Projeto Proxy Server

Roberto Nishino
Versão 1.0
Quinta, 5 de Dezembro de 2019

Conteúdo

Table of contents

2

Namespaces

NamespacesLista de Namespaces

Esta é a lista de todos os Namespaces com suas respectivas descrições:

Ui

Índice Hierárquico

Índice HierárquicoHierarquia de Classes

Esta lista de hierarquias está parcialmente ordenada (ordem alfabética):

- Button
- HttpParser
- QMainWindow
  - Aracne
  - MainWindow
- QObject
  - Client
  - Socket
- QTcpServer
  - Proxy

Índice dos Componentes

Índice dos ComponentesLista de Componentes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

Índice dos Arquivos

Índice dos ArquivosLista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

aracne.cpp
aracne.h
button.cpp
button.h
client.cpp
client.h
httpparser.cpp
httpparser.h
main.cpp
mainwindow.cpp
mainwindow.h
proxy.cpp
proxy.h
socket.cpp
socket.h

Namespaces

Refência do Namespace Ui

UiUi

ClassesClasses

Referência da Classe Aracne

AracneAracne

`#include <aracne.h>`

Diagrama de Hierarquia para Aracne:

IMAGE

Slots Públicos

1 void **new_request** (QByteArray, int)
*Trata nova requisição e envia as informações para o **Client**.*

2 void **new_reply** (QByteArray, int)
*Trata nova resposta e salva as informações para o **Client**.*

Sinais

3 void **send_request** (QByteArray, int)
4 void **send_reply** (QByteArray, int)
5 void **turn_on** ()
6 void **turn_off** ()
7 void **set_mode** (QString)

Métodos Públicos

8 **Aracne** (QWidget *parent=nullptr)
Abre a tela com valores default e server offline.

9 **~Aracne** ()
*Destructor **Aracne**.*

10 void **setServerOff** ()
Muda os status do servidor para desligado.

11 void **setServerOn** ()
Muda os status do servidor para ligado.

Construtores & Destrutores

Aracne:AracneAracne:AracneAracne::Aracne (QWidget * *parent* = nullptr)[explicit]

Abre a tela com valores default e server offline.

Parâmetros:

parent	Tela de UI
ui	**Aracne**

```
 9                                : QMainWindow(parent), ui(new Ui::Aracne) {
10     ui->setupUi(this);
11     this->ui->serverLabel->setText("Server Offline");
```

```
12      this->ui->serverLabel->setStyleSheet("QLabel{color:red;}");
13
14      this->ui->startServerButton->setDisabled(false);
15      this->ui->stopServerButton->setDisabled(true);
16  }
```

~Aracne:AracneAracne:~AracneAracne::~Aracne ()

Destructor **Aracne**.

```
19                  {
20      delete ui;
21  }
```

Métodos

new_reply:AracneAracne:new_replyvoid Aracne::new_reply (QByteArray *reply*, int *clientId*)`[slot]`

Trata nova resposta e salva as informações para o **Client**.

Parâmetros:

reply	String da resposta
clientID	ID do **Client**

Cria uma nova resposta, adiciona na lista de respostas, permite edição e salva para uso no **Client**

```
39                                                      {
40      QListWidgetItem *item = new QListWidgetItem(reply);
41      item->setFlags (item->flags () | Qt::ItemIsEditable);
42      item->setSizeHint(QSize(item->sizeHint().width(), 250));
43      this->ui->repliesWidget->addItem(item);
44      this->repliesId.push(clientId);
45  }
```

new_request:AracneAracne:new_requestvoid Aracne::new_request (QByteArray *request*, int *clientId*)`[slot]`

Trata nova requisição e envia as informações para o **Client**.

