QDebug Class

QDebug 类提供调试信息的输出流。更多的...

Header:	**#include < QDebug>**
CMake:	find_package(Qt6 REQUIRED COMPONENTS Core) target_link_libraries(mytarget PRIVATE Qt6::Core)
qmake:	QT += core
Inherits:	QIODeviceBase
Inherited By:	QQmlInfo

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

公共类型

enum	**VerbosityLevel { MinimumVerbosity, DefaultVerbosity, MaximumVerbosity }**

公共职能

	QDebug(**QIODevice** ***device***)
	QDebug(QString **string*)
	QDebug(QtMsgType *t*)
	QDebug(const QDebug &*o*)
	~QDebug()
bool	**autoInsertSpaces**() const
QDebug &	**maybeQuote**(char *c* = '"')
QDebug &	**maybeSpace**()
QDebug &	**noquote**()
QDebug &	**nospace**()
QDebug &	**quote**()
QDebug &	**resetFormat**()

	QDebug(QIODevice **device*)
void	**setAutoInsertSpaces**(bool *b*)
void	**setVerbosity**(int *verbosityLevel*)
QDebug &	**space**()
void	**swap**(QDebug &*other*)
QDebug &	**verbosity**(int *verbosityLevel*)
int	**verbosity**() const
QDebug &	**operator<<**(QChar *t*)
QDebug &	**operator<<**(bool *t*)
QDebug &	**operator<<**(char *t*)
QDebug &	**operator<<**(short *t*)
QDebug &	**operator<<**(unsigned short *t*)
QDebug &	**operator<<**(char16_t *t*)
QDebug &	**operator<<**(char32_t *t*)
QDebug &	**operator<<**(int *t*)
QDebug &	**operator<<**(unsigned int *t*)
QDebug &	**operator<<**(long *t*)
QDebug &	**operator<<**(unsigned long *t*)
QDebug &	**operator<<**(qint64 *t*)
QDebug &	**operator<<**(quint64 *t*)
QDebug &	**operator<<**(float *t*)
QDebug &	**operator<<**(double *t*)
QDebug &	**operator<<**(const char **t*)
QDebug &	**operator<<**(const char16_t **t*)
QDebug &	**operator<<**(const QString &*t*)
QDebug &	**operator<<**(QStringView *s*)
QDebug &	**operator<<**(QUtf8StringView *s*)
QDebug &	**operator<<**(QLatin1StringView *t*)
QDebug &	**operator<<**(const QByteArray &*t*)
QDebug &	**operator<<**(QByteArrayView *t*)

	QDebug(QIODevice **device*)
QDebug &	**operator<<**(const void **t*)
QDebug &	**operator<<**(const std::basic_string<Char, Args...> &*s*)
QDebug &	**operator<<**(std::basic_string_view<Char, Args...> *s*)
QDebug &	**operator=**(const QDebug &*other*)

静态公共成员

QString	**toString**(**T &&*object***)

相关非会员

QDebug	**operator<<**(**QDebug *debug*, const QMultiMap<Key, T> &*map***)
QDebug	**operator<<**(QDebug *debug*, const std::map<Key, T, Compare, Alloc> &*map*)
QDebug	**operator<<**(QDebug *debug*, const std::multimap<Key, T, Compare, Alloc> &*map*)
QDebug	**operator<<**(QDebug *debug*, const QHash<Key, T> &*hash*)
QDebug	**operator<<**(QDebug *debug*, const QMultiHash<Key, T> &*hash*)
QDebug	**operator<<**(QDebug *debug*, const QPair<T1, T2> &*pair*)
QDebug	**operator<<**(QDebug *debug*, const std::pair<T1, T2> &*pair*)
QDebug	**operator<<**(QDebug *debug*, const QContiguousCache< T> &*cache*)
QDebug	**operator<<**(QDebug *debug*, const QFlags< T> &*flags*)
QDebug	**operator<<**(QDebug *debug*, const QList< T> &*list*)
QDebug	**operator<<**(QDebug *debug*, const QVarLengthArray<T, P> &*array* = P)
QDebug	**operator<<**(QDebug *debug*, const std::list<T, Alloc> &*vec*)
QDebug	**operator<<**(QDebug *debug*, const std::vector<T, Alloc> &*vec*)
QDebug	**operator<<**(QDebug *debug*, const QSet< T> &*set*)
QDebug	**operator<<**(QDebug *debug*, const QMap<Key, T> &*map*)

