QVariant Class

QVariant 类的作用类似于最常见的 Qt 数据类型的联合。更多的...

Header:	**#include**
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
- QVariant 是隐式共享类的一部分。

公共职能

	QVariant()
	QVariant(QMetaType *type*, const void **copy* = nullptr)
	QVariant(QRectF *val*)
	QVariant(const QEasingCurve &*val*)
	QVariant(const QJsonDocument &*val*)
	QVariant(const QPersistentModelIndex &*val*)
	QVariant(const char **val*)
	QVariant(QLatin1StringView *val*)
	QVariant(int *val*)
	QVariant(uint *val*)
	QVariant(qlonglong *val*)
	QVariant(qulonglong *val*)
	QVariant(bool *val*)
	QVariant(double *val*)
	QVariant(float *val*)
	QVariant(QChar *c*)

	QVariant()
	QVariant(QDate *val*)
	QVariant(QTime *val*)
	QVariant(const QBitArray &*val*)
	QVariant(const QByteArray &*val*)
	QVariant(const QDateTime &*val*)
	QVariant(const QHash<QString, QVariant> &*val*)
	QVariant(const QJsonArray &*val*)
	QVariant(const QJsonObject &*val*)
	QVariant(const QList &*val*)
	QVariant(const QLocale &*l*)
	QVariant(const QMap<QString, QVariant> &*val*)
	QVariant(const QRegularExpression &*re*)
	QVariant(const QString &*val*)
	QVariant(const QStringList &*val*)
	QVariant(const QUrl &*val*)
	QVariant(const QJsonValue &*val*)
	QVariant(const QModelIndex &*val*)
	QVariant(QUuid *val*)
	QVariant(QSize *val*)
	QVariant(QSizeF *val*)
	QVariant(QPoint *val*)
	QVariant(QPointF *val*)
	QVariant(QLine *val*)
	QVariant(QLineF *val*)
	QVariant(QRect *val*)
	QVariant(const QVariant &*p*)
	QVariant(QVariant &&*other*)
	~QVariant()
bool	**canConvert**(QMetaType *type*) const

	QVariant()
bool	**canConvert**() const
bool	**canView**() const
void	**clear**()
const void *	**constData**() const
bool	**convert**(QMetaType *targetType*)
void *	**data**()
const void *	**data**() const
bool	**isNull**() const
bool	**isValid**() const
QMetaType	**metaType**() const
void	**setValue**(T &&*value*)
void	**setValue**(const QVariant &*value*)
void	**setValue**(QVariant &&*value*)
void	**swap**(QVariant &*other*)
QBitArray	**toBitArray**() const
bool	**toBool**() const
QByteArray	**toByteArray**() const
QChar	**toChar**() const
QDate	**toDate**() const
QDateTime	**toDateTime**() const
double	**toDouble**(bool **ok* = nullptr) const
QEasingCurve	**toEasingCurve**() const
float	**toFloat**(bool **ok* = nullptr) const
QHash<QString, QVariant>	**toHash**() const
int	**toInt**(bool **ok* = nullptr) const
QJsonArray	**toJsonArray**() const
QJsonDocument	**toJsonDocument**() const
QJsonObject	**toJsonObject**() const
QJsonValue	**toJsonValue**() const

	QVariant()
QLine	**toLine**() const
QLineF	**toLineF**() const
QList	**toList**() const
QLocale	**toLocale**() const
qlonglong	**toLongLong**(bool **ok* = nullptr) const
QMap<QString, QVariant>	**toMap**() const
QModelIndex	**toModelIndex**() const
QPersistentModelIndex	**toPersistentModelIndex**() const
QPoint	**toPoint**() const
QPointF	**toPointF**() const
qreal	**toReal**(bool **ok* = nullptr) const
QRect	**toRect**() const
QRectF	**toRectF**() const
QRegularExpression	**toRegularExpression**() const
QSize	**toSize**() const
QSizeF	**toSizeF**() const
QString	**toString**() const
QStringList	**toStringList**() const
QTime	**toTime**() const
uint	**toUInt**(bool **ok* = nullptr) const
qulonglong	**toULongLong**(bool **ok* = nullptr) const
QUrl	**toUrl**() const
QUuid	**toUuid**() const
int	**typeId**() const
const char *	**typeName**() const
int	**userType**() const
T	**value**() const
T	**view**()
QVariant &	**operator=**(const QVariant &*variant*)

	QVariant()
QVariant &	**operator=**(QVariant &&*other*)

静态公共成员

QPartialOrdering	**compare**(**const QVariant &*lhs*, const QVariant &*rhs*)**
QVariant	**fromStdVariant**(const std::variant<Types...> &*value*)
QVariant	**fromValue**(const T &*value*)

