# QOpenGLWidget Class

QOpenGLWidget 类是用于渲染 OpenGL 图形的小部件。更多的...

| | |
|---|---|
| **Header:** | **#include < QOpenGLWidget>** |
| CMake: | find_package(Qt6 REQUIRED COMPONENTS OpenGLWidgets) target_link_libraries(mytarget PRIVATE Qt6::OpenGLWidgets) |
| qmake: | QT += openglwidgets |
| Inherits: | QWidget |

- 所有成员的列表，包括继承的成员

## 公共类型

| | |
|---|---|
| **enum** | **TargetBuffer { LeftBuffer, RightBuffer }** |
| enum | **UpdateBehavior** { NoPartialUpdate, PartialUpdate } |

## 公共职能

| | |
|---|---|
| | **QOpenGLWidget(QWidget **parent* = nullptr, Qt::WindowFlags *f* = Qt::WindowFlags())** |
| virtual | **~QOpenGLWidget**() |
| QOpenGLContext * | **context**() const |
| QOpenGLWidget::TargetBuffer | **currentTargetBuffer**() const |
| GLuint | **defaultFramebufferObject**() const |
| GLuint | **defaultFramebufferObject**(QOpenGLWidget::TargetBuffer *targetBuffer*) const |
| void | **doneCurrent**() |
| QSurfaceFormat | **format**() const |
| QImage | **grabFramebuffer**() |
| QImage | **grabFramebuffer**(QOpenGLWidget::TargetBuffer *targetBuffer*) |
| bool | **isValid**() const |

| | |
|---|---|
| | **QOpenGLWidget(QWidget **parent* = nullptr, Qt::WindowFlags *f* = Qt::WindowFlags())** |
| void | **makeCurrent**() |
| void | **makeCurrent**(QOpenGLWidget::TargetBuffer *targetBuffer*) |
| void | **setFormat**(const QSurfaceFormat &*format*) |
| void | **setTextureFormat**(GLenum *texFormat*) |
| void | **setUpdateBehavior**(QOpenGLWidget::UpdateBehavior *updateBehavior*) |
| GLenum | **textureFormat**() const |
| QOpenGLWidget::UpdateBehavior | **updateBehavior**() const |

## 信号

| | |
|---|---|
| **void** | **aboutToCompose()** |
| void | **aboutToResize**() |
| void | **frameSwapped**() |
| void | **resized**() |

## protected function

| | |
|---|---|
| **virtual void** | **initializeGL()** |
| virtual void | **paintGL**() |
| virtual void | **resizeGL**(int *w*, int *h*) |

## 重载的protected function

| | |
|---|---|
| **virtual bool** | **event(QEvent **e*) override** |
| virtual int | **metric**(QPaintDevice::PaintDeviceMetric *metric*) const override |
| virtual QPaintEngine * | **paintEngine**() const override |
| virtual void | **paintEvent**(QPaintEvent **e*) override |
| virtual QPaintDevice * | **redirected**(QPoint **p*) const override |
| virtual void | **resizeEvent**(QResizeEvent **e*) override |

# 详细说明

QOpenGLWidget 提供了显示集成到 Qt 应用程序中的 OpenGL 图形的功能。使用起来非常简单：让你的类继承它并像使用其他类一样使用子类QWidget，除了您可以选择使用QPainter和标准 OpenGL 渲染命令。

QOpenGLWidget 提供了三个方便的虚拟函数，您可以在子类中重新实现它们来执行典型的 OpenGL 任务：

- paintGL() - 渲染 OpenGL 场景。每当小部件需要更新时就会被调用。
- resizeGL() - 设置 OpenGL 视口、投影等。每当调整窗口小部件的大小时（以及第一次显示窗口小部件时，因为所有新创建的窗口小部件都会自动获取调整大小事件），都会调用它。
- initializeGL() - 设置 OpenGL 资源和状态。第一次之前被叫过一次resizeGL（） 或者paintGL（） 叫做。

如果您需要从其他地方触发重绘paintGL()（一个典型的例子是使用timers动画场景），你应该调用小部件的update()函数来安排更新。

您的小部件的 OpenGL 渲染上下文在以下情况下变为当前上下文：paintGL(),resizeGL（）， 或者initializeGL（）叫做。如果您需要从其他地方调用标准 OpenGL API 函数（例如在您的小部件的构造函数中或在您自己的绘制函数中），您必须调用makeCurrent（） 第一的。

所有渲染都发生在 OpenGL 帧缓冲区对象中。makeCurrent() 确保它在上下文中绑定。在渲染代码中创建和绑定附加帧缓冲区对象时请记住这一点paintGL()。切勿重新绑定 ID 为 0 的帧缓冲区。而是调用defaultFramebufferObject() 获取应该绑定的ID。