Parâmetros:

clientId	ID do host
request	String da requisição

Cria uma nova requisição, adiciona na lista de requisições, permite edição e envia para o **Client**

```
27                                                          {
28      QListWidgetItem *item = new QListWidgetItem(request);
29      item->setFlags (item->flags () | Qt::ItemIsEditable);
30      item->setSizeHint(QSize(item->sizeHint().width(), 250));
31      this->ui->requestsWidget->addItem(item);
32      this->requestsId.push(clientId);
33  }
```

send_reply:AracneAracne:send_replyvoid Aracne::send_reply (QByteArray , int)[signal]

send_request:AracneAracne:send_requestvoid Aracne::send_request (QByteArray , int)[signal]

set_mode:AracneAracne:set_modevoid Aracne::set_mode (QString)[signal]

setServerOff:AracneAracne:setServerOffvoid Aracne::setServerOff ()

Muda os status do servidor para desligado.

Muda o label para Server Offline, muda a cor para vermelho e desliga o servidor **Proxy**

```
81                          {
82     this->ui->serverLabel->setText("Server Offline");
83     this->ui->serverLabel->setStyleSheet("QLabel{color:red;}");
84
85     this->ui->startServerButton->setDisabled(false);
86     this->ui->stopServerButton->setDisabled(true);
87 }
```

setServerOn:AracneAracne:setServerOnvoid Aracne::setServerOn ()

Muda os status do servidor para ligado.

Muda o label para Server Online, a cor para verde e liga o servidor **Proxy**

```
71                          {
72     this->ui->serverLabel->setText("Server Online");
73     this->ui->serverLabel->setStyleSheet("QLabel{color:green;}");
74
75     this->ui->startServerButton->setDisabled(true);
76     this->ui->stopServerButton->setDisabled(false);
77 }
```

turn_off:AracneAracne:turn_offvoid Aracne::turn_off ()[signal]

turn_on:AracneAracne:turn_onvoid Aracne::turn_on ()[signal]

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

1. **aracne.h**
2. **aracne.cpp**

Referência da Classe Button

ButtonButton

`#include <button.h>`

Métodos Públicos

12 **Button** ()
13 void **handleClick** ()

Atributos Protegidos

14 Ui::Aracne * **_ui**
15 QPushButton * **_button**

Construtores & Destrutores

Button:ButtonButton:ButtonButton::Button ()

```
 4 {
 5     this->_ui = ui;
 6 }
```

Métodos

handleClick:ButtonButton:handleClickvoid Button::handleClick ()

```
 9 {
10     this->_button->t
11 }
```

Atributos

_button:ButtonButton:_buttonQPushButton* Button::_button [protected]

_ui:ButtonButton:_uiUi::Aracne* Button::_ui [protected]

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

3 **button.h**
4 **button.cpp**

Referência da Classe Client

ClientClient

`#include <client.h>`

Diagrama de Hierarquia para Client:

IMAGE

Slots Públicos

16 void **connected** ()

17 void **disconnected** ()

18 void **readyRead** ()

19 void **send_request** (QByteArray, int)

Sinal de enviar Request.

20 void **send_reply** (QByteArray, int)

Sinal de enviar Reply.

Sinais

21 void **new_request** (QByteArray, int)

Métodos Públicos

22 **Client** (QObject *parent=nullptr, bool busy=false, int id=0)

*Constructor **Client**.*

23 void **setSocket** (qintptr Descriptor)

*Trata a conexão no **Socket**.*

24 void **setAracne** (**Aracne** *aracne)

Construtores & Destrutores

Client:ClientClient:ClientClient::Client (QObject * *parent* = `nullptr`, bool *busy* = `false`, int *id* = `0`)`[explicit]`

Constructor **Client**.

Instanciado com o status como busy

```
 5                                                    :
 6     QObject(parent)
 7 {
 8     this->busy = busy;
 9     this->myId = id;
10 }
```

Métodos

connected:ClientClient:connectedvoid Client::connected ()`[slot]`

```
35 {
36     qDebug() << "Client conectado";
37 }
```

disconnected:ClientClient:disconnectedvoid Client::disconnected ()[slot]

```
41 {
42     qDebug() << "Client desconectado";
43 }
```

new_request:ClientClient:new_requestvoid Client::new_request (QByteArray , int)[signal]

readyRead:ClientClient:readyReadvoid Client::readyRead ()[slot]

```
46 {
47     qDebug() << Q_FUNC_INFO << "Socket::readyRead()";
48
49     QByteArray request = socket->readAll();
50
51     qDebug() << Q_FUNC_INFO << request;
52
53     emit new_request(request, this->myId);
54 }
```

send_reply:ClientClient:send_replyvoid Client::send_reply (QByteArray *reply*, int *clientId*)[slot]

Sinal de enviar Reply.