相关非成员

	QVariantHash
	QVariantList
	QVariantMap
T	**qvariant_cast**(const QVariant &*value*)
bool	**operator!=**(const QVariant &*v1*, const QVariant &*v2*)
QDataStream &	**operator<<**(QDataStream &*s*, const QVariant &*p*)
bool	**operator==**(const QVariant &*v1*, const QVariant &*v2*)
QDataStream &	**operator>>**(QDataStream &*s*, QVariant &*p*)

详细说明

由于 C++ 禁止联合包含具有非默认构造函数或析构函数的类型，因此大多数有趣的 Qt 类都不能在联合中使用。如果没有 QVariant，这将是一个问题QObject::property() 以及数据库工作等。

QVariant 对象保存单个值typeId() 一次。（某些类型是多值的，例如字符串列表。）您可以找出变体保存的类型 T，然后使用以下命令将其转换为不同的类型convert()，使用 toT() 函数之一获取其值（例如，toSize()），并使用检查类型是否可以转换为特定类型canConvert()。

名为 toT() 的方法（例如，toInt(),toString()) 是常量。如果您询问存储的类型，它们会返回存储对象的副本。如果您要求可以从存储的类型生成的类型，toT() 会复制并转换对象本身并保持不变。如果您请求的类型无法从存储的类型生成，则结果取决于类型；详细信息请参见函数文档。

下面是一些示例代码来演示 QVariant 的使用：

```
QDataStream out(...);
QVariant v(123);                // The variant now contains an int
int x = v.toInt();              // x = 123
out << v;                       // Writes a type tag and an int to out
```

```
v = QVariant(tr("hello"));      // The variant now contains a QString
int y = v.toInt();              // y = 0 since v cannot be converted to an int
QString s = v.toString();       // s = tr("hello")  (see QObject::tr())
out << v;                       // Writes a type tag and a QString to out
...
QDataStream in(...);            // (opening the previously written stream)
in >> v;                        // Reads an Int variant
int z = v.toInt();              // z = 123
qDebug("Type is %s",            // prints "Type is int"
        v.typeName());
v = v.toInt() + 100;            // The variant now holds the value 223
v = QVariant(QStringList());    // The variant now holds a QStringList
```

您甚至可以存储QList 和QMap<QString, QVariant> 值中的一个变体，这样你就可以轻松构造任意类型的任意复杂的数据结构。这是非常强大和通用的，但可能证明比在标准数据结构中存储特定类型的内存和速度效率更低。

QVariant 还支持空值的概念。如果变体不包含初始化值或包含空指针，则该变体为 null。

```
QVariant x;                              // x.isNull() == true
QVariant y = QVariant::fromValue(nullptr); // y.isNull() == true
```

QVariant 可以扩展以支持除上述提到的类型之外的其他类型QMetaType::Type枚举。看Creating Custom Qt Types了解详情。

关于 GUI 类型的注释

由于QVariant是Qt Core模块的一部分，它无法提供Qt GUI中定义的数据类型的转换函数，例如QColor,QImage， 和QPixmap。换句话说，没有任何`toColor()`功能。相反，您可以使用QVariant::value （） 或者qvariant_cast() 模板函数。例如：

```
QVariant variant;
...
QColor color = variant.value<QColor>();
```

逆转换（例如，从QColor到 QVariant）对于 QVariant 支持的所有数据类型都是自动的，包括 GUI 相关类型：

```
QColor color = palette().background().color();
QVariant variant = color;
```

连续使用canConvert()和convert()

使用时canConvert （） 和convert()连续地可得canConvert() 返回 true，但是convert() 返回 false。这通常是因为canConvert() 仅报告 QVariant 在给定合适数据的类型之间进行转换的一般能力；仍然可以提供实际上无法转换的数据。

例如，`canConvert(QMetaType::fromType<int>())`当调用包含字符串的变量时将返回 true，因为原则上 QVariant 能够将数字字符串转换为整数。但是，如果字符串包含非数字字符，则无法将其转换为整数，并且任何转换尝试都会失败。因此，让两个函数都返回 true 对于成功转换非常重要。

也可以看看QMetaType。

成员函数文档

int QVariant::typeId() const

int QVariant::userType() const

返回存储在变体中的值的存储类型。这与metaType（）。ID（）。

也可以看看metaType()。

*const void *QVariant::constData() const*

*const void *QVariant::data() const*

以无法写入的通用 void* 形式返回指向所包含对象的指针。

也可以看看QMetaType。

QVariant::QVariant()

构造一个无效的变体。

*[explicit]QVariant::QVariant(QMetaType type, const void *copy = nullptr)*

构造类型的变体*type*，并初始化为*copy*如果*copy*不是`nullptr`。

请注意，您必须传递要存储的变量的地址。

通常，你永远不必使用这个构造函数，使用QVariant::fromValue`QMetaType::VoidStar`() 而是根据, 和表示的指针类型构造变体`QMetaType::QObjectStar`。

