**工作纪要**

编写2022年12月5日

审核年月日

批准年月日

**William工作纪要**

[1 产品简介 3](#_Toc203120358)

[1.1 目的 3](#_Toc203120359)

[1.2 背景 3](#_Toc203120360)

[2 参考文档 3](#_Toc203120361)

[3 提交文档 3](#_Toc203120362)

[4 纪要内容 4](#_Toc203120363)

[A004 强制类型转换 7](#_Toc203120364)

[A005 QT的Coin3D三维可视化库的安装 9](#_Toc203120365)

[A006 QStandardItemModel的用法 18](#_Toc203120366)

[A007 for（auto i:v）遍历容器元素 19](#_Toc203120367)

[A008 QMetaType元类型系统 19](#_Toc203120368)

[A009 QT下使用多字节字符集的方法 20](#_Toc203120369)

[A010 QT线程中CreateEvent的用法 20](#_Toc203120370)

[A011 C++头文件里class直接加类名的作用 22](#_Toc203120371)

[A012 QT学习之QString的arg方法 23](#_Toc203120372)

[A013 已知同一组点在不同坐标系下的坐标，如何求解两个坐标系之间的转换关系 24](#_Toc203120373)

[A014 socket编程（C++） 24](#_Toc203120374)

[A015 生成DLL及调用 29](#_Toc203120375)

[A016 自定义数据类型 qRegisterMetaType 的使用 30](#_Toc203120376)

[A017 QT工具类QACTion、边框QFrame、菜单QMenu梳理 30](#_Toc203120377)

[1工具类QAction 30](#_Toc203120378)

[2.自定义类PControl 32](#_Toc203120379)

[3.类QFrame 32](#_Toc203120380)

[4类PMenu继承QMenu、PControl 36](#_Toc203120381)

[5PMenuBar 继承QMenuBar、PControl 38](#_Toc203120382)

[5 QT模板库 38](#_Toc203120383)

[QList的用法 38](#_Toc203120384)

[7按钮列表类PBtnListWidget 41](#_Toc203120385)

[子测试界面类PFramePackage 42](#_Toc203120386)

[子测试界面填充类ptestframe.cpp 42](#_Toc203120387)

[控制面板类ptestctrlwidget.cpp 43](#_Toc203120388)

[8、主框架类PMainFrame 继承QMainWindow ,PFrame 43](#_Toc203120389)

[9滑动效果类PSlideViewWidget继承PFrame 43](#_Toc203120390)

[10模块类PModuleMenuFrame继承PFrame 46](#_Toc203120391)

[11、 QT类QPixmap 46](#_Toc203120392)

[12、 模板类PItemContext 47](#_Toc203120393)

[C++模板之typename和class关键字的区别： 48](#_Toc203120394)

[QT模板库QVector 50](#_Toc203120395)

[13、 自定义类PTestGroupFrame 51](#_Toc203120396)

[A018 QT工具类QACTion、边框QFrame、菜单QMenu梳理 53](#_Toc203120397)

[信号槽Connect 53](#_Toc203120398)

[C++11 lambda表达式，eg: 53](#_Toc203120399)

[A019 QT之容器空间QMdiArea 55](#_Toc203120400)

[A020 自定义类PApp 56](#_Toc203120401)

[A022 Qt之QMap基本用法 56](#_Toc203120402)

[A023 Qt之全局变量extern 58](#_Toc203120403)

[A024 界面部分工具标定、坐标准值、靶球位置 59](#_Toc203120404)

[问题1：ui = new ui CoordTrans(); 59](#_Toc203120405)

[问题2：常用部件及含义 59](#_Toc203120406)

[问题3：表格组件类QTableWIdget 60](#_Toc203120407)

[问题4：表格组件类QTableWIdget、QStackedWidget区别 62](#_Toc203120408)

[A025 矩阵变换的左乘与右乘 63](#_Toc203120409)

[A026 如何将欧拉角转换为方向向量 63](#_Toc203120410)

[A027 Qt中Excel文件导入类Tools 64](#_Toc203120411)

[A028 Arts5025-6.0性能测试模块流程 69](#_Toc203120412)

[PApp 类进行公共菜单类、工具栏、加密检测、序列化输出等设置。 69](#_Toc203120413)

[A029 Arts6025L位姿测试模块流程 72](#_Toc203120414)

[A030 AES加解密 75](#_Toc203120415)

[A031 6Probe姿态算法讨论 76](#_Toc203120416)

[A032 码垛校准模块分析 76](#_Toc203120417)

[A033 QSharedMemory类详解 77](#_Toc203120418)

[A034 QProcessy类详解 78](#_Toc203120419)

[A035 Polyworks与VS联合使用SDK 79](#_Toc203120420)

[A036 C# 基本概念 81](#_Toc203120421)

[A037 QT Pro文件配置项说明 84](#_Toc203120422)

[A037 QT qInstallMessageHandler（输出详细日志） 85](#_Toc203120423)

[A038 C++ initializer\_list详解 85](#_Toc203120424)

[A039 Qt ApplicationAttribute/WidgetAttribute程序级别属性 85](#_Toc203120425)

[A040 C++ std::function详解与实战 86](#_Toc203120426)

[A041 STL unordered\_map 86](#_Toc203120427)

[A042 std::string\_view 使用介绍 87](#_Toc203120428)

[A043 Python库 concurrent.futures—启动并行任务 90](#_Toc203120429)

[《三生万物》：第一天到大办公室 你该做什么？ 91](#_Toc203120430)

[《两晋悲歌》：慕容垂之小段后 94](#_Toc203120431)

# 产品简介

## 目的

*阐述产品或项目的目的。*

## 背景

*阐述产品或项目的背景情况内容。*

# 参考文档

*下表列出了XXX（XXX）产品或项目测试方案所使用的文档，并标明了各文档的可用性：*

|  |  |  |  |
| --- | --- | --- | --- |
| **文档** | **已创建**  **或可用** | **已被接收或**  **已经过复审** | **作者** |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |

# 提交文档

下表列出了XXX（XXX）测试所提交的测试文档，并标明了各文档的可用性：

|  |  |  |  |
| --- | --- | --- | --- |
| 文档 | 已创建  或可用 | 已被接收或  已经过复审 | 作者 |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |
|  | 是□　否□ | 是□　否□ |  |

# 纪要内容

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试编号** | A001 | | | | |
| **工作内容** | 问题：调试弹出The CDB process terminated | | | | |
| **建立日期** | 2021-06-24 | **建立人员** | 林威威 | **修改日期** |  |
|  | 1一般是缺少windows SDK,参考<https://blog.csdn.net/qq_21449473/article/details/107126475>  2.可能缺少第三方库，①一种是查下是否引用了第三方库文件，若有引用，可去项目配置中添加lib和include相关文件（第三方库文件可在pro文件中添加，也可在系统环境变量中添加）；  ②另一种是直接双击生成的exe文件，会提示该项目缺少哪些dll文件，须一一添加，直到运行通过为止。 | | | | |
|  |  | | | | |
| **结论** |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试编号** | A002 | | | | |
| **工作内容** | Qsettings用法总结 功能：记住他的settings,比如窗口大小、位置及一些别的设置，还有常用的recentfiles。  ①这些setting一般是在系统里，比如windows一般都写在系统注册表或者写INI文件，Mac系统一般都在XML文件里。Qsetting就提供了一种方法来存储和回复应用程序的settings.  QSettings的API是基于QVariant，QVariant是一种数据类型的集合，它包含了大部分通常的Qt数据类型，比如：QString，Qect，Qimage等。  ②创建Qsettings的对象时，可以再堆栈上创建，也可以在堆上创建（new）,构造和销毁QSettings对象非常快。  要传递两个参数，第一个是公司名称，第二个是程序名称。例：  Settings = Qsettings(“MySoft”,”QtPad”)  公司名称：MySoft，程序名称：QtPad  假如我们在应用程序中多次要用到Qsettings，为了简单起见，我们可以在主程序中先如下声明。  QtCore.QCoreApplication.setOrganizationName("MySoft")  QtCore.QCoreApplication.setOrganizationDomain("mysoft.com")  QtCore.QCoreApplication.setApplicationName("QtPad")  QSettings存储设置。每个设置都有一个QString和一个QVariant组成，QString指定设置的名称键），QVariant存储于键相关的数据。用setValue()写入设置。  Settings.setValue ("editor/wrapMargin", 68);  可用value()获取设置的值：  int margin = settings.value("editor/wrapMargin").toInt();  如果没有指定名称的设置，QSettings将返回null QVariant, 可以转换为整数0）。通过将第二个参数传递给value（），可以指定另一个默认值：  int margin = settings.value("editor/wrapMargin", 80).toInt();要测试给定的键是否存在，请调用contains（）。要删除与键关联的设置，请调用remove（）。要获取所有键的列表，请调用allKeys（）。要删除所有键，请调用clear（）。  参考[https://blog.csdn.net/zhizhengguan/article/details/115695334](工作纪要.docx) | | | | |
| **建立日期** | 2021-06-28 | **建立人员** | 林威威 | **修改日期** |  |
|  | 当然前提是已经from PyQt4 import QtCore。  ③日志记录系统：loguru. <https://github.com/emilk/loguru>  ④只要包含头文件QApplication或者QCoreapplication，就可以使用qAPP,指向一个当前实例,qApp是个宏：  **QApplication中：#define qApp (static\_cast<QApplication \*>(QCoreApplication::instance()))**  **QCoreapplication中：#define qApp QCoreApplication::instance()**  所以可以肯定的是QCoreApplication有个静态函数instance()返回指向自己的指针。  **s**tatic QCoreApplication \*instance() { return self; }  这个函数返回self，看大片self也定义为静态指针：定义程序唯一的实例时，初始化self指向自身。qApp是一个宏，替换后为调用相应的instance().  ⑤qt中UI:就是designer设计的一个界面类，需要转化后才能看到这个类，如果自己建一个类集成自QWidget，那么他和ui无差别，只是添加不见要手动添加代码，不能像designer那样拖动。  Qt中this:就是一个指向类本身的指针，用this就是指向自己创建的那个类。  QT中qApp:是指向Application的全局变量。 | | | | |
|  |  | | | | |
| **结论** |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试编号** | A003 | | | | |
| **工作内容** | QT Designer中窗口对象的windowFilePath属性  1、windowFilePath属性仅对窗口对象有效，用于关联一个窗口和对应的文件路径。当窗口没有设置标题属性的情况下，窗口标题展示windowFilePath对应的文件名的信息（路径不显示），如果二者都设置，则优先使用窗口标题属性的设置作为标题。  该属性缺省为空，可以通过windowFilePath()和setWindowFilePath(QString )来访问和设置windowFilePath属性。  这个功能在窗口打开文件进行操作时可以使用来标记打开的文件。 | | | | |
| **建立日期** | 2021-06-24 | **建立人员** | 林威威 | **修改日期** |  |
|  | 2、QJsonDocument读写json.  QJsonDocument（JSON文档）：提供一种读取和写入JSON文档的方法，可以通过他的成员函数array（）或object（）检索文档中包含的数组或对象，然后读取JSON数据或者修改数据。  QJsonObject（JSON对象）:封装了一个JSON对象，可以使用他的成员函数size()、insert()和remove()来修改JSON数据。  QJsonArray（JSON数组）:封装了一个JSON数组，用来实现JSON  QJsonValue（值）：用于存储所有值，可用type判断其类型  QJsonParseError(错误)： | | | | |
|  |  | | | | |
| **结论** |  | | | | |

A004 强制类型转换

dynamic\_cast: 通常在基类和派生类之间转换时使用

const\_cast: 主要针对const和volatile的转换.

static\_cast: 一般的转换，no run-time check.通常，如果你不知道该用哪个，就用这个。

reinterpret\_cast: 用于进行没有任何关联之间的转换，比如一个字符指针转换为一个整形数。

1. dynamic\_cast<T\*>(a)

运行期间会检查这个转换是否可能。完成类层次结构中的提升。T必须是一个指针、引用或无类型的指针。A必须是决定一个指针或引用的表达式。

表达式dynamic\_cast<T\*>(a) 将a值转换为类型为T的对象指针。如果类型T不是a的某个基类型，该操作将返回一个空指针。

它不仅仅像static\_cast那样，检查转换前后的两个指针是否属于同一个继承树，它还要检查被指针引用的对象的实际类型，确定转换是否可行。

如果可以，它返回一个新指针，甚至计算出为处理多继承的需要的必要的偏移量。如果这两个指针间不能转换，转换就会失败，此时返回空指针（NULL）。

很明显，为了让dynamic\_cast能正常工作，必须让编译器支持运行期类型信息（RTTI）

2、编译器在编译期处理

将地址a转换成类型T，T和a必须是指针、引用、算术类型或枚举类型。

表达式static\_cast<T\*>(a), a的值转换为模板中指定的类型T。在运行时转换过程中，不进行类型检查来确保转换的安全性。

static\_cast它能在内置的数据类型间互相转换，对于类只能在有联系的指针类型间进行转换。可以在继承体系中把指针转换来、转换去，但是不能转换成继承体系外的一种类型

class A { ... };

class B { ... };

class D : public B { ... };

void f(B\* pb, D\* pd)

{

D\* pd2 = static\_cast<D\*>(pb); // 不安全, pb可能只是B的指针

B\* pb2 = static\_cast<B\*>(pd); // 安全的

A\* pa2 = static\_cast<A\*>(pb); //错误A与B没有继承关系

...