当平台支持时，QOpenGLWidget 允许使用不同的 OpenGL 版本和配置文件。只需通过设置请求的格式setFormat()。但请记住，在同一窗口中拥有多个 QOpenGLWidget 实例要求它们都使用相同的格式，或者至少使用不会使上下文不可共享的格式。为了解决这个问题，最好使用QSurfaceFormat::setDefaultFormat（） 代替setFormat()。

**注：**打电话QSurfaceFormat::setDefaultFormat() 构建之前QApplication当请求 OpenGL 核心配置文件上下文时，实例在某些平台（例如 macOS）上是强制性的。这是为了确保上下文之间的资源共享保持功能，因为所有内部上下文都是使用正确的版本和配置文件创建的。

## *绘画技巧*

如上所述，子类化 QOpenGLWidget 以通过以下方式渲染纯 3D 内容：

- 重新实施initializeGL（） 和resizeGL() 函数设置 OpenGL 状态并提供透视变换。
- 重新实现paintGL() 绘制 3D 场景，仅调用 OpenGL 函数。

还可以使用以下命令将 2D 图形绘制到 QOpenGLWidget 子类上QPainter:

- 在paintGL()，不发出 OpenGL 命令，而是构造一个QPainter用于小部件的对象。
- 使用绘制基元QPainter的成员函数。
- 仍然可以发出直接 OpenGL 命令。但是，您必须确保它们包含在对画家的 beginNativePainting() 和 endNativePainting() 的调用中。

执行绘图时使用QPainter只是，也可以像普通小部件一样执行绘画：通过重新实现paintEvent()。

- 重新实施paintEvent（） 功能。
- 构建一个QPainter针对小部件的对象。将小部件传递给构造函数或QPainter::begin（） 功能。
- 使用绘制基元QPainter的成员函数。
- 绘画完成然后QPainter实例被销毁。或者，致电QPainter::end() 明确。

## *OpenGL 函数调用、标头和 QOpenGLFunction*

在进行 OpenGL 函数调用时，强烈建议避免直接调用函数。相反，更喜欢使用QOpenGLFunctions（当制作便携式应用程序时）或版本化变体（例如，QOpenGLFunctions_3_2_Core类似地，当针对现代的、仅限桌面的 OpenGL 时）。这样，应用程序将在所有 Qt 构建配置中正常工作，包括执行动态 OpenGL 实现加载的配置，这意味着应用程序不直接链接到 GL 实现，因此直接函数调用不可行。

在paintGL() 当前上下文始终可以通过调用来访问QOpenGLContext::currentContext()。从这个上下文中，一个已经初始化的、可以使用的QOpenGLFunctions实例可以通过调用来检索QOpenGLContext::functions()。为每个 GL 调用添加前缀的替代方法是继承QOpenGLFunctions并打电话QOpenGLFunctions::initializeOpenGLFunctions（）在initializeGL()。

至于 OpenGL 标头，请注意，在大多数情况下，不需要直接包含任何标头（例如 GL.h）。与 OpenGL 相关的 Qt 头文件将包括 qopengl.h，而 qopengl.h 又将包括适合系统的头文件。这可能是 OpenGL ES 3.x 或 2.0 标头、可用的最高版本或系统提供的 gl.h。此外，扩展头文件的副本（在某些系统上称为 glext.h）作为 Qt 的一部分提供，适用于 OpenGL 和 OpenGL ES。在可行的情况下，这些将自动包含在平台上。这意味着来自 ARB、EXT、OES 扩展的常量和函数指针类型定义自动可用。

## *代码示例*

首先，最简单的 QOpenGLWidget 子类可能如下所示：

```
class MyGLWidget : public QOpenGLWidget
{
public:
    MyGLWidget(QWidget *parent) : QOpenGLWidget(parent) { }

protected:
    void initializeGL() override
    {
        // Set up the rendering context, load shaders and other resources, etc.:
        QOpenGLFunctions *f = QOpenGLContext::currentContext()->functions();
        f->glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
        ...
    }

    void resizeGL(int w, int h) override
    {
        // Update projection matrix and other size related settings:
        m_projection.setToIdentity();
        m_projection.perspective(45.0f, w / float(h), 0.01f, 100.0f);
        ...
    }

    void paintGL() override
    {
        // Draw the scene:
        QOpenGLFunctions *f = QOpenGLContext::currentContext()->functions();
        f->glClear(GL_COLOR_BUFFER_BIT);
        ...
    }

};
```

或者，可以通过派生来避免每个 OpenGL 调用的前缀QOpenGLFunctions反而：

```
class MyGLWidget : public QOpenGLWidget, protected QOpenGLFunctions
{
    ...
    void initializeGL() override
    {
        initializeOpenGLFunctions();
        glClearColor(...);
        ...
    }
    ...
};
```

要获取与给定 OpenGL 版本或配置文件兼容的上下文，或者请求深度和模板缓冲区，请调用setFormat():

```
QOpenGLWidget *widget = new QOpenGLWidget(parent);
QSurfaceFormat format;
format.setDepthBufferSize(24);
format.setStencilBufferSize(8);
format.setVersion(3, 2);
format.setProfile(QSurfaceFormat::CoreProfile);
widget->setFormat(format); // must be called before the widget or its parent window gets shown
```