Parâmetros:

reply	String da resposta
clientId	ID do **Client**

Envia a resposta de volta e fecha conexão

```
74                                                  {
75     if(clientId == this->myId){
76         socket->write(reply);
77         socket->disconnectFromHost();
78     }
79 }
```

send_request:ClientClient:send_requestvoid Client::send_request (QByteArray *request*, int *clientId*)[slot]

Sinal de enviar Request.

Parâmetros:

request	String da requisição
clientId	ID do **Client**

Cria o socket para fazer a conexão do **Client** para enviar a requisição e aguarda pela resposta

```
60                                                  {
61     if(this->myId == clientId){
62         Socket *socket = new Socket(request, clientId);
63
64         connect(socket, SIGNAL(new_reply(QByteArray, int)), this->aracne,
SLOT(new_reply(QByteArray, int)));
65
66         socket->start();
67     }
```

```
68 }
```

setAracne:ClientClient:setAracnevoid Client::setAracne (Aracne * *aracne*)

```
29                                      {
30     this->aracne = aracne;
31 }
```

setSocket:ClientClient:setSocketvoid Client::setSocket (qintptr *Descriptor*)

Trata a conexão no **Socket**.

Parâmetros:

Descriptor	File Descriptor

```
15 {
16     socket = new QTcpSocket(this);
17
18     qDebug() << "Socket criado";
19
20     connect(socket, SIGNAL(connected()), this, SLOT(connected()));
21     connect(socket, SIGNAL(disconnected()), this, SLOT(disconnected()));
22     connect(socket, SIGNAL(readyRead()), this, SLOT(readyRead()));
23
24     socket->setSocketDescriptor(Descriptor);
25
26     qDebug() << " Client conectado em " << Descriptor;
27 }
```

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

5 **client.h**
6 **client.cpp**

Referência da Classe HttpParser

HttpParserHttpParser

`#include <httpparser.h>`

Métodos Públicos

25 **HttpParser** ()

26 void **parseResponse** (QByteArray)

27 void **parseRequest** (QByteArray)

Trata os headers da requisição.

28 QString **getHostname** ()

Get no Host da requisição.

29 QString **getPath** ()

Get Path.

30 QString **getContentLength** ()

Get tamanho do conteúdo.

31 QString **getContentType** ()

Get no tipo do conteúdo da requisição.

32 void **setHostname** (std::string)

33 void **setRequest** (QByteArray)

34 QByteArray **getRequest** ()

Construtores & Destrutores

HttpParser:HttpParserHttpParser:HttpParserHttpParser::HttpParser ()

```
3                          {
4
5 }
```

Métodos

getContentLength:HttpParserHttpParser:getContentLengthQString HttpParser::getContentLength ()

Get tamanho do conteúdo.

```
64                                      {
65     return QString(this->_http_request[ "Content-Length" ].c_str());
66 }
```

getContentType:HttpParserHttpParser:getContentTypeQString HttpParser::getContentType ()

Get no tipo do conteúdo da requisição.

```
54                                     {
55     return QString(this->_http_request[ "Content-Type" ].c_str());
56 }
```

getHostname:HttpParserHttpParser:getHostnameQString HttpParser::getHostname ()

Get no Host da requisição.

```
49                                  {
50      return QString(this->_http_request[ "Host" ].c_str());
51  }
```

getPath:HttpParserHttpParser:getPathQString HttpParser::getPath ()

Get Path.

```
59                                {
60      return QString(this->_http_request[ "Path" ].c_str());
61  }
```

getRequest:HttpParserHttpParser:getRequestQByteArray HttpParser::getRequest ()

```
73                                    {
74      return this->_request;
75  }
```

parseRequest:HttpParserHttpParser:parseRequestvoid HttpParser::parseRequest (QByteArray *request*)

Trata os headers da requisição.

