QMap Class

template <typename Key, typename T> class QMap

QMap 类是一个提供关联数组的模板类。更多的...

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

注意：该类中的所有函数都是reentrant。

公共类型

class	**const_iterator**
class	**iterator**
class	**key_iterator**
	ConstIterator
	Iterator
	const_key_value_iterator
	difference_type
	key_type
	key_value_iterator
	mapped_type
	size_type

公共职能

	QMap()
	QMap(std::initializer_list<std::pair<Key, T>> *list*)
	QMap(const std::map<Key, T> &*other*)
	QMap(std::map<Key, T> &&*other*)
	QMap(const QMap<Key, T> &*other*)
	QMap(QMap<Key, T> &&*other*)
	~QMap()
auto	**asKeyValueRange**() &
auto	**asKeyValueRange**() const &
auto	**asKeyValueRange**() &&
auto	**asKeyValueRange**() const &&
QMap::iterator	**begin**()
QMap::const_iterator	**begin**() const
QMap::const_iterator	**cbegin**() const
QMap::const_iterator	**cend**() const
void	**clear**()
QMap::const_iterator	**constBegin**() const
QMap::const_iterator	**constEnd**() const
QMap::const_iterator	**constFind**(const Key &*key*) const
QMap::const_key_value_iterator	**constKeyValueBegin**() const
QMap::const_key_value_iterator	**constKeyValueEnd**() const
bool	**contains**(const Key &*key*) const
QMap::size_type	**count**(const Key &*key*) const
QMap::size_type	**count**() const
bool	**empty**() const
QMap::iterator	**end**()
QMap::const_iterator	**end**() const
QPair<QMap::iterator, QMap::iterator>	**equal_range**(const Key &*key*)
QPair<QMap::const_iterator, QMap::const_iterator>	**equal_range**(const Key &*key*) const
QMap::iterator	**erase**(QMap::const_iterator *pos*)

	QMap()
QMap::iterator	**erase**(QMap::const_iterator *first*, QMap::const_iterator *last*)
QMap::iterator	**find**(const Key &*key*)
QMap::const_iterator	**find**(const Key &*key*) const
T &	**first**()
const T &	**first**() const
const Key &	**firstKey**() const
QMap::iterator	**insert**(const Key &*key*, const T &*value*)
QMap::iterator	**insert**(QMap::const_iterator *pos*, const Key &*key*, const T &*value*)
void	**insert**(const QMap<Key, T> &*map*)
void	**insert**(QMap<Key, T> &&*map*)
bool	**isEmpty**() const
Key	**key**(const T &*value*, const Key &*defaultKey* = Key()) const
QMap::key_iterator	**keyBegin**() const
QMap::key_iterator	**keyEnd**() const
QMap::key_value_iterator	**keyValueBegin**()
QMap::const_key_value_iterator	**keyValueBegin**() const
QMap::key_value_iterator	**keyValueEnd**()
QMap::const_key_value_iterator	**keyValueEnd**() const
QList< Key>	**keys**() const
QList< Key>	**keys**(const T &*value*) const
T &	**last**()
const T &	**last**() const
const Key &	**lastKey**() const
QMap::iterator	**lowerBound**(const Key &*key*)
QMap::const_iterator	**lowerBound**(const Key &*key*) const
QMap::size_type	**remove**(const Key &*key*)
QMap::size_type	**removeIf**(Predicate *pred*)
QMap::size_type	**size**() const
void	**swap**(QMap<Key, T> &*other*)

	QMap()
T	**take**(const Key &*key*)
std::map<Key, T>	**toStdMap**() const &
std::map<Key, T>	**toStdMap**() &&
QMap::iterator	**upperBound**(const Key &*key*)
QMap::const_iterator	**upperBound**(const Key &*key*) const
T	**value**(const Key &*key*, const T &*defaultValue* = T()) const
QList< T>	**values**() const
QMap<Key, T> &	**operator=**(const QMap<Key, T> &*other*)
QMap<Key, T> &	**operator=**(QMap<Key, T> &&*other*)
T &	**[operator]**(const Key &*key*)
T	**[operator]**(const Key &*key*) const

相关非成员

qsizetype	**erase_if(QMap<Key, T> &*map*, Predicate *pred*)**
bool	**operator!=**(const QMap<Key, T> &*lhs*, const QMap<Key, T> &*rhs*)
QDataStream &	**operator<<**(QDataStream &*out*, const QMap<Key, T> &*map*)
bool	**operator==**(const QMap<Key, T> &*lhs*, const QMap<Key, T> &*rhs*)
QDataStream &	**operator>>**(QDataStream &*in*, QMap<Key, T> &*map*)

