Qt基礎類型以及字符串類型

C語言Plus 今天

1. 基礎類型

因為Qt是一個C++框架, 因此C++中所有的語法和數據類型在Qt中都是被支持的, 但是Qt中也定義了一些屬於自己的數據類型, 下邊給大家介紹一下這些基礎的數類型。

QT基本數據類型定義在`#include <QtGlobal>`

類型名稱	註釋	備註
qint8	signed char	有符號8位數據
qint16	signed short	16位數據類型
qint32	signed short	32位有符號數據類型
qint64	long long int 或(__int64)	64位有符號數據類型，Windows中定義為
qintptr	qint32 或qint64	指針類型根據系統類型不同而不同，32位2、64位系統為qint64
qlonglong	long long int 或(__int64)	Windows中定義為__int64
qptrdiff	qint32 或qint64	根據系統類型不同而不同，32位系統為qi系統為qint64
qreal	double 或float	除非配置了-qreal float選項，否則默認為
quint8	unsigned char	無符號8位數據類型
quint16	unsigned short	無符號16位數據類型
quint32	unsigned int	無符號32位數據類型
quint64	unsigned long long int 或(unsigned __int64)	無符號64比特數據類型，Windows中定義__int64
quintptr	quint32 或quint64	根據系統類型不同而不同，32位系統為qu系統為quint64
	unsigned long long int 或(unsig	

類型名稱	註釋	備註
qulonglong	unsigned long long int 或(unsigned __int64)	Windows中定義為__int64
uchar	unsigned char	無符號字符類型
uint	unsigned int	無符號整型
ulong	unsigned long	無符號長整型
ushort	unsigned short	無符號短整型
qsizetype	size_t	

2. log輸出

> 在Qt中進行log輸出, 一般不使用c中的`printf` `cout` `QDebug`

基本分類

- qDebug：調試信息提示
- qInfo ：輸出信息
- qWarning ：一般的警告提示
- qCritical ：嚴重的錯誤提示
- qFatal ：致命錯誤提示，會直接中斷程序

C風格輸出

```
qDebug("我是%s，今年%d岁了~","maye",20);
qInfo("maye%d",666);
qWarning("hello %s","warning");
qCritical("helo %s","critical");
qFatal("hello %s","qFatal");    //致命错误会直接中断程序
```

C++風格

```
qDebug()<<"好帅"<<endl;
qInfo()<<"qInfo"<<endl;
qWarning()<<"qWarnning"<<endl;
qCritical()<<"qCritical"<<endl;
//qFatal()<<"qFatal"<<endl;      //致命错误不能用<<输出
```

3. 字符串類型

> C => `char*`
>
> C++ => `std::string`
>
> Qt => `QByteArray` `QString`

3.1 QByteArray

> 在Qt中 我們在使用這種類型的時候可通過這個類的構造函數申請一塊動態內存，用於存儲我們需要處理的字符串數據。`QByteArray` `char*`
>
> 下面給大家介紹一下這個類中常用的一些API函數，`大家要养成遇到问题主动查询帮助文档的好习惯`

構造函數

```
// 构造空对象, 里边没有数据
QByteArray::QByteArray();
// 将data中的size个字符进行构造, 得到一个字节数组对象
// 如果 size==-1 函数内部自动计算字符串长度, 计算方式为: strlen(data)
QByteArray::QByteArray(const char *data, int size = -1);
// 构造一个长度为size个字节, 并且每个字节值都为ch的字节数组
QByteArray::QByteArray(int size, char ch);
```

數據操作

```
// 其他重载的同名函数可参考Qt帮助文档, 此处略
QByteArray &QByteArray::append(const QByteArray &ba);
void QByteArray::push_back(const QByteArray &other);

// 其他重载的同名函数可参考Qt帮助文档, 此处略
QByteArray &QByteArray::prepend(const QByteArray &ba);
void QByteArray::push_front(const QByteArray &other);

// 插入数据, 将ba插入到数组第 i 个字节的位置(从0开始)
// 其他重载的同名函数可参考Qt帮助文档, 此处略
QByteArray &QByteArray::insert(int i, const QByteArray &ba);
```

```
12
13 // 删除数据
14 // 从大字符串中删除len个字符, 从第pos个字符的位置开始删除
15 QByteArray &QByteArray::remove(int pos, int len);
16 // 从字符数组的尾部删除 n 个字节
17 void QByteArray::chop(int n);
18 // 从字节数组的 pos 位置将数组截断 (前边部分留下, 后边部分被删除)
19 void QByteArray::truncate(int pos);
20 // 将对象中的数据清空, 使其为null
21 void QByteArray::clear();
22
23 // 字符串替换
24 // 将字节数组中的 子字符串 before 替换为 after
25 // 其他重载的同名函数可参考Qt帮助文档, 此处略
26 QByteArray &QByteArray::replace(const QByteArray &before, const QByteArray &af
```