**注意：**由应用程序确保从底层窗口系统接口请求深度和模板缓冲区。如果不请求非零深度缓冲区大小，则无法保证深度缓冲区可用，因此与深度测试相关的 OpenGL 操作可能无法按预期运行。常用的深度和模板缓冲区大小请求分别为 24 和 8。

对于 OpenGL 3.0+ 上下文，当可移植性不重要时，版本化QOpenGLFunctions变体可以轻松访问给定版本中可用的所有现代 OpenGL 函数：

```
    ...
    void paintGL() override
    {
        QOpenGLFunctions_3_2_Core *f = QOpenGLContext::currentContext()-
>versionFunctions<QOpenGLFunctions_3_2_Core>();
        ...
        f->glDrawArraysInstanced(...);
        ...
    }
    ...
```

如上所述，全局设置请求的格式更简单且更稳健，以便它在应用程序的生命周期内应用于所有窗口和上下文。下面是一个例子：

```
int main(int argc, char **argv)
{
    QApplication app(argc, argv);

    QSurfaceFormat format;
    format.setDepthBufferSize(24);
    format.setStencilBufferSize(8);
    format.setVersion(3, 2);
    format.setProfile(QSurfaceFormat::CoreProfile);
    QSurfaceFormat::setDefaultFormat(format);
```

```
    MyWidget widget;
    widget.show();

    return app.exec();
}
```

## 多重采样

要启用多重采样，请设置请求的样本数QSurfaceFormat被传递到setFormat()。在不支持它的系统上，该请求可能会被忽略。

多重采样支持需要支持多重采样渲染缓冲区和帧缓冲区位块传送。在 OpenGL ES 2.0 实现中，这些可能不会出现。这意味着多重采样将不可用。对于现代 OpenGL 版本和 OpenGL ES 3.0 及更高版本，这通常不再是问题。

## 螺纹加工

在工作线程上执行离屏渲染，例如生成纹理，然后在 GUI/主线程中使用paintGL()，通过公开小部件的支持QOpenGLContext这样就可以在每个线程上创建与其共享的附加上下文。

通过重新实现，可以在 GUI/主线程之外直接绘制到 QOpenGLWidget 的帧缓冲区paintEvent() 不执行任何操作。上下文的线程关联性必须通过以下方式更改QObject::moveToThread()。在那之后，makeCurrent（） 和doneCurrent() 可在工作线程上使用。之后请小心将上下文移回 GUI/主线程。

仅仅为 QOpenGLWidget 触发缓冲区交换是不可能的，因为它没有真正的屏幕本机表面。由小部件堆栈来管理 GUI 线程上的组合和缓冲区交换。当线程完成更新帧缓冲区时，调用update()**在 GUI/主线程上**安排组合。

当 GUI/主线程执行合成时，必须格外小心，避免使用帧缓冲区。信号aboutToCompose（） 和frameSwapped() 将在合成开始和结束时发出。它们在 GUI/主线程上发出。这意味着通过使用直接连接aboutToCompose() 可以阻塞 GUI/主线程，直到工作线程完成渲染。之后，工作线程必须不再执行渲染，直到frameSwapped() 信号被发射。如果这是不可接受的，工作线程必须实现双缓冲机制。这涉及使用完全由线程控制的替代渲染目标（例如附加帧缓冲区对象）进行绘制，并在适当的时间位块传输到 QOpenGLWidget 的帧缓冲区。

## 上下文共享

当多个 QOpenGLWidget 作为子级添加到同一个顶级 widget 时，它们的上下文将相互共享。这不适用于属于不同窗口的 QOpenGLWidget 实例。

这意味着同一窗口中的所有 QOpenGLWidget 都可以访问彼此的可共享资源，例如纹理，并且不需要额外的“全局共享”上下文。

要设置属于不同窗口的 QOpenGLWidget 实例之间的共享，请设置Qt::AA_ShareOpenGLContexts实例化之前的应用程序属性QApplication。这将触发所有 QOpenGLWidget 实例之间的共享，无需任何进一步的步骤。

创造额外的QOpenGLContext与 QOpenGLWidget 上下文共享纹理等资源的实例也是可能的。只需传递从返回的指针context（） 到QOpenGLContext::setShareContext() 调用之前QOpenGLContext::create()。生成的上下文还可以在不同的线程上使用，从而允许线程化纹理生成和异步纹理上传。

请注意，当涉及底层图形驱动程序时，QOpenGLWidget 期望资源共享的标准一致实现。例如，某些驱动程序（特别是针对移动和嵌入式硬件的驱动程序）在现有上下文和以后创建的其他上下文之间设置共享时存在问题。当尝试利用不同线程之间的共享资源时，其他一些驱动程序可能会以意想不到的方式运行。

