

应用重构操作

Qt Creator允许您通过在上下文菜单中选择操作来快速方便地应用操作（快速修复）来重构代码。可用的操作取决于光标在代码编辑器中的位置。

若要将重构操作应用于C++代码，请右键单击操作数、条件语句、字符串或名称以打开上下文菜单。要将重构操作应用于 QML 代码，请右键单击项目 ID 或名称。

在上下文菜单中，选择“**重构**”，然后选择重构操作。

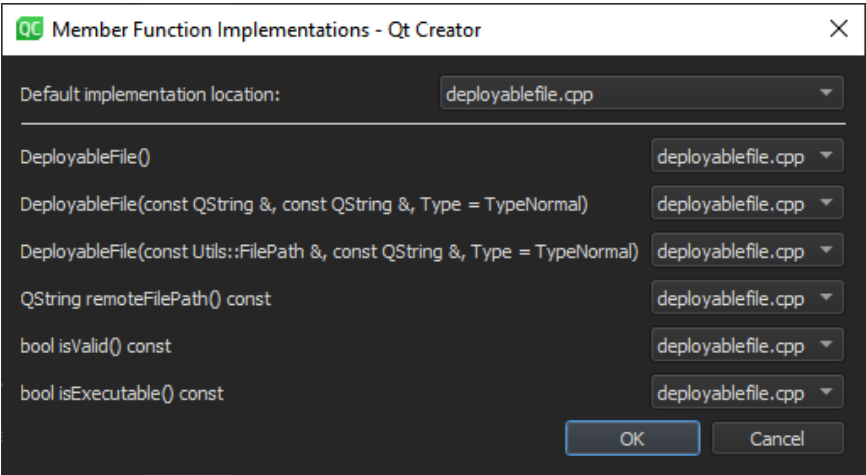
还可以按Alt+Enter打开上下文菜单，其中包含当前光标位置中可用的重构操作。

创建函数

可以应用重构操作来实现成员函数、插入基类的虚拟函数、创建 getter 和 setter 函数以及生成构造函数。可以在项目的生成和运行设置中为所有项目全局指定生成函数的设置，也可以为每个项目单独指定生成函数的设置。

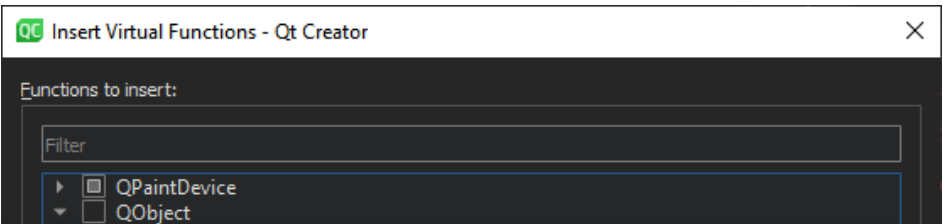
实现成员函数

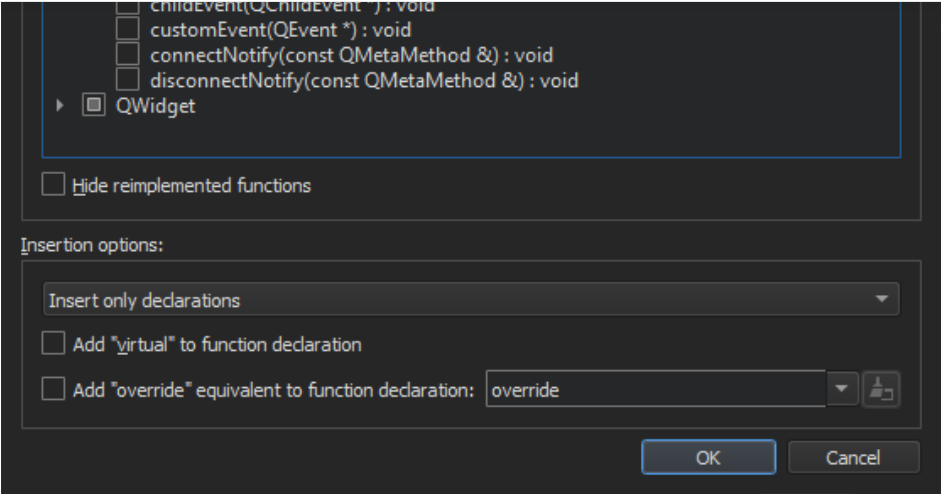
您可以应用为成员函数创建实现重构操作，一次性为所有**成员函数创建实现**。在“成员函数实现”对话框中，可以指定**成员函数**是在类内联生成还是在类外部生成。