详细说明

当开发人员需要将调试或跟踪信息写入设备、文件、字符串或控制台时，就会使用 QDebug。

基本使用

在常见情况下，调用qDebug() 函数获取默认的 QDebug 对象以用于写入调试信息。

```
    qDebug() << "Date:" << QDate::currentDate();
    qDebug() << "Types:" << QString("String") << QChar('x') << QRect(0, 10, 50, 40);
    qDebug() << "Custom coordinate type:" << coordinate;
```

这使用接受的构造函数构造了一个 QDebug 对象QtMsgType的价值QtDebugMsg。同样，qWarning(),qCritical （） 和 qFatal() 函数还返回相应消息类型的 QDebug 对象。

该类还为其他情况提供了几个构造函数，包括一个接受QFile或任何其他QIODevice用于将调试信息写入文件和其他设备的子类。构造函数接受一个QString用于写入字符串以进行显示或序列化。

格式选项

QDebug 格式化输出以便于读取。它会自动在参数之间添加空格，并在周围添加引号QString,QByteArray,QChar论据。

您可以通过以下方式调整这些选项space(),nospace （） 和quote(),noquote （） 方法。此外，QTextStream manipulators 可以通过管道传输到 QDebug 流中。

QDebugStateSaver将格式更改限制在当前范围内。resetFormat() 将选项重置为默认值。

将自定义类型写入流

许多标准类型都可以写入 QDebug 对象，并且 Qt 提供对大多数 Qt 值类型的支持。要添加对自定义类型的支持，您需要实现一个流运算符，如下例所示：

```
QDebug operator<<(QDebug debug, const Coordinate &c)
{
    QDebugStateSaver saver(debug);
    debug.nospace() << '(' << c.x() << ", " << c.y() << ')';

    return debug;
}
```

这在Debugging Techniques和Creating Custom Qt Types文件。

会员类型文档

enum QDebug::VerbosityLevel

该枚举描述了详细级别的范围。

持续的	价值
`QDebug::MinimumVerbosity`	`0`
`QDebug::DefaultVerbosity`	`2`
`QDebug::MaximumVerbosity`	`7`

也可以看看verbosity（） 和setVerbosity()。

成员函数文档

[since 6.5]template <typename Char, typename Args> QDebug &QDebug::operator<<(const std::basic_string<Char, Args...> &s)

[since 6.5]template <typename Char, typename Args> QDebug &QDebug::operator<<(std::basic_string_view<Char, Args...> s)

写入字符串或字符串视图*s*到流并返回对流的引用。

这些运算符仅参与重载决策，如果Char是以下之一

- 字符
- char8_t（仅限 C++20）
- 字符16_t
- char32_t
- wchar_t

该功能是在 Qt 6.5 中引入的。

*[explicit]QDebug::QDebug(QIODevice *device)*

构造一个写入给定的调试流*device*。

*[explicit]QDebug::QDebug(QString *string)*

构造一个写入给定的调试流*string*。

[explicit]QDebug::QDebug(QtMsgType t)

构造一个调试流，写入消息类型的处理程序*t*。

QDebug::QDebug(const QDebug &o)

构造另一个调试流的副本*o*。

QDebug::~QDebug()

刷新所有待写入的数据并销毁调试流。

bool QDebug::autoInsertSpaces() const

`true`如果这样则返回QDebug实例将自动在写入之间插入空格。

也可以看看setAutoInsertSpaces（） 和QDebugStateSaver。

QDebug &QDebug::maybeQuote(char c = '"')

写一个字符*c*到调试流，具体取决于自动插入引号的当前设置，并返回对流的引用。

默认字符是双引号"。

也可以看看quote（） 和noquote()。

QDebug &QDebug::maybeSpace()

根据自动插入空格的当前设置，将空格字符写入调试流，并返回对流的引用。

也可以看看space（） 和nospace()。

QDebug &QDebug::noquote()

