QRegularExpression Class

QRegularExpression 类使用正则表达式提供模式匹配。更多的...

Header:	**#include < QRegularExpression>**
CMake:	find_package(Qt6 REQUIRED COMPONENTS Core) target_link_libraries(mytarget PRIVATE Qt6::Core)
qmake:	QT += core

- 所有成员的列表，包括继承的成员
- 已弃用的成员
- QRegularExpression 是隐式共享类的一部分。

注意：该类中的所有函数都是reentrant。

公共类型

enum	**MatchOption { NoMatchOption, AnchoredMatchOption, AnchorAtOffsetMatchOption, DontCheckSubjectStringMatchOption }**
flags	**MatchOptions**
enum	**MatchType** { NormalMatch, PartialPreferCompleteMatch, PartialPreferFirstMatch, NoMatch }
enum	**PatternOption** { NoPatternOption, CaseInsensitiveOption, DotMatchesEverythingOption, MultilineOption, ExtendedPatternSyntaxOption, …, UseUnicodePropertiesOption }
flags	**PatternOptions**
enum	**WildcardConversionOption** { DefaultWildcardConversion, UnanchoredWildcardConversion }
flags	**WildcardConversionOptions**

公共方法

	QRegularExpression()
	QRegularExpression(const QString &*pattern*, QRegularExpression::PatternOptions *options* = NoPatternOption)
	QRegularExpression(const QRegularExpression &*re*)

	QRegularExpression()
	QRegularExpression(QRegularExpression &&*re*)
	~QRegularExpression()
int	**captureCount**() const
QString	**errorString**() const
QRegularExpressionMatchIterator	**globalMatch**(const QString &*subject*, qsizetype *offset* = 0, QRegularExpression::MatchType *matchType* = NormalMatch, QRegularExpression::MatchOptions *matchOptions* = NoMatchOption) const
QRegularExpressionMatchIterator	**globalMatchView**(QStringView *subjectView*, qsizetype *offset* = 0, QRegularExpression::MatchType *matchType* = NormalMatch, QRegularExpression::MatchOptions *matchOptions* = NoMatchOption) const
bool	**isValid**() const
QRegularExpressionMatch	**match**(const QString &*subject*, qsizetype *offset* = 0, QRegularExpression::MatchType *matchType* = NormalMatch, QRegularExpression::MatchOptions *matchOptions* = NoMatchOption) const
QRegularExpressionMatch	**matchView**(QStringView *subjectView*, qsizetype *offset* = 0, QRegularExpression::MatchType *matchType* = NormalMatch, QRegularExpression::MatchOptions *matchOptions* = NoMatchOption) const
QStringList	**namedCaptureGroups**() const
void	**optimize**() const
QString	**pattern**() const
qsizetype	**patternErrorOffset**() const
QRegularExpression::PatternOptions	**patternOptions**() const
void	**setPattern**(const QString &*pattern*)
void	**setPatternOptions**(QRegularExpression::PatternOptions *options*)
void	**swap**(QRegularExpression &*other*)
bool	**operator!=**(const QRegularExpression &*re*) const
QRegularExpression &	**operator=**(const QRegularExpression &*re*)
QRegularExpression &	**operator=**(QRegularExpression &&*re*)
bool	**operator==**(const QRegularExpression &*re*) const

静态公共成员

QString	**anchoredPattern**(**QStringView *expression***)
QString	**anchoredPattern**(const QString &*expression*)
QString	**escape**(QStringView *str*)

QString	**anchoredPattern(QStringView *expression*)**
QString	escape(const QString &*str*)
QRegularExpression	fromWildcard(QStringView *pattern*, Qt::CaseSensitivity *cs* = Qt::CaseInsensitive, QRegularExpression::WildcardConversionOptions *options* = DefaultWildcardConversion)
QString	wildcardToRegularExpression(QStringView *pattern*, QRegularExpression::WildcardConversionOptions *options* = DefaultWildcardConversion)
QString	wildcardToRegularExpression(const QString &*pattern*, QRegularExpression::WildcardConversionOptions *options* = DefaultWildcardConversion)

相关非成员

size_t	**qHash(const QRegularExpression &*key*, size_t *seed* = 0)**
QDataStream &	operator<<(QDataStream &*out*, const QRegularExpression &*re*)
QDebug	operator<<(QDebug *debug*, const QRegularExpression &*re*)
QDebug	operator<<(QDebug *debug*, QRegularExpression::PatternOptions *patternOptions*)
QDataStream &	operator>>(QDataStream &*in*, QRegularExpression &*re*)

详细说明

正则表达式或*regexps*是处理字符串和文本的非常强大的工具。这在许多情况下都很有用，例如

验证	**正则表达式可以测试子字符串是否满足某些条件，例如是整数或不包含空格。**
搜寻中	*正则表达式提供比简单子字符串匹配更强大的模式匹配，例如，匹配单词mail、letter或communications之一，但不匹配单词email、mailman、mailer、letterbox等。*
搜索和替换	正则表达式可以用不同的子字符串替换所有出现的子字符串，例如，将所有出现的&替换为& 除非&后面已经跟有*amp；*。
字符串分割	正则表达式可用于标识应在何处拆分字符串，例如拆分制表符分隔的字符串。