详细说明

QMap<Key, T> 是 Qt 的泛型之一container classes。它存储（键，值）对并提供键的快速查找。

QMap 和QHash提供非常相似的功能。差异是：

- QHash提供比 QMap 平均更快的查找速度。（看Algorithmic Complexity了解详情。）
- 当迭代一个QHash，项目是任意排序的。使用 QMap，项目始终按键排序。
- a 的密钥类型QHash必须提供运算符 ==() 和全局qHash （键）功能。QMap 的键类型必须提供指定全序的 operator<()。从 Qt 5.8.1 开始，使用指针类型作为键也是安全的，即使底层运算符 <() 不提供全序。

这是一个 QMap 示例QString键和`int`值：

```
QMap<QString, int> map;
```

要将（键，值）对插入到映射中，可以使用operator[] ()：

```
map["one"] = 1;
map["three"] = 3;
map["seven"] = 7;
```

这会将以下三个（键，值）对插入到 QMap 中：（“一”，1）、（“三”，3）和（“七”，7）。将项目插入Map的另一种方法是使用insert():

```
map.insert("twelve", 12);
```

要查找值，请使用运算符[] () 或value():

```
int num1 = map["thirteen"];
int num2 = map.value("thirteen");
```

如果Map中不存在具有指定键的项目，这些函数将返回default-constructed value。

如果你想检查Map是否包含某个键，请使用contains():

```
int timeout = 30;
if (map.contains("TIMEOUT"))
    timeout = map.value("TIMEOUT");
```

还有一个value() 重载，如果没有具有指定键的项目，则使用第二个参数作为默认值：

```
int timeout = map.value("TIMEOUT", 30);
```

一般来说，我们建议您使用contains （） 和value() 而不是用operator[] () 在映射中查找键。原因是，如果不存在具有相同键的项（除非映射是 const），则operator[] () 会默默地将一个项插入到映射中。例如，以下代码片段将在内存中创建 1000 个项目：

```
// WRONG
QMap<int, QWidget *> map;
...
for (int i = 0; i < 1000; ++i) {
    if (map[i] == okButton)
        cout << "Found button at index " << i << endl;
}
```

为了避免这个问题，请在上面的代码中替换`map[i]`为。`map.value(i)`

如果要浏览 QMap 中存储的所有（键，值）对，可以使用迭代器。QMap 两者都提供Java-style iterators（QMapIterator和QMutableMapIterator） 和STL-style iterators（QMap::const_iterator和QMap::iterator）。以下是如何迭代 QMap<QString, int> 使用 Java 风格的迭代器：

```
QMapIterator<QString, int> i(map);
while (i.hasNext()) {
    i.next();
    cout << qPrintable(i.key()) << ": " << i.value() << endl;
}
```

下面是相同的代码，但这次使用了 STL 风格的迭代器：

```
for (auto i = map.cbegin(), end = map.cend(); i != end; ++i)
    cout << qPrintable(i.key()) << ": " << i.value() << endl;
```

这些项目按升序键顺序遍历。

QMap 只允许每个键有一个值。如果你打电话insert() 使用 QMap 中已存在的键，之前的值将被删除。例如：

```
map.insert("plenty", 100);
map.insert("plenty", 2000);
// map.value("plenty") == 2000
```

但是，您可以使用以下方法为每个键存储多个值QMultiMap。

如果您只需要从映射中提取值（而不是键），您还可以使用基于范围：

```
QMap<QString, int> map;
...
for (int value : std::as_const(map))
    cout << value << endl;
```

可以通过多种方式从Map中删除项目。一种方法是打电话remove(); 这将删除具有给定密钥的任何项目。另一种方法是使用QMutableMapIterator::remove()。此外，您可以使用以下命令清除整个Mapclear()。

QMap 的键和值数据类型必须是assignable data types。这涵盖了您可能遇到的大多数数据类型，但编译器不允许您存储例如QWidget作为一个值；相反，存储一个QWidget*。另外，QMap的key类型必须提供operator<()。QMap 使用它来保持其项目的排序，并假设两个键x和y是等价的（如果 和x < y都不y < x为真）。

