

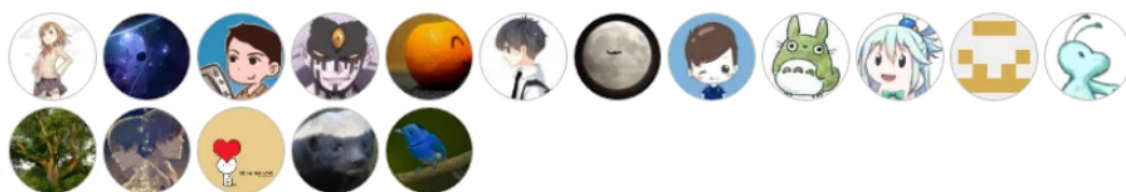
QtDocumentCN

2021 年 2 月 14 日

目录

第一章 贡献	3
第二章 QSql	4
第三章 QSqlDatabase	6

第一章 贡献



第二章 QSql

QSql 命名空间

QSql 命名空间里的各种各样的标识符, 已经被运用在 Qt SQL 各个模块中。[更多](#)

属性	方法
头文件	#include <QSql>
qmake	QT += sql

类型

enum	Location { BeforeFirstRow, AfterLastRow }
enum	NumericalPrecisionPolicy { LowPrecisionInt32, LowPrecisionInt64, LowPrecisionDouble, HighPrecision }
flags	ParamType
enum	ParamTypeFlag { In, Out, InOut, Binary }
enum	TableType { Tables, SystemTables, Views, AllTables }

细节的介绍

[查看 Qt SQL](#)

类型文档

enum QSql::Location

此枚举类型描述特殊的 sql 导航位置

常量	值	介绍
QSql::BeforeFirstRow	-1	在第一个记录之前
QSql::AfterLastRow	-2	在最后一个记录之后

另请参阅 [QSqlQuery::at\(\)](#)

enum QSql::NumericalPrecisionPolicy

数据库中的数值可以比它们对应的 C++ 类型更精确。此枚举列出在应用程序中表示此类值的策略。

常量	值	介绍
QSql::LowPrecisionInt32	0x01	对于 32 位的整形数值。在浮点数的情况下，小数部分将会被舍去。
QSql::LowPrecisionInt64	0x02	对于 64 位的整形数值。在浮点数的情况下，小数部分将会被舍去。
QSql::LowPrecisionDouble	0x04	强制双精度值。这个默认的规则
QSql::HighPrecision	0	字符串将会维持精度

注意：如果特定的驱动发生溢出，这是一个真实行为。像 Oracle 数据库在这种情形下，就会返回一个错误。

```
enum QSql::ParamTypeFlag
```

```
flags QSql::ParamType
```

这个枚举用于指定绑定参数的类型

常量	值	介绍
QSql::In	0x00000001	这个参数被用于向数据库里写入数据
QSql::Out	0x00000002	这个参数被用于向数据库里获得数据
QSql::InOut	In Out	这个参数被用于向数据库里写入数据；使用查询来向数据库里，重写数据
QSql::Binary	0x00000004	如果您想显示数据为原始的二进制数据，那么必须是 OR'd 和其他的标志

类型参数类型定义为 **QFlags**。它被存放在一个 OR 与类型参数标志的值的组合。

第三章 QSqlDatabase

QSqlDatabase 类用于处理数据库的连接

属性	方法
头文件	#include <QSqlDatabase>
qmake	QT += sql

列出所有的成员，包括继承成员

公共类型

返回值	函数名
	QSqlDatabase(const QSqlDatabase &other)
	QSqlDatabase()
QSqlDatabase &	operator=(const QSqlDatabase &other)
	QSqlDatabase()
void	close()
bool	commit()
QString	connectOptions() const
QString	connectionName() const
QString	databaseName() const
QSqlDriver *	driver() const
QString	driverName() const
QSqlQuery	exec(const QString &query = QString()) const
QString	hostName() const
bool	isOpen() const
bool	isOpenError() const
bool	isValid() const
QSqlError	lastError() const
QSql::NumericalPrecisionPolicy	numericalPrecisionPolicy() const
bool	open()
bool	open(const QString &user, const QString &password)
QString	password() const
int	port() const
QSqlIndex	primaryIndex(const QString &tablename) const
QSqlRecord	record(const QString &tablename) const
bool	rollback()
void	setConnectOptions(const QString &options = QString())
void	setDatabaseName(const QString &name)
void	setHostName(const QString &host)
void	setNumericalPrecisionPolicy(QSql::NumericalPrecisionPolicy precisionPolicy)
void	setPassword(const QString &password)
void	setPort(int port)
void	setUserName(const QString &name)
QStringList	tables(QSql::TableType type = QSql::Tables) const
bool	transaction()
QString	userName() const

静态公共成员

返回值	函数名
QSqlDatabase	addDatabase(const QString &type, const QString &connectionName = QLatin1String(defaultConnection))
QSqlDatabase	addDatabase(QSqlDriver *driver, const QString &connectionName = QLatin1String(defaultConnection))
QSqlDatabase	cloneDatabase(const QSqlDatabase &other, const QString &connectionName)
QSqlDatabase	cloneDatabase(const QString &other, const QString &connectionName)
QStringList	connectionNames()
bool	contains(const QString &connectionName = QLatin1String(defaultConnection))
QSqlDatabase	database(const QString &connectionName = QLatin1String(defaultConnection), bool open = true)
QStringList	drivers()
bool	isDriverAvailable(const QString &name)
void	registerSqlDriver(const QString &name, QSqlDriverCreatorBase *creator)
void	removeDatabase(const QString &connectionName)