}

让学生把实践重点回归到心性上来，办法有堵、指、顿、渐渐四类。

堵：是把提问者用“意”想想出来的路都堵死，让他无路可走，从而自然回归心性。

指：是直接把方法、方向指示给他，他就可以照着做了。

顿：目标是让学生立刻了解心性，达到言下顿悟的效果。

渐：不急于指示重点，但不断调整大的方向，不断趋近最终的结果。

第一卷以决疑为主，基本上都是先生在回答学生们各方面的疑问，包括见地的、实践的、还有与其他学说的差别；

第二卷指出并辨析见地，其中最具有代表性的部分就是《答顾东桥书》；

第三卷纯粹讨论见地，解决人们对见地的争议。

总之不要纠结于字面意思，要深度思考心学的见地，实践方法中的内在逻辑，透过文字看本质。

切记 每当你觉得“方便、获益”的时候一定要小心这就是“坑”；

急行知马力 遇事见人心

3、const\_cast<T\*>(a)

编译器在编译期处理

去掉类型中的常量，除了const或不稳定的变址数，T和a必须是相同的类型。

表达式const\_cast<T\*>(a)被用于从一个类中去除以下这些属性：const, volatile, 和 \_\_unaligned。

class A { ... };

void f()

{

const A \*pa = new A;//const对象

A \*pb;//非const对象

//pb = pa; // 这里将出错，不能将const对象指针赋值给非const对象

pb = const\_cast<A\*>(pa); // 现在OK了

...

}

对于本身定义时为const的类型，即使你去掉const性，在你操作这片内容时候也要小心，只能r不能w操作，否则还是会出错

const char\* p = "123";

char\* c = const\_cast<char\*>(p);

c[0] = 1; //表面上通过编译去掉了const性，但是操作其地址时系统依然不允许这么做。

const\_cast操作不能在不同的种类间转换。相反，它仅仅把一个它作用的表达式转换成常量。它可以使一个本来不是const类型的数据转换成const类型的，或者把const属性去掉。

尽量不要使用const\_cast,如果发现调用自己的函数，竟然使用了const\_cast，那就赶紧打住，重新考虑一下设计吧。

世人大多有侥幸心理，对自己有利的事情，哪怕不确定，也愿意相信这件事会发生。但对自己不利的事情，如果不确定，大部分人都不愿意相信这件事情会发生。

4）reinterpret\_cast<T\*>(a)

编译器在编译期处理

任何指针都可以转换成其它类型的指针，T必须是一个指针、引用、算术类型、指向函数的指针或指向一个类成员的指针。

表达式reinterpret\_cast<T\*>(a)能够用于诸如char\* 到 int\*，或者One\_class\* 到 Unrelated\_class\*等类似这样的转换，因此可能是不安全的。

class A { ... };

class B { ... };

void f()

{

A\* pa = new A;

void\* pv = reinterpret\_cast<A\*>(pa);

// pv 现在指向了一个类型为B的对象，这可能是不安全的。

...

}

A005 QT的Coin3D三维可视化库的安装

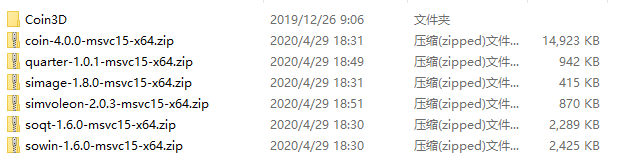
1. Coin3D介绍

Coin3D是一个基于OpenGL的三维图形库，源于Open Inventor2.1API,Coin仍然与之兼容。Coin是一个基于OpenGL的保留模式三维图形渲染库。它在C++中实现，公开发布，源代码开放供您阅读。应用程序编程接口（API）与SGI的Open Inventor完全兼容，后者实际上是复杂可视化应用程序的标准3D图形API。

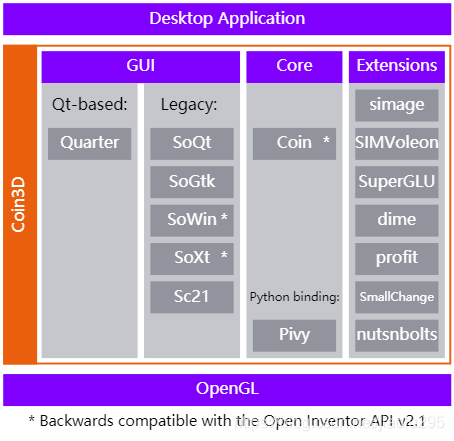
Coin3D安装，官网：<https://github.com/coin3d>。

https://coin3d.github.io/Coin/html/

分别下载coin, quarter, simage, simvoleon, soqt, sowin这几个库，要在window，Qt下使用的话，coin，soqt/quarter, sowin是必选的，simage用于图像的相关处理，simvoleon用于三维模拟结果方面。



2、架构图



下表列出了Coin3D所包含的组件及其主要用途。绿色为必须组件，黄色根据不同平台选用，青色可选，Windows上必须Coin、SoWin、SoQt、Quarter。



3、主要类介绍

SoQt

SoSeparator类是一个状态保持组节点。SoSeparator节点的子图不会影响当前节点的装填，因为在遍历子徒之前推送（push）遍历状态，在遍历子徒之后弹出（pop）遍历状态。

Ref():void SoBase::ref(void) const 增加对象的引用计数。在某些情况下，这可能需要从用户代码显式执行（为了避免过早删除），但当对象相互依赖时，通常从Coin库中的其他实例调用。

Unref():void SoBase::unref(void) const 减少对象的应用计数。如果引用计数达到0，对象将删除自身。显式调用此方法时要小心，通常需要将对ref()的用户级调用与对unref（）或UnrefNodelete（）的调用相匹配，以避免内存泄漏。

AddChild():void SoGroup::addChild(SoNode\* node) 将子节点添加到当前节点正在管理的子节点列表中。

SoQtExaminerViewer类是一个3d模型检查查看器。被认为是最“通用”的查看器，他通常用于快速原型中，已检查简单模型和完成场景

SetSceneGraph():void SoQtViewer::setSceneGraph(SoNode \*root) 为查看器提供一个要渲染和交互的场景图。

SoTransformation类是转换节点的抽象基类。要在三维世界空间中定位和定向几何体，可以使用各种变换节点类型。这些节点都继承

SoTransformation基类。除非在SoSeparator节点下使用，否则变换将通过场景图累积。在SoSeparator节点下，变换矩阵堆栈将被推送、弹出（push/pop）

SoMatrixTransform类是一个转换节点。是最灵活的转换节点，可以使用它在当前模型转换矩阵的基础上累积任何类型的转换矩阵。

Coin3D 三维可视化教程 4

在Inventor中基本数据类型的名字都是以字母Sb开始的，例如SbColor、SbViewVolume。Inventor场景数据库是由一个或多个3D场景的数据信息所组成的。数据库SoDB可以包含多个图形场景。每个场景都是有一组此昂管的3D对象和属性构成的。场景数据库有两个基本操作（方法）可供调用。首先需要初始化数据库：SoDB::init()这是在编写Inventor程序时的第一个调用函数。如果在开发的程序中使用Inventor组件库的话，那么当调用SoXt::init()的饿时候，场景数据库将会自动被初始化。如果没有使用组件库，但使用了有交互作用的节点或者节点工具包（Node Kits）时，必须首先调用SoInteraction::init()函数，这个函数会同时初始化数据库、交互作用模块和节点工具包。其次，可以讲一个Inventor文件读取到场景数据库中，增加新的图形场景。

其次，可以将一个Inventor文件读到场景数据库中，增加新的图形场景。①SoSeparator\* readAll(SoInput \*in);②SbBool read(SoInput \*in,SoNede \*& rootNode ) const;③SbBool read(SoInput \*in,SoPath \*& path ) const。方法①Inventor 将从参数in所指定文件中读取场景数据，并返回一个指向SoSeparator节点的指针，这个SoSeparator节点将是所读文件中所有场景的跟节点。方法②将从in所指定文件中读取场景数据，然后将文件中的场景跟节点作为结果返回给rootNode.方法③同②。节点是组成图形场景的基本元素。它内部包含用于定义3D形体、属性、组的数据和方法。当创建了一个节点，这个节点贝贝当做跟节点自动保存到场景数据库中（when a node is created,it is automatically inserted int the database as a root node）。通常情况下，可以通过将节点与数据库中的其他节点相连接，构造出一种具有分层关系的场景。

节点分三大类：形体节点（Shape nodes）代表3D几何模型；属性节点（Property nodes）,表现对象的外观或其他场景特征；组节点（Group nodes）是一种将节点聚合包含进场景中的容器（containers）。

创建节点 使用C++ new操作符穿件一个节点，SoSphere \*headSphere = new SoSphere 。不能创建节点数组、不能使用delete操作符删除。理解“引用计数”的原理对于我们使用Inventor是至关重要的，应为我么必须知道在什么条件下，节点会被自动删除。

节点里有什么 每个节点都是有一组叫做“域”的用于描述节点参数的数据元素组成。例如：一个点光源节点（SoPointLight）包含了四个域：intensity亮度、color颜色、location位置、on开关。Inventor定义了许多域的类型。每种域类型都有唯一的方法来访问其数据。在每个节点中，根据域的用途，每个域都被命名了相应的名称。

当对节点执行动作时发生了什么 每个节点都会执行自己的动作行为。当需要对场景执行一个特定的动作时，我们必须首先要创建一个动作类的实例，然后再将这个动作应用到场景的根节点上。数据库对每个动作都管理着一个叫做遍历状态（tranersal state）的数据结构，这个数据结构实在某个给定时刻内动作的元素与参数的结合。通常，当执行一次动作时，动作将按照从上到下，从左到右的顺序遍历整个场景。在遍历的过程中，节点将根据他们对此次动作所做的特定行为来修改遍历状态。

举例来说，SoMaterial节点用于遍历状态中各种材质元素的当前值，SoDrawStyle节点用于设置遍历状态中绘制风格元素的当前值。

形体节点 用于表示一种3D几何对象，代表的是一些受属性节点和组节点影响的实际物体。在执行渲染动作期间，他们实际上会将自己星星绘制在屏幕上。形体节点包括：SoSphere、SoIndexedFaceSet和SoText3等类。

属性节点 代表的是像“表面材质”、“绘制风格”、“几何变换”等这些用于展现场景外观和特有性质的节点。可分为若干类，场景图一般使用三种不同的图标来表示他们：

变换类图标：表示执行集合变换的属性节点，像SoTransform、SoRatation等。这些都是从SoTransformation派生出来的。

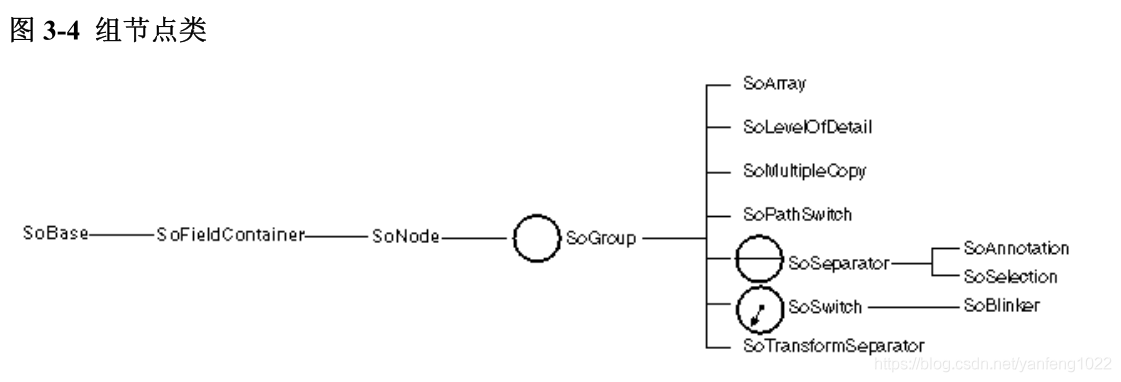
外观类图标：表示可以修改对象外观属性节点，像SoMaterial、SoBaseColor 、SoMaterialBinding、SoComplexity、SoDrawStyle、SoLightModel 和SoFont。

度量类图标：包含有坐标线、法线或几何信息节点，像SoCoordinate3、SoCoordinate4、SoProfileCoordinate2、SoProfileCoordinate3、SoNormal 和 SoNormalBinding。

通常，在便利过程中，属性节点将自己的新值替换掉在遍历状态中相应元素的值。但是几何变换节点是一个例外，他们会和当前几何变换进行累积运算。

Coin3D 三维可视化教程 5

组 组节点是一个用于手机子节点对象的容器。组节点可以讲属性、形体、和其他组节点收集到场景中。下图展示了部分组节点类的类树图。Inventor中有多重不同类型的组节点，每种组节点都有特定“分组”特性。当创建完一个组节点后，初始情况下它是没有子节点在其中的。SoGroup是所有组节点的基类。所有送她派生出来的节点类都有一个addChild()方法。



子节点的顺序 addChild()方法会将某个特定节点增加到组节点内子节点列表末端位置上。每个加到组节点中的子节点都分配有一个关联的索引值。组中第一个子节点索引为0。以此类推。