插入虚函数

可以应用“**插入基类的虚拟函数**”重构操作，在类内部或外部或实现文件（如果存在）中插入声明和相应的定义。



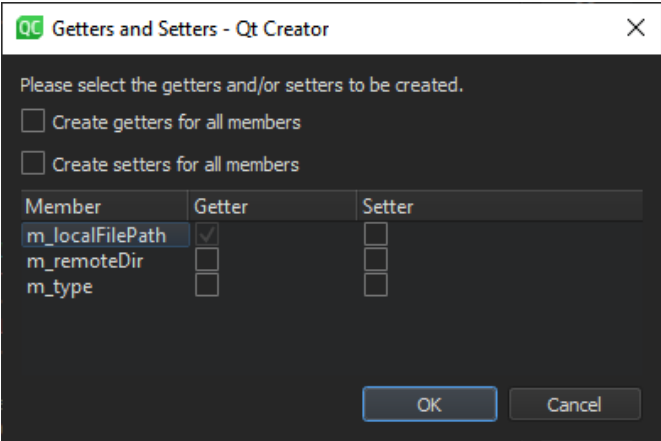


选择要插入到可用函数列表中的函数。您可以过滤列表并从中隐藏重新实现的函数。

您可以添加`virtual`或等效于函数声明的覆盖。

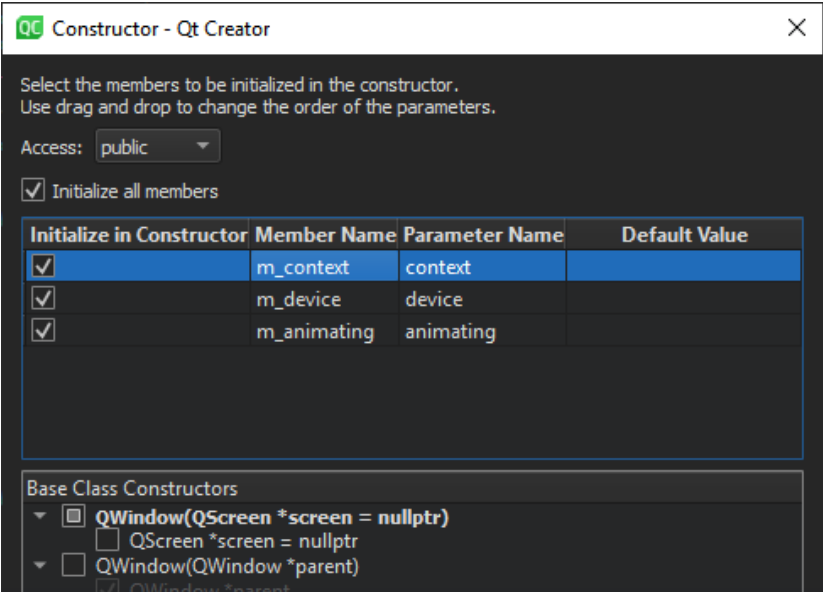
创建接模和二传器

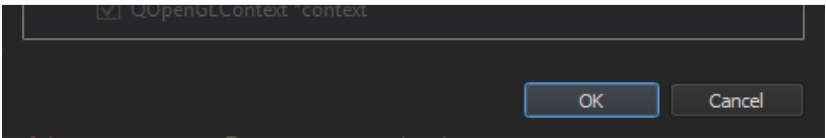
您可以应用“**创建 getter 和 Setter**成员函数”重构操作，为成员变量创建 `getter` 和 `setter` 成员函数，或者仅为 `getter` 或 `setter` 创建。