例子：

```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

class Employee
{
public:
    Employee() {}
    Employee(const QString &name, QDate dateOfBirth);
    ...

private:
    QString myName;
    QDate myDateOfBirth;
};

inline bool operator<(const Employee &e1, const Employee &e2)
{
    if (e1.name() != e2.name())
        return e1.name() < e2.name();
    return e1.dateOfBirth() < e2.dateOfBirth();
}

#endif // EMPLOYEE_H
```

在示例中，我们首先比较员工的姓名。如果相等，我们会比较他们的出生日期来打破平局。

可以参考： QMapIterator,QMutableMapIterator,QHash， 和QSet。

成员类型文档

QMap::ConstIterator

Qt 风格的同义词QMap::const_iterator。

QMap::Iterator

Qt 风格的同义词QMap::iterator。

QMap::const_key_value_iterator

QMap::const_key_value_iterator typedef 提供了一个 STL 风格的迭代器QMap。

QMap::const_key_value_iterator 本质上与QMap::const_iterator不同之处在于，operator*() 返回一个键/值对而不是一个值。

可以参考： QKeyValueIterator。

[alias]QMap::difference_type

ptrdiff_t 的类型定义。提供 STL 兼容性。

[alias]QMap::key_type

键的类型定义。提供 STL 兼容性。

QMap::key_value_iterator

QMap::key_value_iterator typedef 提供了一个 STL 风格的迭代器QMap。

QMap::key_value_iterator 本质上与QMap::iterator不同之处在于，operator*() 返回一个键/值对而不是一个值。

可以参考： QKeyValueIterator。

[alias]QMap::mapped_type

T 的 Typedef。为 STL 兼容性而提供。

[alias]QMap::size_type

typedef 为 int。提供 STL 兼容性。

成员函数文档

[since 6.4]auto QMap::asKeyValueRange() &

[since 6.4]auto QMap::asKeyValueRange() &&

[since 6.4]auto QMap::asKeyValueRange() const &

[since 6.4]auto QMap::asKeyValueRange() const &&

返回一个范围对象，该对象允许以键/值对的形式迭代此映射。例如，这个范围对象可以在基于范围的 for 循环中与结构化绑定声明结合使用：

```
QMap<QString, int> map;
map.insert("January", 1);
map.insert("February", 2);
// ...
map.insert("December", 12);

for (auto [key, value] : map.asKeyValueRange()) {
    cout << qPrintable(key) << ": " << value << endl;
    --value; // convert to JS month indexing
}
```

请注意，通过这种方式获得的键和值都是对映射中的键和值的引用。具体来说，改变值将修改映射本身。

该功能是在 Qt 6.4 中引入的。

可以参考： QKeyValueIterator。

QMap::QMap()

构造一个空Map。

可以参考： clear()。

QMap::QMap(std::initializer_list<std::pair<Key, T>> list)

使用初始值设定项列表中每个元素的副本构造一个映射*list*。

[explicit]QMap::QMap(const std::map<Key, T> &other)

构造一个副本*other*。

可以参考： toStdMap()。

[explicit]QMap::QMap(std::map<Key, T> &&other)

通过移动来构建Map*other*。

可以参考： toStdMap()。

[default]QMap::QMap(const QMap<Key, T> &other)

构造一个副本*other*。

此操作发生在constant time，因为 QMap 是implicitly shared。这使得从函数返回 QMap 的速度非常快。如果共享实例被修改，它将被复制（写时复制），这需要linear time。

可以参考： operator=。

[default]QMap::QMap(QMap<Key, T> &&other)

Move-构造一个QMap实例。

[default]QMap::~QMap()

破坏Map。对该映射中的值以及该映射上的所有迭代器的引用都将变得无效。

QMap::iterator QMap::begin()

返回一个STL-style iterator指向Map中的第一个项目。

可以参考： constBegin（） 和end()。

QMap::const_iterator QMap::begin() const

这是一个重载功能。

QMap::const_iterator QMap::cbegin() const

返回一个常量STL-style iterator指向Map中的第一个项目。

可以参考： begin（） 和cend()。

QMap::const_iterator QMap::cend() const

返回一个常量STL-style iterator指向Map中最后一个项目之后的虚构项目。

可以参考： cbegin（） 和end()。

void QMap::clear()