重要的子节点顺序 每种节点类对于给定的数据库动作都有自己的相应方式。在本届的讨论中，我们家是只处理渲染动作。

如果当前渲染的节点是组节点，那么组节点将按照顺序对每个子节点调用渲染动作，调用顺序通常都是按找子节点在场景中从左到右的顺序。

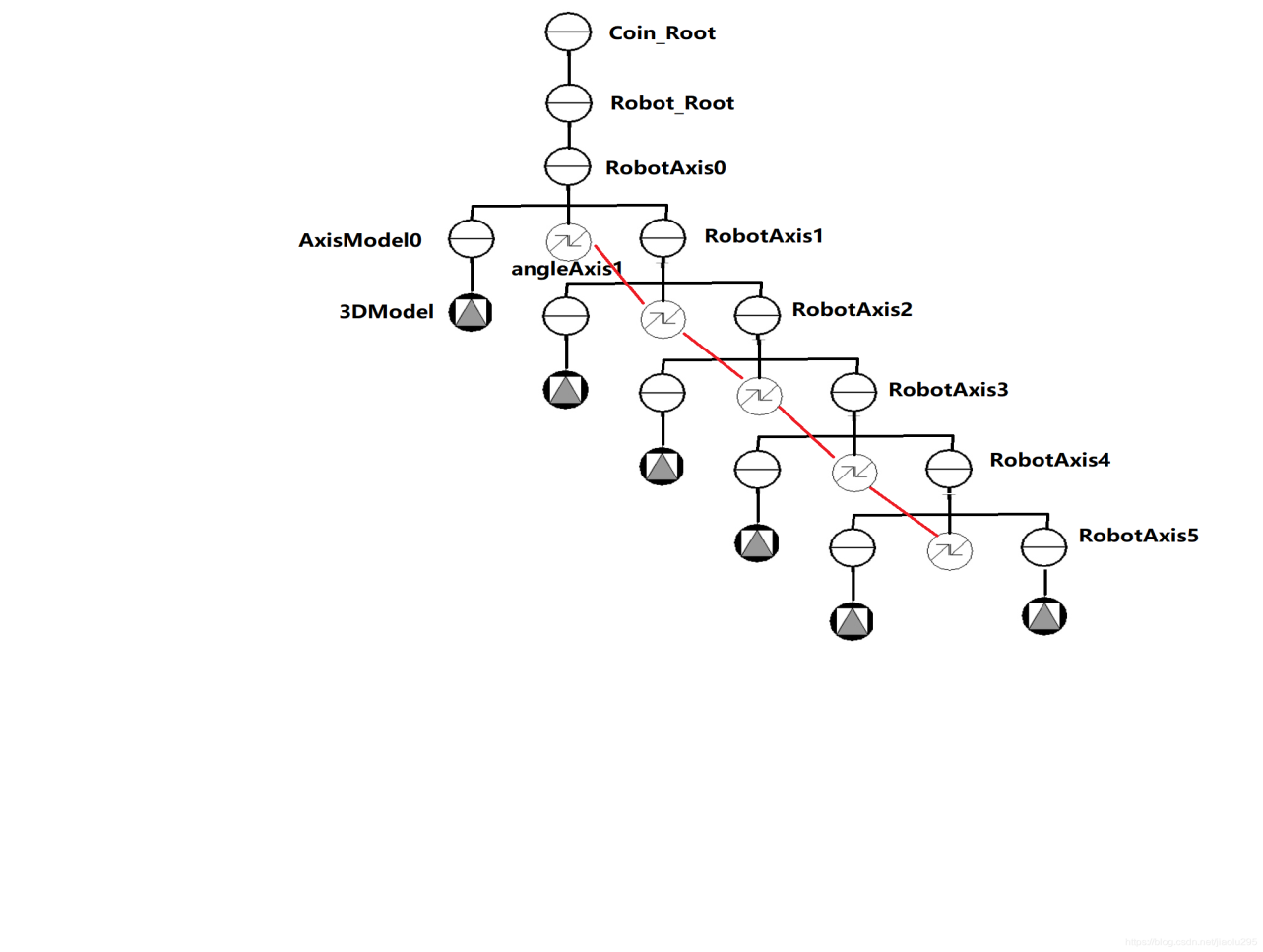
每个子节点一次执行他们自己的渲染方法，这些方法在某些方面会影响遍历状态。如果子节点是一个属性节点，那么它可能会修改诸如像散射光颜色、物体缩放比例、线宽度等当前遍历状态的元素数据。绝大多数的属性节点只是简单地（使用自己的数值）替换掉遍历状态中对应元素的数值(例如，青铜材质的节点将使用自己的数值替换掉遍历状态中当前材质的数值)。几何变换是一种例外情况，它们是彼此互相结合，累积产生合成变换。

如果当前渲染的节点是形体节点的话，那么形体节点将使用当前的遍历状态来绘制自己。

隔离节点 使用从SoGroup派生出来的子类SoSparator节点，可以隔离其子节点所产生的影响。SoSeparator节点在遍历其子节点之前，会保存当前的遍历状态，当遍历完所有子节点后，SoSepartor会回复以前的遍历状态。SoSeparator中的节点将不会对位于其上、右的任何节点产生影响。

4、坐标系介绍

Coin3D对3D数据使用的右手坐标系，从屏幕内指向外，表示Z轴的正方向。所有角度单位都是弧度。对象都是在自己的局部坐标空间下进行表述的，即众所周知的“对象坐标系空间（object coordinat space）”。当场景中的所有物体都已经进行完坐标变换后，那么他们就都在“世界坐标系空间”下描述了（world coordinate space）。照相机和灯光节点处于世界坐标系空间下。例子中五轴机器人仿真模型节点图如图所示：



具体流程：

1. 读入三维模型

SoVRMLGroup \* Robot::openModel(QString fileName)

{

SoInput \* myInput = new SoInput;

if (myInput->openFile(fileName.toStdString().data()))

{

SoVRMLGroup \*model = SoDB::readAllVRML(myInput);

myInput->closeFile();

delete myInput;

return model;

}

myInput->closeFile();

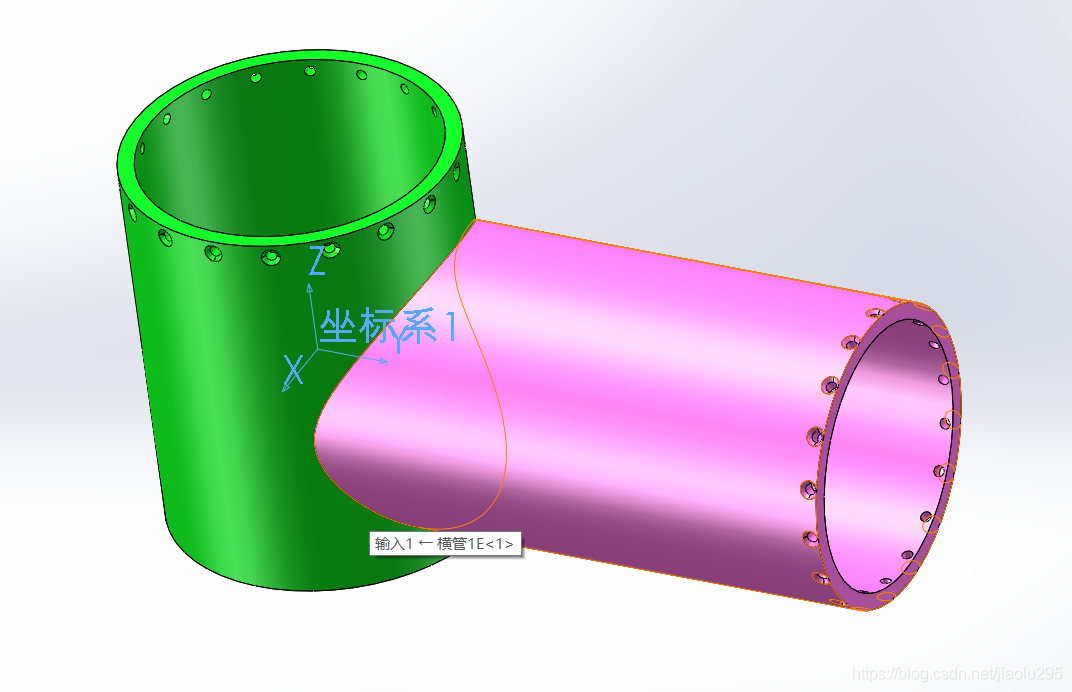
delete myInput;

return NULL;

}

1. 进行坐标变换

方式一：针对三维模型（将坐标系建立在旋转轴原点）



思路：将模型的坐标系相对于当前局部坐标系变换到目标坐标系，变换完成后部件就在世界坐标系下面描述了。核心代码如下：

#include <Inventor/nodes/SoTransform.h>

#include <Inventor/nodes/SoRotationXYZ.h>

SoTransform\* angleAxisTransform = new SoTransform;//模型变换节点

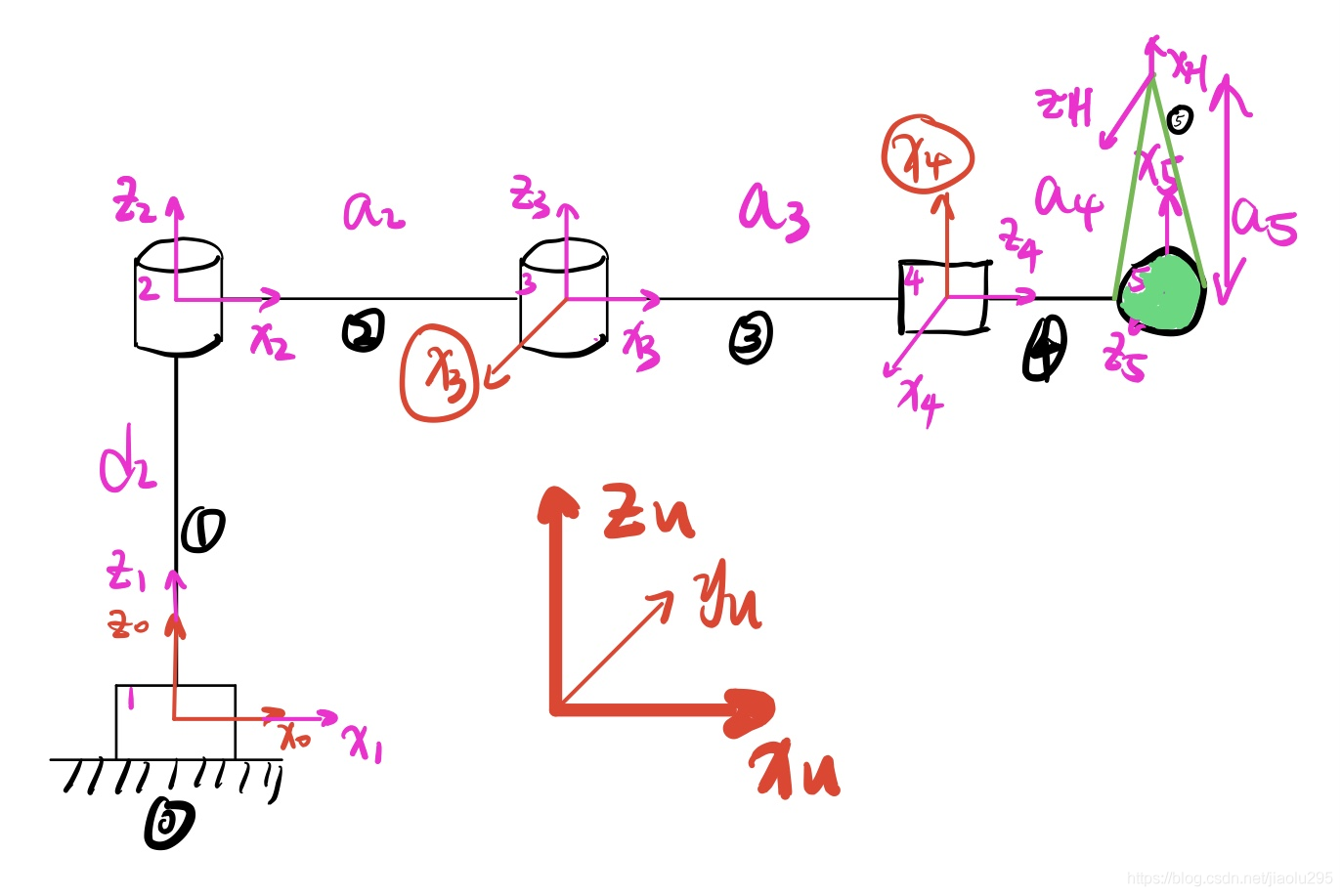
SoRotationXYZ\* angleAxisRotation = new SoRotationXYZ;

angleAxisTransform->translation.setValue(0, 0, 500);//将模型沿Z轴平移500mm

angleAxisTransform->rotation.setValue(SbVec3f(1, 0, 0), 1.5707963f);//将模型绕X轴旋转90度

angleAxisRotation->axis.setValue("Z");//设置旋转轴为Z（变换后的坐标系Z轴）

方式二：将坐标系与根据DH发建立的坐标系对应（原点对应，方向也对应）



[(39条消息) QT与Coin3D实现机器人的仿真\_jiaolu☞的博客-CSDN博客\_coin3d](https://blog.csdn.net/jiaolu295/article/details/117195749)

思路：根据节点SoMatrixTransform，将坐标变换描述成DH法中T0\*T1\*T2\*T3\*T4\*T5的形式，优点是可以进行正逆运动学解算操作。注意：变换节点是累乘的，所以angleAxis会向下传递，这样就有了前面轴转动后面部件跟踪一起转动的效果

A006 QStandardItemModel的用法

QStandardItemModel是标准的以项数据为基础的标准数据模型类，通常与QTableView组合成Model/View结构，实现通用的二维数据的管理功能。

1. QStandardItemModel：基于项数据的标准数据模型，可以处理二维数据。维护一个二维的项数据数组，每个项是一个QStandardItem类的变量，用于存储箱的数据、字体格式、对齐方式等。
2. QTableView:二维数据表视图组件，有多个行和多格列，每个基本现实单元是一个单元格，通过setModel()函数设置一个QStandardItemModel类的数据模型之后，一个单元格显示QStandardItemModel数据模型中的一个项。
3. QItemSelectionModel:一个用于跟踪视图组件的单元格选择状态的类，当在QTableView选择某个单元格、或多个单元格时，通过QItemSelectionModel可以获得选中单元格的模型索引，为单元格的选择操作提供方便，

这几个类之间的关系是：QTableView 是界面视图组件，其关联的数据模型是 QStandardItemModel，关联的项选择模型是 QItemSelectionModel，QStandardItemModel 的数据管理的基本单元是 QStandardItem。

参考链接：[Qt QStandardItemModel用法（超级详细） (biancheng.net)](http://c.biancheng.net/view/1869.html)

A007 for（auto i:v）遍历容器元素

C++11的新特性，v是一个可比案例的容器或流，比如Vector类型，i就用来在遍历过程汇总获得容器里的每一个元素。

例：遍历字符串

Std::string str = “hello, world”;

for(auto ch : str)

{

std::cout << ch << std::endl;

}

A008 QMetaType元类型系统

QMetaType类管理Qt的元类型系统，他讲类型名称与类型关联，从而可以在run-time时创建、销毁。Q\_DECLARE\_METATYPE() 宏静态，使类型可以用于基于模板的方法，qRegidterMetaType()方法动态让类型可以用于非模板方法。

QT内置类型都已注册enum QMetaType::Type,enum QMetaType::Type从0开始，其中0为UnknowType,1024为用户自定义类型起始位值。

Q\_DECLARE\_METATYPE()宏与int qMetaTypeId()：

此二者用于编译时，自定义类型只要满足以下条件就可以用宏 Q\_DECLARE\_METATYPE()类注册为元类型。

public default constructor

public copy constructor

public destructor

使用Q\_DECLARE\_METATYPE(T)时类型T（包括指针类型）必须是完全定义的，正常情况下就在类型定义之后注册，或者在每次 都要包含的私有头文件中。

对于前置声明类型的指针可以使用宏Q\_DECLARE\_OPAQUE\_POINTER()，之后可以正常注册。

如果T在namespace中， Q\_DECLARE\_METATYPE(T)必须在namespace之外，T要包含完全限定名：

namespace MyNamespace

{

...

