# Qt Learn

## Qt学习--D指针

1. 为什么要用D指针：

D指针是C++特有的一种“设计模式”，主要为了解决二进制兼容的问题

1. 什么是二进制兼容：

A library is binary compatible（二进制编译）, if a program linked dynamically to a former version of the library continues running with newer versions of the library without the need to recompile.

If a program needs to be recompiled to run with a new version of library but doesn’t require any further modifications, the library is source compatible(源代码兼容).

二进制兼容对于库尤其重要，因为不能指望每次升级了库之后，要求所有使用了这个库的应用程序都要重新编译。（就Qt而言，一个使用Qt5.5编译的应用程序，如果能够保持二进制兼容，就能够直接在Qt5.6上正常运行）。

1. 如何实现二进制兼容：

要求每一个结构以及每一个对象的数据模型保持不变（即不能在类中增加，删除数据成员，其根本是保持数据成员在数据模型中的位移保持不变）。

我们让所有的公共类都只持有一个指针，这个指针指向了一个包含了所有数据的私有数据结构。这个私有的数据结构可以在未来的版本中任意修改，而且不会影响到使用了这个库的程序。因为这个数据结构是私有的，外部根本不能使用。外部只能看到一个不变的公共类，因为它只持有一个指针（就是D指针）。

//////////////////////////////////////////////////////////////////////////////////

widget.h

class WidgetPrivate;

class Widget : public QWidget

{

QRect geometry() const;

private:

WidgetPrivate \* d\_ptr;

}

widget\_p.h

struct WidgetPrivate

{

WidgetPrivate(Widget \*q):q\_ptr(q){}

QRect geometry;

QString str;

Widget \* q\_ptr;

}

widget.cpp

Widget::Widget():d\_ptr(new WidgetPrivate){}

QRect Widget::geometry() const{return d\_ptr->geometry;}

//////////////////////////////////////////////////////////////////////////////////

1. D指针其他好处
2. 隐藏实现细节：只需要发布头文件和二进制文件，cpp文件闭源
3. 头文件很干净，没有实现细节的干扰，可以作为API实现
4. 供实现的头文件都被移动到了实现的源代码文件中，编译会很快。
5. D指针如何使用它所在的公共类中的函数：Q指针

通过新增的Q指针，我们就可以访问到外部的公共类。

## 信号槽

1. 形式：

connect ( sender, signal, receiver, slot)

sender, receiver没什么区别，都是QObject指针

信号槽要求信号和槽的参数一致（所谓一致，是参数类型一致）

槽函数的参数可以比信号的少

只有继承了QObject类的类，才具有信号槽的能力。所以，为了使用信号槽，处理必须继承QObject，之外，还需要再第一行代码写上Q\_OBJECT。

slots (public, protected, private)

signals

emit

信号可以连信号

## QMetaObject

The Qt Meta-object system in Qt is responsible for the signals and slots inter-object communication mechanism(通信机制), runtime type infomation, and the qt property system.

A single QMetaObject instance is created for each QObject subclass that is used in an application, and this instance stores all the meta-information for the QObject subclass. This object is availabel as QObject::metaObject();

1. The functions you are most likely to find useful are these:

* className() return the name of a class;
* superClass() return the supperclass’s meta-object
* Enumerator() and enumeratorCount() are provide information about a class’s enumertator
* propertyCount() and property()
* Constructor() and constructorCount()

The index functions indexOfConstruct(), indexOfMethod(), indexOfEnumerator() and indexOfProperty() map names of ,,, to indexed in the meta-obj. For example, Qt uses indexOfMethod() internally when you connect a signal to a slot.

Classes can also have a list of name--value paris of additionanl class information, stored in QMetaClassInfo objects.