从Map上删除所有项目。

可以参考： remove()。

QMap::const_iterator QMap::constBegin() const

返回一个常量STL-style iterator指向Map中的第一个项目。

可以参考： begin（） 和constEnd()。

QMap::const_iterator QMap::constEnd() const

返回一个常量STL-style iterator指向Map中最后一个项目之后的虚构项目。

可以参考： constBegin（） 和end()。

QMap::const_iterator QMap::constFind(const Key &key) const

返回一个 const 迭代器，指向带有 key 的项*key*在Map中。

如果Map不包含带有键的项目*key*，函数返回constEnd()。

可以参考： find()。

QMap::const_key_value_iterator QMap::constKeyValueBegin() const

返回一个常量STL-style iterator指向Map中的第一个条目。

可以参考： keyValueBegin()。

QMap::const_key_value_iterator QMap::constKeyValueEnd() const

返回一个常量STL-style iterator指向Map中最后一个条目之后的虚构条目。

可以参考： constKeyValueBegin()。

bool QMap::contains(const Key &key) const

`true`如果Map包含带有键的项目，则返回*key*; 否则返回`false`.

可以参考： count()。

QMap::size_type QMap::count(const Key &key) const

返回与键关联的项目数*key*。

可以参考： contains()。

QMap::size_type QMap::count() const

这是一个重载功能。

与...一样size()。

bool QMap::empty() const

提供此函数是为了兼容 STL。它相当于isEmpty()，如果Map为空则返回true；否则返回 false。

QMap::iterator QMap::end()

返回一个STL-style iterator指向Map中最后一个项目之后的虚构项目。

可以参考： begin（） 和constEnd()。

QMap::const_iterator QMap::end() const

这是一个重载功能。

QPair<QMap::iterator, QMap::iterator> QMap::equal_range(const Key &key)

返回一对界定值范围的迭代器`[first, second)`，这些值存储在*key*。

QPair<QMap::const_iterator, QMap::const_iterator> QMap::equal_range(const Key &key) const

这是一个重载功能。

QMap::iterator QMap::erase(QMap::const_iterator pos)

删除迭代器指向的（键，值）对*pos*从映射中获取，并返回一个迭代器到映射中的下一项。

注意： 迭代器*pos* *必须*有效且可取消引用。

可以参考： remove()。

[since 6.0]QMap::iterator QMap::erase(QMap::const_iterator first, QMap::const_iterator last)

删除迭代器范围 [指向的 (key, value) 对*first*,*last*）从Map上。返回映射中最后一个删除元素后面的项目的迭代器。

注意：该范围`[first, last)` *必须*是 中的有效范围`*this`。

这个函数是在Qt 6.0中引入的。

可以参考：remove()。

QMap::iterator QMap::find(const Key &key)

返回一个指向带有 key 的项目的迭代器*key*在Map中。

如果Map不包含带有键的项目*key*，函数返回end()。

可以参考：constFind(),value(),values(),lowerBound（），和upperBound()。

QMap::const_iterator QMap::find(const Key &key) const

这是一个重载功能。

T &QMap::first()

返回对映射中第一个值的引用，即映射到最小键的值。该函数假设Map不为空。

当非共享（或调用 const 版本）时，这会在以下位置执行constant time。

可以参考：last(),firstKey（），和isEmpty()。

const T &QMap::first() const

这是一个重载功能。

const Key &QMap::firstKey() const

返回对映射中最小键的引用。该函数假设Map不为空。

这执行于constant time。

可以参考：lastKey(),first(),keyBegin（），和isEmpty()。

QMap::iterator QMap::insert(const Key &key, const T &value)

使用 key 插入一个新项目*key*和一个值*value*。

如果已经有一个带有该密钥的项目*key*，该项目的值被替换为*value*。

可以参考：QMultiMap::insert()。

QMap::iterator QMap::insert(QMap::const_iterator pos, const Key &key, const T &value)

这是一个重载功能。

使用 key 插入一个新项目*key*和价值*value*并有提示*pos*建议在哪里进行插入。

如果constBegin() 用作提示，表示*key*小于Map中的任何键，同时constEnd() 表明*key*（严格）大于Map中的任何键。否则提示应满足条件（*pos*- 1).key() <*key*<= 位置。key()。如果提示*pos*是错误的，它会被忽略并完成常规插入。

如果已经有一个带有该密钥的项目*key*，该项目的值被替换为*value*。

如果提示正确并且映射未共享，则插入会按摊销方式执行constant time。

从排序数据创建映射时，首先插入最大的键constBegin() 比按排序顺序插入要快constEnd（），自从constEnd() - 1（需要检查提示是否有效）需要logarithmic time。