本文档绝不是使用正则表达式进行模式匹配的完整参考，以下部分将要求读者对类 Perl 正则表达式及其模式语法有一些基本知识。

关于正则表达式的好参考包括：

- *掌握正则表达式*（第三版）作者：Jeffrey EF Friedl，ISBN 0-596-52812-4；
- 这pcrepattern(3)手册页，描述 PCRE 支持的模式语法（Perl 兼容正则表达式的参考实现）；
- 这Perl's regular expression documentation和Perl's regular expression tutorial。

介绍

QRegularExpression 实现与 Perl 兼容的正则表达式。它完全支持 Unicode。有关 QRegularExpression 支持的正则表达式语法的概述，请参阅前面提到的 pcrepattern(3) 手册页。正则表达式由两部分组成：**模式字符串**和一组更改模式字符串含义的**模式选项**。

您可以通过将字符串传递给 QRegularExpression 构造函数来设置模式字符串：

```
QRegularExpression re("a pattern");
```

这将模式字符串设置为`a pattern`。您还可以使用setPattern() 函数在现有的 QRegularExpression 对象上设置模式：

```
QRegularExpression re;
re.setPattern("another pattern");
```

请注意，由于 C++ 文字字符串规则，您必须使用另一个反斜杠转义模式字符串内的所有反斜杠：

```
// matches two digits followed by a space and a word
QRegularExpression re("\\d\\d \\w+");

// matches a backslash
QRegularExpression re2("\\\\");
```

或者，您可以使用raw string literal，在这种情况下，您不需要转义模式中的反斜杠，其间的所有字符都`R"(...)"`被视为原始字符。正如您在以下示例中看到的，这简化了编写模式：

```
// matches two digits followed by a space and a word
QRegularExpression re(R"(\d\d \w+)");
```

这pattern() 函数返回当前为 QRegularExpression 对象设置的模式：

```
QRegularExpression re("a third pattern");
QString pattern = re.pattern(); // pattern == "a third pattern"
```

图案选项

*可以通过设置一个或多个模式选项*来修改模式字符串的含义。例如，可以通过设置模式来设置不区分大小写的模式QRegularExpression::CaseInsensitiveOption。

您可以通过将选项传递给 QRegularExpression 构造函数来设置选项，如下所示：

```
// matches "Qt rocks", but also "QT rocks", "QT ROCKS", "qT rOcKs", etc.
QRegularExpression re("Qt rocks", QRegularExpression::CaseInsensitiveOption);
```

或者，您可以使用setPatternOptions() 函数作用于现有的 QRegularExpressionObject：

```
QRegularExpression re("^\\d+$");
re.setPatternOptions(QRegularExpression::MultilineOption);
// re matches any line in the subject string that contains only digits (but at least one)
```

可以使用以下方法获取当前在 QRegularExpression 对象上设置的模式选项patternOptions（） 功能：

```
QRegularExpression re = QRegularExpression("^two.*words$", QRegularExpression::MultilineOption
                                                           |
QRegularExpression::DotMatchesEverythingOption);

QRegularExpression::PatternOptions options = re.patternOptions();
// options == QRegularExpression::MultilineOption |
QRegularExpression::DotMatchesEverythingOption
```

请参阅QRegularExpression::PatternOption有关每个模式选项的更多信息的枚举文档。

匹配类型和匹配选项

最后两个参数match() 和globalMatch() 函数设置匹配类型和匹配选项。匹配类型是一个值QRegularExpression::MatchType枚举；“传统”匹配算法是通过使用来选择的NormalMatch匹配类型（默认）。还可以启用正则表达式与主题字符串的部分匹配：请参阅partial matching部分了解更多详细信息。

匹配选项是一组一个或多个QRegularExpression::MatchOption价值观。它们改变了正则表达式与主题字符串的特定匹配的方式。请参阅QRegularExpression::MatchOption枚举文档以获取更多详细信息。

正常匹配

为了执行匹配，您只需调用match() 函数传递一个字符串进行匹配。我们将此字符串称为*主题字符串*。结果match() 函数是一个QRegularExpressionMatch可用于检查比赛结果的对象。例如：

```
// match two digits followed by a space and a word
QRegularExpression re("\\d\\d \\w+");
QRegularExpressionMatch match = re.match("abc123 def");
bool hasMatch = match.hasMatch(); // true
```

如果匹配成功，则可以使用（隐式）捕获组编号 0 来检索与整个模式匹配的子字符串（另请参阅有关extracting captured substrings）：

```
QRegularExpression re("\\d\\d \\w+");
QRegularExpressionMatch match = re.match("abc123 def");
if (match.hasMatch()) {
    QString matched = match.captured(0); // matched == "23 def"
    // ...
}
```

还可以通过将偏移量作为参数传递来在主题字符串内的任意偏移量处开始匹配match （） 功能。在以下示例中"12 abc"不匹配，因为匹配从偏移量 1 开始：

```
QRegularExpression re("\\d\\d \\w+");
QRegularExpressionMatch match = re.match("12 abc 45 def", 1);
if (match.hasMatch()) {
    QString matched = match.captured(0); // matched == "45 def"
    // ...
}
```

