Классификация методов класса `QString`

Конструкторы

- **QString** ()
- **QString** (const QChar * *unicode*, int *size*)
- **QString** (QChar *ch*)
- **QString** (int *size*, QChar *ch*)
- **QString** (const QLatin1String & *str*)
- **QString** (const QString & *other*)
- **QString** (const char * *str*)
- **QString** (const QByteArray & *ba*)

Деструктор

- **~QString** ()

Методы доступа

Анализ содержимого строки

- bool **isEmpty** () const
- bool **isNull** () const
- int **length** () const

Получение “сырых” данных

- const QChar * **constData** () const
- QChar * **data** ()
- const QChar * **data** () const
- DataPtr & **data_ptr** ()

Сравнение строк

- int **compare** (const QString & *other*) const
- int **compare** (const QString & *other*, Qt::CaseSensitivity *cs*) const
- int **compare** (const QLatin1String & *other*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const

- int **compare** (const QString & *s1*, const QString & *s2*, Qt::CaseSensitivity *cs*)
- int **compare** (const QString & *s1*, const QString & *s2*)
- int **compare** (const QString & *s1*, const QLatin1String & *s2*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)
- int **compare** (const QLatin1String & *s1*, const QString & *s2*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)

Поиск подстроки

- bool **contains** (const QString & *str*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **contains** (QChar *ch*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **contains** (const QRegExp & *rx*) const

- int **indexOf** (const QString & *str*, int *from* = 0, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **indexOf** (QChar *ch*, int *from* = 0, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **indexOf** (const QRegExp & *rx*, int *from* = 0) const

- int **lastIndexOf** (const QString & *str*, int *from* = -1, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **lastIndexOf** (QChar *ch*, int *from* = -1, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **lastIndexOf** (const QRegExp & *rx*, int *from* = -1) const

- bool **startsWith** (const QString & *s*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **startsWith** (const QLatin1String & *s*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **startsWith** (const QChar & *c*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **endsWith** (const QString & *s*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **endsWith** (const QLatin1String & *s*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- bool **endsWith** (const QChar & *c*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const

- int **count** (const QString & *str*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **count** (QChar *ch*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **count** (const QRegExp & *rx*) const
- int **count** () const

Модифицирующие методы

Расширение строки

- QString & **append** (const QString & *str*)
- QString & **append** (const QLatin1String & *str*)
- QString & **append** (const QByteArray & *ba*)
- QString & **append** (const char * *str*)
- QString & **append** (QChar *ch*)

- QString & **prepend** (const QString & *str*)
- QString & **prepend** (const QLatin1String & *str*)
- QString & **prepend** (const QByteArray & *ba*)
- QString & **prepend** (const char * *str*)
- QString & **prepend** (QChar *ch*)

- void **push_back** (const QString & *other*)
- void **push_back** (QChar *ch*)
- void **push_front** (const QString & *other*)
- void **push_front** (QChar *ch*)

Формирование/заполнение строки

- QString **arg** (const QString & *a*, int *fieldWidth* = 0, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (const QString & *a1*, const QString & *a2*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*, const QString & *a4*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*, const QString & *a4*, const QString & *a5*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*, const QString & *a4*, const QString & *a5*, const QString & *a6*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*, const QString & *a4*, const QString & *a5*, const QString & *a6*, const QString & *a7*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*, const QString & *a4*, const QString & *a5*, const QString & *a6*, const QString & *a7*, const QString & *a8*) const
- QString **arg** (const QString & *a1*, const QString & *a2*, const QString & *a3*, const QString & *a4*, const QString & *a5*, const QString & *a6*, const QString & *a7*, const QString & *a8*, const QString & *a9*) const
- QString **arg** (int *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (uint *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (long *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (ulong *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (qlonglong *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (qulonglong *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (short *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (ushort *a*, int *fieldWidth* = 0, int *base* = 10, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (QChar *a*, int *fieldWidth* = 0, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (char *a*, int *fieldWidth* = 0, const QChar & *fillChar* = QLatin1Char(' ')) const
- QString **arg** (double *a*, int *fieldWidth* = 0, char *format* = 'g', int *precision* = -1, const QChar & *fillChar* = QLatin1Char(' ')) const

- QString & **fill** (QChar *ch*, int *size* = -1)

- QString & **setNum** (int *n*, int *base* = 10)
- QString & **setNum** (uint *n*, int *base* = 10)
- QString & **setNum** (long *n*, int *base* = 10)
- QString & **setNum** (ulong *n*, int *base* = 10)
- QString & **setNum** (qlonglong *n*, int *base* = 10)
- QString & **setNum** (qulonglong *n*, int *base* = 10)
- QString & **setNum** (short *n*, int *base* = 10)
- QString & **setNum** (ushort *n*, int *base* = 10)
- QString & **setNum** (double *n*, char *format* = 'g', int *precision* = 6)
- QString & **setNum** (float *n*, char *format* = 'g', int *precision* = 6)

- QString & **sprintf** (const char * *cformat*, ...)