受保护的成员函数

返回值	函数名
	QSqlDatabase(QSqlDriver *driver)
	QSqlDatabase(const QString &type)

详细的介绍

`QSqlDatabase` 类提供接口用于数据库的连接。一个 `QSqlDatabase` 实例对象表示连接。这个连接提供数据库所需要的驱动，这个驱动来自于 `QSqlDriver`。换言之，您可以实现自己的数据库驱动，通过继承 `QSqlDriver`。查看[如何实现自己的数据库驱动](#)来获取更多的信息。

通过调用一个静态的 `addDatabase()` 函数，来创建一个连接（即：实例化一个 `QSqlDatabase` 类），并且可以指定驱动或者驱动类型去使用（依赖于数据库的类型）和一个连接的名称。一个连接是通过它自己的名称，而不是通过数据库的名称去连接的。对于一个数据库您可以有多个连接。`QSqlDatabase` 也支持默认连接，您可以不传递连接名参数给 `addDatabase()` 来创建它。随后，这个默认连接假定您在调用任何静态函数情况下，而不去指定连接名称。下面的一段代码片段展示了如何去创建和打开一个默认连接，去连接 PostgreSQL 数据库：

```
QSqlDatabase db = QSqlDatabase::database();
```

`QSqlDatabase` 是一个值类。通过一个 `QSqlDatabase` 实例对数据库连接所做的操作将影响表示相同连接的其他 `QSqlDatabase` 实例。使用 `cloneDatabase()` 在基于已存在数据库的连接来创建独立的数据库的连接。

警告：强烈建议不要将 `QSqlDatabase` 的拷贝作为类成员，因为这将阻止关闭时正确清理实例。如果需要访问已经存在 `QSqlDatabase`，应该使用 `database()` 访问。如果您选择使用作为成员变量的 `QSqlDatabase`，则需要在删除 `QCoreApplication` 实例之前删除它，否则可能会导致未定义的行为。

如果您想创建多个数据库连接，可以调用 `addDatabase()`，并且给一个独一无二的参数（即：连接名称）。使用带有连接名的 `database()` 函数，来获取该连接。使用带有连接名的 `removeDatabase()` 函数，来删除一个连接。如果尝试删除由其他 `QSqlDatabase` 对象引用的连接，`QSqlDatabase` 将输出警告。可以使用 `contains()` 查看给定的连接名是否在连接列表中。

	一些实用的方法
<code>tables()</code>	返回数据表的列表
<code>primaryIndex()</code>	返回数据表的主索引
<code>record()</code>	返回数据表字段的元信息
<code>transaction()</code>	开始一个事务
<code>commit()</code>	保存并完成一个事务
<code>rollback()</code>	取消一个事务
<code>hasFeature()</code>	检查驱动程序是否支持事务
<code>lastError()</code>	返回有关上一个错误的信息
<code>drivers()</code>	返回可用的数据库驱动名称
<code>isDriverAvailable()</code>	检查特定驱动程序是否可用
<code>registerSqlDriver()</code>	注册自定义驱动程序

注意: `QSqlDatabase::exec()` 方法已经被弃用。请使用 `QSqlQuery::exec()`

注意: 使用事务时, 必须在创建查询之前启动事务。

成员函数文档

```
[protected] QSqlDatabase::QSqlDatabase(QSqlDriver *driver)
```

这是一个重载函数

使用给定驱动程序来创建连接

```
[protected] QSqlDatabase::QSqlDatabase(const QString &type)
```

这是一个重载函数

通过引用所给的数据库驱动类型来创建一个连接。如果不给定数据库驱动类型, 那么这个数据库连接将会没有什么作用。

当前可用的驱动类型:

驱动类别	介绍
QDB2	IBM DB2
QIBASE	Borland InterBase 驱动
QMYSQL	MySQL 驱动
QOCI	Oracle 调用的接口驱动
QODBC	ODBC 驱动 (包含 Microsoft SQL Server)
QPSQL	PostgreSQL 驱动
QSQLITE	SQLite 第三版本或者以上
QSQLITE2	SQLite 第二版本
QTDS	Sybase Adaptive Server

其他第三方驱动程序, 包括自己自定义的驱动程序, 都可以动态加载。

请参阅 [SQL Database Drivers](#), `registerSqlDriver()` 和 `drivers()`。


```
QSqlDatabase::QSqlDatabase(const QSqlDatabase &other)
```

创建一个其它的副本

`QSqlDatabase::QSqlDatabase()` 创建一个无效的 `QSqlDatabase` 空对象。使用 `addDatabase()`, `removeDatabase()` 和 `database()` 来获得一个有效的 `QSqlDatabase` 对象。

```
QSqlDatabase &QSqlDatabase::operator=(const QSqlDatabase &other)
```

给这个对象赋一个其他对象的值

```
QSqlDatabase::QSqlDatabase()
```

销毁这个对象，并且释放所有配置的资源注意：当最后的连接被销毁，这个析构函数就会暗中的调用 `close()` 函数，去删除这个数据库的其他连接。

另请参阅 `close()`。