提取捕获的子字符串

这QRegularExpressionMatch对象还包含有关模式字符串中捕获组捕获的子字符串的信息。这captured() 函数将返回第 n 个捕获组捕获的字符串：

```
QRegularExpression re("^(\\d\\d)/(\\d\\d)/(\\d\\d\\d\\d)$");
QRegularExpressionMatch match = re.match("08/12/1985");
if (match.hasMatch()) {
    QString day = match.captured(1); // day == "08"
    QString month = match.captured(2); // month == "12"
    QString year = match.captured(3); // year == "1985"
    // ...
}
```

模式中的捕获组从 1 开始编号，隐式捕获组 0 用于捕获与整个模式匹配的子字符串。

还可以使用以下方法检索每个捕获的子字符串的起始和结束偏移量（在主题字符串内） capturedStart() 和 capturedEnd （） 功能：

```
QRegularExpression re("abc(\\d+)def");
QRegularExpressionMatch match = re.match("XYZabc123defXYZ");
if (match.hasMatch()) {
    int startOffset = match.capturedStart(1); // startOffset == 6
    int endOffset = match.capturedEnd(1); // endOffset == 9
    // ...
}
```

所有这些函数都有一个重载QString作为参数以提取*命名的*捕获子字符串。例如：

```
QRegularExpression re("^(?<date>\\d\\d)/(?<month>\\d\\d)/(?<year>\\d\\d\\d\\d)$");
QRegularExpressionMatch match = re.match("08/12/1985");
if (match.hasMatch()) {
    QString date = match.captured("date"); // date == "08"
    QString month = match.captured("month"); // month == "12"
    QString year = match.captured("year"); // year == 1985
}
```

全球匹配

*全局匹配*对于查找主题字符串中给定正则表达式的所有出现非常有用。假设我们要从给定字符串中提取所有单词，其中单词是与模式匹配的子字符串`\w+`。

QRegularExpression::globalMatch返回一个QRegularExpressionMatchIterator，这是一个类似 Java 的前向迭代器，可用于迭代结果。例如：

```
QRegularExpression re("(\\w+)");
QRegularExpressionMatchIterator i = re.globalMatch("the quick fox");
```

由于它是一个类似 Java 的迭代器，QRegularExpressionMatchIterator将指向第一个结果之前。每个结果都作为QRegularExpressionMatch目的。这hasNext如果至少还有一个结果，() 函数将返回 true，并且next() 将返回下一个结果并推进迭代器。继续前面的示例：

```
QStringList words;
while (i.hasNext()) {
    QRegularExpressionMatch match = i.next();
    QString word = match.captured(1);
    words << word;
}
// words contains "the", "quick", "fox"
```

您还可以使用peekNext() 获得下一个结果而不推进迭代器。

也可以简单地使用结果QRegularExpression::globalMatch在基于范围的 for 循环中，例如这样：

```
// using a raw string literal, R"(raw_characters)", to be able to use "\w"
// without having to escape the backslash as "\\w"
QRegularExpression re(R"(\w+)");
QString subject("the quick fox");
for (const QRegularExpressionMatch &match : re.globalMatch(subject)) {
    // ...
}
```

可以将起始偏移量和一个或多个匹配选项传递给globalMatch() 函数，与普通匹配完全相同match()。

部分匹配

当到达主题字符串末尾时获得部分匹配，但需要更多字符才能成功完成匹配。请注意，部分匹配通常比正常匹配效率低得多，因为无法采用匹配算法的许多优化。

必须通过指定匹配类型来显式请求部分匹配PartialPreferCompleteMatch或者PartialPreferFirstMatch打电话时QRegularExpression::match或者QRegularExpression::globalMatch。如果找到部分匹配，则调用hasMatch() 函数上QRegularExpressionMatch返回的对象match() 将返回`false`，但是hasPartialMatch() 将返回`true`。

当找到部分匹配时，不会返回捕获的子字符串，并且与整个匹配对应的（隐式）捕获组 0 捕获主题字符串的部分匹配子字符串。

请注意，如果找到了部分匹配，则请求部分匹配仍然可以导致完全匹配；在这种情况下，hasMatch() 将返回true并且hasPartialMatch() false。从来没有发生过这样的情况：QRegularExpressionMatch报告部分匹配和完整匹配。

部分匹配主要用于两种场景：实时验证用户输入和增量/多段匹配。

验证用户输入

假设我们希望用户以特定格式输入日期，例如“MMM dd, yyyy”。我们可以使用如下模式检查输入有效性：

```
^(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec) \d\d?, \d\d\d\d$
```