}

**Q\_DECLARE\_METATYPE**(MyNamespace::MyStruct)

Q\_DECLARE\_METATYPE在compile time注册类型，使类型T可以用于所有的模板类，包括QVariant。  
如果要在QueuedConnections的信号槽连接中使用类型T，还需要使用qRegisterMetaType()来注册，因为QueuedConnections信号槽参数是动态查找的。  
qMetaTypeId<T>()可以在编译时获取注册的类型id，如果T没有使用Q\_DECLARE\_METATYPE注册，会产生编译错误。

qRegisterMetaType()方法与QMetaType::type()

qRegisterMetaType()就是将类型与类型名关联，之后可以用QMetaType::type()查找类型id

typedef QString CustomString;

qRegisterMetaType<CustomString>("CustomString");

QMetaType::type()类似qMetaTypeId返回类型id，不过是在runtime查找注册的类型，而qMetaTypeId在编译时。

const char \* QMetaType::typeName(int typeId)

同一类型的不同alias注册后的id为同一值，查找时获取的是最原始的类型名。

A009 QT下使用多字节字符集的方法

QT项目里的源代码默认的都是Unicode,原因是qmake.conf里的定义。在QT下使用多字节字符集的方法，需在.pro文件中做如下修改：

DEFINES  -= UNICODE

DEFINES  += UMBCS即可设置当前工程为多字节字符集

DEFINES += QT\_DEPRECATED\_WARNINGS；定义编译选项，当Qt的某些功能被标记为过时的，那么编译器会发出警告。

GreaterThan(QT\_MAJOR\_VERSION,4);QT+=widgets:这是Qt5为了兼容Qt4而专门设计的，语句大意为如果Qt版本大于4，则将widgets模块加入到QT变量中，从这一点，我们可以推敲出Qt4的widgets模块是和gui模块融为一体的，Qt5已将其分离出来。

A010 QT线程中CreateEvent的用法

CreateEvent:创建windows事件，用于判断线程退出，线程锁定方面；

功能描述：创建或打开一个命名的或无命名的时间对象；

Event有两种状态：发信号、不发信号。

SetEvent/ResetEvent分别将EVENT置位发信号与不发信号。

WaitForSingObject()等待，直到参数所指定的OBJECT成为发信号时才返回，OBJECT可以是EVENT,也可以是其他内核对象。

当创建一个线程时，这个线程是一个循环，不会运行一次就停止。这样带来一个问题，在这个死循环里要找到合适的条件退出，是怎么实现的呢?Windows里往往采用事件的方式来通知从线程运行函数退出来。原理：在这个死循环里不断使用WaitForSingleObject函数来检查事件是否满足，如果满足就退出线程，不满足就继续运行。当在线程里运行阻塞的函数时，需要在退出线程时，先要把阻塞状态变成非租塞状态，同时使用阻塞的SOCKET时，那么要先关闭SOCKET，再发送事件信号，才可以 退出线程的。

创建或打开一个命名或无名的事件对象。

1. CreateEvent

函数原型：

HANDLE CreateEvent(

  LPSECURITY\_ATTRIBUTES lpEventAttributes,   // 安全属性

  BOOL bManualReset,   // 复位方式

  BOOL bInitialState,   // 初始状态

  LPCTSTR lpName   // 对象名称

);

参数：

lpEventAttributes: [输入]一个指向SECURITY\_ATTRIBUTES结构的指针，确定返回的句柄是否可被子进程继承。如果lpEventAttributes是NULL，此句柄不能被继承。

bManualReset：

[输入]指定将事件对象创建成手动复原还是自动复原。如果是TRUE，那么必须用ResetEvent函数来手工将事件的状态复原到无信号状态。如果设置为FALSE，当事件被一个等待线程释放以后，系统将会自动将事件状态复原为无信号状态。

bInitialState：

[输入]指定事件对象的初始状态。如果为TRUE，初始状态为有信号状态；否则为无信号状态。

lpName：

[输入]指定事件的对象的名称，是一个以0结束的字符串指针。名称的字符格式限定在MAX\_PATH之内。名字是对大小写敏感的。

如果lpName指定的名字，与一个存在的命名的事件对象的名称相同，函数将请求EVENT\_ALL\_ACCESS来访问存在的对象。这时候，由于 bManualReset和bInitialState参数已经在创建事件的进程中设置，这两个参数将被忽略。如果lpEventAttributes是 参数不是NULL，它将确定此句柄是否可以被继承，但是其安全描述符成员将被忽略。如果lpName为NULL，将创建一个无名的事件对象。

2. WaitForSingleObject的用法

WaitForSingleObject的用法

DWORD WINAPI WaitForSingleObject(

\_\_in HANDLE hHandle,　　　　　　// 句柄

\_\_in DWORD dwMilliseconds　　　// 时间间隔

);

hHandle[in]对象句柄。可以指定一系列的对象，如Event、Job、Memory resource notification、Mutex、Process、Semaphore、Thread、Waitable timer等。

dwMilliseconds[in]定时时间间隔，单位为milliseconds（毫秒）.如果指定一个非零值，函数处于等待状态直到hHandle标记的对象被触发，或者时间到了。如果dwMilliseconds为0，对象没有被触发信号，函数不会进入一个等待状态，它总是立即返回。如果dwMilliseconds为INFINITE，对象被触发信号后，函数才会返回。

参考链接：<https://www.cnblogs.com/amwuau/p/8579050.html>

QMutex:提供的是线程之间的访问顺序化。其目的是保护一个对象、数据结构或者代码段、同一时间只有一个线程可以访问他。

QMutex和QMutexLocker是基于互斥量的线程同步类，QMutex定义的实例是一个互斥量，三个参数分别为：

Lock（）：锁定互斥量，如果另一个线程锁定了这个互斥量，他将阻塞执行直到其他线程解锁这个互斥量。

Unlock(0:解锁一个互斥量，需要与lock()配对使用。

tryLock():试图锁定一个互斥量，如果成功锁定就返回TRUE，如果其他线程已经锁定了这个互斥量，发挥false，但不阻塞程序执行。

问题：设置线程有信号/无信号是什么意思？

A011 C++头文件里class直接加类名的作用

在写 头文件的时间 经常会

有一些 class QFile 等这类 看似 是声明的东西..例如下面

#ifndef WINDOW\_H

#define WINDOW\_H

QT\_BEGIN\_NAMESPACE

class QTcpServer;

class QTcpSocket;

class QProgressBar;

class QLabel;

class QDialogButtonBox;

class QFile;

QT\_END\_NAMESPACE其 实直接在头文件里面#include < QDialogButtonBox> #include< QFile>也是可以的，像它这样做，好像是可以降低各个文件编译时的关联度，不会在改动了一下部分类的时候，引 发其他大量文件的重新编译，在做小工程的时候没什么区别，但是做大了，编译一次需要好几个小时的时候，这样做的优势就显现出来了

A012 QT学习之QString的arg方法

在QTde Qstring中，argue方法类似于C中的printf中使用的格式输出幅（只有有点类似）。

1. 使用arg(str1,str2,str3)这种方法进行替换。
2. 使用arg（str1）.arg(str2).arg(str3)进行替换。
3. 使用arg(int,int,int)进行替换。
4. 使用arg(str1,str2,str3)这种方法进行替换。

要求str1、str2、str3都为const QString &类型，即：

QString QString::arg(const QString & a1, const QString & a2, const QString & a3) const

同时，arg( )里的参数实现从1个到9个，也就是说最多只能有9个！即在实现中，arg的定义中参数最多的为：

QString QString::arg(const QString & a1, const QString & a2, const QString & a3, const QString & a4, const QString & a5, const QString & a6, const QString & a7, const QString & a8, const QString & a9) const

1. 使用arg（str1）.arg(str2).arg(str3)进行替换

简单说就是挨着替换呗。但请注意下面的形式

str = QString("%1 %2").arg("%1World", "Hello");

qDebug()<<str;

//输出为："%1World Hello"

str = QString("%1 %2").arg("%1World").arg("Hello");

qDebug()<<str;

//输出为："HelloWorld %2"

//第一个arg执行完后变为：QString("%1World %2").arg("Hello")

//再次执行后"Hello"替换的为%1

arg( )里的参数在进行替换时，只会按照数字从小到大的顺序进行替换，只有比当前数字小的所有数字都替换完成了，才会替换当前数字，否则将当前数字和%按字符串处理。且数字为自然数！

参考链接：[QT学习之QString的arg方法 - 九二 - 博客园 (cnblogs.com)](https://www.cnblogs.com/lomper/p/4135387.html)

A013 已知同一组点在不同坐标系下的坐标，如何求解两个坐标系之间的转换关系

问题描述：

已知同一组点在不同坐标系下的坐标（例如有点ABCDEF六个点在坐标系1下的坐标A1,B1,C1,D1,E1,F1同时又已知ABCDEF六个点在坐标系2下的坐标A2,B2,C2,D2,E2,F2），如何求解两个坐标系之间的转换关系？

问题解决：

方法一：

参考：

<https://blog.csdn.net/sinat_29886521/article/details/77506426>

补充说明：

该方法中的H为协方差矩阵，有关资料可以查阅https://www.jianshu.com/p/25bdab15a9d6

该方法的原始参考网址为：http://nghiaho.com/?page\_id=671

存在问题：在该方法中用到了矩阵知识中的奇异值分解法求解矩阵的+号逆，但是在我理解应该是求解Ma=b其中a和b分别代表着两组点集，然后通过求解＋号逆来求解方程的最小二乘解，得到M，但是这样求解出来的矩阵并不是一个酉矩阵，也就是说不是传统意义上的平移旋转矩阵，但是上面链接中的方法却可以得到一个合理的位姿变换矩阵，这和协方差矩阵有很大关系，最后求解的应该是协方差矩阵的齐次方程的小二解吧，具体也不太懂，数学太差了！

请能看懂的朋友在下面评论一下具体的原理，感激不尽！

方法二：

通过点集建立两个坐标系，直接求解坐标系的转换关系M，使得MA=B。

方法三：

如我上面所述，直接求+号逆，但是问题就是结果不是传统的位姿变换矩阵，懂得原理的朋友可以讲一下，我自己的理解是关于某个空间中的转轴变换的一个矩阵，而且该转轴不和原来的原点重合，不知道对不对。

A014 socket编程（C++）

1. WSAStartup()函数以及DLL的加载

WinSock（Windows Socket）编程依赖于系统提供的动态链接库(DLL)，有两个版本：

较早的DLL是 wsock32.dll，大小为 28KB，对应的头文件为 winsock1.h；

最新的DLL是 ws2\_32.dll，大小为 69KB，对应的头文件为 winsock2.h。

几乎所有的 Windows 操作系统都已经支持 ws2\_32.dll，包括个人操作系统 Windows 95 OSR2、Windows 98、Windows Me、Windows 2000、XP、Vista、Win7、Win8、Win10 以及服务器操作系统 Windows NT 4.0 SP4、Windows Server 2003、Windows Server 2008 等，所以你可以毫不犹豫地使用最新的 ws2\_32.dll。

[WSAStartup()函数以及DLL的加载 (biancheng.net)](http://c.biancheng.net/view/2130.html)

使用DLL之前，需调用WSASartup()函数进行初始化，以指明WinSock规范的版本，原型：

int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);

wVersionRequested 为 WinSock 规范的版本号，低字节为主版本号，高字节为副版本号（修正版本号）；lpWSAData 为指向 WSAData 结构体的指针。