## *资源初始化和清理*

QOpenGLWidget 关联的 OpenGL 上下文保证在任何时候都是当前的initializeGL（） 和paintGL() 被调用。在此之前不要尝试创建 OpenGL 资源initializeGL（） 叫做。例如，在子类的构造函数中尝试编译着色器、初始化顶点缓冲区对象或上传纹理数据将失败。这些操作必须推迟到initializeGL()。一些 Qt 的 OpenGL 帮助器类，例如QOpenGLBuffer或者QOpenGLVertexArrayObject，具有匹配的延迟行为：它们可以在没有上下文的情况下实例化，但所有初始化都会延迟到create() 或类似的调用。这意味着它们可以用作 QOpenGLWidget 子类中的普通（非指针）成员变量，但是create() 或类似函数只能从initializeGL()。但请注意，并非所有类都是这样设计的。如有疑问，请将成员变量设置为指针并动态创建和销毁实例initializeGL() 和析构函数。

释放资源还需要当前上下文。因此，执行此类清理的析构函数预计会调用makeCurrent()，然后再继续销毁任何 OpenGL 资源或包装器。避免延迟删除通过deleteLater() 或养育机制QObject。无法保证在相关实例真正被销毁时正确的上下文将是当前的。

因此，在资源初始化和销毁方面，典型的子类通常如下所示：

```
class MyGLWidget : public QOpenGLWidget
{
    ...

private:
    QOpenGLVertexArrayObject m_vao;
    QOpenGLBuffer m_vbo;
    QOpenGLShaderProgram *m_program;
    QOpenGLShader *m_shader;
    QOpenGLTexture *m_texture;
};

MyGLWidget::MyGLWidget()
    : m_program(0), m_shader(0), m_texture(0)
{
    // No OpenGL resource initialization is done here.
}

MyGLWidget::~MyGLWidget()
{
    // Make sure the context is current and then explicitly
    // destroy all underlying OpenGL resources.
    makeCurrent();

    delete m_texture;
    delete m_shader;
    delete m_program;

    m_vbo.destroy();
    m_vao.destroy();

    doneCurrent();
}
```

```
void MyGLWidget::initializeGL()
{
    m_vao.create();
    if (m_vao.isCreated())
        m_vao.bind();

    m_vbo.create();
    m_vbo.bind();
    m_vbo.allocate(...);

    m_texture = new QOpenGLTexture(QImage(...));

    m_shader = new QOpenGLShader(...);
    m_program = new QOpenGLShaderProgram(...);

    ...
}
```

这适用于大多数情况，但作为通用解决方案并不完全理想。当小部件被重新设置父级以便它最终出现在完全不同的顶级窗口中时，需要更多的东西：通过连接到aboutToBeDestroyed() 信号QOpenGLContext，只要 OpenGL 上下文即将释放，就可以执行清理操作。

**注意：**对于在其生命周期内多次更改其关联的顶级窗口的小部件，组合清理方法（如下面的代码片段所示）至关重要。每当小部件或其父级重新设置父级以使顶级窗口变得不同时，小部件的关联上下文就会被销毁并创建一个新的上下文。然后调用initializeGL()，其中所有 OpenGL 资源都必须重新初始化。因此，执行正确清理的唯一选择是连接到上下文的 aboutToBeDestroyed() 信号。请注意，当信号发出时，所讨论的上下文可能不是当前的上下文。因此，最好的做法是致电makeCurrent() 在连接的槽中。此外，必须从派生类的析构函数执行相同的清理步骤，因为当小部件被销毁时，可能不会调用连接到信号的槽或 lambda。

```
MyGLWidget::~MyGLWidget()
{
    cleanup();
}

void MyGLWidget::initializeGL()
{
    ...
    connect(context(), &QOpenGLContext::aboutToBeDestroyed, this, &MyGLWidget::cleanup);
}

void MyGLWidget::cleanup()
{
    makeCurrent();
    delete m_texture;
    m_texture = 0;
    ...
    doneCurrent();
    disconnect(context(), &QOpenGLContext::aboutToBeDestroyed, this, &MyGLWidget::cleanup);
}
```

**注：**当Qt::AA_ShareOpenGLContexts设置后，小部件的上下文永远不会改变，即使在重新设置父级时也不会改变，因为小部件的关联纹理也可以从新的顶层上下文访问。因此，设置此标志并不强制对上下文的 aboutToBeDestroyed() 信号进行操作。

由于上下文共享，正确的清理尤为重要。即使每个 QOpenGLWidget 的关联上下文与 QOpenGLWidget 一起被销毁，该上下文中的可共享资源（如纹理）将保持有效，直到 QOpenGLWidget 所在的顶级窗口被销毁为止。此外，像这样的设置Qt::AA_ShareOpenGLContexts某些 Qt 模块可能会触发更广泛的共享上下文，从而可能导致相关资源在应用程序的整个生命周期内保持活动状态。因此，最安全、最稳健的方法始终是对 QOpenGLWidget 中使用的所有资源和资源包装器执行显式清理。