注意：请小心提示。从较旧的共享实例提供迭代器可能会崩溃，但也存在它会悄悄损坏映射和*pos*Map。

可以参考：QMultiMap::insert()。

void QMap::insert(const QMap<Key, T> &map)

将所有项目插入*map*进入这张Map。

如果两个映射都有一个键，则其值将替换为存储在*map*。

可以参考：QMultiMap::insert()。

void QMap::insert(QMap<Key, T> &&map)

将所有项目从*map*进入这张Map。

如果两个映射都有一个键，则其值将替换为存储在*map*。

如果*map*被共享，则项目将被复制。

bool QMap::isEmpty() const

`true`如果Map不包含任何项目则返回；否则返回 false。

可以参考： size()。

Key QMap::key(const T &value, const Key &defaultKey = Key()) const

这是一个重载功能。

返回第一个有值的键*value*， 或者*defaultKey*如果Map不包含有价值的项目*value*。如果不*defaultKey*如果函数返回一个default-constructed key。

这个函数可能会很慢（linear time）， 因为QMap的内部数据结构针对按键（而不是按值）快速查找进行了优化。

可以参考： value（） 和keys()。

QMap::key_iterator QMap::keyBegin() const

返回一个常量STL-style iterator指向Map中的第一个键。

可以参考： keyEnd（） 和firstKey()。

QMap::key_iterator QMap::keyEnd() const

返回一个常量STL-style iterator指向Map中最后一个键之后的虚构项目。

可以参考： keyBegin（） 和lastKey()。

QMap::key_value_iterator QMap::keyValueBegin()

返回一个STL-style iterator指向Map中的第一个条目。

可以参考： keyValueEnd()。

QMap::const_key_value_iterator QMap::keyValueBegin() const

返回一个常量STL-style iterator指向Map中的第一个条目。

可以参考： keyValueEnd()。

QMap::key_value_iterator QMap::keyValueEnd()

返回一个STL-style iterator指向Map中最后一个条目之后的虚构条目。

可以参考： keyValueBegin()。

QMap::const_key_value_iterator QMap::keyValueEnd() const

返回一个常量STL-style iterator指向Map中最后一个条目之后的虚构条目。

可以参考： keyValueBegin()。

QList< Key> QMap::keys() const

返回一个列表，其中包含映射中按升序排列的所有键。

保证与使用的顺序相同values()。

该函数创建一个新列表，在linear time。可以通过迭代来避免所需的时间和内存使用keyBegin（） 到keyEnd()。

可以参考： QMultiMap::uniqueKeys(),values（），和key()。

QList< Key> QMap::keys(const T &value) const

这是一个重载功能。

返回包含与值关联的所有键的列表*value*按升序排列。

这个函数可能会很慢（linear time）， 因为QMap的内部数据结构针对按键（而不是按值）快速查找进行了优化。

T &QMap::last()

返回对映射中最后一个值的引用，即映射到最大键的值。该函数假设Map不为空。

当非共享（或调用 const 版本）时，这会在以下位置执行logarithmic time。

可以参考： first(),lastKey（），和isEmpty()。

const T &QMap::last() const

这是一个重载功能。

const Key &QMap::lastKey() const

返回对映射中最大键的引用。该函数假设Map不为空。

这执行于logarithmic time。

可以参考： firstKey(),last(),keyEnd（），和isEmpty()。

QMap::iterator QMap::lowerBound(const Key &key)

返回一个指向第一个带有 key 的项目的迭代器*key*在Map中。如果Map不包含带有键的项目*key*，该函数返回一个迭代器，指向具有更大键的最近项。

可以参考： upperBound（） 和find()。

QMap::const_iterator QMap::lowerBound(const Key &key) const

这是一个重载功能。

QMap::size_type QMap::remove(const Key &key)

删除所有具有该密钥的项目*key*从Map上。返回删除的项目数，如果映射中存在该键，则返回 1，否则返回 0。

可以参考： clear（） 和take()。

[since 6.1]template < typename Predicate> QMap::size_type QMap::removeIf(Predicate pred)

删除谓词所属的所有元素*pred*从Map返回 true。

该函数支持采用 type 参数`QMap<Key, T>::iterator`或 type 参数的谓词`std::pair<const Key &, T &>`。

返回删除的元素数（如果有）。

该功能是在 Qt 6.1 中引入的。

可以参考： clear（） 和take()。

QMap::size_type QMap::size() const

返回映射中（键，值）对的数量。

可以参考： isEmpty（） 和count()。

void QMap::swap(QMap<Key, T> &other)