禁用自动插入引号字符QChar,QString和QByteArray内容并返回对流的引用。

当禁用引用时，打印这些类型时不带引号字符，也不转义不可打印的字符。

也可以看看quote（） 和maybeQuote()。

QDebug &QDebug::nospace()

禁用自动插入空格并返回对流的引用。

也可以看看space（） 和maybeSpace()。

QDebug &QDebug::quote()

启用周围自动插入引号字符QChar,QString和QByteArray内容并返回对流的引用。

默认情况下启用引用。

也可以看看noquote（） 和maybeQuote()。

QDebug &QDebug::resetFormat()

重置流格式化选项，使其恢复到其原始构造状态。

也可以看看space（） 和quote()。

void QDebug::setAutoInsertSpaces(bool b)

启用在写入之间自动插入空格，如果*b*是真的; 否则自动插入空格将被禁用。

也可以看看autoInsertSpaces（） 和QDebugStateSaver。

void QDebug::setVerbosity(int verbosityLevel)

将流的详细程度设置为*verbosityLevel*。

允许的范围是 0 到 7。默认值为 2。

也可以看看verbosity（） 和VerbosityLevel。

QDebug &QDebug::space()

将空格字符写入调试流并返回对该流的引用。

流会记住为将来的写入启用了自动插入空格。

也可以看看nospace（） 和maybeSpace()。

void QDebug::swap(QDebug &other)

将此调试流实例交换为*other*。这个功能非常快并且永远不会失败。

[static, since 6.0]template < typename T> QString QDebug::toString(T &&object)

流*object*变成一个QDebug对字符串进行操作，然后返回该字符串的实例。

当您需要对象的文本表示形式进行调试但又不能使用operator<<. 例如：

```
    QTRY_VERIFY2(list.isEmpty(), qPrintable(QString::fromLatin1(
        "Expected list to be empty, but it has the following items:
%1")).arg(QDebug::toString(list)));
```

该字符串使用流式传输nospace()。

这个函数是在Qt 6.0中引入的。

QDebug &QDebug::verbosity(int verbosityLevel)

将流的详细程度设置为*verbosityLevel*并返回对流的引用。

允许的范围是 0 到 7。默认值为 2。

也可以看看verbosity(),setVerbosity（），和VerbosityLevel。

int QDebug::verbosity() const

返回调试流的详细程度。

流操作员可以检查该值以确定是否需要详细输出并根据级别打印更多信息。值越高表示需要更多信息。

允许的范围是 0 到 7。默认值为 2。

也可以看看setVerbosity（） 和VerbosityLevel。

QDebug &QDebug::operator<<(QChar t)

写出人物，*t*，到流并返回对流的引用。通常情况下，QDebug将控制字符和非 US-ASCII 字符打印为其 C 转义序列或其 Unicode 值 (\u1234)。要打印不可打印的字符而不进行转换，请启用noquote() 功能，但要注意一些QDebug后端可能不是 8 位干净的，并且可能无法表示t.

QDebug &QDebug::operator<<(bool t)

写入布尔值，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(char t)

写出人物，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(short t)

写入有符号短整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(unsigned short t)

然后写入无符号短整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(char16_t t)

写入 UTF-16 字符，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(char32_t t)

写入 UTF-32 字符，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(int t)

写入有符号整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(unsigned int t)

然后写入无符号整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(long t)

写入有符号长整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(unsigned long t)

然后写入无符号长整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(qint64 t)

写入有符号的 64 位整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(quint64 t)

然后写入无符号 64 位整数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(float t)

写入 32 位浮点数，*t*，到流并返回对流的引用。

QDebug &QDebug::operator<<(double t)

写入 64 位浮点数，*t*，到流并返回对流的引用。

*QDebug &QDebug::operator<<(const char *t)*

写入以 '\0' 结尾的 UTF-8 字符串，*t*，到流并返回对流的引用。该字符串在输出时不会被引用或转义。注意QDebug内部缓冲区为 UTF-16，并且可能需要使用区域设置的编解码器转换为 8 位才能使用某些后端，这可能会导致乱码输出 (mojibake)。建议限制为 US-ASCII 字符串。