（此模式不会捕获无效日期，但为了示例的目的，我们保留它）。

我们希望在用户键入时使用*此正则表达式验证输入，以便我们可以在提交后立即报告输入错误（例如，用户键入了错误的键）*。为此，我们必须区分三种情况：

- 输入不可能与正则表达式匹配；
- 输入与正则表达式匹配；
- 现在输入与正则表达式不匹配，但如果添加更多字符，就会匹配。

请注意，这三种情况准确地代表了 a 的可能状态QValidator（参见QValidator::State枚举）。

特别是，在最后一种情况下，我们希望正则表达式引擎报告部分匹配：我们成功地将模式与主题字符串匹配，但匹配无法继续，因为遇到了主题结尾。但请注意，匹配算法应继续并尝试所有可能性，如果找到完整（非部分）匹配，则应报告此匹配，并将输入字符串视为完全有效。

此行为是由PartialPreferCompleteMatch比赛类型。例如：

```
QString pattern("^(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec) \\d\\d?, \\d\\d\\d\\d$");
QRegularExpression re(pattern);

QString input("Jan 21,");
QRegularExpressionMatch match = re.match(input, 0,
QRegularExpression::PartialPreferCompleteMatch);
bool hasMatch = match.hasMatch(); // false
bool hasPartialMatch = match.hasPartialMatch(); // true
```

如果将相同的正则表达式与主题字符串匹配导致完全匹配，则照常报告：

```
QString input("Dec 8, 1985");
QRegularExpressionMatch match = re.match(input, 0,
QRegularExpression::PartialPreferCompleteMatch);
bool hasMatch = match.hasMatch(); // true
bool hasPartialMatch = match.hasPartialMatch(); // false
```

另一个具有不同模式的示例，显示了更喜欢完全匹配而不是部分匹配的行为：

```
QRegularExpression re("abc\\w+X|def");
QRegularExpressionMatch match = re.match("abcdef", 0,
QRegularExpression::PartialPreferCompleteMatch);
bool hasMatch = match.hasMatch(); // true
bool hasPartialMatch = match.hasPartialMatch(); // false
QString captured = match.captured(0); // captured == "def"
```

在这种情况下，子模式abc\\w+X部分匹配主题字符串；但是，子模式def与主题字符串完全匹配，因此报告完全匹配。

如果匹配时找到多个部分匹配（但没有完全匹配），则QRegularExpressionMatch对象将报告找到的第一个对象。例如：

```
QRegularExpression re("abc\\w+X|defY");
QRegularExpressionMatch match = re.match("abcdef", 0,
QRegularExpression::PartialPreferCompleteMatch);
bool hasMatch = match.hasMatch(); // false
bool hasPartialMatch = match.hasPartialMatch(); // true
QString captured = match.captured(0); // captured == "abcdef"
```

增量/多段匹配

增量匹配是部分匹配的另一个用例。假设我们要查找大文本中正则表达式的出现次数（即与正则表达式匹配的子字符串）。为此，我们希望将大文本以较小的块“馈送到”正则表达式引擎。明显的问题是，如果与正则表达式匹配的子字符串跨越两个或多个块，会发生什么情况。

在这种情况下，正则表达式引擎应该报告部分匹配，以便我们可以添加新数据再次匹配并（最终）获得完整匹配。这意味着正则表达式引擎可能会假设主题字符串*末尾之外*还有其他字符。这不能从字面上理解——引擎永远不会尝试访问主题中最后一个字符之后的任何字符。

QRegularExpression 在使用时实现此行为PartialPreferFirstMatch比赛类型。此匹配类型一发现就报告部分匹配，并且不会尝试其他匹配替代方案（即使它们可能导致完全匹配）。例如：

```
QRegularExpression re("abc|ab");
QRegularExpressionMatch match = re.match("ab", 0, QRegularExpression::PartialPreferFirstMatch);
bool hasMatch = match.hasMatch(); // false
bool hasPartialMatch = match.hasPartialMatch(); // true
```

发生这种情况是因为当匹配交替运算符的第一个分支时，发现了部分匹配，因此匹配停止，而不尝试第二个分支。另一个例子：

```
QRegularExpression re("abc(def)?");
QRegularExpressionMatch match = re.match("abc", 0, QRegularExpression::PartialPreferFirstMatch);
bool hasMatch = match.hasMatch(); // false
bool hasPartialMatch = match.hasPartialMatch(); // true
```

这显示了量词看似违反直觉的行为：因为?是贪婪的，所以引擎在匹配 后首先尝试继续匹配"abc"；但随后匹配到达主题字符串的末尾，因此报告部分匹配。在下面的例子中，这一点更令人惊讶：

```
QRegularExpression re("(abc)*");
QRegularExpressionMatch match = re.match("abc", 0, QRegularExpression::PartialPreferFirstMatch);
bool hasMatch = match.hasMatch(); // false
bool hasPartialMatch = match.hasPartialMatch(); // true
```

如果我们记得引擎期望主题字符串只是我们正在寻找匹配的整个文本的子字符串（也就是说，我们之前说过，引擎假设还有其他匹配项），那么就很容易理解这种行为超出主题字符串末尾的字符）。

由于*量词是贪婪的，因此报告完全匹配可能是一个错误，因为在当前主题之后"abc"可能会出现其他"abc". 例如，完整的文本可能是“abcabcX”，因此报告的*正确*匹配（在完整的文本中）将是"abcabc"；通过仅匹配前导，"abc"我们会得到部分匹配。