Вставка подстроки

- QString & **insert** (int *position*, const QString & *str*)
- QString & **insert** (int *position*, const QLatin1String & *str*)
- QString & **insert** (int *position*, const QChar * *unicode*, int *size*)
- QString & **insert** (int *position*, QChar *ch*)

Удаление подстроки, очистка строки

- void **chop** (int *n*)
- void **clear** ()

- QString & **remove** (int *position*, int *n*)
- QString & **remove** (const QString & *str*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)
- QString & **remove** (QChar *ch*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)
- QString & **remove** (const QRegExp & *rx*)

- void **truncate** (int *position*)

Замена подстроки

- QString & **replace** (int *position*, int *n*, const QString & *after*)
- QString & **replace** (int *position*, int *n*, const QChar * *unicode*, int *size*)
- QString & **replace** (int *position*, int *n*, QChar *after*)
- QString & **replace** (const QString & *before*, const QString & *after*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)
- QString & **replace** (QChar *ch*, const QString & *after*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)
- QString & **replace** (QChar *before*, QChar *after*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)
- QString & **replace** (const QRegExp & *rx*, const QString & *after*)

Порождающие методы

Получение подстроки/символа

- const QChar **at** (int *position*) const

- QString **left** (int *n*) const
- QString **leftJustified** (int *width*, QChar *fill* = QLatin1Char(' '), bool *truncate* = false) const
- QString **right** (int *n*) const
- QString **rightJustified** (int *width*, QChar *fill* = QLatin1Char(' '), bool *truncate* = false) const

- QString **mid** (int *position*, int *n* = -1) const

- QString **section** (QChar *sep*, int *start*, int *end* = -1, SectionFlags *flags* = SectionDefault) const
- QString **section** (const QString & *sep*, int *start*, int *end* = -1, SectionFlags *flags* = SectionDefault) const
- QString **section** (const QRegExp & *reg*, int *start*, int *end* = -1, SectionFlags *flags* = SectionDefault) const

- QStringList **split** (const QString & *sep*, SplitBehavior *behavior* = KeepEmptyParts, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- QStringList **split** (const QChar & *sep*, SplitBehavior *behavior* = KeepEmptyParts, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- QStringList **split** (const QRegExp & *rx*, SplitBehavior *behavior* = KeepEmptyParts) const

Преобразование строки к другому типу данных

- QByteArray **toAscii** () const
- QString **toCaseFolded** () const
- double **toDouble** (bool * *ok* = 0) const
- float **toFloat** (bool * *ok* = 0) const
- int **toInt** (bool * *ok* = 0, int *base* = 10) const
- QByteArray **toLatin1** () const
- QByteArray **toLocal8Bit** () const
- long **toLong** (bool * *ok* = 0, int *base* = 10) const
- qlonglong **toLongLong** (bool * *ok* = 0, int *base* = 10) const
- short **toShort** (bool * *ok* = 0, int *base* = 10) const
- std::string **toStdString** () const
- std::wstring **toStdWString** () const
- uint **toUInt** (bool * *ok* = 0, int *base* = 10) const
- ulong **toULong** (bool * *ok* = 0, int *base* = 10) const
- qulonglong **toULongLong** (bool * *ok* = 0, int *base* = 10) const
- ushort **toUShort** (bool * *ok* = 0, int *base* = 10) const
- QVector<uint> **toUcs4** () const
- QByteArray **toUtf8** () const
- int **toWCharArray** (wchar_t * *array*) const

Порождение строки на основе существующей

- QString **simplified** () const
- QString **trimmed** () const
- QString **toUpper** () const
- QString **toUpper** () const

Преобразование других типов данных в строку

- QString **fromAscii** (const char * *str*, int *size* = -1)
- QString **fromLatin1** (const char * *str*, int *size* = -1)
- QString **fromLocal8Bit** (const char * *str*, int *size* = -1)
- QString **fromRawData** (const QChar * *unicode*, int *size*)
- QString **fromStdString** (const std::string & *str*)
- QString **fromStdWString** (const std::wstring & *str*)
- QString **fromUcs4** (const uint * *unicode*, int *size* = -1)
- QString **fromUtf8** (const char * *str*, int *size* = -1)
- QString **fromUtf16** (const ushort * *unicode*, int *size* = -1)
- QString **fromWCharArray** (const wchar_t * *string*, int *size* = -1)