生成构造函数

可以应用“**生成构造函数**重构”操作为类创建公共、受保护或私有**构造函数**。选择要在构造函数中初始化的类成员。拖放参数以指定它们在构造函数中的顺序。

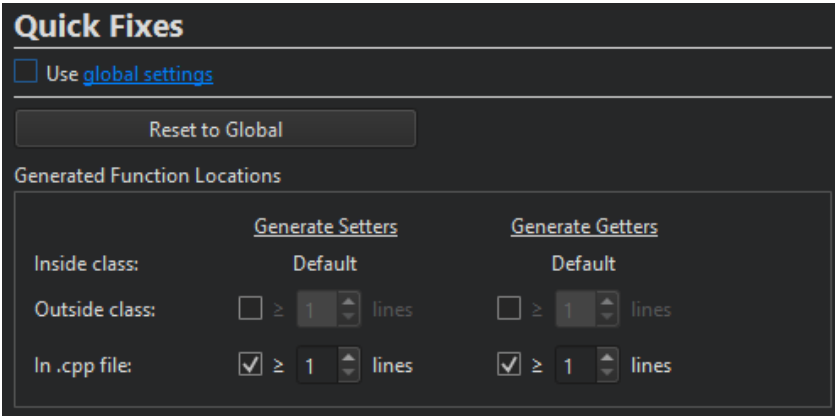




指定重构操作的设置

可以为所有项目全局指定重构操作的设置，也可以为每个项目单独指定重构操作的设置。若要指定全局选项，请选择“**编辑>首选项>C++>快速修复**”。

若要为特定项目指定自定义设置，请选择“**项目>项目设置>快速修复**”，然后取消选择“**使用全局设置**”。



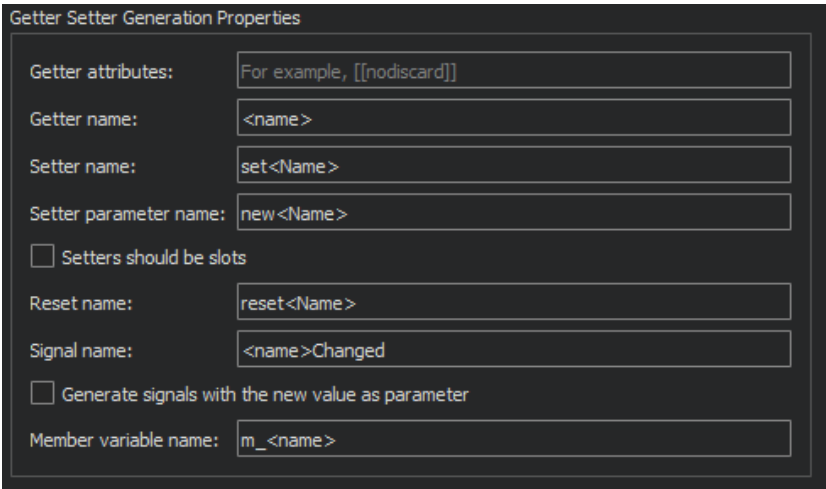
若要还原为全局设置，请选择“**重置为全局**”。若要删除自定义设置，请选择“**使用全局设置**”，然后选择“**删除自定义设置文件**”。

功能位置

在“**生成的函数位置**”组中，可以确定重构操作是在头文件（类内部或外部）还是在实现文件中生成 getter 和 setter 函数。

函数名称和属性

在“**吸气剂生成属性**”组中，您可以指定**吸气手**和**二传手**名称、属性和参数的其他设置。您可以指定应将 setter 函数创建为**插槽**，并且应使用新值作为参数生成信号。



命名空间处理

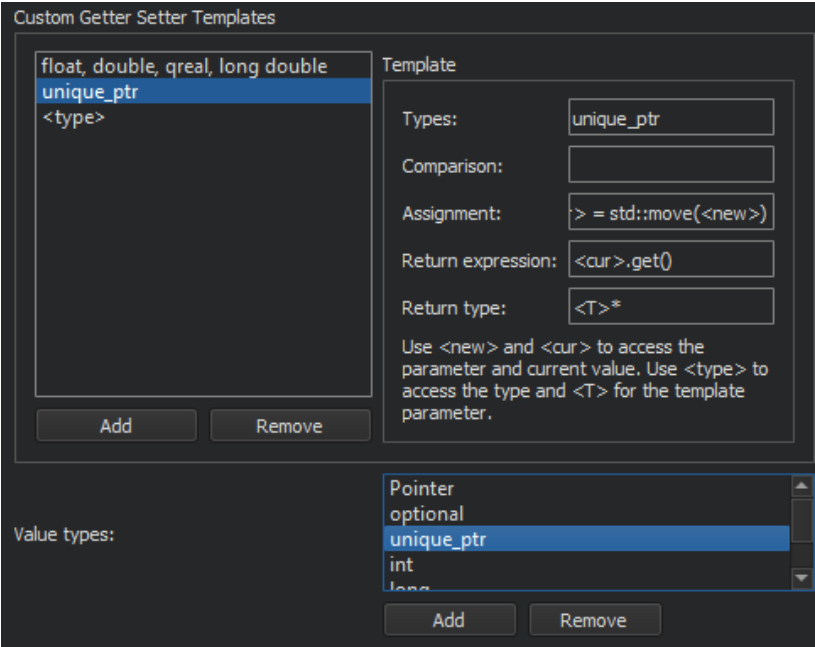
在“**缺少命名空间处理**”组中，选择是生成缺少的命名空间、根据需要添加还是重写类型以匹配现有**命名空间**。using namespace