如果*type*不支持复制和默认构造，变体将无效。

也可以看看QVariant::fromValue（） 和QMetaType::Type。

QVariant::QVariant(QRectF val)

构造一个新的变体，其矩形值为*val*。

QVariant::QVariant(const QEasingCurve &val)

构造一个具有缓动曲线值的新变体，*val*。

QVariant::QVariant(const QJsonDocument &val)

使用 json 文档值构造一个新变体，*val*。

QVariant::QVariant(const QPersistentModelIndex &val)

构造一个新的变体QPersistentModelIndex价值，*val*。

*QVariant::QVariant(const char *val)*

构造一个新的变量，其字符串值为*val*。该变体创建了一个深层副本*val*变成一个QString假设输入采用 UTF-8 编码*val*。

注意*val*被转换为QString用于存储在变体中和QVariant::userType（） 将返回QMetaType::QString对于变体。

QT_NO_CAST_FROM_ASCII您可以通过在编译应用程序时进行定义来禁用此运算符。

QVariant::QVariant(QLatin1StringView val)

构造一个新的变体QString来自 Latin-1 字符串的值*val*。

QVariant::QVariant(int val)

构造一个具有整数值的新变体，*val*。

QVariant::QVariant(uint val)

使用无符号整数值构造一个新变体，*val*。

QVariant::QVariant(qlonglong val)

构造一个具有 long long 整数值的新变体，*val*。

QVariant::QVariant(qulonglong val)

使用无符号长整型值构造一个新变体，*val*。

QVariant::QVariant(bool val)

构造一个具有布尔值的新变体，*val*。

QVariant::QVariant(double val)

构造一个具有浮点值的新变体，*val*。

QVariant::QVariant(float val)

构造一个具有浮点值的新变体，*val*。

QVariant::QVariant(QChar c)

使用 char 值构造一个新变体，*c*。

QVariant::QVariant(QDate val)

构造一个具有日期值的新变体，*val*。

QVariant::QVariant(QTime val)

构造一个具有时间值的新变体，*val*。

QVariant::QVariant(const QBitArray &val)

使用位数组值构造一个新变体，*val*。

QVariant::QVariant(const QByteArray &val)

使用字节数组值构造一个新变体，*val*。

QVariant::QVariant(const QDateTime &val)

构造一个具有日期/时间值的新变体，*val*。

QVariant::QVariant(const QHash<QString, QVariant> &val)

构造一个新的变体，其哈希值为QVariants,*val*。

QVariant::QVariant(const QJsonArray &val)

使用 json 数组值构造一个新变体，*val*。

QVariant::QVariant(const QJsonObject &val)

使用 json 对象值构造一个新变体，*val*。

QVariant::QVariant(const QList< QVariant> &val)

使用列表值构造一个新变体，*val*。

QVariant::QVariant(const QLocale &l)

构造一个具有区域设置值的新变体，*l*。

QVariant::QVariant(const QMap<QString, QVariant> &val)

使用以下地图构造一个新变体QVariants,*val*。

QVariant::QVariant(const QRegularExpression &re)

使用正则表达式值构造一个新变体*re*。

QVariant::QVariant(const QString &val)

使用字符串值构造一个新变体，*val*。

QVariant::QVariant(const QStringList &val)

使用字符串列表值构造一个新变体，*val*。

QVariant::QVariant(const QUrl &val)

构造一个新的变体，其 url 值为*val*。

QVariant::QVariant(const QJsonValue &val)

使用 json 值构造一个新变体，*val*。

QVariant::QVariant(const QModelIndex &val)

构造一个新的变体QModelIndex价值，*val*。

QVariant::QVariant(QUuid val)

使用 uuid 值构造一个新变体，*val*。

QVariant::QVariant(QSize val)

构造一个新的变体，其大小值为*val*。

QVariant::QVariant(QSizeF val)

构造一个新的变体，其大小值为*val*。

QVariant::QVariant(QPoint val)

构造一个新的变体，其点值为*val*。

QVariant::QVariant(QPointF val)

构造一个新的变体，其点值为*val*。

QVariant::QVariant(QLine val)

构造一个新的变体，其行值为*val*。

QVariant::QVariant(QLineF val)

构造一个新的变体，其行值为*val*。

QVariant::QVariant(QRect val)

构造一个新的变体，其矩形值为*val*。

QVariant::QVariant(const QVariant &p)

构造变体的副本，*p*，作为参数传递给此构造函数。

QVariant::QVariant(QVariant &&other)

移动构造一个 QVariant 实例，使其指向与*other*正在指着。

QVariant::~QVariant()