1. 创建套接字

<https://www.cnblogs.com/helloworldcode/p/10610581.html>

原型：

/\* \* \_domain 套接字使用的协议族信息 \* \_type 套接字的传输类型 \* \_\_protocol 通信协议 \* \*/

int socket (int \_\_domain, int \_\_type, int \_\_protocol) \_\_THROW;

socket起源于UNIX，在Unix一切皆文件哲学的思想下，socket是一种"打开—读/写—关闭"模式的实现，可以将该函数类比常用的open()函数，服务器和客户端各自维护一个"文件"，在建立连接打开后，可以向自己文件写入内容供对方读取或者读取对方内容，通讯结束时关闭文件。

参数介绍：

第一个参数：关于协议族信息可选字段如下，只列出一般常见的字段。

| **地址族** | **含义** |
| --- | --- |
| AF\_INET | IPv4网络协议中采用的地址族 |
| AF\_INET6 | IPv6网络协议中采用的地址族 |
| AF\_LOCAL | 本地通信中采用的UNIX协议的地址族（用的少） |

第二个参数：套接字类型。常用的有SOCKET\_RAW，SOCK\_STREAM和SOCK\_DGRAM。

| **套接字类型** | **含义** |
| --- | --- |
| SOCKET\_RAW | 原始套接字(SOCKET\_RAW)允许对较低层次的协议直接访问，比如IP、 ICMP协议。 |
| SOCK\_STREAM | SOCK\_STREAM是数据流，一般为TCP/IP协议的编程。 |
| SOCK\_DGRAM | SOCK\_DGRAM是数据报，一般为UDP协议的网络编程； |

第三个参数：最终采用的协议。常见的协议有IPPROTO\_TCP、IPPTOTO\_UDP。如果第二个参数选择了SOCK\_STREAM，那么采用的协议就只能是IPPROTO\_TCP；如果第二个参数选择的是SOCK\_DGRAM，则采用的协议就只能是IPPTOTO\_UDP。

1. 向套接字分配网络地址—bind()

/\* \* \_\_fd:socket描述字，也就是socket引用

\* myaddr:要绑定给sockfd的协议地址

\* \_\_len:地址的长度

\*/

int bind (int \_\_fd, const struct sockaddr\* myaddr, socklen\_t \_\_len) \_\_THROW;

第一个参数：socket文件描述符\_\_fd即套接字创建时返回的对象;

第二个参数：myaddr则是填充了一些网络地址信息，包含通信所需要的相关信息，其结构体具体如下：

struct sockaddr

{

sa\_family\_t sin\_family; /\* Common data: address family and length. \*/

char sa\_data[14]; /\* Address data. \*/

};

在具体传参的时候，会用该结构体的变体sockaddr\_in形式去初始化相关字段，该结构体具体形式如下，结构体sockaddr中的sa\_data就保存着地址信息需要的IP地址和端口号，对应着结构体sockaddr\_in的sin\_port和sin\_addr字段。

struct sockaddr\_in{

sa\_family\_t sin\_family; //前面介绍的地址族

uint16\_t sin\_port; //16位的TCP/UDP端口号

struct in\_addr sin\_addr; //32位的IP地址

char sin\_zero[8]; //不使用

}

in\_addr 结构定义如下：

/\*

Internet address.

\*/

typedef uint32\_t in\_addr\_t;

struct in\_addr

{

in\_addr\_t s\_addr;

};

而sin\_zero 无特殊的含义，只是为了与下面介绍的sockaddr结构体一致而插入的成员。因为在给套接字分配网络地址的时候会调用bind函数，其中的参数会把sockaddr\_in转化为sockaddr的形式，如下：

struct sockaddr\_in serv\_addr;

...

bind(serv\_socket, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)；

需要注意的是s\_addr是一种uint32\_t类型的数据，而且在网络传输时，统一都是以大端序的网络字节序方式传输数据，而我们通常习惯的IP地址格式是点分十进制，例如：“219.228.148.169”，这个时候就会调用以下函数进行转化，将IP地址转化为32位的整数形数据，同时进行网络字节转换：

in\_addr\_t inet\_addr (const char \*\_\_cp) \_\_THROW; //或者 int inet\_aton (const char \*\_\_cp, struct in\_addr \*\_\_inp) \_\_THROW; //windows无此函数

1. 进入等待连接请求状态

给套接字分配了所需的信息后，就可以调用listen()函数对来自客户端的连接请求进行监听（客户端此时要调用connect()函数进行连接）

/\* Prepare to accept connections on socket FD.

N connection requests will be queued before further requests are refused.

Returns 0 on success, -1 for errors. \*/

extern int listen (int \_\_fd, int \_\_n) \_\_THROW;

第一个参数：socket文件描述符\_\_fd，分配所需的信息后的套接字。

第二个参数：连接请求的队列长度，如果为6，表示队列中最多同时有6个连接请求。

这个函数的fd(socket套接字对象)就相当于一个门卫，对连接请求做处理，决定是否把连接请求放入到server端维护的一个队列中去。

1. 受理客户端的连接请求

listen()中的sock(\_\_fd : socket对象)发挥了服务器端接受请求的门卫作用，此时为了按序受理请求，给客户端做相应的回馈，连接到发起请求的客户端，此时就需要再次创建另一个套接字，该套接字可以用以下函数创建：

extern int accept (int \_\_fd, struct sockaddr \*addr, socklen\_t \*addr\_len);

函数成功执行时返回socket文件描述符，失败时返回-1。

第一个参数：socket文件描述符\_\_fd，要注意的是这个套接字文件描述符与前面几步的套接字文件描述符不同。

第二个参数：保存发起连接的客户端的地址信息。

第三个参数： 保存该结构体的长度。

1. send/write发送信息

linux下的发送函数为：

ssize\_t write (int \_\_fd, const void \*\_\_buf, size\_t \_\_n) ;

windows下发送函数为：

ssize\_t send (int sockfd, const void \*buf, size\_t nbytes, int flag) ;

第四个参数是传输数据时可指定的信息，一般设置为0。

1. recv/read接受信息

linux下的接收函数为：

/\* Read NBYTES into BUF from FD.

Return the number read, -1 for errors or 0 for EOF.

This function is a cancellation point and therefore not marked with \_\_THROW. \*/

ssize\_t read (int \_\_fd, void \*\_\_buf, size\_t \_\_nbytes);

而在windows下的接收函数为

ssize\_t recv(int sockfd, void \*buf, size\_t nbytes, int flag) ;

1. 关闭连接

/\* Close the file descriptor FD.

This function is a cancellation point and therefore not marked with \_\_THROW. \*/

int closesocket (int \_\_fd);

退出连接，此时要注意的是：**调用close()函数即表示向对方发送了EOF结束标志信息**。

1. 关闭套接字

WSACLeanup();

客户端参考：<https://www.cnblogs.com/helloworldcode/p/10610581.html>

A015 生成DLL及调用

1. 生成步骤：
   1. 新建QT Project ->Library->C++;
   2. 共享库选择需要的模块；
   3. [(16条消息) Qt 动态库的编译和调用\_风云的的博客-CSDN博客\_qt编译动态库](https://blog.csdn.net/kidults/article/details/86595694)

A016 自定义数据类型 qRegisterMetaType 的使用

QT中使用自定义类型，需要使用qRegisterMetaType对子定义类型进行注册，方法：

* 1. 注册位置：在第一次使用此类连接跨线程的signal/sloat之前，一般在当前类的构造函数中进行注册。
  2. 注册方法：在当前类的顶部包含：#include<QMetaType>,构造函数中加入代码：qRegisterMetaType<MyClass>(“Myclass”);
  3. Myclass的应用类型需要单独注册：qRegisterMetaType<MyClass>(“MyClass”);

A017 QT工具类QACTion、边框QFrame、菜单QMenu梳理

## 1工具类QAction

QAction类提供了抽象的用户界面action,这些action可以被放置在窗口部件中。应用程序可以通过菜单、工具栏按钮自己键盘快捷键来调用通用的命令。由于用户期望每个命令都能以相同方式执行，而不管命令所使用的用户界面，这个时候Action就是十分有用。

Actions可以被添加到菜单和工具栏中，并且可以自动保持在菜单和工具栏中的同步。例如，在一个字处理软件中，如果用户在工具栏中按下了Bold按钮，那么菜单中的Bold选项就会自动被选中。

Actions可以作为独立的对象被创建，但是我们也可以在构建菜单的时候创建他们；QMenu类包含了非常简便的方法用于创建适合用作菜单的actions。

QAction可以包括一个图标，菜单文本，快捷键，状态文本，"What`s This"文本以及一个tooltip。这些内容的绝大部分都可以在构造函数中设置。也可以通过setIcon()，setIconText()，setShortCut()，setStatusTip()，setWhatsThis和SetToolTip()函数分别设置。对于菜单项，我们还可以通过setFont()单独为它们设置font。

一旦QAction被创建了，那么就必须将它添加到相关的菜单和工具栏上，然后将它们链接到实现相应action功能的槽函数上。例如：

openAct = new QAction(QIcon(":/images/open.png"), tr("&Open..."), this);

openAct->setShortcuts(QKeySequence::Open);

openAct->setStatusTip(tr("Open an existing file"));

connect(openAct, SIGNAL(triggered()), this, SLOT(open()));

fileMenu->addAction(openAct);

fileToolBar->addAction(openAct);

建议将actions作为使用它们的窗口的孩子创建。在绝大多数情况下，actions都是应用程序主窗口的孩子。

**自定义类PAction：**集成Qt类QACTion、自定义类PControl。设置对象（框架窗口、上下文、菜单动作、工具动作、菜单栏工具栏）等自定义类、类型的文本、浮标、图片显示以及触发显示等。

应用：ARTS6025中Paction类：构造函数中设置菜单栏（PCT\_MATION）、工具栏（PCT\_TACTION）中action的名称、图片、并连接。两肱抱丹混元劲，借力腾空沾身起。有触必应随感而发，劲随意走运转由心。形神皆妙与道合真。

**使用QACTion中setText（**）如果间操作添加到菜单中，菜单选项将由图标（如果有）、文本和快捷方式（如果有）组成，如果文本没有在构造函数中显示设置，或者没有使用setText（），那么动作的猫叔图标文本将被用作文本。没有默认文本。

**QKeySequence:setShortcut**(QKeySequence(QString(“Ctrl+”)+shortcut));在最常见的行驶中，建序列描述了必须一起使用以执行某些操作的键的组合。建序列与QACTion对想一起使用，已制定可以使用那些键盘快捷键来出发动作。

**构造函数PAction():**三个参数：悬浮显示内容、控件类型、快捷方式；完成悬浮显示、设置图片等功能。

**类PAction中函数Hook():**连接按钮触发显示。涉及信号和槽的五种返回值：

1. QT：：AutoConnection 0 自动连接，默认方式。信号发出的线程和槽的对象在一个线程的时候相当于DireectConnection,如果是在不同的线程，相当于QueuedConenctioin。
2. Qt::DirectConnection 1 直接连接，相当于直接调用槽函数，但是当信号发出的线程和槽的对象不再一个线程的时候，则槽函数是在发出的信号中执行的。
3. Qt：：QueuedConnection 2 队列连接，内部通过postEvent实现的。不是实时调用的，槽函数永远在槽函数对象所在的线程中执行。如果信号采纳数是引用类型，则会另外复制一份线程安全的。
4. Qt::BlockingQueuedConnection 3 阻塞连接，此连接方式只能用于信号发出的线程（一般是选好对象的线程）和槽函数的对象不再一个线程中才能用。通过信号量+postEvent实现的。不是实时调用的，槽函数永远在槽函数对象所在的线程中执行。但是发出信号后，当前线程会阻塞，等待槽函数执行完毕后才继续执行。
5. Qt：：UniqueConnection ) 0x80 防止重复连接。如果当前信号和槽已经连接过了就不在连接了。严格说来不算连接方式。

AutoConenction模式为默认模式，可以看做是DirectConnectioin和QueuedConnection的自动选择，发出信号、调用槽的方式也可简单的分为两种：同步调用和异步调用。同步调用：发出信号后，当前线程等待槽函数执行完毕后才继续执行。异步调用：发出信号后，立即执行剩下的逻辑，不关心槽函数什么时候执行。五行十二形：五行指的是金木水火土即劈、崩、钻、炮、横五种力，十二象形对应龙、虎、候、鸡、马、鹞、燕、蛇、、鼍、鹱、鹰、熊等十二种动物。

|  |  |  |  |
| --- | --- | --- | --- |
| 线程/模型 | DirectConnection | QueuedConnection | BlockingQueuedConnection |
| 相同线程 | 直接调用，同步调用 | 通过事件队列进行调用，异步调用 | 不可用 |
| 不同线程 | 直接调用，同步调用。槽函数在发出信号的线程执行。有线程安全隐患。 | 通过事件进行队列调用。一部调用。槽函数在对象所在的线程执行。线程安全。 | 通过事件进行阻塞调用。同步调用函数在对象所在的线程执行。线程安全。 |
| Qt时间循环依赖 | 直接调用，不依赖Qt时间循环 | 通过事件进行队列调用。依赖槽函数所在对象的线程必须启用Qt事件循环。 | 通过事件进行队列调用，用信号量阻塞、依赖、槽函数所在对象的线程必须启用Qt事件循环。 |

[Qt信号槽的一些事（第一次知道信号还有返回值，以及Qt::UniqueConnection） - findumars - 博客园 (cnblogs.com)](https://www.cnblogs.com/findumars/p/5176419.html)

## 2.自定义类PControl

**构造函数PControl()：构**造函数中初始化对象类型（PCONTYPE）并给对象类型变量m\_UIType赋值。

**析构函数中~PControl():**调用ClearAllControl（）；

**清除UI函数ClearUI():**使用函数GetMenbers（）得到对象类型，再用for auto遍历该对象的所有存在的对象，如果对象指针存在则断开连接、父指针设置为空并隐藏。对象减少标志位构造器PList m\_unrefmenbers.append末尾增加。