自定义参数类型

在“自定义 Getter Setter 模板”组中，指定特定数据类型的 getter 或 setter 函数的代码应如何显示。这对于赋值无法使用的类型是必需的，例如在预定义的设置中，或者 where 不适合比较，例如在浮点类型的预定义设置中。例如，如果您有特殊类型，则可以指定函数，，应用于比较而不是默认值。operator=unique_ptoperator==MyClassmyCompare==

若要为自定义参数类型指定特殊处理，请选择“添加”并设置参数类型、比较、返回表达式和返回类型。在返回类型字段中，您可以使用和 访问参数和当前值。用于访问类型和模板参数。<new><cur><type><T>



通常，参数是使用引用传递的。若要将特定类型的参数作为值传递，请在“值类型”字段中列出它们。将删除命名空间和模板参数。实际类型必须包含给定的类型。例如，匹配但不匹配，匹配但不匹配。
constintint32_tvector<int>vectorstd::pmr::vector<int>std::optional<vector<int>>

重构操作摘要

如果您使用 Clang代码模型来解析C++文件，您还可以使用已集成到 Qt Creator 中的Clang fix-it 提示。除了激活重构操作的标准方法之外，还可以在代码编辑器左旁注的上下文菜单中选择适用于某一行的操作。

重构C++代码

可以将以下类型的重构操作应用于C++代码：

- 更改二进制操作数
- 简化 if 和 while 条件（例如，将声明移出 if 条件）
- 修改字符串（例如，将字符串的编码设置为 Latin-1，将字符串标记为可翻译，并将符号名称转换为驼峰大小写）
- 创建变量声明
- 创建函数声明和定义

下表总结了C++代码的重构操作。当光标位于“激活”列中所述的位置时，该操作可用。

重构操作	描述	激活
添加大括号	向不包含复合语句的 if 语句添加大括号。例如，重写 <pre>if (a) b;</pre>	if

	<pre>if (a) { b; }</pre>	
将声明移出条件	<p>将声明移出 if 或 while 条件以简化条件。例如，重写</p> <pre>if (Type name = foo()) {}</pre> <p>如</p> <pre>Type name = foo; if (name) {}</pre>	引入变量的名称
使用 重写条件	<p>根据德摩根定律重写表达式。例如，重写：</p> <pre>!a && !b</pre> <p>如</p> <pre>!(a b)</pre>	&&
使用运算符重写	<p>重写一个表达式，使其取反并使用反运算符。例如，重写：</p> <pre>> a op b</pre> <p>如</p> <pre>!(a invop b)</pre> <pre>> (a op b)</pre> <p>如</p> <pre>!(a invop b)</pre> <pre>> !(a op b)</pre> <p>如</p>	<=或<>===!=

拆分声明	<p>将简单声明拆分为多个声明。例如，重写：</p> <pre>int *a, b;</pre> <p>如</p> <pre>int *a; int b;</pre>	类型名称或变量名称
拆分如果语句	<p>将 if 语句拆分为多个语句。例如，重写：</p> <pre>if (something && something_else) { }</pre> <p>如</p> <pre>if (something) { if (something_else) { } }</pre> <p>和</p> <pre>if (something something_else) x;</pre> <p>跟</p> <pre>if (something) x; else if (something_else) x;</pre>	&&或
交换操作数	<p>使用反运算符以反顺序重写表达式。例如，重写：</p> <pre>a op b</pre> <p>如</p> <pre>b flipop a</pre>	<=或<>>===!=&&