摧毁了QVariant以及所包含的对象。

[since 6.0]bool QVariant::canConvert(QMetaType type) const

返回true变体的类型是否可以转换为请求的类型，*type*。这种转换是在调用时自动完成的toInt(),toBool（），...方法。

这个函数是在Qt 6.0中引入的。

也可以看看QMetaType::canConvert()。

template < typename T> bool QVariant::canConvert() const

返回变true体是否可以转换为模板类型T，否则返回 false。

例子：

```
QVariant v = 42;

v.canConvert<int>();              // returns true
v.canConvert<QString>();          // returns true

MyCustomStruct s;
v.setValue(s);

v.canConvert<int>();              // returns false
v.canConvert<MyCustomStruct>();   // returns true
```

AQVariant包含一个指向派生类型的指针QObject如果一个函数也将返回 trueqobject_cast到模板类型T就会成功。请注意，这仅适用于QObject使用的子类Q_OBJECT宏。

也可以看看convert()。

template < typename T> bool QVariant::canView() const

返回是否可以在此变体上创建`true`模板类型的可变视图，否则返回。`T``false`

也可以看看value()。

void QVariant::clear()

将此变体转换为类型QMetaType::UnknownType并释放所有使用的资源。

[static, since 6.0]QPartialOrdering QVariant::compare(const QVariant &lhs, const QVariant &rhs)

比较对象*lhs*和*rhs*用于订购。

退货 QPartialOrdering::Unordered 如果不支持比较或值无序。否则，返回QPartialOrdering::Less,QPartialOrdering::Equivalent或者QPartialOrdering::Greater如果*lhs*小于、等于或大于*rhs*，分别。

如果变体包含具有不同元类型的数据，则这些值被视为无序，除非它们都是数字或指针类型，其中将使用常规数字或指针比较规则。

注意：：如果进行数值比较并且至少有一个值为 NaN，QPartialOrdering::Unordered被返回。

如果两个变体都包含相同元类型的数据，该方法将使用QMetaType::compare方法来确定两个变体的顺序，这也表明它无法在两个值之间建立顺序。

这个函数是在Qt 6.0中引入的。

也可以看看QMetaType::compare（） 和QMetaType::isOrdered()。

[since 6.0]bool QVariant::convert(QMetaType targetType)

将变体转换为请求的类型，*targetType*。如果无法完成转换，变体仍会更改为请求的类型，但会处于清除的空状态，类似于由QVariant（类型）。

`true`如果变体的当前类型已成功转换，则返回；否则返回`false`.

AQVariant包含一个指向派生类型的指针QObject如果一个函数也将转换并返回 trueqobject_cast到所描述的类型*targetType*会成功的。请注意，这仅适用于QObject使用的子类Q_OBJECT宏。

注意：由于未初始化或先前转换失败而转换为 null 的 QVariants 将始终失败，更改类型，保持 null 并返回`false`。

这个函数是在Qt 6.0中引入的。

也可以看看canConvert（） 和clear()。

*void *QVariant::data()*

以可写入的通用 void* 形式返回指向所包含对象的指针。

该函数分离QVariant。当被呼叫时null-QVariant， 这QVariant调用后不会为空。

也可以看看QMetaType。

[static]template QVariant QVariant::fromStdVariant(const std::variant<Types...> &value)

返回一个QVariant与活动变体的类型和值*value*。如果活动类型是 std::monostate 默认值QVariant被返回。

注意：使用此方法，您不需要将变体注册为 Qt 元类型，因为 std::variant 在存储之前已解析。然而，应该注册组件类型。

也可以看看fromValue()。

[static]template < typename T> QVariant QVariant::fromValue(const T &value)

返回一个QVariant包含一份副本*value*。行为完全像setValue（） 否则。

例子：

```
MyCustomStruct s;
return QVariant::fromValue(s);
```

也可以看看setValue（） 和value()。

bool QVariant::isNull() const

`true`如果这是一个 null 变量，则返回，否则返回 false。

如果变量不包含初始化值或空指针，则该变量被视为 null。

注意：此行为已从 Qt 5 开始更改，如果变体包含内置类型的对象，并且 isNull() 方法为该对象返回 true，则 isNull() 也会返回 true。

也可以看看convert()。

bool QVariant::isValid() const

`true`如果该变体的存储类型不是则返回QMetaType::UnknownType; 否则返回`false`.