Parâmetros:

request	String da requisição

Find request type

Find path

Find HTTP version

Map all headers from a key to a value

```
 9                                                       {
10      char *msg = request.data();
11      char *head = msg;
12      char *mid;
13      char *tail = head;
14
15      if( request.size() == 0 ){
16          return;
17      }
18
20      while( *head++ != ' ');
21      this->_http_request[ "Type" ] = std::string( ( char * ) msg ).substr( 0 ,
( head - 1) - tail );
22
24      tail = head;
25      while( *head++ != ' ');
26      this->_http_request[ "Path" ] = std::string( ( char * ) msg ).substr(
tail - ( char *)msg , ( head - 1) - tail );
27
29      tail = head;
30      while( *head++ != '\r');
31      this->_http_request[ "Version" ] = std::string( ( char * ) msg ).substr(
tail - ( char *)msg , ( head - 1) - tail );
32
```

```
34      while( true )
35      {
36          tail = head + 1;
37          while( *head++ != '\r' );
38          mid = strstr( tail, ":" );
39
40          // Look for the failed strstr
41          if( tail > mid )
42              break;
43
44          this->_http_request[ std::string( ( char * ) msg ).substr( tail - (
char *)msg , ( mid ) - tail  ) ] = std::string( ( char * ) msg ).substr( mid + 2 -
( char *) msg , ( head - 3 ) - mid );
45      }
46 }
```

parseResponse:HttpParserHttpParser:parseResponsevoid HttpParser::parseResponse (QByteArray)

setHostname:HttpParserHttpParser:setHostnamevoid HttpParser::setHostname (std::string)

setRequest:HttpParserHttpParser:setRequestvoid HttpParser::setRequest (QByteArray *request*)

```
69                                                  {
70      this->_request = request;
71 }
```

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

7 **httpparser.h**
8 **httpparser.cpp**

Referência da Classe MainWindow

MainWindowMainWindow

`#include <mainwindow.h>`

Diagrama de Hierarquia para MainWindow:

IMAGE

Métodos Públicos

35 **MainWindow** (QWidget *parent=nullptr)

36 **~MainWindow** ()

Construtores & Destrutores

MainWindow:MainWindowMainWindow:MainWindowMainWindow::MainWindow (QWidget * *parent* = `nullptr`)

```
5     : QMainWindow(parent)
6     , ui(new Ui::MainWindow)
7 {
8     ui->setupUi(this);
9 }
```

~MainWindow:MainWindowMainWindow:~MainWindowMainWindow::~MainWindow ()

```
12 {
13     delete ui;
14 }
```

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

9 **mainwindow.h**

10 **mainwindow.cpp**

Referência da Classe Proxy

ProxyProxy

`#include <proxy.h>`

Diagrama de Hierarquia para Proxy:

IMAGE

Slots Públicos

37 void **turn_on** ()

38 void **turn_off** ()

39 void **set_mode** (QString)

*Seta o modo do **Proxy** para Standard.*

Métodos Públicos

40 **Proxy** (QObject *parent=nullptr, **Aracne** *aracne=nullptr, unsigned short port=8228)

*Constructor **Proxy**.*

41 void **start** ()

Métodos Protegidos

42 void **incomingConnection** (qintptr socketDescriptor) Q_DECL_OVERRIDE

*Trata conexão vindo do **Socket**.*

Construtores & Destrutores

Proxy:ProxyProxy:ProxyProxy::Proxy (QObject * *parent* = `nullptr`, Aracne * *aracne* = `nullptr`, unsigned short *port* = `8228`)

Constructor **Proxy**.

Parâmetros:

port	Porta para o **Proxy** escutar

```
 6                                                                    :
 7     QTcpServer (parent)
 8 {
 9     this->_aracne = aracne;
10     this->_port = port;
11     connect(this->_aracne, SIGNAL(turn_on()), this, SLOT(turn_on()));
12     connect(this->_aracne, SIGNAL(turn_off()), this, SLOT(turn_off()));
13 }
```

Métodos

incomingConnection:ProxyProxy:incomingConnectionvoid Proxy::incomingConnection (qintptr *socketDescriptor*) `[protected]`

Trata conexão vindo do **Socket**.