*[since 6.0]QDebug &QDebug::operator<<(const char16_t *t)*

写入以 u'\0' 结尾的 UTF-16 字符串，*t*，到流并返回对流的引用。该字符串在输出时不会被引用或转义。注意QDebug内部缓冲区为 UTF-16，并且可能需要使用区域设置的编解码器转换为 8 位才能使用某些后端，这可能会导致乱码输出 (mojibake)。建议限制为 US-ASCII 字符串。

这个函数是在Qt 6.0中引入的。

QDebug &QDebug::operator<<(const QString &t)

写入字符串，*t*，到流并返回对流的引用。通常情况下，QDebug打印引号内的字符串并将不可打印字符转换为其 Unicode 值 (\u1234)。

要打印不可打印的字符而不进行转换，请启用noquote() 功能。请注意，一些QDebug后端可能不是 8 位干净的。

输出示例：

```
    QString s;

    s = "a";
    qDebug().noquote() << s;    // prints: a
    qDebug() << s;              // prints: "a"

    s = "\"a\r\n\"";
    qDebug() << s;              // prints: "\"a\r\n\""

    s = "\033";                 // escape character
    qDebug() << s;              // prints: "\u001B"

    s = "\u00AD";               // SOFT HYPHEN
    qDebug() << s;              // prints: "\u00AD"

    s = "\u00E1";               // LATIN SMALL LETTER A WITH ACUTE
    qDebug() << s;              // prints: "á"

    s = "a\u0301";              // "a" followed by COMBINING ACUTE ACCENT
    qDebug() << s;              // prints: "á";

    s = "\u0430\u0301";         // CYRILLIC SMALL LETTER A followed by COMBINING ACUTE ACCENT
    qDebug() << s;              // prints: "á"
```

QDebug &QDebug::operator<<(QStringView s)

写入字符串视图，*s*，到流并返回对流的引用。通常情况下，QDebug打印引号内的字符串并将不可打印字符转换为其 Unicode 值 (\u1234)。

要打印不可打印的字符而不进行转换，请启用noquote() 功能。请注意，一些QDebug后端可能不是 8 位干净的。

请参阅QString过载为例。

[since 6.0]QDebug &QDebug::operator<<(QUtf8StringView s)

写入字符串视图，*s*，到流并返回对流的引用。

通常情况下，QDebug打印引号内的数据并将控制字符或非 US-ASCII 字符转换为其 C 转义序列 (\xAB)。这样，输出始终是 7 位干净的，并且可以根据需要从输出中复制字符串并将其粘贴回 C++ 源中。

要打印不可打印的字符而不进行转换，请启用noquote() 功能。请注意，一些QDebug后端可能不是 8 位干净的。

这个函数是在Qt 6.0中引入的。

QDebug &QDebug::operator<<(QLatin1StringView t)

写入字符串，*t*，到流并返回对流的引用。通常情况下，QDebug打印引号内的字符串并将不可打印字符转换为其 Unicode 值 (\u1234)。

要打印不可打印的字符而不进行转换，请启用noquote() 功能。请注意，一些QDebug后端可能不是 8 位干净的。

请参阅QString过载为例。

QDebug &QDebug::operator<<(const QByteArray &t)

写入字节数组，*t*，到流并返回对流的引用。通常情况下，QDebug打印引号内的数组，并将控制字符或非 US-ASCII 字符转换为其 C 转义序列 (\xAB)。这样，输出始终是 7 位干净的，并且可以根据需要从输出中复制字符串并将其粘贴回 C++ 源中。

要打印不可打印的字符而不进行转换，请启用noquote() 功能。请注意，一些QDebug后端可能不是 8 位干净的。

输出示例：

```
QByteArray ba;

ba = "a";
qDebug().noquote() << ba;    // prints: a
qDebug() << ba;              // prints: "a"

ba = "\"a\r\n\"";
qDebug() << ba;              // prints: "\"a\r\n\""

ba = "\033";                 // escape character
```

```
qDebug() << ba;             // prints: "\x1B"

ba = "\xC3\xA1";
qDebug() << ba;             // prints: "\xC3\xA1"

ba = QByteArray("a\0b", 3);
qDebug() << ba              // prints: "\a\x00""b"
```