子字符串查找和判断

```
1 // 判断字节数组中是否包含子字符串 ba, 包含返回true, 否则返回false
2 bool QByteArray::contains(const QByteArray &ba) const;
3 bool QByteArray::contains(const char *ba) const;
4 // 判断字节数组中是否包含子字符 ch, 包含返回true, 否则返回false
5 bool QByteArray::contains(char ch) const;
6
7 // 判断字节数组是否以字符串 ba 开始, 是返回true, 不是返回false
8 bool QByteArray::startsWith(const QByteArray &ba) const;
9 bool QByteArray::startsWith(const char *ba) const;
10 // 判断字节数组是否以字符 ch 开始, 是返回true, 不是返回false
11 bool QByteArray::startsWith(char ch) const;
12
13 // 判断字节数组是否以字符串 ba 结尾, 是返回true, 不是返回false
14 bool QByteArray::endsWith(const QByteArray &ba) const;
15 bool QByteArray::endsWith(const char *ba) const;
16 // 判断字节数组是否以字符 ch 结尾, 是返回true, 不是返回false
17 bool QByteArray::endsWith(char ch) const;
```

遍歷

```
// 使用迭代器
iterator QByteArray::begin();
iterator QByteArray::end();

// 使用数组的方式进行遍历
// i的取值范围 0 <= i < size()
char QByteArray::at(int i) const;
char QByteArray::operator[](int i) const;
```

查看字節數

```
// 返回字节数组对象中字符的个数
 int QByteArray::length() const;
 int QByteArray::size() const;
 int QByteArray::count() const;

 // 返回字节数组对象中 子字符串ba 出现的次数
 int QByteArray::count(const QByteArray &ba) const;
 int QByteArray::count(const char *ba) const;
 // 返回字节数组对象中 字符串ch 出现的次数
 int QByteArray::count(char ch) const;
```

類型轉換

```
// 将QByteArray类型的字符串 转换为 char* 类型
 char *QByteArray::data();
 const char *QByteArray::data() const;

 // int, short, long, float, double -> QByteArray
 // 其他重载的同名函数可参考Qt帮助文档, 此处略
 QByteArray &QByteArray::setNum(int n, int base = 10);
 QByteArray &QByteArray::setNum(short n, int base = 10);
 QByteArray &QByteArray::setNum(qlonglong n, int base = 10);
 QByteArray &QByteArray::setNum(float n, char f = 'g', int prec = 6);
 QByteArray &QByteArray::setNum(double n, char f = 'g', int prec = 6);
```

```
[static] QByteArray QByteArray::number(int n, int base = 10);
[static] QByteArray QByteArray::number(qlonglong n, int base = 10);
[static] QByteArray QByteArray::number(double n, char f = 'g', int prec = 6);

// QByteArray -> int, short, long, float, double
int QByteArray::toInt(bool *ok = Q_NULLPTR, int base = 10) const;
short QByteArray::toShort(bool *ok = Q_NULLPTR, int base = 10) const;
long QByteArray::toLong(bool *ok = Q_NULLPTR, int base = 10) const;
float QByteArray::toFloat(bool *ok = Q_NULLPTR) const;
double QByteArray::toDouble(bool *ok = Q_NULLPTR) const;

// std::string -> QByteArray
[static] QByteArray QByteArray::fromStdString(const std::string &str);
// QByteArray -> std::string
std::string QByteArray::toStdString() const;

// 所有字符转换为大写
QByteArray QByteArray::toUpper() const;
// 所有字符转换为小写
QByteArray QByteArray::toLower() const;
```

3.2 QString

> QString也是封裝了字符串, 但是內部的編碼為`utf8`
>
> 下面給大家介紹一下這個類中常用的一些API函數。

構造函數

```
// 构造一个空字符串对象
QString();
// 将 char* 字符串 转换为 QString 类型
QString(const char *str);
// 将 QByteArray 转换为 QString 类型
QString(const QByteArray &ba);
// 其他重载的同名构造函数可参考Qt帮助文档, 此处略
```