错误处理

由于模式字符串中的语法错误，QRegularExpression 对象可能无效。这isValid如果正则表达式有效，() 函数将返回 true，否则返回 false：

```
QRegularExpression invalidRe("(unmatched|parenthesis");
bool isValid = invalidRe.isValid(); // false
```

您可以通过调用以下命令来获取有关特定错误的更多信息errorString （） 功能; 此外，patternErrorOffset() 函数将返回模式字符串内的偏移量

```
QRegularExpression invalidRe("(unmatched|parenthesis");
if (!invalidRe.isValid()) {
    QString errorString = invalidRe.errorString(); // errorString == "missing )"
    int errorOffset = invalidRe.patternErrorOffset(); // errorOffset == 22
    // ...
}
```

如果尝试使用无效的 QRegularExpression 进行匹配，则返回QRegularExpressionMatch对象也将是无效的（也就是说，它的isValid() 函数将返回 false）。这同样适用于尝试全局匹配。

不支持的 Perl 兼容正则表达式功能

QRegularExpression 不支持与 Perl 兼容的正则表达式中可用的所有功能。最值得注意的是，不支持捕获组的重复名称，并且使用它们可能会导致未定义的行为。

这可能会在 Qt 的未来版本中发生变化。

调试使用 QRegularExpression 的代码

QRegularExpression 内部使用即时编译器 (JIT) 来优化匹配算法的执行。JIT 大量使用自修改代码，这可能会导致 Valgrind 等调试工具崩溃。如果要使用 QRegularExpression（例如，Valgrind 的--smc-check命令行选项）调试程序，则必须启用对自修改代码的所有检查。启用此类检查的缺点是您的程序运行速度会相当慢。

为了避免这种情况，如果您在调试模式下编译 Qt，则默认情况下禁用 JIT。QT_ENABLE_REGEXP_JIT通过分别将环境变量设置为非零或零值，可以覆盖默认值并启用或禁用 JIT 使用（在调试或发布模式下）。

也可以看看QRegularExpressionMatch和QRegularExpressionMatchIterator。

成员类型文档

枚举 QRegularExpression:: MatchOption 标志 QRegularExpression:: MatchOptions

持续的	价值	描述
QRegularExpression::NoMatchOption	0x0000	未设置匹配选项。
QRegularExpression::AnchoredMatchOption	AnchorAtOffsetMatchOption	请改用 AnchorAtOffsetMatchOption。
QRegularExpression::AnchorAtOffsetMatchOption	0x0001	比赛被限制在传递给的偏移量处精确开始match() 才能成功，即使模式字符串不包含任何在该点锚定匹配的元字符。请注意，传递此选项不会将匹配的结尾锚定到主题的结尾；如果您想完全锚定正则表达式，请使用anchoredPattern()。该枚举值已在 Qt 6.0 中引入。
QRegularExpression::DontCheckSubjectStringMatchOption	0x0002	在尝试匹配之前，不会检查主题字符串的 UTF-16 有效性。使用此选项时要格外小心，因为尝试匹配无效字符串可能会导致程序崩溃和/或构成安全问题。该枚举值已在 Qt 5.4 中引入。

MatchOptions 类型是QFlags 的类型定义。它存储 MatchOption 值的 OR 组合。

enum QRegularExpression::MatchType

MatchType 枚举定义应尝试针对主题字符串的匹配类型。

持续的	价值	描述
QRegularExpression::NormalMatch	0	一场正常的比赛就完成了。
QRegularExpression::PartialPreferCompleteMatch	1	模式字符串与主题字符串部分匹配。如果找到部分匹配，则会将其记录下来，并像往常一样尝试其他匹配的替代方案。如果随后找到完整匹配，则优先于部分匹配；在这种情况下，仅报告完整的匹配。相反，如果未找到完整匹配项（仅找到部分匹配项），则报告部分匹配项。
QRegularExpression::PartialPreferFirstMatch	2	模式字符串与主题字符串部分匹配。如果发现部分匹配，则匹配停止并报告部分匹配。在这种情况下，不会尝试其他匹配替代方案（可能导致完全匹配）。此外，此匹配类型假设主题字符串只是较大文本的子字符串，并且（在该文本中）在主题字符串末尾之外还有其他字符。这可能会带来令人惊讶的结果；请参阅中的讨论partial matching部分了解更多详细信息。
QRegularExpression::NoMatch	3	没有进行匹配。该值作为默认构造的匹配类型返回QRegularExpressionMatch或者QRegularExpressionMatchIterator。使用此匹配类型对于用户来说不是很有用，因为不会发生任何匹配。该枚举值已在 Qt 5.1 中引入。