掉期Map*other*有了这张Map。这个操作非常快并且永远不会失败。

T QMap::take(const Key &key)

用钥匙移除物品*key*从Map并返回与其关联的值。

如果Map中不存在该项目，该函数仅返回一个default-constructed value。

如果不使用返回值，remove() 效率更高。

可以参考： remove()。

std::map<Key, T> QMap::toStdMap() const &

返回与此等效的 STL 映射QMap。

[since 6.0]std::map<Key, T> QMap::toStdMap() &&

这是一个重载功能。

注意： 调用该函数会留下thisQMap在部分形成的状态下，唯一有效的操作是销毁或分配新值。

这个函数是在Qt 6.0中引入的。

QMap::iterator QMap::upperBound(const Key &key)

返回一个迭代器，指向紧随最后一个带有 key 的项目之后的项目*key*在Map中。如果Map不包含带有键的项目*key*，该函数返回一个迭代器，指向具有更大键的最近项。

例子：

```
QMap<int, QString> map;
map.insert(1, "one");
map.insert(5, "five");
map.insert(10, "ten");

map.upperBound(0);      // returns iterator to (1, "one")
map.upperBound(1);      // returns iterator to (5, "five")
map.upperBound(2);      // returns iterator to (5, "five")
map.upperBound(10);     // returns end()
map.upperBound(999);    // returns end()
```

可以参考： lowerBound（） 和find()。

QMap::const_iterator QMap::upperBound(const Key &key) const

这是一个重载功能。

T QMap::value(const Key &key, const T &defaultValue = T()) const

返回与键关联的值*key*。

如果Map不包含带有键的项目*key*，函数返回*defaultValue*。如果不*defaultValue*被指定，该函数返回一个default-constructed value。

可以参考： key(),values(),contains（） ， 和[operator]()。

QList< T> QMap::values() const

返回一个列表，其中包含映射中的所有值（按键的升序排列）。

该函数创建一个新列表，在linear time。可以通过迭代来避免所需的时间和内存使用keyValueBegin （） 到keyValueEnd()。

可以参考： keys （） 和value()。

[default]QMap<Key, T> &QMap::operator=(const QMap<Key, T> &other)

分配*other*到此映射并返回对此映射的引用。

[default]QMap<Key, T> &QMap::operator=(QMap<Key, T> &&other)

移动分配*other*对此QMap实例。

T &QMap::operator[] (const Key &key)

返回与键关联的值*key*作为可修改的参考。

如果Map不包含带有键的项目*key*，该函数插入一个default-constructed value用钥匙进入Map*key*，并返回对其的引用。

可以参考： insert（） 和value()。

T QMap::operator [] (const Key &key) const

这是一个重载功能。

与...一样value()。

相关非成员

[since 6.1]template <typename Key, typename T, typename Predicate> qsizetype erase_if(QMap<Key, T> &map, Predicate pred)

删除谓词所属的所有元素*pred*从Map返回 true*map*。

该函数支持采用 type 参数`QMap<Key, T>::iterator`或 type 参数的谓词`std::pair<const Key &, T &>`。

返回删除的元素数（如果有）。

该功能是在 Qt 6.1 中引入的。

bool operator!=(const QMap<Key, T> &lhs, const QMap<Key, T> &rhs)

返回`true`如果*lhs*不等于*rhs*; 否则返回 false。

如果两个映射包含相同的（键，值）对，则认为它们相等。

该功能需要键和值类型来实现`operator==()`。

可以参考： operator==()。

template <typename Key, typename T> QDataStream &operator<<(QDataStream &out, const QMap<Key, T> &map)

写Map*map*流式传输*out*。

该功能需要键和值类型来实现`operator<<()`。

可以参考： Format of the QDataStream operators。

bool operator==(const QMap<Key, T> &lhs, const QMap<Key, T> &rhs)

返回`true`如果*lhs*等于*rhs*; 否则返回 false。

如果两个映射包含相同的（键，值）对，则认为它们相等。

该功能需要键和值类型来实现`operator==()`。

可以参考： operator!=()。

template <typename Key, typename T> QDataStream &operator>>(QDataStream &in, QMap<Key, T> &map)

从流中读取Map*in*进入*map*。

该功能需要键和值类型来实现`operator>>()`。

可以参考： Format of the QDataStream operators。