數據操作

```
// 尾部追加数据
 QString& append(const QString &str);
 QString& append(const char *str);
 QString& append(const QByteArray &ba);
 void push_back(const QString &other);

 // 头部添加数据
 QString& prepend(const QString &str);
 QString& prepend(const char *str);
 QString& prepend(const QByteArray &ba);
 void QString::push_front(const QString &other);

 // 插入数据, 将 str 插入到字符串第 position 个字符的位置(从0开始)
 QString& insert(int position, const QString &str);
 QString& insert(int position, const char *str);
 QString& insert(int position, const QByteArray &str);

 // 删除数据
 // 从大字符串中删除len个字符, 从第pos个字符的位置开始删除
 QString& remove(int position, int n);

 // 从字符串的尾部删除 n 个字符
 void  chop(int n);
 // 从字节串的 position 位置将字符串截断 (前边部分留下, 后边部分被删除)
 void  truncate(int position);
 // 将对象中的数据清空, 使其为null
 void  clear();

 // 字符串替换
 // 将字节数组中的 子字符串 before 替换为 after
 // 参数 cs 为是否区分大小写, 默认区分大小写
 QString& replace(const QString &before, const QString &after, Qt::CaseSensiti
```

子字符串查找和判斷

```
// 参数 cs 为是否区分大小写，默认区分大小写
// 其他重载的同名函数可参考Qt帮助文档，此处略

// 判断字符串中是否包含子字符串 str，包含返回true，否则返回false
bool  contains(const QString &str, Qt::CaseSensitivity cs = Qt::CaseSensitive

// 判断字符串是否以字符串 ba 开始，是返回true，不是返回false
bool startsWith(const QString &s, Qt::CaseSensitivity cs = Qt::CaseSensitive)

// 判断字符串是否以字符串 ba 结尾，是返回true，不是返回false
bool endsWith(const QString &s, Qt::CaseSensitivity cs = Qt::CaseSensitive) c
```

遍歷

```
// 使用迭代器
iterator  begin();
iterator  end();

// 使用数组的方式进行遍历
const QChar  at(int position) const
const QChar  operator[](int position) const;
```

查看字節數

```
// 返回字节数组对象中字符的个数
int  length() const;
int  size() const;
int  count() const;

// 返回字节串对象中 子字符串 str 出现的次数
// 参数 cs 为是否区分大小写，默认区分大小写
int  count(const QStringRef &str, Qt::CaseSensitivity cs = Qt::CaseSensitive)
```

類型轉換

```
// int, short, long, float, double -> QString
 // 其他重载的同名函数可参考Qt帮助文档，此处略
 QString& setNum(int n, int base = 10);
 QString& setNum(short n, int base = 10);
 QString& setNum(long n, int base = 10);
 QString& setNum(float n, char format = 'g', int precision = 6);
 QString&QString::setNum(double n, char format = 'g', int precision = 6);
 [static] QString QString::number(long n, int base = 10);
 [static] QString QString::number(int n, int base = 10);
 [static] QString QString::number(double n, char format = 'g', int precision =

 // QString -> int, short, long, float, double
 int QString::toInt(bool *ok = Q_NULLPTR, int base = 10) const;
 short QString::toShort(bool *ok = Q_NULLPTR, int base = 10) const;
 long QString::toLong(bool *ok = Q_NULLPTR, int base = 10) const
 float QString::toFloat(bool *ok = Q_NULLPTR) const;
 double QString::toDouble(bool *ok = Q_NULLPTR) const;


 // 所有字符转换为大写
 QString QString::toUpper() const;
 // 所有字符转换为小写
 QString QString::toLower() const;
```

字符串格式化

C語言中有sprintf()函數，QString也提供了一個asprintf()函數。

```
 QString res =  asprintf("fileName:%s size:%d","./av.jpg",20);
 qDebug()<<res<<endl;
```

不過QString還提供的另一種格式化字符串輸出的函數arg()，更為方便。

```
 QString arg(const QString &a, int fieldWidth = 0, QChar fillChar = QLatin1Cha
 QString arg(int a, int fieldWidth = 0, int base = 10, QChar fillChar = QLatir
 //用于填充字符串中的%1,%2…为给定格式的整形数字，其中第一个参数是要填充的数字，第二个参

 // 示例程序
```

```
QString str =  QString("%1 %2 %3").arg(1).arg(2);
str = str.arg("hello");
qDebug()<<str<<endl;     //"hello 2 1"

QString text = QString("%1:%2:%3").arg(1,2,10,QChar('0')).arg(35).arg(59);
qDebug()<<text<<endl;    //"01:35:59"
```

3.2 不同字符串類型相互轉換

```
// std::string -> QString
[static] QString QString::fromStdString(const std::string &str);
// QString -> std::string
std::string QString::toStdString() const;

#QString -> QByteArray
// 转换为本地编码，跟随操作系统
QByteArray QString::toLocal8Bit() const;
// 转换为 Latin-1 编码的字符串 不支持中文
QByteArray QString::toLatin1() const;
// 转换为 utf8 编码格式的字符串 (常用)
QByteArray QString::toUtf8() const;

#QByteArray -> QString
//使用QString的构造函数即可
```

“以前的我VS現在的我！”哈哈哈哈哈有被真實到！

喵大白話