枚举 QRegularExpression:: PatternOption 标志 QRegularExpression:: PatternOptions

PatternOption 枚举定义了模式字符串解释方式的修饰符，以及模式与主题字符串匹配的方式。

持续的	价值	描述
QRegularExpression::NoPatternOption	0x0000	未设置模式选项。

持续的	价值	描述
QRegularExpression::CaseInsensitiveOption	0x0001	该模式应以不区分大小写的方式与主题字符串匹配。该选项对应于 Perl 正则表达式中的 /i 修饰符。
QRegularExpression::DotMatchesEverythingOption	0x0002	模式字符串中的点元字符 (.) 允许匹配主题字符串中的任何字符，包括换行符（通常，点不匹配换行符）。该选项对应于/sPerl 正则表达式中的修饰符。
QRegularExpression::MultilineOption	0x0004	模式字符串中的脱字号 (^) 和美元 ( ) 元$字符分别允许在主题字符串中的任何换行符之后和之前以及主题字符串的开头和结尾处进行匹配。该选项对应于/mPerl 正则表达式中的修饰符。
QRegularExpression::ExtendedPatternSyntaxOption	0x0008	模式字符串中任何未转义且位于字符类之外的空格都将被忽略。此外，字符类外部的未转义的尖号 (#) 会导致所有后续字符（直到第一个换行符（包括））被忽略。这可用于增加模式字符串的可读性以及将注释放入正则表达式中；如果模式字符串是从文件加载或由用户编写的，则这特别有用，因为在 C++ 代码中，始终可以使用字符串文字的规则将注释放在模式字符串之外。该选项对应于/xPerl 正则表达式中的修饰符。
QRegularExpression::InvertedGreedinessOption	0x0010	量词的贪婪性被反转：*、+、?、{m,n}等变得懒惰，而它们的懒惰版本（*?、+?、??、{m,n}?等）变得贪婪。Perl 正则表达式中没有与此选项等效的选项。
QRegularExpression::DontCaptureOption	0x0020	非命名捕获组不捕获子字符串；命名捕获组仍然按预期工作，以及对应于整个匹配的隐式捕获组编号 0。Perl 正则表达式中没有与此选项等效的选项。
QRegularExpression::UseUnicodePropertiesOption	0x0040	\w、等字符类的含义\d及其对应字符（\W、\D等）的含义从仅匹配 ASCII 字符更改为匹配具有相应 Unicode 属性的任何字符。例如，\d更改为匹配具有 Unicode Nd（十进制数字）属性的任何字符；\w匹配具有 Unicode L（字母）或 N（数字）属性的任何字符，加上下划线等。该选项对应于/uPerl 正则表达式中的修饰符。

PatternOptions 类型是QFlags 的类型定义。它存储 PatternOption 值的 OR 组合。

[since 6.0]枚举 QRegularExpression:: WildcardConversionOption 标志 QRegularExpression:: WildcardConversionOptions

WildcardConversionOption 枚举定义了通配符 glob 模式转换为正则表达式模式的方式的修饰符。

持续的	价值	描述
`QRegularExpression::DefaultWildcardConversion`	`0x0`	未设置转换选项。
`QRegularExpression::UnanchoredWildcardConversion`	`0x1`	转换不会锚定该模式。这允许通配符表达式的部分字符串匹配。

该枚举是在 Qt 6.0 中引入或修改的。

WildcardConversionOptions 类型是QFlags 的类型定义。它存储 WildcardConversionOption 值的 OR 组合。

成员函数文档

QRegularExpression::QRegularExpression()

构造一个带有空模式且没有模式选项的 QRegularExpression 对象。

也可以看看setPattern （） 和setPatternOptions()。

[explicit]QRegularExpression::QRegularExpression(const QString &pattern, QRegularExpression::PatternOptions options = NoPatternOption)

使用给定构造一个 QRegularExpression 对象*pattern*作为模式和*options*作为模式选项。

也可以看看setPattern （） 和setPatternOptions()。

QRegularExpression::QRegularExpression(const QRegularExpression &re)

构造一个 QRegularExpression 对象作为以下对象的副本*re*。

也可以看看operator=()。

[since 6.1]QRegularExpression::QRegularExpression(QRegularExpression &&re)

通过从以下位置移动来构造 QRegularExpression 对象*re*。

请注意，移出的 QRegularExpression 只能被销毁或分配给。调用除析构函数或赋值运算符之一之外的其他函数的效果是未定义的。

该功能是在 Qt 6.1 中引入的。

也可以看看operator=()。

QRegularExpression::~QRegularExpression()

摧毁了QRegularExpression目的。

[static]QString QRegularExpression::anchoredPattern(QStringView expression)

返回*expression*包裹在\A和\z锚点之间，用于精确匹配。

[static]QString QRegularExpression::anchoredPattern(const QString &expression)

这是一个过载功能。

int QRegularExpression::captureCount() const

返回模式字符串内捕获组的数量，如果正则表达式无效，则返回 -1。

注意：隐式捕获组 0不包含在返回的数字中。

也可以看看isValid()。

QString QRegularExpression::errorString() const

返回检查正则表达式有效性时发现的错误的文本描述，如果未发现错误，则返回“无错误”。

也可以看看isValid（） 和patternErrorOffset()。

[static]QString QRegularExpression::escape(QStringView str)

转义所有字符*str*以便它们在用作正则表达式模式字符串时不再具有任何特殊含义，并返回转义字符串。例如：

```
QString escaped = QRegularExpression::escape("a(x) = f(x) + g(x)");
// escaped == "a\\(x\\)\\ \\=\\ f\\(x\\)\\ \\+\\ g\\(x\\)"
```