[since 6.0]QMetaType QVariant::metaType() const

返回QMetaType存储在变体中的值。

这个函数是在Qt 6.0中引入的。

template <typename T, typename> void QVariant::setValue(T &&value)

存储一个副本*value*。如果T是一个类型QVariant不支持,QMetaType用于存储值。如果出现以下情况，将会发生编译错误QMetaType不处理类型。

例子:

```
QVariant v;

v.setValue(5);
int i = v.toInt();         // i is now 5
QString s = v.toString();  // s is now "5"

MyCustomStruct c;
v.setValue(c);

...

MyCustomStruct c2 = v.value<MyCustomStruct>();
```

也可以看看value(),fromValue（），和canConvert()。

void QVariant::setValue(const QVariant &value)

副本*value*在此之上QVariant。相当于简单的赋值*value*对此QVariant。

void QVariant::setValue(QVariant &&value)

动作*value*在此之上QVariant。相当于简单的移动赋值*value*对此QVariant。

void QVariant::swap(QVariant &other)

掉期变体*other*与此变体。这个操作非常快并且永远不会失败。

QBitArray QVariant::toBitArray() const

将变体返回为QBitArray如果变体有userType()QMetaType::QBitArray; 否则返回一个空位数组。

也可以看看canConvert（） 和convert()。

bool QVariant::toBool() const

如果变体具有，则将变体作为布尔值返回userType() 布尔。

`true` 如果变体有则返回userType()QMetaType::Bool,QMetaType::QChar,QMetaType::Double,QMetaType::Int,QMetaType::LongLong,QMetaType::UInt， 或者QMetaType::ULongLong并且该值非零， 或者如果变体具有类型QMetaType::QString或者QMetaType::QByteArray并且其小写内容不是以下之一：空、“0”或“false”；否则返回`false`.

也可以看看canConvert（） 和convert()。

QByteArray QVariant::toByteArray() const

将变体返回为QByteArray如果变体有userType()QMetaType::QByteArray或者QMetaType::QString （使用转换QString::fromUtf8()); 否则返回一个空字节数组。

也可以看看canConvert（） 和convert()。

QChar QVariant::toChar() const

将变体返回为QChar如果变体有userType()QMetaType::QChar,QMetaType::Int， 或者QMetaType::UInt; 否则返回无效QChar。

也可以看看canConvert（） 和convert()。

QDate QVariant::toDate() const

将变体返回为QDate如果变体有userType()QMetaType::QDate,QMetaType::QDateTime， 或者QMetaType::QString; 否则返回无效日期。

如果类型()是QMetaType::QString，如果字符串不能被解析为一个无效的日期将返回Qt::ISODate格式化日期。

也可以看看canConvert（） 和convert()。

QDateTime QVariant::toDateTime() const

将变体返回为QDateTime如果变体有userType()QMetaType::QDateTime,QMetaType::QDate， 或者QMetaType::QString; 否则返回无效的日期/时间。

如果类型()是QMetaType::QString，如果字符串不能被解析为一个无效的日期/时间，将会返回一个无效的日期/时间Qt::ISODate格式化日期/时间。

也可以看看canConvert（） 和convert()。

*double QVariant::toDouble(bool *ok = nullptr) const*

如果变体具有，则将变体作为双精度值返回userType()QMetaType::Double,QMetaType::Float,QMetaType::Bool,QMetaType::QByteArray,QMetaType::Int,QMetaType::LongLong,QMetaType::QString,QMetaType::UInt， 或者QMetaType::ULongLong; 否则返回 0.0。

如果*ok*是非空的：**ok*如果该值可以转换为双精度型，则设置为 true；否则**ok*设置为 false。

也可以看看canConvert（） 和convert()。

QEasingCurve QVariant::toEasingCurve() const

将变体返回为QEasingCurve如果变体有userType()QMetaType::QEasingCurve; 否则返回默认缓动曲线。

也可以看看canConvert（） 和convert()。

*float QVariant::toFloat(bool *ok = nullptr) const*

如果变体有，则以浮点形式返回变体userType()QMetaType::Double,QMetaType::Float,QMetaType::Bool,QMetaType::QByteArray,QMetaType::Int,QMetaType::LongLong,QMetaType::QString,QMetaType::UInt， 或者QMetaType::ULongLong; 否则返回 0.0。

如果*ok*是非空的：**ok*如果该值可以转换为双精度型，则设置为 true；否则**ok*设置为 false。

也可以看看canConvert（） 和convert()。

QHash<QString, QVariant> QVariant::toHash() const

将变体返回为QHash[QString](https://doc-qt-io.translate.goog/qt-6/qstring.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp),[QVariant](https://doc-qt-io.translate.goog/qt-6/qvariant.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp) 如果变体具有 type()QMetaType::QVariantHash; 否则返回一个空地图。

也可以看看canConvert（） 和convert()。

*int QVariant::toInt(bool *ok = nullptr) const*

如果变体有，则以 int 形式返回变体 userType()QMetaType::Int,QMetaType::Bool,QMetaType::QByteArray,QMetaType::QChar,QMetaType::Double,QMetaType::LongLong,QMetaType::QString,QMetaType::UInt， 或者QMetaType::ULongLong; 否则返回 0。

如果*ok*是非空的：**ok*如果该值可以转换为 int，则设置为 true；否则**ok*设置为 false。

警告：如果该值可转换为QMetaType::LongLong但太大而无法用 int 表示，因此产生的算术溢出不会反映在*ok*。一个简单的解决方法是使用QString::toInt()。