## 存储容器

Qt提供了一套基于模板的容器类，这些容器类不继承object，都提供了隐式数据共享、不可变的特性，并且为速度做了优化，具有较低的内存占用量。而且是线程安全，与平台无关的（即不因编译器的不同而具有不同的实现）

能够存储在容器中的数据必须是可赋值数据类型。（所谓可赋值数据类型，是指具有默认构造函数、拷贝构造函数和赋值运算符的类型）。绝大多数数据类型，包括基本类型（int, double），指针，Qt数据类型（QString, QDate, QTime）都是可赋值数据类型。但是，QObject及其子类都不是（也就是说，不能使用QList<QWidget>这种容器，因为QWidget的拷贝构造函数和赋值运算符不可用。如果需要这种类型的容器，只能存储其指针，也就是QList<QWidget\*>）

1. 顺序存储容器：QList, QVector，
2. 关联存储容器：

总结：

1. QList<T>：
2. 存储形式--列表；内部使用数组实现，同事提供基于索引的快速访问
3. It stores items in a list that provides fase index\_based access and index\_based insertions and removals. QVector shoule be the default first choise, QVector will usually give better performance than QList. Becase QVector always stores it items sequentially in memory. Where QList will allocate its items on the heap unless sizeof(T)<=sizeof(void\*) and T has been declares to be either a Q\_MOVABLE\_TYPE or a Q\_PRIMITIVE\_TYPE using Q\_DECLARE\_TYPEINFO. If you need a real linked list, which guarantees constant time insertions mid-list and uses iterators to items rather than indexes, uss QLinkedList.
4. Iterators into a QLinkedList and references into heap-allocating QList remain valid as long as the referenced items remain in the container. This is not true for iterators and references into a QVector and non-heap-allocating QLists.
5. The array representation allows very fast insertions and index-based access. The prepend() hand append() operations are also very fast becase QList pre allocates memory at both ends of its iternal array.
6. Qt includes QStringList class that inherits QList<QString> and adds a few convenience functions, such as ...
7. QList stores a list of items. The default constructor cetates an empty list. You can use the initializer-list constructor to create a list with elements.
8. Qlist provides basic functions to add, move and remove items.
9. QLinkList:不是基于索引的随机访问，为了访问元素，需要使用遍历器进行遍历。在中间插入数据的性能远优于QList
10. QVector：用于再内存的连续区域存储一系列给定类型的值。在头部和中间插入数据代价高（因为会引起数据的移动）。
11. QStack：QVector的子类，先进后出
12. Qqueue: 是list的子类，先进先出

## 容器的算法复杂性

数据量增长时，容器的每一个函数有多快。

* 常量时间：O(1) 运行时间与容器中数据量无关。（QLinkedList::insert()函数）
* 对数时间：O(log n) 运行时间时容器数据量的对数关系
* 线性时间：O(n) 运行时间与数量相关
* 线性对数时间：O(n log n) 线性对数时间 > 线性时间，线性对数时间 < 平方时间
* 平方时间:：O(n^2) 平方时间与容器数据量是平方关系

# C++

## -stdio.h

C library to perform input/output operations

This library uses what are called strams to operate with physical devices such as keyboards, printers, terminals or with any other type of files supported by the system.

Stream are handled in the cstdio library as pointer to FILE objects.

There also exist three standard stream: stdin, stdout, stderr, which are automatically created and opened for all programs using the library.

重载赋值运算符

本质：函数。名字由operator关键字+要定义的运算符的符号组成。

运算符函数有：返回类型，参数列表（参数表示运算符的运算对象）

运算符通常返回一个指向其左侧运算对象的引用。（标准库通常要求保存在容器中的类型要具有赋值运算符，且其返回值是左侧运算对象的引用）

析构函数

作用：释放对象使用的资源，并销毁对象的非static数据成员。

析构函数是类的一个成员函数，没有返回值，也不接受参数。

由于析构函数不接受参数，所以不能被重载（唯一的）

销毁成员的顺序：按成员初始化顺序的逆序销毁

什么时候会调用析构函数

1. 变量在离开其作用域时被销毁
2. 当一个对象被销毁时，其成员被销毁
3. 容器被销毁时，其元素被销毁
4. 动态分配的对象，当对指向它的指针应用delete运算符时被销毁
5. 临时对象，当创建它的完整表达式结束时被销毁