Parâmetros:

socketDescriptor	File Descriptor

```
44                                                      {
45     if(!this->serverOn) return;
46     else if(this->mode == "standard") this->standardMode(socketDescriptor);
47     else return;
48 }
```

set_mode:ProxyProxy:set_modevoid Proxy::set_mode (QString *mode*)[slot]

Seta o modo do **Proxy** para Standard.

Futuramente haverá outros modos como o Spider por exemplo

```
29                                  {
30     if(mode != "standard"){
31         qDebug() << "Incorrect mode: " << mode;
32         return;
33     }
34     qDebug() << "Changing mode: " << mode;
35     this->mode = mode;
36 }
```

start:ProxyProxy:startvoid Proxy::start ()

```
15                  {
16     if(!listen(QHostAddress::Any, this->_port)){
17         qDebug() << "Server could not start";
18     } else{
19         qDebug() << "Server started on port " << this->_port;
20     }
21 }
```

turn_off:ProxyProxy:turn_offvoid Proxy::turn_off ()[slot]

```
38                     {
39     this->serverOn = false;
40 }
```

turn_on:ProxyProxy:turn_onvoid Proxy::turn_on ()[slot]

```
23                    {
24     this->serverOn = true;
25 }
```

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

11 **proxy.h**
12 **proxy.cpp**

Referência da Classe Socket

SocketSocket

`#include <socket.h>`

Diagrama de Hierarquia para Socket:

IMAGE

Slots Públicos

43 void **connected** ()
Pega a reply do host e fecha a conexão.

44 void **disconnected** ()
Envia a resposta para ser tratada.

Sinais

45 void **new_reply** (QByteArray, int)

Métodos Públicos

46 **Socket** (QByteArray request=nullptr, int clientId=0)
*Constructor do **Socket**.*

47 void **start** ()
Faz a conexão com o host.

Construtores & Destrutores

Socket:SocketSocket:SocketSocket::Socket (QByteArray *request* = `nullptr`, int *clientId* = `0`) `[explicit]`

Constructor do **Socket**.

```
 5                                                   {
 6     this->myId = clientId;
 7     this->request = request;
 8     this->originalRequest = request;
 9     QThreadPool::globalInstance()->setMaxThreadCount(5);
10 }
```

Métodos

connected:SocketSocket:connectedvoid Socket::connected () `[slot]`

Pega a reply do host e fecha a conexão.

```
37                        {
38     qDebug() << "Conectado";
39
40     this->socket->write(this->originalRequest);
41     while(this->socket->waitForReadyRead(3000)){
42         while(this->socket->bytesAvailable() > 0){
```

```
43              this->reply.append(this->socket->readAll());
44              this->socket->flush();
45          }
46      }
47
48      qDebug() << this->reply;
49      this->socket->disconnectFromHost();
50      this->socket->deleteLater();
51  }
```

disconnected:SocketSocket:disconnectedvoid Socket::disconnected ()[slot]

Envia a resposta para ser tratada.

```
54                          {
55      qDebug() << "Desconectado";
56      emit new_reply(this->reply, this->myId);
57  }
```

new_reply:SocketSocket:new_replyvoid Socket::new_reply (QByteArray , int)[signal]

start:SocketSocket:startvoid Socket::start ()

Faz a conexão com o host.

Usa o Parser para os GET na requisição e prepara a conexão com o host

```
14                  {
15      qDebug() << Q_FUNC_INFO << "Starting thread for client " << this->myId;
16      qDebug() << Q_FUNC_INFO << "Request: " << this->request;
17
18      this->socket = new QTcpSocket(this);
19
20      connect(this->socket, SIGNAL(connected()), this, SLOT(connected()));
21      connect(this->socket, SIGNAL(disconnected()), this,
SLOT(disconnected()));
22
23      qDebug() << Q_FUNC_INFO << "Conectando...";
24
25      HttpParser parser;
26
27      parser.parseRequest(this->request);
28
29      char hostname[32];
30      strcpy(hostname, parser.getHostname().toStdString().c_str());
31
32      qDebug() << Q_FUNC_INFO << "Hostname: " << hostname;
33      this->socket->connectToHost(hostname, 80);
34  }
```