**插入控件函数InsertControl():**将对象插入到指定索引的对象标志位构造器PList m\_unrefmenbers，计数标志位m\_refCount加1，并更新UI。

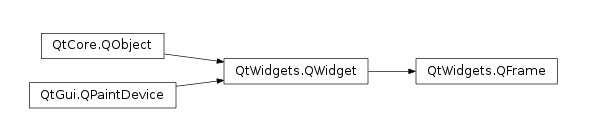
**增加控件函数AddControls();**将QVector类型的对象加入到对象标志位构造器PList m\_unrefmenbers中，技术标志位加1 并更新。

**删除指定索引的对象RemoveControl：**

## 3.类QFrame

继承QFrame、PControl。其中QFrame是一个基类可直接使用。主要用来控制一些边框样式，例如：凸起、凹下、阴影、线宽。其继承QWidget,功能有创建QFrame对象、框架形状、框架阴影、框架的几个线宽、框架样式、框架矩形。

1、继承图：



**构造函数PFrame():**继承自QFrame(),初始化器类型PControl、类型m\_ID.并设置模态属性对于顶层窗口，在Designer中设置的背景样式表默认是不生效的。如果设置了顶层窗口的背景样式表，要想生效需要在顶层窗口的构造函数中设置其属性setAttribute(Qt::WA\_StyledBackground)，告诉窗口应该使用设置的样式表进行渲染空间背景。

**关闭事件函数closeEvent(QCloseEvent\*):**输入参数为“当鼠标点击窗口左上角的关闭按钮时所触发的函数”触发信号signal\_close(windowTitle).该信号在PTestgroupFrame中连接到PSheetWidget类中用在在多文本窗口中关闭对应子窗口。

**启动事件函数showEvent(QShowEvent\*):**QShowEvent()显示事件有两种，由窗口系统（自发）引起的显示事件和内部显示事件。自发（QEvent::spontaneous()）显示事件在窗口系统显示窗口之后立即发送；在图标化后重新显示顶级窗口时，也会发送他们。内部显示事件在窗口小部件变得可见之前传递。

**获取指定索引、名称对象函数：**GetFrameByName（QString name）/ GetFrameByName（int ID）.获取指定索引、名称对象的指针。

**创建菜单栏函数CreateMenuBar():**创建菜单栏对象menuBar插入到主框架并返回该菜单栏指针。在该指针上课继续使用InsertControl插入具体菜单对象。

**创建工具栏函数CreateToolBar（）：同上。**

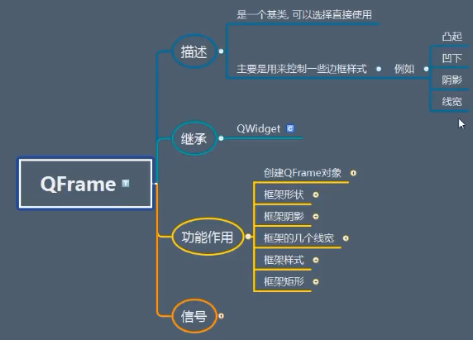
**附加函数Attach():**在管理类对象g\_app上操作，首先设置对应框架的菜单栏为空、隐藏并断开所有的工具栏连接。然后获取对象的类型添加到对应的菜单栏、工具栏上。加入你什么都看不见、听不见、摸不到什么都没想却又没死掉、没睡着非常清醒那么你是谁？ 想象不出来样子但应该还是我。确实没法去想象至于进入那种状态才能体会其中的妙趣，元神的概念无法用语言准确去描述。人的日常思维随着外援与信念变化流转不惜成为“识神”当识神退去之后那种村崔的意识状态就接近于所谓的“元神”。人们在偶尔的灵光一现中可能捕捉到这种状态却很难稳定的维持也不能随意的进出这种状态。如果可以稳定的出入这种状态不论采用何种休整方式都可以成成为“元神出现”的境界。这并不等于一时世界是一片空白元神自然的外感会衍生处很多念是一种很玄妙的体验。识神随时变化而元神清明纯粹所谓伤了元神就是本该清明纯粹的元神因某种缘由留下了种种痕迹或阴影也会反过来影响到感应和判断。比如没有人却看见了人没有声音却听见了声音俗话说见鬼了、撞邪了道理莫过如此。这种影响可能是短暂的可以自我调节克服也可能是永久的无法磨灭。如果情况很严重导致主体对外界客体做出错误的反应那就是行为失控这个人疯了。如果在某种特定的刺激下行为才会失控那就是间歇性精神病。

“行者既觉知魔事即当却之。却法有二：一者修止却之。凡见一切外诸恶魔境悉知虚诳不忧不怖亦不取不舍妄计分别息心寂然彼自当灭。二者修观却之。若见如上所说种种魔境用止不去即当反观能见之心不见处所彼何所恼如是观时寻当灭谢……。

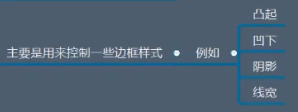
若诸魔境恼乱行人、或经年月不去但当端心正念坚固不惜身命莫怀忧惧。当诵大乘方等诸经治魔咒默念诵之存念三宝。若出禅定亦当诵咒自防忏悔惭愧、及诵波罗提木叉。邪不干正久久自灭。魔事众多说不可尽善须识之。

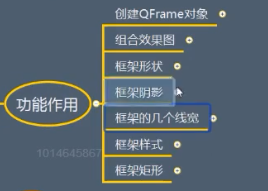
是故初心行人必须亲近善知识为有如此等难事。是魔入人心能令行者心神狂乱或喜或忧因是成患致死……取要言之若欲遣邪归正当观诸法实相善修止观无邪不破。故释论云：除诸法实相其余一切皆是魔事。如偈中说：若分别忆想即是魔罗网。不动不分别是则为法印。”（注：此段出自《修止观坐禅法要》原文较长书中只是节录。）

2、QFrame描述：主要用来控制一些边框样式。

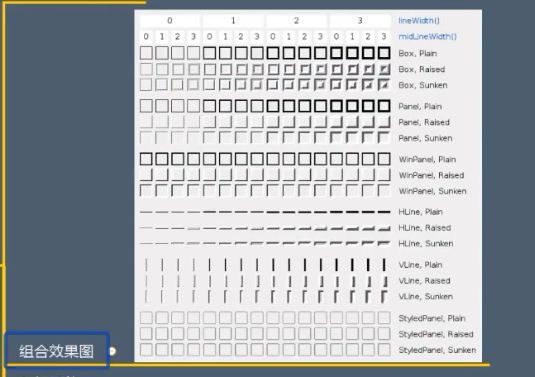


1. 继承QWdget: 所以可以直接展示。
2. QFrame功能作用：

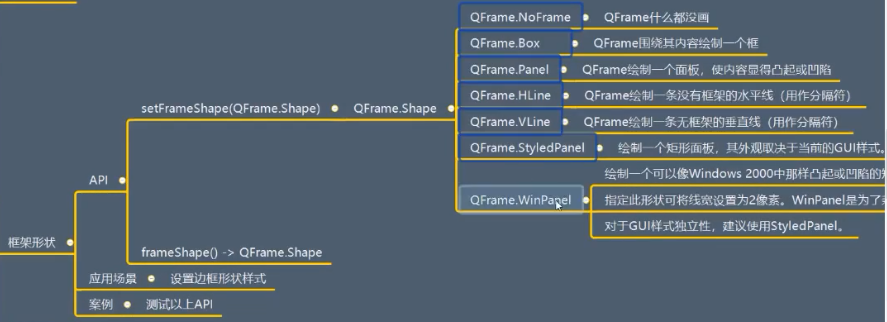




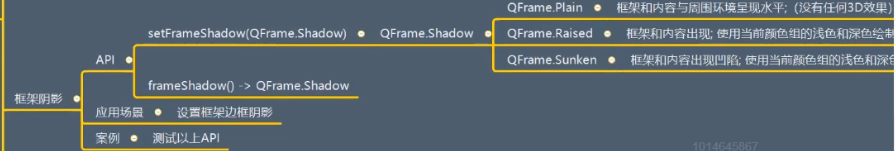
1. QFrame功能作用之组合效果图：它有形状、阴影现况组合在一起。一般根据这个效果图选择所需参数。



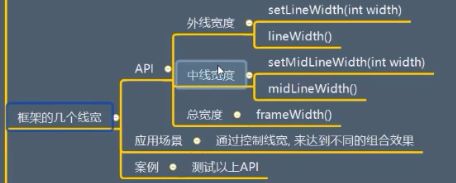
1. QFrame功能作用之框架形状：



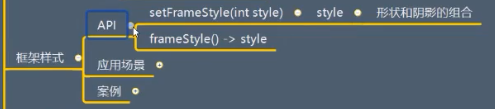
1. QFrame功能作用之框架阴影



7、QFrame能作用之框架的几个线宽：



8QFrame框架样式：



只需要将形状 和 阴影  按位或  就可以了。

9Frame 功能作用之框架矩形：



## 4类PMenu继承QMenu、PControl

PMenu功能：初始化对象、增加控件、菜单、分隔符、更新控件界面等。

**构造函数PMenu():**继承QMenu（），初始化对象类型为PCT\_MENU.使用setStyleSheet设置菜单栏样式，使用setMinimumwidth(200)设置其宽度；调用初始化函数Init();

**增加动作函数AddAction():**输入动作文本和快捷键方式；将该操作插入到对象的标志构造器m\_remenbers中，并返回该操作。

**界面更新函数UpdateUI() ：**首先使用clear()删除菜单拥有的、没有显示在任何其他小部件中的操作将被删除。之后获取对象的数据成员通过对象类型（PCT\_MACTION/PCT\_MENU/PCT\_SEPARATOR）来添加对应的动作。

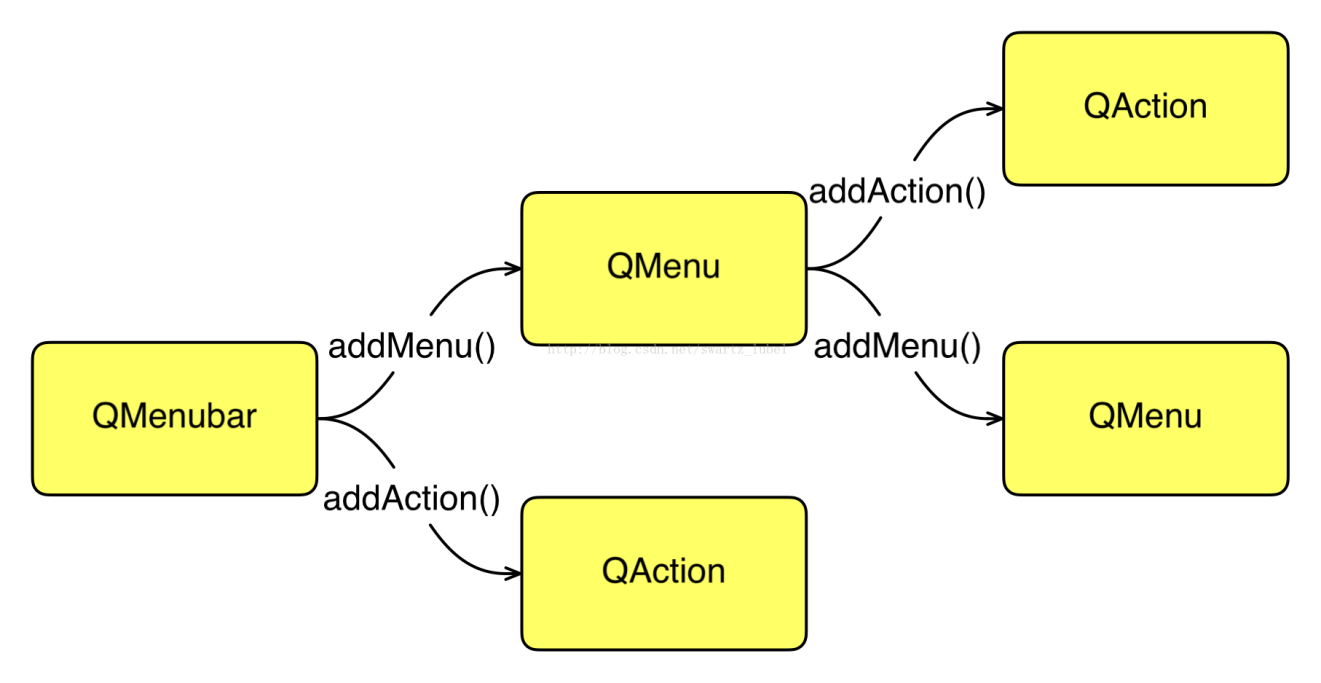
四面荷花三面柳 一城山色半城湖

QT建立菜单有三个类很重要：

QMenuBar:就是所有窗口的菜单栏，在此基础上添加不同的QMenu、QAction;

QMenu:菜单栏里面菜单，可以显示文本和图标，但是并不负责执行操作，有点类似label的作用；

QAction：Qt将用户与界面进行交互的元素抽象为一种“动作/操作”，使用QAction才是真正的部件。



**PMenu构造函数中使用setStyleSheet来设置图形界面的外观：**例

关于样式表的级联属性:

看下面代码的不同

1 btn1->setStyleSheet("QPushButton{color:red}"); //设定前景颜色，就是字体颜色

2 btn1->setStyleSheet("QPushButton{background:yellow}"); //设定背景颜色为红色

和

1 btn1->setStyleSheet("QPushButton{color:red；background:yellow}");

　　第一个代码只能显示黄色背景，第二个显示红色字体，黄色背景。所以对于同一个部件，要在同一个setStyleSheet(...)中完全写出来，否则对于该部件来讲，只有最后一个setStyleSheet(...)起作用。

18 btn3=new QPushButton(this);

19 btn3->setStyleSheet("QPushButton{background-image:url(image/1.png);background-repeat: repeat-xy;background-position: center;background-attachment: fixed;background-attachment: fixed;background-attachment: fixed;;background-clip: padding}");

20 //设定按钮的背景图片,background-repeat可以设定背景图片的重复规则，这里设定仅在xy方向都重复，所以图片会被重复一次

21 //background-position用来设定图片的位置，是左(left)还是右(right)，还是在中间(center),是上(top)还是底部(bottom)

22 //background-attachment用来这定背景图片是否卷动或者和窗口大小相匹配，默认是卷动的

23 btn3->setText("Button3");

24

25 btn4=new QPushButton(this);

26 btn4->setStyleSheet("QPushButton{border: 3px solid red;border-radius:8px}"); //设定边框宽度以及颜色

27 //可以使用border-top，border-right，border-bottom，border-left分别设定按钮的上下左右边框，

28 //同样有border-left-color, border-left-style, border-left-width.等分别来设定他们的颜色，样式和宽度

29 //border-image用来设定边框的背景图片。

30 //border-radius用来设定边框的弧度。可以设定圆角的按钮

31 btn4->setText("Button4");

32

33 //字体设定

34 //font-family来设定字体所属家族，

35 //font-size来设定字体大小

36 //font-style来设定字体样式

37 //font-weight来设定字体深浅

38 //height用来设定其高低

39 //selection-color用来设定选中时候的颜色

40 edit1=new QLineEdit(this);

41 edit1->setStyleSheet("QLineEdit{font: bold italic large /"Times New Roman/";font-size:25px;color:rgb(55,100,255);height:50px;border:4px solid rgb(155,200,33);border-radius:15px;selection-color:pink}");

42

## 5PMenuBar 继承QMenuBar、PControl

**类PToolBar继承QToolBar、PControl**

这两个类结构类似包含初始化函数Init(重载PControl)、得到类名函数GetClassName、得到类指针函数GetQWidgetPtr、增加菜单、动作函数AddMenu、AddAction.