-
- QString **number** (long *n*, int *base* = 10)
- QString **number** (ulong *n*, int *base* = 10)
- QString **number** (int *n*, int *base* = 10)
- QString **number** (uint *n*, int *base* = 10)
- QString **number** (qlonglong *n*, int *base* = 10)
- QString **number** (qulonglong *n*, int *base* = 10)
- QString **number** (double *n*, char *format* = 'g', int *precision* = 6)

Операции

Операция присваивания

- QString & **operator=** (const QString & *other*)
- QString & **operator=** (const QLatin1String & *str*)
- QString & **operator=** (const QByteArray & *ba*)
- QString & **operator=** (const char * *str*)
- QString & **operator=** (char *ch*)
- QString & **operator=** (QChar *ch*)

Логические операции

- bool **operator!=** (const QString & *other*) const
- bool **operator!=** (const QLatin1String & *other*) const
- bool **operator!=** (const QByteArray & *other*) const
- bool **operator!=** (const char * *other*) const
- bool **operator<** (const QString & *other*) const
- bool **operator<** (const QLatin1String & *other*) const
- bool **operator<** (const QByteArray & *other*) const
- bool **operator<** (const char * *other*) const
- bool **operator<=** (const QString & *other*) const
- bool **operator<=** (const QLatin1String & *other*) const
- bool **operator<=** (const QByteArray & *other*) const
- bool **operator<=** (const char * *other*) const
- bool **operator==** (const QString & *other*) const
- bool **operator==** (const QLatin1String & *other*) const
- bool **operator==** (const QByteArray & *other*) const
- bool **operator==** (const char * *other*) const
- bool **operator>** (const QString & *other*) const
- bool **operator>** (const QLatin1String & *other*) const
- bool **operator>** (const QByteArray & *other*) const
- bool **operator>** (const char * *other*) const
- bool **operator>=** (const QString & *other*) const
- bool **operator>=** (const QLatin1String & *other*) const
- bool **operator>=** (const QByteArray & *other*) const
- bool **operator>=** (const char * *other*) const

- bool **operator!=** (const char * *s1*, const QString & *s2*)
- bool **operator<** (const char * *s1*, const QString & *s2*)
- bool **operator<=** (const char * *s1*, const QString & *s2*)
- bool **operator==** (const char * *s1*, const QString & *s2*)
- bool **operator>** (const char * *s1*, const QString & *s2*)
- bool **operator>=** (const char * *s1*, const QString & *s2*)

Расширение строки

- QString & **operator+=** (const QString & *other*)
- QString & **operator+=** (const QLatin1String & *str*)
- QString & **operator+=** (const QByteArray & *ba*)
- QString & **operator+=** (const char * *str*)
- QString & **operator+=** (char *ch*)
- QString & **operator+=** (QChar *ch*)

Обращение к символу строки

- QCharRef **operator[]** (int *position*)
- const QChar **operator[]** (int *position*) const
- QCharRef **operator[]** (uint *position*)
- const QChar **operator[]** (uint *position*) const

Склеивание строк

- const QString **operator+** (const QString & *s1*, const QString & *s2*)
- const QString **operator+** (const QString & *s1*, const char * *s2*)

Основные методы класса QString

Методы доступа

Длина строки

- int **length** () const

Поиск подстроки

- int **indexOf** (const QString & *str*, int *from* = 0, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- int **lastIndexOf** (const QString & *str*, int *from* = -1, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const

Модифицирующие методы

Вставка подстроки

- QString & **insert** (int *position*, const QString & *str*)

Удаление подстроки

- QString & **remove** (int *position*, int *n*)

Замена подстроки

- QString & **replace** (int *position*, int *n*, const QString & *after*)
- QString & **replace** (const QString & *before*, const QString & *after*, Qt::CaseSensitivity *cs* = Qt::CaseSensitive)

Порождающие методы

Получение части строки

- QString **mid** (int *position*, int *n* = -1) const

Разбиение строки на подстроки

- QStringList **split** (const QString & *sep*, SplitBehavior *behavior* = KeepEmptyParts, Qt::CaseSensitivity *cs* = Qt::CaseSensitive) const
- QStringList **split** (const QRegExp & *rx*, SplitBehavior *behavior* = KeepEmptyParts) const

Преобразование строки к массиву символов ASCII, заканчивающегося нуль-символом

- QByteArray **toAscii** () const

Операции

Операция присваивания

- `QString & operator= ( const QString & other )`
- `QString & operator= ( const char * str )`

Операция равенства

- `bool operator== ( const QString & other ) const`
- `bool operator== ( const char * other ) const`
- `bool operator== ( const char * s1, const QString & s2 )`

Операция расширения строки

- `QString & operator+= ( const QString & other )`
- `QString & operator+= ( const char * str )`