转换为八进制	将整数文本转换为八进制表示形式	数字文本
转换为 Objective-C 字符串文字	<p>如果文件类型为 Objective-C (++)，则将字符串文本转换为 Objective-C 字符串文本。例如，重写以下字符串</p> <pre>"abcd" QLatin1String("abcd") QLatin1Literal("abcd")</pre> <p>如</p> <pre>@"abcd"</pre>	字符串文本
包含在 QLatin1Char () 中	<p>将字符的编码设置为 Latin-1，除非该字符已包含在 QLatin1Char、QT_TRANSLATE_NOOP、tr、trUtf8、QLatin1Literal 或QLatin1String 中。例如，重写</p> <pre>'a'</pre> <p>如</p> <pre>QLatin1Char('a')</pre>	字符串文本
括在 QLatin1String () 中	<p>将字符串的编码设置为 Latin-1，除非该字符串已包含在 QLatin1Char、QT_TRANSLATE_NOOP、tr、trUtf8、QLatin1Literal 或QLatin1String 中。例如，重写</p> <pre>"abcd"</pre> <p>如</p> <pre>QLatin1String("abcd")</pre>	字符串文本
标记为可翻译	<p>标记可翻译的字符串。例如，使用以下选项之一重写，具体取决于其中哪些选项可用："abcd"</p> <pre>tr("abcd") QCoreApplication::translate("CONTEXT", "abcd") QT_TRANSLATE_NOOP("GLOBAL", "abcd")</pre>	字符串文本
在 ... 中添加定义 ...	<p>在头文件（类内部或外部）或实现文件中插入函数声明的定义存根。对于自由函数，在函数声明之后或实现文件中插入定义。限定名称在可能的情况下最小化，而不是始终完全展开。例如，重写</p> <pre>Class Foo {</pre>	函数名称

	<div>作为（类内）</div> <div><pre>Class Foo { void bar() { } };</pre></div> <div>作为（课外）</div> <div><pre>Class Foo { void bar(); }; void Foo::bar() { }</pre></div> <div>作为（在实现文件中）</div> <div><pre>// Header file Class Foo { void bar(); }; // Implementation file void Foo::bar() { }</pre></div>	
添加声明 Function	将与成员函数定义匹配的成员函数声明插入到类声明中。该函数可以是,,,,或。 publicprotectedprivatepublic slotprotected slotprivate slot	函数名称
添加类成员	为正在初始化的类成员添加成员声明（如果尚未声明）。必须输入成员的数据类型。	标识符
为成员函数创建实现	一次性为所有成员函数创建实现。在“成员函数实现”对话框中，可以指定 成员函数 是在类内联生成还是在类外部生成。	函数名称
使用下一个/上一个参数切换	将参数在参数列表中下移或上移一个位置。	函数的声明或定义中的参数
提取功能	将所选代码移动到新函数，并将代码块替换为对新函数的调用。在“ 提取函数重构 ”对话框中输入函数的名称。	选定的代码块
提取常量作为函数参数	将选定的文本及其所有匹配项替换为函数参数。该参数将使用原始文本作为默认值。 newParameternewParameter	选定的代码块
添加本地声明	添加被分派人的类型（如果工作分配右侧的类型已知）。例如，重写	受让人
	<div>a = foo();</div> <div>如</div>	

```
a = foo();
```