## *限制和其他考虑因素*

将其他小部件放在下面并使 QOpenGLWidget 透明将不会导致预期的结果：下面的小部件将不可见。这是因为实际上 QOpenGLWidget 是在所有其他常规非 OpenGL 小部件之前绘制的，因此透明类型的解决方案是不可行的。其他类型的布局，例如在 QOpenGLWidget 之上放置小部件，将按预期运行。

当绝对必要时，可以通过设置来克服此限制Qt::WA_AlwaysStackOnTopQOpenGLWidget 上的属性。但请注意，这会破坏堆叠顺序，例如，在 QOpenGLWidget 之上不可能有其他小部件，因此它只能在需要半透明 QOpenGLWidget 且其他小部件在下面可见的情况下使用。

请注意，当下面没有其他小部件并且目的是拥有半透明窗口时，这并不适用。在这种情况下，传统的设置方法Qt::WA_TranslucentBackground在顶层窗口就足够了。请注意，如果仅在 QOpenGLWidget 中需要透明区域，则Qt::WA_NoSystemBackground`false`启用后需要返回Qt::WA_TranslucentBackground。此外，通过以下方式请求 QOpenGLWidget 上下文的 alpha 通道setFormat() 可能也是必要的，具体取决于系统。

QOpenGLWidget支持多种更新行为，就像QOpenGLWindow。在保留模式下，先前渲染的内容paintGL() 调用在下一个中可用，允许增量渲染。在非保留模式下，内容会丢失并且paintGL() 实现预计会重绘视图中的所有内容。

在 Qt 5.5 之前，QOpenGLWidget 的默认行为是保留渲染内容paintGL() 调用。自 Qt 5.5 起，默认行为不再保留，因为这提供了更好的性能，并且大多数应用程序不需要以前的内容。这也类似于基于 OpenGL 的语义QWindow并匹配默认行为QOpenGLWindow 因为每个帧的颜色和辅助缓冲区都是无效的。要恢复保留的行为，请调用setUpdateBehavior() 与`PartialUpdate`.

**注意：**当将 QOpenGLWidget 动态添加到小部件层次结构中时，例如，通过将新的 QOpenGLWidget 父级设置为相应的顶级小部件已显示在屏幕上的小部件，如果 QOpenGLWidget 是第一个，则关联的本机窗口可能会隐式销毁并重新创建其窗口内的同类。这是因为窗口类型从RasterSurface到OpenGLSurface这具有特定于平台的影响。此行为是 Qt 6.4 中的新行为。

一旦 QOpenGLWidget 添加到部件层次结构中，顶层窗口的内容就会通过基于 OpenGL 的渲染进行刷新。QOpenGLWidget 以外的小部件继续使用基于软件的绘图器绘制其内容，但最终的合成是通过 3D API 完成的。

**注意：**由于与其他窗口的组合方式，显示 QOpenGLWidget 需要关联的顶级窗口后备存储中的 Alpha 通道QWidget基于内容的作品。如果没有alpha通道，QOpenGLWidget渲染的内容将不可见。当使用低于 24 的颜色深度时，这在远程显示设置（例如使用 Xvnc）中的 Linux/X11 上尤其重要。例如，16 的颜色深度通常会映射到使用具有以下格式的后备存储图像QImage::Format_RGB16(RGB565)，没有为 Alpha 通道留下空间。因此，如果在将 QOpenGLWidget 的内容与窗口中的其他小部件正确合成时遇到问题，请确保服务器（例如 vncserver）配置为 24 或 32 位深度而不是 16 位深度。

## 备择方案

将 QOpenGLWidget 添加到窗口中将为整个窗口启用基于 OpenGL 的合成。在某些特殊情况下，这可能并不理想，并且需要具有单独的本机子窗口的旧 QGLWidget 样式行为。了解此方法局限性的桌面应用程序（例如，当涉及重叠、透明度、滚动视图和 MDI 区域时），可以使用QOpenGLWindow和QWidget::createWindowContainer()。这是 QGLWidget 的现代替代品，并且由于缺少额外的合成步骤而比 QOpenGLWidget 更快。强烈建议将此方法的使用限制在没有其他选择的情况下。请注意，此选项不适用于大多数嵌入式和移动平台，并且已知在某些桌面平台（例如 macOS）上也存在问题。稳定的跨平台解决方案始终是 QOpenGLWidget。

## 立体渲染

从6.5开始QOpenGLWidget开始支持立体渲染。要启用它，请设置QSurfaceFormat::StereoBuffers在创建窗口之前使用 QSurfaceFormat::SetDefaultFormat() 全局标记。

**注意：**使用setFormat() 不一定有效，因为该标志的内部处理方式不同。

这将触发 paintGL() 每帧调用两次，每帧调用一次 QOpenGLWidget::TargetBuffer。在 paintGL（），称呼 currentTargetBuffer() 查询当前正在绘制哪一个。

