则表达式结构解释：

^:	这个符号是 开始锚点，表示匹配字符串的开始位置。即正则表达式会从字符串的开头开始进行匹配。
[a-zA-Z0-9!@#$%^&*]:	方括号 [] 表示定义了一个 字符集，该字符集中的字符可以出现在匹配的字符串中。具体来说： ▪a-z：匹配任意小写字母（从 a 到 z）。 ▪A-Z：匹配任意大写字母（从 A 到 Z）。 ▪0-9：匹配任意数字（从 0 到 9）。 ▪!@#$%^&*：匹配这些特殊字符：!、@、#、$、%、^、&、*。 这意味着，匹配的字符串可以包含大小写字母、数字以及这些特定的特殊字符。
{6,15}:	○这是一个 量词，表示前面字符集中的字符必须出现 至少6次，最多15次。 ○也就是说，匹配的字符串长度必须在 6到15个字符之间（包括6个字符和15个字符）。
$:	这个符号是 结束锚点，表示匹配字符串的结尾位置。即正则表达式会确保整个字符串（从头到尾）都符合条件。

》》》》一点改进：

由于在 on_confirm_button_clicked() 这个函数中需要重复进行这个操作：
而我觉得看起来不顺眼，就将其优化成了 宏定义

```
    if(!valid){
        return;
    }

    valid = checkEmailValid();
    if(!valid){
        return;
    }

    valid = checkPassValid();
    if(!valid){
        return;
    }

    valid = checkVarifyValid();
    if(!valid){
        return;
    }
```

以下是改进的结果：

```
#include <QDialog>
#include <type_traits>

// 这个宏填入的参数必须是 std::function<bool()> 类型的函数名称 ！！！
#define CHECK_VALID(func){\
    bool valid = func();\
    if(!valid)\
    {\
        return;\
    }\
}

namespace Ui
{
```

```
void RegisterDialog::on_confirm_button_clicked()
{
    CHECK_VALID(CheckUserValid);
    CHECK_VALID(CheckEmailValid);
    CHECK_VALID(CheckPassValid);
    CHECK_VALID(CheckConfirmValid);
    CHECK_VALID(CheckVerifyValid);

    QJsonObject jsonObj;
    jsonObj["user"] = ui->user
    jsonObj["email"] = ui->ema
    jsonObj["password"] = MD5Encrypt(ui->pass_edit->text());
    jsonObj["confirm"] = MD5Encrypt(ui->confirm_edit->text());
    jsonObj["verifycode"] = ui->verify_edit->text();
    HttpMgr::GetInstance()->PostHttpReq(QUrl(gateUrlPrefix + "/user_regist
```

commit
Author: Not Committed Yet <>
Date:
Modified line in registerdialog.cpp

可以看到，这样极大的节省了空间。