如

	其中 Type 是返回类型foo()	
转换为骆驼壳	将符号名称转换为驼峰大小写，其中名称的元素连接起来，不带分隔符，并且每个元素的初始字符大写。例如，重写萨桑达斯 an_example_symbolanExampleSymbolAN_EXAMPLE_SYMBOLAnExampleSymbol	标识符
完整的转换声明	将所有可能的情况添加到 switch 语句的类型enum	switch
生成缺少的Q_PROPERTY成员	将缺少的成员添加到：Q_PROPERTY > read功能 > write函数，如果有写入 > onChanged信号，如果有通知 > 具有名称的数据成员m_<propertyName>	Q_PROPERTY
生成Q_PROPERTY和缺失的成员	生成Q_PROPERTY并向其中添加缺少的成员，如上所述。	类成员
生成常量Q_PROPERTY和缺失成员	生成常量Q_PROPERTY并向其中添加缺少的成员，如上所述。	类成员
使用重置功能生成Q_PROPERTY和缺失的成员	生成一个Q_PROPERTY并向其中添加缺少的成员，如上所述，但具有附加函数。reset	类成员
应用更改	通过在编辑函数签名时检查匹配的声明或定义并将更改应用于匹配的代码，使函数声明和定义保持同步。	函数签名。当此操作可用时，将显示一个灯泡图标：💡
为未声明或转发声明的标识符添加#include	将指令添加到当前文件以使符号的定义可用。#include	未声明的标识符
添加转发声明	为未声明的标识符操作添加前向声明。	未声明的标识符
重新设置指针或引用的格式	根据当前项目的代码样式设置，使用指针或引用重新设置声明的格式。如果未打开任何项目，则使用当前的全局代码样式设置。 例如，重写： <div>char*s;</div> 如 <div>char *s;</div> 应用于所选内容时，将重写所选内容中的所有适当声明。	带有指针或引用的声明以及包含此类声明的选择
创建 getter 和 setter 成员函数	为成员变量创建 getter 和 setter 成员函数，或者只创建 getter 或 setter。	类定义中的成员变量
生成吸气剂和二传手	为成员变量创建 getter 和 setter 成员函数。	类定义中的成员变量
生成吸气剂	为成员变量创建 getter 成员函数。	类定义中的成员变量
生成资源库	为成员变量创建资源库成员函数。	类定义中的成员变量
生成构造函数	为类创建构造函数。	类定义
移动函数定义	将函数定义移动到实现文件、类外部或移回其声明。例如，重写： <div></div>	函数签名

	<pre>{ // do stuff here } };</pre> <p>如</p> <pre>class Foo { void bar(); }; void Foo::bar() { // do stuff here }</pre>	
移动所有函数定义	<p>将所有函数定义移动到实现文件或类外部。例如，重写：</p> <pre>class Foo { void bar() { // do stuff here } void baz() { // do stuff here } };</pre> <p>如</p> <pre>class Foo { void bar(); void baz(); }; void Foo::bar() { // do stuff here } void Foo::baz() { // do stuff here }</pre>	类名
赋值到局部变量	<p>添加一个局部变量，该变量存储函数调用或新表达式的返回值。例如，重写：</p> <pre>QString s; s.toLatin();</pre> <p>如</p>	函数调用或类名

	<div>和</div> <div><pre>new Foo;</pre></div> <div>如</div> <div><pre>Foo * localFoo = new Foo;</pre></div>	
插入基类的虚函数	在类内部或外部或实现文件（如果存在）中插入声明和相应的定义。有关更多信息，请参见 插入虚拟函数 。	类或基类名称
优化 for 循环	将后递增运算符重写为递增前运算符，将递减后运算符重写为递减前运算符。它还将字符串或数字文本和 id 表达式以外的其他内容从 for 循环的条件移动到你初始值设定项。例如，重写： <div><pre>for (int i = 0; i < 3 * 2; i++)</pre></div> <div>如</div> <div><pre>for (int i = 0, total = 3 * 2; i < total; ++i)</pre></div>	for
转义字符串文本为 UTF-8	对字符串文本到十六进制转义序列中的非 ASCII 字符进行转义。字符串文本以 UTF-8 形式处理。	字符串文本
取消转义字符串文本为 UTF-8	取消转义字符串文本中的八进制或十六进制转义序列。字符串文本以 UTF-8 形式处理。	字符串文本
转换为堆栈变量	将选定的指针转换为堆栈变量。例如，重写： <div><pre>QByteArray *foo = new QByteArray("foo"); foo->append("bar");</pre></div>	指针变量
Qt 创建者手册 8.0.2 Topics >		
	<div><pre>QByteArray foo("foo"); foo.append("bar");</pre></div> <div>此操作仅限于在函数范围内工作。此外，指针和引用的编码样式尚未得到尊重。</div>	
转换为指针	将选定的堆栈变量转换为指针。例如，重写： <div><pre>QByteArray foo = "foo"; foo.append("bar");</pre></div> <div>如</div>	堆栈变量