A documentação para esta classe foi gerada a partir dos seguintes arquivos:

13 **socket.h**
14 **socket.cpp**

ArquivosArquivos

Referência do Arquivo aracne.cpp

aracne.cpparacne.cpp#include "aracne.h"
#include "ui_aracne.h"
#include <iostream>

Referência do Arquivo aracne.h

aracne.haracne.h#include <QDebug>
#include <QMainWindow>
#include "string"
#include "list"
#include "QListWidgetItem"
#include <queue>

Componentes

48 class **Aracne**

Namespaces

49 **Ui**

Referência do Arquivo button.cpp

button.cppbutton.cpp`#include "button.h"`

Referência do Arquivo button.h

button.hbutton.h`#include "ui_aracne.h"`
`#include <QApplication>`

Componentes

50 class **Button**

Referência do Arquivo client.cpp

client.cppclient.cpp`#include "client.h"`

Referência do Arquivo client.h

client.hclient.h#include <QObject>
#include <QTcpSocket>
#include <QDebug>
#include <QThreadPool>
#include "socket.h"
#include "aracne.h"

Componentes

51 class **Client**

Referência do Arquivo httpparser.cpp

httpparser.cpphttpparser.cpp`#include "httpparser.h"`

Referência do Arquivo httpparser.h

httpparser.hhttpparser.h`#include "QByteArray"`

```
#include "QtDebug"
#include <iostream>
#include <map>
#include <string>
```

Componentes

52 class **HttpParser**

Referência do Arquivo main.cpp

main.cppmain.cpp`#include "aracne.h"`
`#include "proxy.h"`
`#include "client.h"`
`#include <QApplication>`

Funções

53 int **main** (int argc, char *argv[])

Funções

main:main.cppmain.cpp:mainint main (int *argc*, char * *argv*[])

```
      6                                 {
      7     QApplication a(argc, argv);
      8     QCoreApplication::setApplicationName("Aracne");
      9
     10     QCommandLineParser parser;
     11     parser.addHelpOption();
     12     parser.addVersionOption();
     13     QCommandLineOption portOption(QStringList() << "p" << "port",
     14                                   QCoreApplication::translate("main", "Porta
   proxy."),
     15                                   QCoreApplication::translate("main",
   "porta."));
     16     parser.addOption(portOption);
     17
     18     parser.process(a);
     19
     20     QString portValue = parser.value(portOption);
     21
     22     Aracne w;
     23     w.show();
     24
     25     Proxy *proxy = portValue.length() == 0 ? new Proxy(nullptr, &w) :
     26                                              new Proxy(nullptr, &w,
   static_cast<unsigned short>(std::stoul(portValue.toStdString())));
     27     proxy->start();
     28
     29     return a.exec();
     30 }
```

Referência do Arquivo mainwindow.cpp

mainwindow.cppmainwindow.cpp`#include "mainwindow.h"`
`#include "ui_mainwindow.h"`

Referência do Arquivo mainwindow.h

mainwindow.hmainwindow.h`#include <QMainWindow>`

Componentes

54 class **MainWindow**

Namespaces

55 **Ui**

Referência do Arquivo proxy.cpp

proxy.cppproxy.cpp#include "proxy.h"

Referência do Arquivo proxy.h

proxy.hproxy.h#include "aracne.h"
#include "client.h"
#include <QObject>
#include <QTcpServer>
#include <QTcpSocket>
#include <QAbstractSocket>

Componentes

56 class **Proxy**

Referência do Arquivo socket.cpp

socket.cppsocket.cpp`#include "socket.h"`

Referência do Arquivo socket.h

socket.hsocket.h`#include "aracne.h"`

```
#include <QtCore>
#include <QtNetwork>
#include <QByteArray>
#include "httpparser.h"
```

Componentes

57 class **Socket**

Índice

ÍndiceINDEX

38