也可以看看canConvert（） 和convert()。

QJsonArray QVariant::toJsonArray() const

将变体返回为QJsonArray如果变体有userType()QJsonArray; 否则返回默认构造的QJsonArray。

也可以看看canConvert（） 和convert()。

QJsonDocument QVariant::toJsonDocument() const

将变体返回为QJsonDocument如果变体有userType()QJsonDocument; 否则返回默认构造的QJsonDocument。

也可以看看canConvert（） 和convert()。

QJsonObject QVariant::toJsonObject() const

将变体返回为QJsonObject如果变体有userType()QJsonObject; 否则返回默认构造的QJsonObject。

也可以看看canConvert（） 和convert()。

QJsonValue QVariant::toJsonValue() const

将变体返回为QJsonValue如果变体有userType()QJsonValue; 否则返回默认构造的QJsonValue。

也可以看看canConvert（） 和convert()。

QLine QVariant::toLine() const

将变体返回为QLine如果变体有userType()QMetaType::QLine; 否则返回无效QLine。

也可以看看canConvert（） 和convert()。

QLineF QVariant::toLineF() const

将变体返回为QLineF如果变体有userType()QMetaType::QLineF; 否则返回无效QLineF。

也可以看看canConvert（） 和convert()。

QList[QVariant](https://doc-qt-io.translate.goog/qt-6/qvariant.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp#QVariant) QVariant::toList() const

如果变体具有，则将变体作为 QVariantList 返回userType()QMetaType::QVariantList。如果没有，QVariant将尝试将类型转换为列表，然后返回它。对于已将转换器注册到 QVariantList 或使用以下命令声明为顺序容器的任何类型，这都会成功Q_DECLARE_SEQUENTIAL_CONTAINER_METATYPE。如果这些条件都不成立，则该函数将返回一个空列表。

也可以看看canConvert（） 和convert()。

QLocale QVariant::toLocale() const

将变体返回为QLocale如果变体有userType()QMetaType::QLocale; 否则返回无效QLocale。

也可以看看canConvert（） 和convert()。

*qlonglong QVariant::toLongLong(bool *ok = nullptr) const*

如果变体具有，则将变体作为 long long int 返回userType()QMetaType::LongLong,QMetaType::Bool,QMetaType::QByteArray,QMetaType::QChar,QMetaType::Double,QMetaType::Int,QMetaType::QString,QMetaType::UInt，或者QMetaType::ULongLong; 否则返回 0。

如果*ok*为非空：*``ok如果该值可以转换为 int，则设置为 true；否则*``ok设置为 false。

也可以看看canConvert（） 和convert()。

QMap<QString, QVariant> QVariant::toMap() const

如果变体具有 type()，则将变体作为 QVariantMap 返回QMetaType::QVariantMap。如果没有，QVariant将尝试将类型转换为映射，然后返回它。对于已将转换器注册到 QVariantMap 或使用以下命令声明为关联容器的任何类型，这都会成功Q_DECLARE_ASSOCIATIVE_CONTAINER_METATYPE。如果这些条件都不成立，则该函数将返回一个空地图。

也可以看看canConvert（） 和convert()。

QModelIndex QVariant::toModelIndex() const

将变体返回为QModelIndex如果变体有userType()QModelIndex; 否则返回默认构造的QModelIndex。

也可以看看canConvert(),convert（），和toPersistentModelIndex()。

QPersistentModelIndex QVariant::toPersistentModelIndex() const

将变体返回为 QPersistentModelIndex 如果变体有 userType()QPersistentModelIndex; 否则返回默认构造的 QPersistentModelIndex。

也可以看看canConvert(),convert（），和toModelIndex()。

QPoint QVariant::toPoint() const

将变体返回为QPoint如果变体有userType()QMetaType::QPoint或者QMetaType::QPointF; 否则返回 nullQPoint。

也可以看看canConvert（） 和convert()。

QPointF QVariant::toPointF() const

将变体返回为QPointF如果变体有userType()QMetaType::QPoint或者QMetaType::QPointF; 否则返回 nullQPointF。

也可以看看canConvert（） 和convert()。

*qreal QVariant::toReal(bool *ok = nullptr) const*

如果变体有，则将变体作为 qreal 返回userType()QMetaType::Double,QMetaType::Float,QMetaType::Bool,QMetaType::QByteArray,QMetaType::Int,QMetaType::LongLong,QMetaType::QString,QMetaType::UInt， 或者QMetaType::ULongLong; 否则返回 0.0。