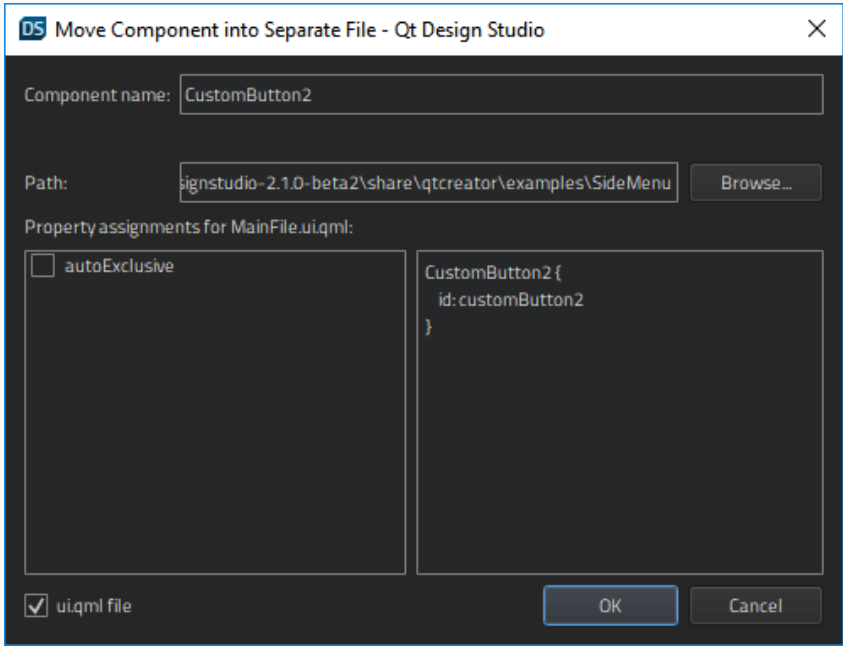
	<pre>foo->append(bar);</pre>	
	此操作仅限于在函数范围内工作。此外，指针和引用的编码样式尚未得到尊重。	
相应地删除和调整类型名称using namespace	删除本地范围内的出现次数并相应地调整类型名称。using namespace	using命令
删除全局范围内出现的所有项并相应地调整类型名称using namespace	删除全局范围内出现的所有项，并相应地调整类型名称。using namespace	using命令
将 connect () 转换为 Qt 5 样式	将 Qt 4QObject: : connect () 转换为 Qt 5 样式。	QObject: : connect () (Qt 4 样式)

重构 QML 代码

您可以将以下类型的重构操作应用于 QML 代码：

- › 重命名 ID
- › 拆分初始值设定项
- › 将 QML 类型移动到单独的文件中，以便在其他 .qml 文件中重用它

下表总结了 QML 代码的重构操作。当光标位于“激活”列中所述的位置时，该操作可用。

重构操作	描述	激活
将组件移动到单独的文件中	<p>将 QML 类型移动到单独的文件中。为新组件命名，并选择是为新组件还是为原始组件设置属性。</p> 	QML 类型名称。
拆分初始值设定项	<p>将单行类型重新格式化为多行类型。例如，重写</p> <pre>Item { x: 10; y: 20; width: 10 }</pre> <p>如</p> <pre>Item { x: 10; y: 20; width: 10; }</pre>	QML 类型属性

	<pre>} </pre>	
在加载器中包装组件	将类型包装在组件类型中，并在加载器类型中动态加载它。这样做通常是为了缩短启动时间。	QML 类型名称
添加邮件抑制注释	在该行前面加上阻止生成消息的批注注释。	来自静态分析的错误、警告或提示

[< 重构](#)

[美化源代码 >](#)

©2022 Qt有限公司 此处包含的文档贡献的版权归 他们各自的所有者。此处提供的文档根据自由软件基金会发布的[GNU 自由文档许可证版本 1.3](#)的条款进行许可。Qt和相应的徽标是Qt有限公司在芬兰和/或全球其他国家的商标。所有其他商标均为财产 其各自所有者。



The Qt Company



联系我们

公司

关于我们

投资者

编辑部

职业

办公地点

发牌

条款和条件

开源

常见问题

支持

支持服务

专业服务

合作伙伴

训练

对于客户

支持中心

下载

Qt登录

联系我们

客户成功案例

社区

为Qt做贡献

论坛

维基

下载

市场

反馈

登录

© 2022 Qt公司

https://doc.qt.io/qtcreator/creator-editor-quick-fixes.html

13/13