这对于从任意字符串构建模式非常方便：

```
QString pattern = "(" + QRegularExpression::escape(name) +
                  "|" + QRegularExpression::escape(nickname) + ")";
QRegularExpression re(pattern);
```

注意：该函数实现了 Perl 的 quotemeta 算法，并使用反斜杠转义所有字符*str*`[A-Z]`、 、`[a-z]`和范围中的字符`[0-9]`以及下划线 (_) 字符除外。与 Perl 的唯一区别是里面有一个文字 NUL*str*使用序列`"\\0"`(反斜杠 + `'0'`) 而不是`"\\\0"`(反斜杠 + `NUL`) 进行转义。

[static]QString QRegularExpression::escape(const QString &str)

这是一个过载功能。

[static, since 6.0]QRegularExpression QRegularExpression::fromWildcard(QStringView pattern, Qt::CaseSensitivity cs = Qt::CaseInsensitive, QRegularExpression::WildcardConversionOptions options = DefaultWildcardConversion)

返回 glob 模式的正则表达式*pattern*。正则表达式将区分大小写，如果*cs*是Qt::CaseSensitive，并根据转换*options*。

相当于

```
auto reOptions = cs == Qt::CaseSensitive ? QRegularExpression::NoPatternOption :
                                           QRegularExpression::CaseInsensitiveOption;
return QRegularExpression(wildcardToRegularExpression(str, options), reOptions);
```

这个函数是在Qt 6.0中引入的。

QRegularExpressionMatchIterator QRegularExpression::globalMatch(const QString &subject, qsizetype offset = 0, QRegularExpression::MatchType matchType = NormalMatch, QRegularExpression::MatchOptions matchOptions = NoMatchOption) const

尝试根据给定的正则表达式执行全局匹配*subject*字符串，从位置开始*offset*在主题内部，使用类型匹配*matchType*并尊重所给予的*matchOptions*。

返回的QRegularExpressionMatchIterator位于第一个匹配结果（如果有）之前。

也可以看看QRegularExpressionMatchIterator和global matching。

[since 6.5]QRegularExpressionMatchIterator QRegularExpression::globalMatchView(QStringView subjectView, qsizetype offset = 0, QRegularExpression::MatchType matchType = NormalMatch, QRegularExpression::MatchOptions matchOptions = NoMatchOption) const

这是一个过载功能。

尝试根据给定的正则表达式执行全局匹配*subjectView*字符串视图，从该位置开始*offset*在主题内部，使用类型匹配*matchType*并尊重所给予的*matchOptions*。

返回的QRegularExpressionMatchIterator位于第一个匹配结果（如果有）之前。

注：数据参考*subjectView*只要存在就必须保持有效QRegularExpressionMatchIterator或者QRegularExpressionMatch使用它的对象。

该功能是在 Qt 6.5 中引入的。

也可以看看QRegularExpressionMatchIterator和global matching。

bool QRegularExpression::isValid() const

`true`如果正则表达式是有效的正则表达式（即不包含语法错误等），则返回，否则返回 false。使用errorString() 获取错误的文本描述。

也可以看看errorString（） 和patternErrorOffset()。

QRegularExpressionMatch QRegularExpression::match(const QString &subject, qsizetype offset = 0, QRegularExpression::MatchType matchType = NormalMatch, QRegularExpression::MatchOptions matchOptions = NoMatchOption) const

尝试将正则表达式与给定的匹配*subject*字符串，从位置开始*offset*在主题内部，使用类型匹配*matchType*并尊重所给予的*matchOptions*。

返回的QRegularExpressionMatch对象包含比赛的结果。

也可以看看QRegularExpressionMatch和normal matching。

[since 6.5]QRegularExpressionMatch QRegularExpression::matchView(QStringView subjectView, qsizetype offset = 0, QRegularExpression::MatchType matchType = NormalMatch, QRegularExpression::MatchOptions matchOptions = NoMatchOption) const

这是一个过载功能。

尝试将正则表达式与给定的匹配*subjectView*字符串视图，从该位置开始*offset*在主题内部，使用类型匹配*matchType*并尊重所给予的*matchOptions*。

返回的QRegularExpressionMatch对象包含比赛的结果。

注：数据参考*subjectView*只要存在就必须保持有效QRegularExpressionMatch使用它的对象。

该功能是在 Qt 6.5 中引入的。

也可以看看QRegularExpressionMatch和normal matching。

QStringList QRegularExpression::namedCaptureGroups() const

返回一个列表captureCount() + 1 个元素，包含模式字符串中命名捕获组的名称。对列表进行排序，使得列表中位于位置的元素i是第 -th 捕获组的名称i（如果它有名称）；如果该捕获组未命名，则为空字符串。

例如，给定正则表达式

```
    (?<day>\d\d)-(?<month>\d\d)-(?<year>\d\d\d\d) (\w+) (?<name>\w+)
```

nameCaptureGroups() 将返回以下列表：

```
    ("", "day", "month", "year", "", "name")
```