注意如何QDebug需要以 C 和 C++ 语言连接字符串文字的方式关闭并重新打开字符串，以便字母“b”不会被解释为先前的十六进制转义序列的一部分。

[since 6.0]QDebug &QDebug::operator<<(QByteArrayView t)

写入观察到的字节数组的数据，*t*，到流并返回对流的引用。

通常情况下，QDebug打印引号内的数据并将控制字符或非 US-ASCII 字符转换为其 C 转义序列 (\xAB)。这样，输出始终是 7 位干净的，并且可以根据需要从输出中复制字符串并将其粘贴回 C++ 源中。

要打印不可打印的字符而不进行转换，请启用noquote() 功能。请注意，一些QDebug后端可能不是 8 位干净的。

请参阅QByteArray过载为例。

这个函数是在Qt 6.0中引入的。

*QDebug &QDebug::operator<<(const void *t)*

写一个指针，*t*，到流并返回对流的引用。

QDebug &QDebug::operator=(const QDebug &other)

分配*other*调试流到此流并返回对此流的引用。

相关非会员

template <typename Key, typename T> QDebug operator<<(QDebug debug, const QMultiMap<Key, T> &map)

写入内容*map*到*debug*。和Key都T需要支持流式传输QDebug。

template <typename Key, typename T, typename Compare, typename Alloc> QDebug operator<<(QDebug debug, const std::map<Key, T, Compare, Alloc> &map)

写入内容*map*到*debug*。和Key都T需要支持流式传输QDebug。

template <typename Key, typename T, typename Compare, typename Alloc> QDebug operator<<(QDebug debug, const std::multimap<Key, T, Compare, Alloc> &map)

写入内容*map*到*debug*。和Key都T需要支持流式传输QDebug。

template <typename Key, typename T> QDebug operator<<(QDebug debug, const QHash<Key, T> &hash)

写入内容*hash*到*debug*。和Key都T需要支持流式传输QDebug。

template <typename Key, typename T> QDebug operator<<(QDebug debug, const QMultiHash<Key, T> &hash)

写入内容*hash*到*debug*。和Key都T需要支持流式传输QDebug。

template <typename T1, typename T2> QDebug operator<<(QDebug debug, const QPair<T1, T2> &pair)

写入内容*pair*到*debug*。和T1都T2需要支持流式传输QDebug。

template <typename T1, typename T2> QDebug operator<<(QDebug debug, const std::pair<T1, T2> &pair)

写入内容*pair*到*debug*。和T1都T2需要支持流式传输QDebug。

template < typename T> QDebug operator<<(QDebug debug, const QContiguousCache< T> &cache)

写入内容*cache*到*debug*。T需要支持流式传输QDebug。

template < typename T> QDebug operator<<(QDebug debug, const QFlags< T> &flags)

写*flags*到*debug*。

template < typename T> QDebug operator<<(QDebug debug, const QList< T> &list)

写入内容*list*到*debug*。T需要支持流式传输QDebug。

[since 6.3]template <typename T, qsizetype P> QDebug operator<<(QDebug debug, const QVarLengthArray<T, P> &array = P)

写入内容*array*到*debug*。T需要支持流式传输QDebug。

该功能是在 Qt 6.3 中引入的。

template <typename T, typename Alloc> QDebug operator<<(QDebug debug, const std::list<T, Alloc> &vec)

写入列表的内容*vec*到*debug*。T需要支持流式传输QDebug。

template <typename T, typename Alloc> QDebug operator<<(QDebug debug, const std::vector<T, Alloc> &vec)

写入向量的内容*vec*到*debug*。T需要支持流式传输QDebug。

template < typename T> QDebug operator<<(QDebug debug, const QSet< T> &set)

写入内容*set*到*debug*。T需要支持流式传输QDebug。

template <typename Key, typename T> QDebug operator<<(QDebug debug, const QMap<Key, T> &map)

写入内容*map*到*debug*。和Key都T需要支持流式传输QDebug。