**注意：**为了更好地控制左右颜色缓冲区，请考虑使用QOpenGLWindow+QWidget::createWindowContainer（） 反而。

**注意：**这种类型的 3D 渲染有一定的硬件要求，例如显卡需要设置立体支持。

*OpenGL 是 Silicon Graphics, Inc. 在美国和其他国家/地区的商标。*

**也可以看看**QOpenGLFunctions,QOpenGLWindow,Qt::AA_ShareOpenGLContexts， 和UpdateBehavior。

# 会员类型文档

### *[since 6.5]enum QOpenGLWidget::TargetBuffer*

指定启用立体渲染时要使用的缓冲区，通过设置切换QSurfaceFormat::StereoBuffers。

**注意：** LeftBuffer 始终是默认值，并在图形驱动程序禁用或不支持立体渲染时用作后备值。

| 持续的 | 价值 |
| --- | --- |
| `QOpenGLWidget::LeftBuffer` | `0` |
| `QOpenGLWidget::RightBuffer` | `1` |

该枚举是在 Qt 6.5 中引入或修改的。

### *enum QOpenGLWidget::UpdateBehavior*

该枚举描述了更新语义QOpenGLWidget。

| 持续的 | 价值 | 描述 |
| --- | --- | --- |
| `QOpenGLWidget::NoPartialUpdate` | `0` | QOpenGLWidget将在之后丢弃颜色缓冲区和辅助缓冲区的内容QOpenGLWidget被渲染到屏幕上。这与通过调用可以预期的行为相同QOpenGLContext::swapBuffers默认启用 openglQWindow作为论点。当帧缓冲区对象用作渲染目标时，NoPartialUpdate 可以在移动和嵌入式空间中常见的某些硬件架构上具有一些性能优势。使用 glDiscardFramebufferEXT（如果支持）或 glClear 使帧缓冲区对象在帧之间失效。请参阅 EXT_discard_framebuffer 的文档以获取更多信息：https://www.khronos.org/registry/gles/extensions/EXT/EXT_discard_framebuffer.txt |
| `QOpenGLWidget::PartialUpdate` | `1` | 帧缓冲区对象颜色缓冲区和辅助缓冲区在帧之间不会失效。 |

**也可以看看**updateBehavior（） 和setUpdateBehavior()。

## 成员函数文档

### *[explicit]QOpenGLWidget::QOpenGLWidget(QWidget *parent = nullptr, Qt::WindowFlags f = Qt::WindowFlags())*

构造一个小部件，它是*parent*，小部件标志设置为*f*。

### *[virtual]QOpenGLWidget::~QOpenGLWidget()*

摧毁了QOpenGLWidget例如，释放其资源。

这QOpenGLWidget的上下文在析构函数中成为当前上下文，允许安全销毁可能需要释放属于该小部件提供的上下文的 OpenGL 资源的任何子对象。

**警告：**如果您有包装 OpenGL 资源的对象（例如QOpenGLBuffer,QOpenGLShaderProgram等）作为 OpenGLWidget 子类的成员，您可能需要添加对makeCurrent() 也在该子类的析构函数中。由于 C++ 对象销毁的规则，这些对象将在调用此函数之前被*销毁（但在子类的析构函数运行之后），因此使 OpenGL 上下文在此函数中处于当前状态对于安全处置来说为时已晚。*

**也可以看看**makeCurrent。

## *[signal]void QOpenGLWidget::aboutToCompose()*

当小部件的顶级窗口即将开始合成其纹理时，会发出此信号QOpenGLWidget孩子和其他小部件。

## *[signal]void QOpenGLWidget::aboutToResize()*

当小部件的大小更改时会发出此信号，因此将重新创建帧缓冲区对象。

## *QOpenGLContext *QOpenGLWidget::context() const*

返回QOpenGLContext由该小部件使用或0尚未初始化。

**注意：**当通过以下方式重新调整小部件的父级时，小部件使用的上下文和帧缓冲区对象会发生变化setParent()。

**也可以看看**QOpenGLContext::setShareContext （） 和defaultFramebufferObject()。

## *[since 6.5]QOpenGLWidget::TargetBuffer QOpenGLWidget::currentTargetBuffer() const*

返回当前活动的目标缓冲区。默认情况下，这将是左缓冲区，右缓冲区仅在以下情况下使用QSurfaceFormat::StereoBuffers已启用。当启用立体渲染时，可以在paintGL() 了解当前正在使用什么缓冲区。paintGL() 将被调用两次，每个目标一次。

