QListIterator Class

template < typename T> class QListIterator

QListIterator 类提供了一个 Java 风格的 const 迭代器QList和QQueue。更多的...

Header:	**#include < QListIterator>**
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

公共职能

	QListIterator(**const QList< T> &*list***)
bool	**findNext**(const T &*value*)
bool	**findPrevious**(const T &*value*)
bool	**hasNext**() const
bool	**hasPrevious**() const
const T &	**next**()
const T &	**peekNext**() const
const T &	**peekPrevious**() const
const T &	**previous**()
void	**toBack**()
void	**toFront**()
QListIterator< T> &	**operator=**(const QList< T> &*container*)

详细说明

QList两者都有Java-style iterators和STL-style iterators。STL 风格的迭代器效率更高，应该是首选。

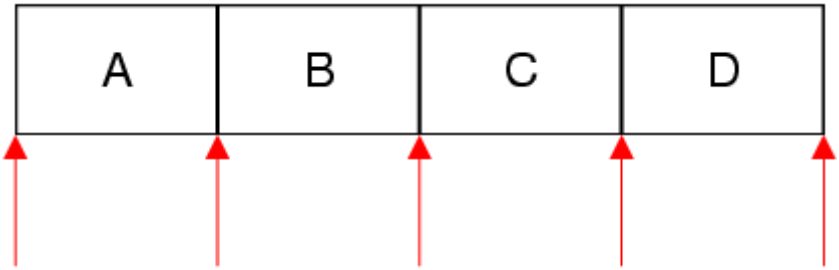
使用迭代器的另一种方法是使用索引位置。最多QList成员函数将索引作为其第一个参数，从而可以在不使用迭代器的情况下访问、修改和删除项目。

QListIterator< T> 允许您迭代QList< T>，一个QQueue< T> 或QStack< T>。如果您想在迭代列表时修改列表，请使用QMutableListIterator< T> 代替。

QListIterator 构造函数采用QList作为论证。构造后，迭代器位于列表的最开头（第一项之前）。以下是按顺序迭代所有元素的方法：

```
QList<float> list;
...
QListIterator<float> i(list);
while (i.hasNext())
    float f = i.next();
```

这next() 函数返回列表中的下一项并推进迭代器。与 STL 样式的迭代器不同，Java 样式的迭代器指向项*之间*而不是直接*指向*项。第一次致电next() 将迭代器前进到第一项和第二项之间的位置，并返回第一项；第二次致电next() 将迭代器前进到第二项和第三项之间的位置，并返回第二项；等等。



以下是如何以相反的顺序迭代元素：

```
QListIterator<float> i(list);
i.toBack();
while (i.hasPrevious())
    float f = i.previous();
```

如果您想查找特定值的所有出现位置，请使用findNext （） 或者findPrevious() 循环中。

可以在同一个列表上使用多个迭代器。如果在 QListIterator 处于活动状态时修改列表，则 QListIterator 将继续迭代原始列表，忽略修改后的副本。

也可以看看QMutableListIterator和QList::const_iterator。

成员函数文档

QListIterator::QListIterator(const QList< T> &list)

构造一个迭代器用于遍历*list*。迭代器设置为位于列表的前面（第一项之前）。

也可以看看operator=()。

QListIterator< T> &QListIterator::operator=(const QList< T> &container)

使迭代器运行*list*。迭代器设置为位于列表的前面（第一项之前）。

也可以看看toFront（） 和toBack()。

void QListIterator::toFront()

将迭代器移动到容器的前面（第一项之前）。

也可以看看toBack（） 和next()。

void QListIterator::toBack()

将迭代器移动到容器的后面（最后一项之后）。

也可以看看toFront（） 和previous()。

bool QListIterator::hasNext() const

`true`如果迭代器前面至少有一项，即迭代器不在*容器*的后面，则返回；否则返回`false`.

也可以看看hasPrevious（） 和next()。

const T &QListIterator::next()

返回下一项并将迭代器前进一个位置。

在位于容器后面的迭代器上调用此函数会导致未定义的结果。

也可以看看hasNext(),peekNext（） ， 和previous()。

const T &QListIterator::peekNext() const

返回下一项而不移动迭代器。

在位于容器后面的迭代器上调用此函数会导致未定义的结果。

也可以看看hasNext(),next（） ， 和peekPrevious()。

bool QListIterator::hasPrevious() const

`true`如果迭代器后面至少有一项，即迭代器不在*容器*的前面，则返回；否则返回`false`.

也可以看看hasNext（） 和previous()。

const T &QListIterator::previous()

返回前一项并将迭代器向后移动一个位置。

在位于容器前面的迭代器上调用此函数会导致未定义的结果。

也可以看看hasPrevious(),peekPrevious（），和next()。

const T &QListIterator::peekPrevious() const

返回前一项而不移动迭代器。

在位于容器前面的迭代器上调用此函数会导致未定义的结果。

也可以看看hasPrevious(),previous（），和peekNext()。

bool QListIterator::findNext(const T &value)

搜索次数*value*从当前迭代器位置开始向前。返回`true`如果*value*被发现；否则返回`false`.

通话结束后，如果*value*找到后，迭代器就位于匹配项之后；否则，迭代器位于容器的后面。

也可以看看findPrevious()。

bool QListIterator::findPrevious(const T &value)

搜索次数*value*从当前迭代器位置向后开始。返回`true`如果*value*被发现；否则返回 false。

通话结束后，如果*value*找到后，迭代器就位于匹配项之前；否则，迭代器位于容器的前面。

也可以看看findNext()。