如果*ok*是非空的：**ok*如果该值可以转换为双精度型，则设置为 true；否则**ok*设置为 false。

也可以看看canConvert（） 和convert()。

QRect QVariant::toRect() const

将变体返回为QRect如果变体有userType()QMetaType::QRect; 否则返回无效QRect。

也可以看看canConvert（） 和convert()。

QRectF QVariant::toRectF() const

将变体返回为QRectF如果变体有userType()QMetaType::QRect或者QMetaType::QRectF; 否则返回无效QRectF。

也可以看看canConvert（） 和convert()。

QRegularExpression QVariant::toRegularExpression() const

将变体返回为QRegularExpression如果变体有userType()QRegularExpression; 否则返回空QRegularExpression。

也可以看看canConvert（） 和convert()。

QSize QVariant::toSize() const

将变体返回为QSize如果变体有userType()QMetaType::QSize; 否则返回无效QSize。

也可以看看canConvert（） 和convert()。

QSizeF QVariant::toSizeF() const

将变体返回为QSizeF如果变体有userType()QMetaType::QSizeF; 否则返回无效QSizeF。

也可以看看canConvert（） 和convert()。

QString QVariant::toString() const

将变体返回为QString如果变体有一个userType（） 包括但不仅限于：

QMetaType::QString,QMetaType::Bool,QMetaType::QByteArray,QMetaType::QChar,QMetaType::QDate,QMetaType::QDateTime,QMetaType::Double,QMetaType::Int,QMetaType::LongLong,QMetaType::QStringList,QMetaType::QTime,QMetaType::UInt， 或者QMetaType::ULongLong。

对不受支持的变体调用 QVariant::toString() 会返回空字符串。

也可以看看canConvert（） 和convert()。

QStringList QVariant::toStringList() const

将变体返回为QStringList如果变体有userType()QMetaType::QStringList,QMetaType::QString， 或者QMetaType::QVariantList可以转换为的类型QString; 否则返回一个空列表。

也可以看看canConvert（） 和convert()。

QTime QVariant::toTime() const

将变体返回为QTime如果变体有userType()QMetaType::QTime,QMetaType::QDateTime， 或者QMetaType::QString; 否则返回无效时间。

如果类型()是QMetaType::QString，如果字符串无法解析为a，则返回无效时间Qt::ISODate格式化时间。

也可以看看canConvert（） 和convert()。

uint QVariant::toUInt(bool *ok = nullptr) const

如果变体有，则将变体作为无符号整数返回userType()QMetaType::UInt,QMetaType::Bool,QMetaType::QByteArray,QMetaType::QChar,QMetaType::Double,QMetaType::Int,QMetaType::LongLong,QMetaType::QString， 或者QMetaType::ULongLong; 否则返回 0。

如果*ok*是非空的：**ok*如果该值可以转换为无符号整数，则设置为 true；否则**ok*设置为 false。

警告：如果该值可转换为QMetaType::ULongLong但太大而无法用 unsigned int 表示，因此产生的算术溢出不会反映在*ok*。一个简单的解决方法是使用QString::toUInt()。

也可以看看canConvert（） 和convert()。

*qulonglong QVariant::toULongLong(bool *ok = nullptr) const*

如果变体具有 type()，则将变体作为 unsigned long long int 返回QMetaType::ULongLong,QMetaType::Bool,QMetaType::QByteArray,QMetaType::QChar,QMetaType::Double,QMetaType::Int,QMetaType::LongLong,QMetaType::QString， 或者QMetaType::UInt; 否则返回 0。

如果*ok*是非空的：**ok*如果该值可以转换为 int，则设置为 true；否则**ok*设置为 false。

也可以看看canConvert（） 和convert()。

QUrl QVariant::toUrl() const

将变体返回为QUrl如果变体有userType()QMetaType::QUrl; 否则返回无效QUrl。

也可以看看canConvert（） 和convert()。

QUuid QVariant::toUuid() const

将变体返回为QUuid如果变体有 type()QMetaType::QUuid,QMetaType::QByteArray或者QMetaType::QString; 否则返回默认构造的QUuid。

也可以看看canConvert（） 和convert()。

*const char *QVariant::typeName() const*

返回存储在变体中的类型的名称。返回的字符串描述了用于存储数据的 C++ 数据类型：例如，“QFont“，”QString”，或“QVariantList”。无效的变体返回 0。

template < typename T> T QVariant::value() const

返回转换为模板类型的存储值T。称呼canConvert() 来判断某个类型是否可以转换。如果该值无法转换，则default-constructed value将被退回。

如果该类型T支持QVariant，该函数的行为完全相同toString(),toInt（） ETC。

例子：

```
QVariant v;

MyCustomStruct c;
if (v.canConvert<MyCustomStruct>())
    c = v.value<MyCustomStruct>();

v = 7;
int i = v.value<int>();                          // same as v.toInt()
QString s = v.value<QString>();                  // same as v.toString(), s is now "7"
MyCustomStruct c2 = v.value<MyCustomStruct>(); // conversion failed, c2 is empty
```