该功能是在 Qt 6.5 中引入的。

**也可以看看**paintGL()。

## *GLuint QOpenGLWidget::defaultFramebufferObject() const*

0如果尚未初始化，则返回帧缓冲区对象句柄。

**注意：** framebuffer对象属于由返回的上下文context() 并且可能无法从其他上下文访问。

**注意：**当通过以下方式重新调整小部件的父级时，小部件使用的上下文和帧缓冲区对象会发生变化setParent()。此外，帧缓冲区对象在每次调整大小时都会发生变化。

**也可以看看**context()。

## [since 6.5]GLuint QOpenGLWidget::defaultFramebufferObject(QOpenGLWidget::TargetBuffer targetBuffer) const

返回指定目标缓冲区的帧缓冲区对象句柄，或者0如果尚未初始化。

仅当以下情况时调用此重载才有意义QSurfaceFormat::StereoBuffers由硬件启用和支持。如果没有，此方法将返回默认缓冲区。

**注意：** framebuffer对象属于由返回的上下文context() 并且可能无法从其他上下文访问。当重新调整小部件的父级时，小部件使用的上下文和帧缓冲区对象会发生变化setParent()。此外，帧缓冲区对象在每次调整大小时都会发生变化。

该功能是在 Qt 6.5 中引入的。

**也可以看看**context()。

## void QOpenGLWidget::doneCurrent()

释放上下文。

大多数情况下没有必要调用此函数，因为小部件将确保调用时正确绑定和释放上下文paintGL()。

## [override virtual protected]bool QOpenGLWidget::event(QEvent *e)

重新实现：QWidget::event（QEvent *事件）。

## QSurfaceFormat QOpenGLWidget::format() const

返回此小部件及其顶级窗口使用的上下文和表面格式。

在创建小部件及其顶层、调整大小并显示之后，此函数将返回上下文的实际格式。如果平台无法满足请求，则这可能与请求的格式不同。也可以获得比要求更大的颜色缓冲区大小。

当widget的窗口和相关的OpenGL资源尚未初始化时，返回值是通过设置的格式setFormat()。

**也可以看看**setFormat（） 和context()。

## *[signal]void QOpenGLWidget::frameSwapped()*

该信号在小部件的顶级窗口完成组合并从其潜在阻塞返回后发出QOpenGLContext::swapBuffers（） 称呼。

## *QImage QOpenGLWidget::grabFramebuffer()*

渲染并返回帧缓冲区的 32 位 RGB 图像。

**注意：**这是一个潜在昂贵的操作，因为它依赖于 glReadPixels() 来读回像素。这可能会很慢并且可能会导致 GPU 管道停顿。

## *[since 6.5]QImage QOpenGLWidget::grabFramebuffer(QOpenGLWidget::TargetBuffer targetBuffer)*

渲染并返回指定目标缓冲区的帧缓冲区的 32 位 RGB 图像。此重载仅在以下情况下调用才有意义QSurfaceFormat::StereoBuffers已启用。如果禁用立体渲染或硬件不支持，则抓取右侧目标缓冲区的帧缓冲区将返回默认图像。

**注意：**这是一个潜在昂贵的操作，因为它依赖于 glReadPixels() 来读回像素。这可能会很慢并且可能会导致 GPU 管道停顿。

该功能是在 Qt 6.5 中引入的。

## *[virtual protected]void QOpenGLWidget::initializeGL()*

该虚函数在第一次调用之前被调用一次paintGL（） 或者resizeGL()。在子类中重新实现它。

此函数应设置任何所需的 OpenGL 资源。

无需致电makeCurrent() 因为当调用该函数时这已经完成了。但请注意，帧缓冲区在此阶段尚不可用，因此请避免从此处发出绘制调用。将此类调用推迟到paintGL（） 反而。

**也可以看看**paintGL（） 和resizeGL()。

## *bool QOpenGLWidget::isValid() const*

如果小部件和 OpenGL 资源（如上下文）已成功初始化，则返回*true* 。请注意，在显示小部件之前，返回值始终为 false。

## *void QOpenGLWidget::makeCurrent()*

通过将相应的上下文设置为当前上下文并绑定该上下文中的帧缓冲区对象，准备为此小部件渲染 OpenGL 内容。

大多数情况下没有必要调用该函数，因为它会在调用之前自动调用paintGL()。

**也可以看看**context(),paintGL（），和doneCurrent()。

## *[since 6.5]void QOpenGLWidget::makeCurrent(QOpenGLWidget::TargetBuffer targetBuffer)*

通过将传入缓冲区的上下文设为当前上下文并绑定该上下文中的帧缓冲区对象，准备为此小部件渲染 OpenGL 内容。

**注意：**只有在启用立体渲染时调用才有意义。如果在禁用时请求正确的缓冲区，则不会发生任何情况。

大多数情况下没有必要调用该函数，因为它会在调用之前自动调用paintGL()。

该功能是在 Qt 6.5 中引入的。

**也可以看看**context(),paintGL（），和doneCurrent()。

## *[override virtual protected]int QOpenGLWidget::metric(QPaintDevice::PaintDeviceMetric metric) const*

重新实现：QWidget::metric(QPaintDevice::PaintDeviceMetric m) const。

## *[override virtual protected]QPaintEngine *QOpenGLWidget::paintEngine() const*

重新实现：QWidget::paintEngine() const。

## *[override virtual protected]void QOpenGLWidget::paintEvent(QPaintEvent *e)*

重新实现：QWidget::paintEvent（QPaintEvent *事件）。

处理绘画事件。

呼唤QWidget::update() 将导致发送绘制事件*e*，从而调用该函数。（注意，这是异步的，并且会在从返回后的某个时刻发生update()）。经过一些准备后，该函数将调用虚拟函数paintGL() 更新内容QOpenGLWidget的帧缓冲区。然后，小部件的顶级窗口会将帧缓冲区的纹理与窗口的其余部分合成。