**构造函数PMenuBar():**使用setStyleSheet()设置菜单栏的样式，setFixedHeight(40)设置菜单栏高度。

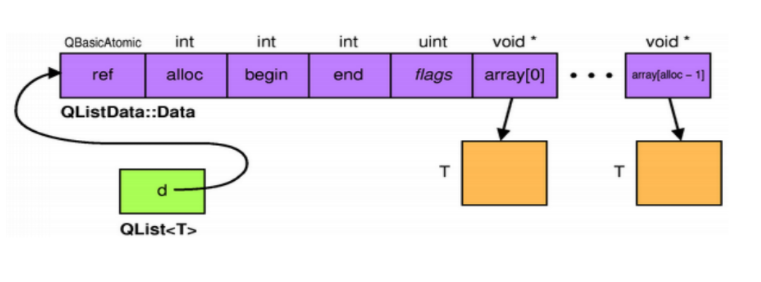
# QT模板库

QTL(Qt Template Library)是一套提供对象容器的模板。QTL提供了对象的链表、对象的矢量（动态数组）、从一个类型到另一个类型的映射（或称为字典）和祥光的迭代器和算法。容器是包含和管理其他对象的一个对象，并且提供迭代器对被包含的对象进行访问。

## QList的用法

QList是一种典型的数据链表，而不是真正意义上的链表。根据参数T的不同，QList有两种组织方式

种组织方式。通常请款下，QList有一个指针数组用于保存用new操作符在堆上分配的对象的指针，是的在容器的中间插入和删除的速度比参数不可拷贝的类型QVector快，因为指针是可以用内存拷贝的。



是QT的一种泛型容器类。它以链表方式存储一组值，并能对这组数据进行快速索引，还提供了快速插入和删除等操作。QList、QLinkedList、和QVector提供的操作极其相似：

QVector应该是默认首选的容器；

QList是最广泛用于QtAPI中传递值的容器；

QList不是指针结构体类型的链表，真正的结构体指针链表是QLinkedList.

**QList声明及初始化，**可以保存基本数据类型，如int，double，BOOL等；但是在存储QOBject对象及其子类对象时，不能直接存对象本身，而是只能存对象地址；

1. 声明QList

声明简单：

//基本数据类型

QList<int> integerList;

QList<QDate> dateList;

//存储QObject对象的地址

QList <QWidget \*> ptr\_widgets;

QList <QObject \*> ptr\_objects;

1. 初始化QList

初始化也很简单；若没初始化，链表默认为空的。

QList<QString> list = { "one", "two", "three" };

QWidget \* widget1 = new QWidget();

QWidget \* widget2 = new QWidget();

QList<QWidget \*> ptr\_widgets = {widget1, widget2};

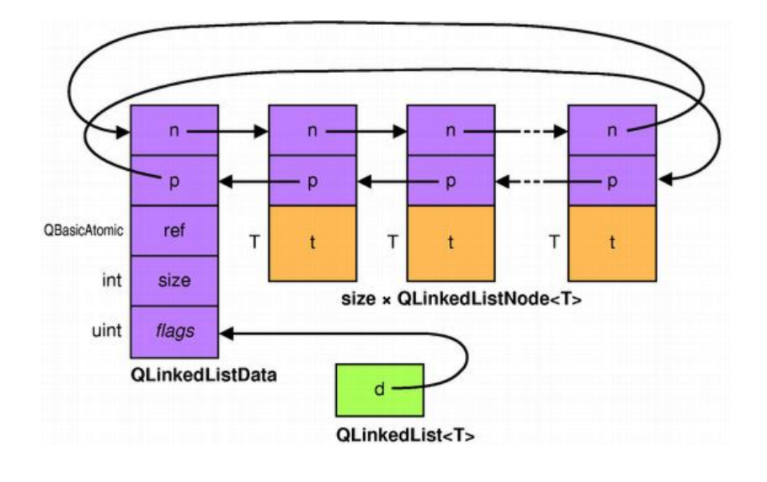
③ 增加节点：insert() 插入节点到；append() 尾部添加节点；prepend()头部添加节点；operator<<()尾部添加节点或链表；operator+=（）尾部添加节点；

④ 删除节点：removeAt()删除节点；removeFirst()删除头结点；removeLast（）删除尾节点（）；

1. 移动节点：move()立案表内移动节点位置；swap()连个链表交换值。

**QLinekedList用法：**

QLinkedList<T>提供一个双向链表，插入删除操作在常数时间完成。



虽然插入、删除操作是常数时间，但QLinkedList<T>只有在对象数量超过200对象时才能看出优势。对于小于200个对象的链表，QList<T>提供更好的性能，耗费更小的内存。

QLinkedLIst<T>的优点是数据的插入和删除很快，但在随机位置的访问会很慢。与QVector<T>不同，QLinkedList<T>并没有提供重载的[]操作符，只有使用append（）函数，或者<<操作符进行数据的添加。

**QVector模板**

QVector<T>提供了[]的重载，但对于只读操作，由于不需要深拷贝，使用at()函数的效率会较高。

QVector<double> v(2);

V(0) = 1.1;

V(1) = 1.2;

如果实现不知道vector的长度，可以创建一个空参数的vector，然后使用append（）函数添加数据：

QVector<double> v;

v.append(1.1);

v.append(1.2);

QVector <T>重载了<<,可以直接使用<<操作符，

QVector<double>v;

V<<1.1<<1.2;

如果QVector<T>中的数据没有被现实地赋值，那么，数据项将使用加入类的默认构造函数进行舒适化，如果是基本数据类型和指针，则初始化为0.

## 7按钮列表类PBtnListWidget

**构造函数PBtnListwidget():**setFixedwidth(150)设置按钮列表的宽度。SetStyleSheet()设置按钮风格样式。

**包含增加按钮函数AddBtn**：首先获取按钮列表的最后一行QLayoutItem\*stretch,然后移除最后一行的空白页，添加输入参数按钮对象btn，再添加空白行，最后设置按钮使能并将该按钮增加到按钮组。

QVBoxLayout直布局指的是将所有控件从上到下（或者从下到上）依次摆放。QVBoxLayout 中还可以放置其它控件，比如 QLabel 文本框、QLineEdit 单行输入框等。

项目中使用 QVBoxLayout 布局控件，需提前引入<QVBoxLayout>头文件。每个 QVBoxLayout 控件本质都是 QVBoxLayout 类的实例对象，该类提供了两个构造函数，分别是：

QVBoxLayout()

QVBoxLayout(QWidget \*parent)

创建 QVBoxLayout 布局时，可以指定父窗口，那么它将作为父窗口中布局其它控件的工具；也可以暂时不指定父窗口，待全部设置完毕后再将其添加到某个窗口中。

常用方法：

**void QBoxLayout::addWidget**(QWidget \*widget, int stretch = 0, Qt::Alignment alignment = Qt::Alignment())。 向布局管理器中添加指定的widget控件。默认情况下，新加入控件的 stretch 拉伸系数为 0，表示该控件不会过多占用布局管理器的空间；alignment 是一个枚举类型参数，默认的枚举值也是 0，表示该控件会填满占用的整个空间。

**void QBoxLayout::addStretch(**int stretch = 0)添加一个空白行，整个窗口中除了控件占用的区域外，其他区域可以有多个空白行分摊，分摊比例取余于各个空白行设置的stretch的值。strech 参数的默认值为 0，表示当窗口很小时，空白行可以不占据窗口空间。当窗口中包含多个 strech 值为 0 的空白行时，它们会平分窗口中的空白区域。

**通过setLayout()方法，**将layout被成功地添加到widget窗口中。当然，我们也可以在创建 layout 对象时指定 widget 作为它的父窗口，两种方式是完全等价的。

**获取所有按钮函数GetAllBtn():**使用findChildren返回对应按钮组列表。

**Qt之findChildren()：在 Qt 中，findChildren() 函数用于在对象的子树中查找所有符合条件的子对象，并返回它们的指针。该函数返回一个 QObjectList 类型的对象，其中包含所有符合条件的子对象。**

**Q**T编程中常会有多个控件嵌套，如：主窗体是QMainWindow，中央挂件是QTabWidget, 而每一个QTabWidget子窗口中又有QTextEdit，这样得组合就是经典得多标签页文本编辑器样式。通常我们可以很容易得获得QMainWindow上QTabWidget控件得对象，但是想要直接操作QTabWidget中嵌套得QTextEdit就不能直接引用.

QList<T> QObject::findChildren(const QString &name=QString()) const

QList<QLineEdit \*> lineEdit = tabWidget.findChildren<QLineEdit \*>();

这句话的意思就是在tabWidget挂件中寻找所有得QLineEdit控件，然后返回给QList容器（QList容器中每一个元素都为 QLineEdit \*类型），下面给出一个实例：

描述：主窗体为QMainWindow，中央挂件为QTabWidget，每个子页面包括若干个QLineEdit（注意：Tab1页面有两个，Tab2页面有1个），当单击“setText”按钮时候，自动给每一个QLineEdit填充文字。

关键代码如下：﻿

void MainWindow::on\_pushButton\_clicked()

{

QList<QLineEdit \*> lineEdit=ui->tabWidget->findChildren<QLineEdit \*>();

for (int i=0;i<lineEdit.count();i++)

lineEdit.at(i)->setText("www.hicjiajia.cn");

}

这样对我们操作多嵌套式得控件确实很方便，但是又有一个问题了，这种方法会将所有QLineEdit都给影响到了，但是我现在想只让某个特定得QLineEdit来响应怎摸办，还好，findChildren()函数还是可以带参数得：

QList<QLineEdit \*> lineEdit = tabWidget.findChildren<QLineEdit \*>("lineEdit");

## 子测试界面类PFramePackage

**构造函数PFramePackage():**继承PFrame(),初始化m\_content()。使用setWindowIcon（）设置测试界面图标。并调用初始化函数Init（）；

**初始化函数Init():**调用自定义布局函数SetLayout(),设置子界面布局没有空白行，边距设为0.当对应子界面存在时创建菜单栏、工具栏并插入g\_app中的“FileMenu”“HomeAction”.

## 子测试界面填充类ptestframe.cpp

在ptestframe.cpp中建立性能测试及校准模块子类PJoinModule、PSixAxisModule;在两个子类的初始化函数中添加各自模块的控制面板及3d、测试条件结果部分。

Explict关键字：智能修饰只有一个参数的类构造函数，指定构造函数或转换函数（C++11起）为显式，即它不能用于隐式转化和赋值初始化。

## 控制面板类ptestctrlwidget.cpp

类PJointCtrl构造函数：①注册测试状态枚举变量TestStatus，设置界面风格；②实用代码设置垂直排列、循环次数、位姿点数、开始（停止、下一步、暂停）等按钮；③初始化标签（循环、姿态）状态，设置开始等按钮的信号槽函数。

事件函数PJointCtrl::showEvent():将开始按钮、下一步按钮关联到菜单中

状态更新函数：OnUpdateTestStatus(int status):

## 8、主框架类PMainFrame 继承QMainWindow ,PFrame

**构造函数PMainFrame中实现：**设置主框架名称SetName(g\_app->GetName()),其中g\_app->GetName()为PApp中函数，在main.cpp中设置框架名称。SetName为类PFrame中函数通过setWindowTitle设置框架名称。调用初始化函数。