Операция обращения к символу строки

- `QCharRef operator[] ( int position )`

Операция склеивания строк

- `const QString operator+ ( const QString & s1, const QString & s2 )`
- `const QString operator+ ( const QString & s1, const char * s2 )`

Описание класса `QString`

Создание строки

Для создания пустой строки необходимо объявить переменную типа **`QString`**.

```
QString str; // пустая строка
```

Инициализация строки

Значение строки можно задать путем присвоения ей либо строковой константы, либо массива символов, заканчивающегося нуль-символом.

```
QString str = "invalid data";

char str_mass[]= "no solution";   // строка языка Си

str = str_mass;
```

Строку можно задать как результат склеивания нескольких строк.

```
QString str1= "invalid data";
QString str2= "!!!";
QString str3= " no solution";
QString str;

str = str1 + str2 + str3;     // строки  str1, str2, str3 не
                              // изменяются

QString str = QString() + "invalid data" + "!!!" +
" no solution";
```

Анализ содержимого строки

Для определения того факта, что строка заполнена используется метод **`length`**`()`, который возвращает длину строки.

```
QString str; // пустая строка
printf("String length=%d", str.length());// результат 0

str = "invalid data";
printf("String length=%d", str.length());// результат 12
```

Проверить содержимое всей строки можно через операцию сравнения.

```
QString str = "invalid data";
bool falg;

if(str == "no solution")      flag= true;
else                          flag= false;
```

Возможно посимвольное обращение к строке.

```
// Печать первого символа строки
printf("First symbol=%c", str[0].toAscii());
```

Однако можно работать и с подстроками. Для поиска подстрок используются методы **indexOf**() и **lastIndexOf**().

```
QString x = "sticky question";
QString y = "sti";

x.indexOf(y);                 // returns 0
x.indexOf(y, 1);              // returns 10
x.indexOf(y, 10);             // returns 10
x.indexOf(y, 11);             // returns -1
```

```
QString x = "crazy azimuths";
QString y = "az";

x.lastIndexOf(y);             // returns 6
x.lastIndexOf(y, 6);          // returns 6
x.lastIndexOf(y, 5);          // returns 2
x.lastIndexOf(y, 1);          // returns -1
```

Для получения части строки в заданной позиции используется метод **mid**().

```
QString x = "Nine pineapples";
QString y = x.mid(5, 4);            // y == "pine"
QString z = x.mid(5);               // z == "pineapples"
```

Для выделения элементов, составляющих строку, используется метод **split**().

```
QString str = "a,,b,c";
QStringList list1 = str.split(",");
// list1: [ "a", "", "b", "c" ]

QStringList list2 = str.split(",", QString::SkipEmptyParts);
// list2: [ "a", "b", "c" ]
```

```
QString str = "a,..,b,c";
QStringList list1 = str.split(",");
// list1: [ "a", "..", "b", "c" ]

QStringList list2 = str.split(QRegExp("[,.]"),
                                        QString::SkipEmptyParts);
// list2: [ "a", "b", "c" ]
```

Модификация строки

Строку можно модифицировать посимвольно или целыми подстроками.

Для посимавольной модификации строки используется операция [].

```
QString str = "abcdef";

for(int i=0; i<str.length(); i++)
{
    str[i]= '.';
}
// str = "....."
```

Используя методы **insert**(), **remove**(), и **replace**() можно выполнить вставку, удаление и замену подстроки.

```
QString str = "Meal";
str.insert(1, QString("ontr"));
// str == "Montreal"
```

```
QString s = "Montreal";
s.remove(1, 4);
// s == "Meal"
```

```
QString x = "Say yes!";
QString y = "no";
x.replace(4, 3, y);
// x == "Say no!"

QString str = "colour behaviour flavour neighbour";
str.replace(QString("ou"), QString("o"));
// str == "color behavior flavor neighbor"
```

Преобразование строки

Для использования стандартных функций ввода-вывода языка Си строку типа `QString` необходимо преобразовать к массиву символов, заканчивающемуся нуль-символом. Такое преобразование можно выполнить с использованием метода **`toAscii`**`()`.

```
QString str = "abcdef";

printf(str.toAscii().data());

// другой вариант использовать макрос qPrintable()
printf(qPrintable(str));
```

Основные методы класса `QStringList`

Склеивание строк

- `QString join ( const QString & separator ) const`

Добавление строки

- `void append ( const T & value )`

Вставка строки

- `void insert ( int i, const T & value )`

Перемещение строки

- `void move ( int from, int to )`

Удаление строки

- `void removeAt ( int i )`

Кол-во строк

- `int count () const`

Обращение к строке

- `T & operator[] ( int i )`