如果QVariant包含一个指向派生类型的指针QObject那么T可能是任何QObject类型。如果指针存储在QVariant可qobject_cast到T，然后返回结果。否则nullptr返回。请注意，这仅适用于QObject使用的子类Q_OBJECT宏。

如果QVariant包含一个顺序容器并且T是QVariantList，容器中的元素会被转换成QVariants 并作为 QVariantList 返回。

```
QList<int> intList = {7, 11, 42};

QVariant variant = QVariant::fromValue(intList);
if (variant.canConvert<QVariantList>()) {
    QSequentialIterable iterable = variant.value<QSequentialIterable>();
    // Can use foreach:
    foreach (const QVariant &v, iterable) {
        qDebug() << v;
    }
    // Can use C++11 range-for:
    for (const QVariant &v : iterable) {
        qDebug() << v;
    }
    // Can use iterators:
    QSequentialIterable::const_iterator it = iterable.begin();
    const QSequentialIterable::const_iterator end = iterable.end();
    for ( ; it != end; ++it) {
        qDebug() << *it;
    }
}
```

也可以看看setValue(),fromValue(),canConvert（），和Q_DECLARE_SEQUENTIAL_CONTAINER_METATYPE()。

template < typename T> T QVariant::view()

T返回存储值的模板类型的可变视图。称呼canView() 来查明是否支持这种观点。如果无法创建此类视图，则返回转换为模板类型的存储值T。称呼canConvert() 来判断某个类型是否可以转换。如果该值既无法查看也无法转换，则default-constructed value将被退回。

也可以看看canView（） 和Q_DECLARE_SEQUENTIAL_CONTAINER_METATYPE()。

QVariant &QVariant::operator=(const QVariant &variant)

分配变量的值*variant*到这个变体。

QVariant &QVariant::operator=(QVariant &&other)

移动分配*other*对此QVariant实例。

相关非成员

[alias]QVariantHash

同义词QHash[QString](https://doc-qt-io.translate.goog/qt-6/qstring.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp),[QVariant](https://doc-qt-io.translate.goog/qt-6/qvariant.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp)。

[alias]QVariantList

同义词QList[QVariant](https://doc-qt-io.translate.goog/qt-6/qvariant.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp)。

[alias]QVariantMap

同义词QMap[QString](https://doc-qt-io.translate.goog/qt-6/qstring.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp),[QVariant](https://doc-qt-io.translate.goog/qt-6/qvariant.html?_x_tr_sl=auto&_x_tr_tl=zh-CN&_x_tr_hl=zh-CN&_x_tr_pto=wapp)。

template < typename T> T qvariant_cast(const QVariant &value)

返回给定的*value*转换为模板类型T。

这个函数相当于QVariant::value()。

也可以看看QVariant::value()。

bool operator!=(const QVariant &v1, const QVariant &v2)

返回`false`如果*v1*和*v2*是平等的；否则返回`true`.

QVariant使用包含的 type() 的相等运算符来检查相等性。

不同类型的变体在比较时总是不相等，但有一些例外：

- 如果两种类型都是数字类型（整数和浮点数），Qt 将使用标准 C++ 类型提升规则来比较这些类型。
- 如果一种类型是数字，另一种类型是QString，Qt 将尝试转换QString匹配的数字类型，如果成功则比较它们。
- 如果两个变体都包含指向QObject派生类型，QVariant将检查类型是否相关并指向同一对象。

QDataStream &operator<<(QDataStream &s, const QVariant &p)

写一个变体*p*到stream*s*。

也可以看看Format of the QDataStream operators。

bool operator==(const QVariant &v1, const QVariant &v2)

返回`true`如果*v1*和*v2*是平等的；否则返回`false`.

QVariant使用包含的 type() 的相等运算符来检查相等性。

不同类型的变体在比较时总是不相等，但有一些例外：

- 如果两种类型都是数字类型（整数和浮点数），Qt 将使用标准 C++ 类型提升规则来比较这些类型。
- 如果一种类型是数字，另一种类型是QString，Qt 将尝试转换QString匹配的数字类型，如果成功则比较它们。
- 如果两个变体都包含指向QObject派生类型，QVariant将检查类型是否相关并指向同一对象。

函数的结果不受以下结果的影响QVariant::isNull，这意味着两个值可以相等，即使其中一个为空而另一个不为空。

QDataStream &operator>>(QDataStream &s, QVariant &p)

读取变体*p*从stream中*s*。

注意：如果流包含非内置类型（请参阅QMetaType::Type），这些类型必须使用注册qRegisterMetaType（） 或者QMetaType::registerType() 才能正确加载变体。如果发现未注册的类型，QVariant将在流中设置损坏标志，停止处理并打印警告。例如，对于QList 它将打印以下内容：

> QVariant::load：名称未知的用户类型QList<整数>

也可以看看Format of the QDataStream operators。