## *[virtual protected]void QOpenGLWidget::paintGL()*

每当需要绘制小部件时就会调用此虚拟函数。在子类中重新实现它。

无需致电makeCurrent() 因为当调用该函数时这已经完成了。

在调用此函数之前，上下文和帧缓冲区已绑定，并且通过调用 glViewport() 设置视口。框架不设置其他状态，也不执行任何清除或绘制操作。

**注意：**为了确保可移植性，不要期望在initializeGL()仍然存在。相反，设置所有必要的状态，例如，通过在 paintGL() 中调用glEnable()。这是因为某些平台（例如带有 WebGL 的 WebAssembly）在某些情况下可能对 OpenGL 上下文有限制，这可能导致使用与QOpenGLWidget也可用于其他目的。

什么时候QSurfaceFormat::StereoBuffers启用后，该函数将被调用两次 - 每个缓冲区一次。通过调用查询当前绑定的缓冲区currentTargetBuffer()。

**注意：**即使硬件不支持立体渲染，每个目标的帧缓冲区也会被绘制。实际上，只有左侧缓冲区在窗口中可见。

**也可以看看**initializeGL(),resizeGL（），和currentTargetBuffer()。

## *[override virtual protected]QPaintDevice QOpenGLWidget::redirected(QPoint **p) const*

## *[override virtual protected]void QOpenGLWidget::resizeEvent(QResizeEvent *e)*

重新实现：QWidget::resizeEvent（QResizeEvent *事件）。

处理在中传递的调整大小事件*e*事件参数。调用虚函数resizeGL()。

**注意：**避免在派生类中重写此函数。如果这不可行，请确保QOpenGLWidget的实现也会被调用。否则，底层帧缓冲区对象和相关资源将无法正确调整大小，并导致错误渲染。

## *[virtual protected]void QOpenGLWidget::resizeGL(int w, int h)*

只要调整小部件的大小，就会调用此虚拟函数。在子类中重新实现它。新的尺寸被传入*w*和*h*。

无需致电makeCurrent() 因为当调用该函数时这已经完成了。此外，帧缓冲区也被绑定。

**也可以看看**initializeGL（）和paintGL()。

## *[signal]void QOpenGLWidget::resized()*

由于调整小部件的大小而重新创建帧缓冲区对象后，会立即发出此信号。

## *void QOpenGLWidget::setFormat(const QSurfaceFormat &format)*

设置所需的表面*format*。

当未通过此函数显式设置格式时，返回的格式QSurfaceFormat::defaultFormat（） 将会被使用。这意味着当有多个 OpenGL 小部件时，对此函数的单独调用可以替换为对QSurfaceFormat::setDefaultFormat() 在创建第一个小部件之前。

**注意：** 当目的是使其他小部件可见时，通过此函数请求 alpha 缓冲区不会产生所需的结果。相反，使用Qt::WA_AlwaysStackOnTop启用半透明QOpenGLWidget下面可见其他小部件的实例。但请记住，这会破坏堆叠顺序，因此将不再可能在顶部放置其他小部件QOpenGLWidget。

**也可以看看**format(),Qt::WA_AlwaysStackOnTop， 和QSurfaceFormat::setDefaultFormat()。

## *void QOpenGLWidget::setTextureFormat(GLenum texFormat)*

设置自定义内部纹理格式*texFormat*。

使用 sRGB 帧缓冲区时，需要指定一种格式，例如GL_SRGB8_ALPHA8. 这可以通过调用该函数来实现。

**注意：** 如果在小部件已经显示并因此执行初始化之后调用该函数，则该函数无效。

**注意：** 此功能通常必须与 QSurfaceFormat::setDefaultFormat() 调用将颜色空间设置为QSurfaceFormat::sRGBColorSpace。

**也可以看看**textureFormat()。

## *void QOpenGLWidget::setUpdateBehavior(QOpenGLWidget::UpdateBehavior updateBehavior)*

将此小部件的更新行为设置为*updateBehavior*。

**也可以看看**updateBehavior()。

## *GLenum QOpenGLWidget::textureFormat() const*

如果小部件已经初始化，则返回活动的内部纹理格式；如果已设置但小部件尚未可见，或者`nullptr`如果 setTextureFormat() 未被调用，并且该小部件尚未变得可见。

**也可以看看**setTextureFormat()。

## *QOpenGLWidget::UpdateBehavior QOpenGLWidget::updateBehavior() const*

返回小部件的更新行为。

**也可以看看**setUpdateBehavior()。