

2006--QMap 类&QHash 类&QVector 类--零声教育 vico 老师

一、QMap 类

QMap<Key,T>提供一个从类型为 Key 的键到类型为 T 的值的映射。通常, QMap 存储的数据形式是一个键对应一个值, 并且按照键 Key 的次序存储数据。为了能够支持一键多值的情况, QMap 提供 QMap<Key,T>::insertMulti()和 QMap<Key,T>::values()函数。QMultiMap 类来实例化一个 QMap 对象。

【案例分析】

```
#include <QCoreApplication>

#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    // QMap 类
    // 1:创建 QMap 实例, 第一个参数为 QString 类型的键, 第二个参数为 int
    类型的值
    QMap<QString,int> qmap;

    // 插入数据信息, 它有两方式进行操作
    qmap["Chinese"]=119;
    qmap["English"]=120;

    qmap.insert("Math",115);
    qmap.insert("Physics",99);
    qmap.insert("Chemistry",100);
    qDebug()<<qmap;

    // 删除数据信息 key 键
    qmap.remove("Chemistry");
    qDebug()<<qmap<<endl;

    // 遍历 QMap 类的实例: 数据信息
```

```

// 1:迭代器 (java 类型的迭代操作)
 QMapIterator<QString, int> itr(qmap);
 while(itr.hasNext())
 {
     itr.next();
     qDebug() <<itr.key() <<": " <<itr.value();
 }

// 2:STL 类型的迭代
 qDebug() <<endl;
 QMap<QString, int>::const_iterator stritr=qmap.constBegin();
 while(stritr!=qmap.constEnd())
 {
     qDebug() <<stritr.key() <<": " <<stritr.value();
     stritr++;
 }

// key 键/T 键-->来查找
 qDebug() <<endl;
 qDebug() <<"key-->T: " <<qmap.value("Math");
 qDebug() <<"T-->key: " <<qmap.key(99) <<endl;

// 修改键值
// 一个键对应一个值, 再次调用 insert() 函数将覆盖之前的值
 qmap.insert("Math", 118);
 qDebug() <<qmap.value("Math");

// 查询是否包含某个键
 qDebug() <<endl;
 qDebug() <<"result=" <<qmap.contains("Chinese");
 qDebug() <<"result=" <<qmap.contains("Chemistry");

// 输出所有 QMap 实例化: Key 键和 T 值
 qDebug() <<endl;
 QList<QString> aKeys=qmap.keys();
 qDebug() <<aKeys;
 QList<int> aValues=qmap.values();
 qDebug() <<aValues;

// 一个键对应多个值
// 直接使用 QMapMultiMap 类来实例化一个 QMap 对象
 qDebug() <<endl;
 QMapMultiMap<QString, QString> mulmap;
 mulmap.insert("student", "no");

```

```

mulmap.insert("student", "name");
mulmap.insert("student", "sex");
mulmap.insert("student", "age");
mulmap.insert("student", "high");
mulmap.insert("student", "weight");
QDebug() << mulmap; // 从输出结果可以看出 mulmap 仍然是一个 QMap 对象

return a.exec();
}

```

二、QHash 类

QHash<Key,T>具有与 QMap 几乎完全相同的 API。QHash 维护着一张哈希表（Hash Table），哈希表的大小与 QHash 的数据项的数目相适应。

QHash 以任意的顺序组织它的数据。当存储数据的顺序无关紧要时，建议使用 QHash 作为存放数据的容器。

【案例分析】

```

#include <QCoreApplication>

#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    // QHash 类
    QHash<QString, int> qhash;

    qhash["key 1"]=3;
    qhash["key 1"]=8;
    qhash["key 4"]=4;
    qhash["key 2"]=2;
}

```

```

qhash.insert("key 3", 30);

QList<QString> list=qhash.keys();
for(int i=0;i<list.length();i++)
    qDebug()<<list[i]<<" "<<qhash.value(list[i]);

// QHash 内部的迭代器 QHashIterator 类
qDebug()<<endl;
QHash<QString, int> hash;
hash["key 1"]=33;
hash["key 2"]=44;
hash["key 3"]=55;
hash["key 4"]=66;
hash.insert("key 3", 100);

QHash<QString, int>::const_iterator iterator;
for(iterator=hash.begin();iterator!=hash.end();iterator++)
    qDebug()<<iterator.key()<<"-->"<<iterator.value();

return a.exec();
}

```

QMap 与 QHash 区别:

- QMap 与 QHash 的功能差不多，但 QHash 的查找速度更快;
- QMap 是按照键的顺序存储数据，而 QHash 是任意顺序存储的;
- QMap 的键必须提供 "<" 运算符，而 QHash 的键必须提供 "==" 运算符和一个名为 qHash() 的全局散列函数。

三、QVector 类

`QVector<T>`在相邻的内存当中存储给定数据类型 `T` 的一组数值。在一个 `QVector` 的前部或者中间位置进行插入操作的速度是很慢的，这是因为这样的操作将导致内存中的大量数据被移动，这是由 `QVector` 存储数据的方式决定的。

【案例分析】

```
#include <QCoreApplication>

#include <QDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    // QVector 类
    // QVector<T>是 Qt 的一个容器类

    QVector<int> qvr;
    // 第一种方式赋值
    qvr<<10;
    qvr<<20;
    qvr<<30;
    qvr<<40;
    // 第二方式赋值
    qvr.append(50);
    qvr.append(60);
    qvr.append(70);
    qvr.append(80);
    qvr.append(90);
    qvr.append(100);

    qDebug()<<qvr<<endl;

    // 求出 QVector 类容器的实例化：元素个数
    qDebug()<<"qvr count="<<qvr.count()<<endl;

    // 遍历所有元素
    for(int i=0;i<qvr.count();i++)
```

```
        qDebug() << qvr[i];

// 删除 qvr 容器里面的元素
qDebug() << endl;
qvr.remove(0); // 删除第 0 个元素
for(int i=0; i<qvr.count(); i++)
    qDebug() << qvr[i];

qvr.remove(2, 3); // 从第 2 个元素开始，删除后面 3 个元素
qDebug() << endl;
for(int i=0; i<qvr.count(); i++)
    qDebug() << qvr[i];

// 判断容器是否包含某个元素
qDebug() << endl;
qDebug() << "result=" << qvr.contains(90);
qDebug() << "result=" << qvr.contains(901) << endl;

return a.exec();
}
```