这对应于捕获组#0（对应于整个比赛）没有名称，捕获组#1的名称为“day”，捕获组#2的名称为“month”，等等。

如果正则表达式无效，则返回空列表。

也可以看看isValid(),QRegularExpressionMatch::captured（），和QString::isEmpty()。

void QRegularExpression::optimize() const

立即编译模式，包括 JIT 编译（如果启用了 JIT）以进行优化。

也可以看看isValid（）和Debugging Code that Uses QRegularExpression。

QString QRegularExpression::pattern() const

返回正则表达式的模式字符串。

也可以看看setPattern（）和patternOptions()。

qsizetype QRegularExpression::patternErrorOffset() const

返回模式字符串内的偏移量，在检查正则表达式的有效性时发现错误。如果没有发现错误，则返回 -1。

也可以看看pattern(),isValid（），和errorString()。

QRegularExpression::PatternOptions QRegularExpression::patternOptions() const

返回正则表达式的模式选项。

也可以看看setPatternOptions（）和pattern()。

void QRegularExpression::setPattern(const QString &pattern)

将正则表达式的模式字符串设置为*pattern*。模式选项保持不变。

也可以看看pattern（）和setPatternOptions()。

void QRegularExpression::setPatternOptions(QRegularExpression::PatternOptions options)

设置给定的*options*作为正则表达式的模式选项。模式字符串保持不变。

也可以看看patternOptions（） 和setPattern()。

void QRegularExpression::swap(QRegularExpression &other)

交换正则表达式*other*用这个正则表达式。这个操作非常快并且永远不会失败。

[static]QString QRegularExpression::wildcardToRegularExpression(QStringView pattern, QRegularExpression::WildcardConversionOptions options = DefaultWildcardConversion)

返回给定 glob 的正则表达式表示形式*pattern*。该转换的目标是文件路径通配，这特别意味着路径分隔符受到特殊处理。这意味着它不仅仅是从“*”到“.*”的基本翻译。

```
QString wildcard = QRegularExpression::wildcardToRegularExpression("*.jpeg");
// Will match files with names like:
//    foo.jpeg
//    f_o_o.jpeg
//    föö.jpeg
```

默认情况下，返回的正则表达式是完全锚定的。换句话说，不需要调用anchoredPattern() 再次计算结果。要获取未锚定的正则表达式，请传递UnanchoredWildcardConversion作为转换*options*。

此实现严格遵循 glob 模式通配符的定义：

C	**除下面提到的字符外，任何字符都代表其自身。因此c匹配字符*c*。**
?	匹配任何单个字符。它与 相同。在完整的正则表达式中。
*	匹配零个或多个任意字符。**它与完整正则表达式中的**.*相同。
[ABC]	匹配括号中给出的一个字符。
[交流]	匹配括号中给定范围内的一个字符。
[!abc]	匹配括号中未给出的一个字符。**它与完整正则表达式中的**[^abc]相同。
[!ac]	匹配不在括号中给定范围内的一个字符。**它与完整正则表达式中的**[^ac]相同。

注意：在此上下文中，反斜杠 () 字符不是*转义*字符。为了匹配特殊字符之一，请将其放在方括号中（例如，`[?]`）。

有关实施的更多信息可以在以下位置找到：

- The Wikipedia Glob article
- `man 7 glob`

也可以看看escape()。

[static]QString QRegularExpression::wildcardToRegularExpression(const QString &pattern, QRegularExpression::WildcardConversionOptions options = DefaultWildcardConversion)

这是一个过载功能。

bool QRegularExpression::operator!=(const QRegularExpression &re) const

`true`如果正则表达式不同于则返回*re*，否则为 false。

也可以看看operator==()。

QRegularExpression &QRegularExpression::operator=(const QRegularExpression &re)

指定正则表达式*re*到此对象，并返回对该副本的引用。图案和图案选项均被复制。

QRegularExpression &QRegularExpression::operator=(QRegularExpression &&re)

移动指定正则表达式*re*到此对象，并返回对结果的引用。图案和图案选项均被复制。

请注意，移出QRegularExpression只能被销毁或分配给。调用除析构函数或赋值运算符之一之外的其他函数的效果是未定义的。

bool QRegularExpression::operator==(const QRegularExpression &re) const

true如果正则表达式等于则返回*re*，否则为 false。二QRegularExpression如果对象具有相同的模式字符串和相同的模式选项，则它们相等。

也可以看看operator!=()。

相关非成员

size_t qHash(const QRegularExpression &key, size_t seed = 0)

返回哈希值*key*， 使用*seed*为计算提供种子。

QDataStream &operator<<(QDataStream &out, const QRegularExpression &re)

写入正则表达式*re*流式传输*out*。

也可以看看Serializing Qt Data Types。

QDebug operator<<(QDebug debug, const QRegularExpression &re)

写入正则表达式*re*进入调试对象*debug*用于调试目的。

也可以看看Debugging Techniques。

QDebug operator<<(QDebug debug, QRegularExpression::PatternOptions patternOptions)

写入模式选项*patternOptions*进入调试对象*debug*用于调试目的。

也可以看看Debugging Techniques。

QDataStream &operator>>(QDataStream &in, QRegularExpression &re)

从流中读取正则表达式*in*进入*re*。

也可以看看Serializing Qt Data Types。