**初始化函数Init（）：**调用布局函数SetLayout,创建状态栏函数CreateStatusBar()

**布局函数SetLayout（）：**创建中心窗口m\_centralwidget,布局对象m\_mainLayout.通过该对象设置其左右边距、上下边距。

**QVBoxlayout类**可调用函数setMargin()设置空间于窗体的左右边距，setSpacing()设置各个控件之间的上下间距；setCentralWidget()设置为中心窗体部件。

**QStatusBar**提供一个水平条来显示状态信息，包括：临时信息（一般的提示信息）、正常信息（页数）、永久信息（版本信息）。使用showMessage()来显示一个临时消息，他会出现在状态栏的最左边，用addWidget()讲一个QLabel加到状态栏上用于显示正常信息，生成到状态栏的最左边，可能会被临时消息覆盖。使用addPermanentWidget()来添加一个QLabel赖在状态栏的最右边显示永久信息，不会被临时信息覆盖。在状态栏的最右端还有一个QSizeGrip部件用来调整窗口大小，我们可以通过setSizeGripEnabled()函数来禁用它。也许你想通过使用Qt Designer来可视化地完成窗口布局。

## 9滑动效果类PSlideViewWidget继承PFrame

**构造函数PSlideViewWidget():**设置滑块风格、名称、页边距等。使用了**setObjectName(),**

eg:QLabel \*m\_pLb1 = new QLabel(this);

pLb1->setObjectName(“testLb1”);

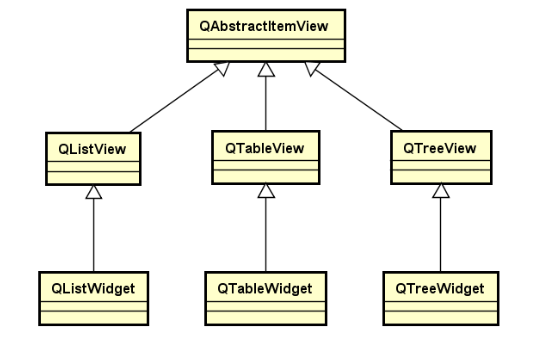
QLabel是一个总体的抽象；m\_pLb1是抽象的一个实例；而ObjectName相当于给这个实例命了个名字，唯一的一个标识符，指针的名字相当于是内部约定的代号，比如说代码内部可以通过m\_pLbl来对这个label进行相应的操作；但是如果外界想要访问这个label，则无法通过指针名m\_pLbl来进行访问，因为这个名字仅仅限于内部使用，对于外部的人来说，如果想要访问这个控件，必须要知道这个控件的ObjectName，这个ObjectName相当于是一个键值对的“键”，其对应的“值”就是这个label，一个完整的映射关系。

**初始化函数Init():**SetLayout()设置页面布局样式；创建“下一个”“上一个”按钮**，**并将按钮连接到对应的向右、左滑动函数Scroll()。新建列表对象m\_listwidget = new QListwidget；设置项目布局应该的流向方向、拓展方式、滑动槽等。在创建水平、竖直布局对象，窗口对象m\_mainwidget.并在窗口对象中一次加入“上一个”按钮、“水平布局”对象hMainLayout()、竖直布局对象vLayout()，并在竖直布局对象中加入列表布局对话框m\_listwidget.

**QListWidget列表框的创建：**

QListWidget是QT提供的控件类，专门用来创建列表。继承关系如下QListWidget -> QListView -> QAbstractItemView -> QAbstractScrollArea -> QFrame -> QWidget

QListWidget是继承自QListView的，其他的Widget容器也是继承自对应的view容器，QListView继承自抽象基类QAbastractItemView,三者关系如下：



QListView里没有自己的“模型”要自己建模来保存数据，这可以很大程度上降低数据冗余提高效率，要求对view/model框架比较熟。

QListWidget是QListView的子类，在QListWidget中已经帮我们定义好了一个模型，可以直接在QListWidget里面添加数据。

View类别容器和Widget类别容器主要区别在于“模型”。模型是将数据从视图中分离出来的产物，提供和数据交互的接口。Widget类容器内置了一个简单的模型，可以直接调用实例的addItem()、addItems()方法来个widget添加元素，相应变化会直接反映到界面上。而View类别采用了所谓的view/model框架，对数据和视图进行了分离和解耦。Widget部件并没有被设计为可以从视图中分离数据。View和Widget类型看起来相同，但他们与数据的交互方式不同。

所有模型都是QAbstractItemModel的子类。这个类定义了供视图和委托访问数据的接口。模型并不存储数据本身。这意味着可以将数据存储在一个数据结构中、另外的类中、文件中、数据库中等。QAbstractItemModel提供的接口足以应付表格、列表和树的形式显示的数据。但是需要为裂变或者表格设计另外的模型，直接继承直接继承QAbstractListModel和QAbstractTableModel类可能更好一些，因为这两个类已经实现了很多通用函数。

​ Qt 内置了许多标准模型：

* QStringListModel：存储简单的字符串列表。
* QStandardItemModel：可以用于树结构的存储，提供了层次数据。
* QFileSystemModel：本地系统的文件和目录信息。
* QSqlQueryModel、QSqlTableModel和QSqlRelationalTableModel：存取数据库数据。

如果这些标准模型不能满足需要，就必须继承QAbstractItemModel、QAbstractListModel或者QAbstractTableModel，创建自己的模型类。

**QAbstractitemView()**提供接口：void setVerticalScrollMode(ScrollMode mode)；

setVerticalScrollMode(ScrollMode mode)。

ScrollPerItem 值为0；按一个Item滚动；

ScrollPerPixel 值为1 ；按像素滚动，使用次滚动方式时，除非使用setSingleStep()显示设置，否则单步长度将自动调整。通过将单步长度设置为-1可以恢复自动调整。

另有类QScroller支持滚动的小部件或试图启用滚动。

**setHorizontalScrollBarPolicy（QScrollBar \*scrollBar）：**将现有的水平滚动条替换为ScrollBar，并在新的滚动条上设置所有以前滚动调属性。然后删除前一个滚动条。

ScrollBarAsNeeded:自动，当里面的内容显示不下时才出现滚动；

ScrollBarAlwaysOff:关闭，不会显示滚动条；

ScrollBarAlwaysOn:打开一直显示滚动条；

**QlistWidget中有一个setItemWidget函数接口：**常用来设置显示自定义的QWidget,在列表小部件顶的位置显示静态内容。如果现实自定义的动态内容或者实现自定义的编辑器小部件，使用QListView和子类QStyledItemDelegate代替。

QListwidget::item():返回占用列表中给定行的项，否则返回空指针。

QListwidget::coutn():返回属性里诶包中的项数，包括任何隐藏的项。

QListwidget::setItemwidget();输入参数QListWidgetItem \*item、QWidget\*widget,该函数用来设置显示自定义的QWidget,然后将item与我们自定义的QWidget相关联。

**增加对象函数AddItem():**首先添加一个QListwidget()空对象；创建按钮btn并设置按钮名称、样式风格；使用setItemwidget()函数将滑动列表框与该按钮连接起来；最后设置该按钮的对应的连接槽函数。

**设置间隔函数SetSpacing():**使用setMaximumHeight()设置最大高度，设置最小高度、风格。

**窗口大小设置函数OnResize():**首先使用函数GetItemCoverwidth()得到目标块的宽度;将窗口宽度除以目标快的宽度得到目标块的需要显示的个数showNum；取showNum,列表个数m\_Listwidge->count()中的较小值给showNum；将showNum\*单个目标快的宽度得到列表框的总宽度；如果显示的小于总体的则将上一个按钮设置为Disable,  
**滑动效果函数Scroll():**调用scrollToItem(const QTableWidgetItem \*item,QAbstractItemView:ScrollHinthint = EnsureVisible**),**滚动视图确保item()可见。第一个参数设定item,第二个参数设定item显示的位置，总共右手拿种，分别是top、center和bottom。

## 10模块类PModuleMenuFrame继承PFrame

**构造函数PModuleMenuFrame中（）:**使用SetName设置模块名称()QString.toStdString().c\_str.其中toStdString将QString转化为C++标准string,c\_str()生成一个const char\*指针，指向以空字符终止的数组，内容与string相同，这个数组的数据是临时的，当有一个改变这些数据的成员函数被调用后，其中的数据就会失效。因此要现用现转，或者把它的数据复制到用户自己可以管理的内存中。

**初始化函数Init():①**创建“文件”“查看”“设置”“帮助”菜单；创建工具栏，添加操作“打开测试”“密码管理”“帮助”等。②设置窗口m\_tabwidget分配内存、设置大小并将该窗口添加到框架下。

1. 定义创建滑动效果框PSlideViewWidget \*g1:调用模块类PModuleMenuFrame中函数CreateModuleG创建滑动对象slideView、布局对象lay、窗口对象w;将滑块对象加入到布局对象lay中再将布局对象加入到窗口对象w中，设置窗口图片、大小；最后将窗口对象加入到QTableWidget \*m\_tableWidget中，最后返回滑动对象slideView.

**设置页面布局（SetLayout）**。完成主框架中菜单栏、工具栏窗口部件的添加，用到PSlideViewWidget类.

**测试类PTestGroupFrame继承PFrame**

**构造函数PTestGroupFrame:**设置主框架名称、设计空间布局大小数型、初始化当前框架。

Init()中测试按钮列表PBtnListWidget()、测试控制按钮类PTestCtrlWidget()、提供可以同时显示多个文档窗口的区域的类QMdiArea(0.

QMdiArea(Multiple Document Interface Area)提供了一个可以同时显示多个文档窗口的区域

## QT类QPixmap

**QPixmap类是一种off-screen图像表示形式，**可以用作绘画设备。用于绘图设备的图像显示，它皆可以作为一个绘图对象，也可以加载到一个控件中，通常是标签或按钮，用于在标签或按钮上显示图像。可读取的图像文件类型有BMP、GIF、JPG、PNG、PBM、PGM、PPM、XBM、XPM等。

QPixmap与QImage的区别：QPixmap依赖于硬件，QImage不依赖于硬件。

QPixmap主要用于绘图，针对屏幕显示而最佳化设计，QImage主要是为图像I/O、图片访问和像素修改而设计的。当图片小的情况下，直接用QPixmap进行加载，画图时无所谓，当图片大的时候如果直接用QPixmap进行加载，会占很大的内存，一般一张几十K的图片，用QPixmap加载进来会放大很多倍，所以一般图片大的情况下，用QImage进行加载，然后转乘QPixmap用户绘制。QPixmap绘制效果是最好的。

## 模板类PItemContext

包含头文件#include<QListwidgetItem> , QListwidgetItem通过setCheckState可以设置列表项的勾选状态，但其样式并不是通过设置item的样式来实现的，而是通过QListWidget::indicator来实现的。

QListWidget::indicator{xxx;}

QListWidget::indicator:checked{image::url(xxx)} 选中状态

QListWidget::indicator:unchecked{} 未选中那个状态

例：①创建一个列表：

QListWidget::

Listwin = new QListWidget(this);

listWin->move(50,50);move的原点是父窗口的左上角，如果没有父窗口，则桌面即为父窗口，x往右递增，y往下递增

listWin->setFixedSize(100,200);

QStringList fruits = QStringList()<<”apple”

<<”banana”<<”orange”<<”pear”;

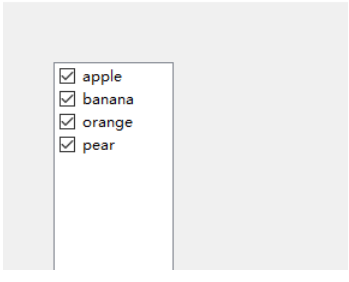
For(auto fruit:fruit){

QListWidgetItem\* listItem = new QListWidgetItem(fruit);

listItem->setCheckState(Qt::Checked);

listWin->addItem(listItem);

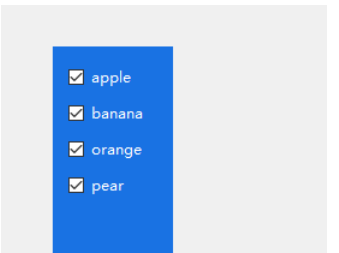
}



②设置列表的样式

listWin->setStyleSheet(QSting(“”QListWidget{border:none;backedground-color:#1972e3;padding:10px;outline:none;}”

“QListWidget::item{height:30px;color:white;}”));



未设置勾选框的样式是，通常不能和设计搭配。

1. 置勾选框的样式；

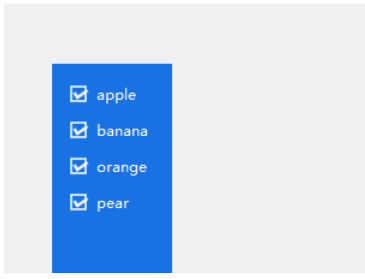
listWin->setStyleSheet(QSting(“”QListWidget{border:none;backedground-color:#1972e3;padding:10px;outline:none;}”

“QListWidget::item{height:30px;color:white;}”

“QListWidget::indicator{width:18px;height:18px;}”

“QListWidget::indicator:checked{image:url(:/icon\_checked\_24.png);}”

“QListWidget::indicator:unchecked{ image:url(:/icon\_checked\_24.png);}”));



**构造函数PItemContext（）：**输入参数1为对象名称字符类型，参数2为按钮类型默认为空。继承类PControl(PCT\_CONTEXT)并初始化对象类型为上下文PCT\_CONTEXT。新建窗口对象m\_frame并用参数1命名；若参数2为空创建按钮并conect到m\_frame用于显示show。最后调用初始化函数Init().

**设置按钮图片函数SetBtnImage():**调用setStyleSheet()设置按钮图片，名字。

C++模板之typename和class关键字的区别：

CArray属于MFC,是一个数组模板类。MFC的数组支持的数组类似于常规数组，可以存放任何数据类型。常规数组在使用前必须将其定义成能够容纳所有可能需要的元素，即先确定大小，而MFC数组类创建的对象可是跟踪需要动态地增大或减小，数组的起始下标是0，而上限可以使固定的，也可以随着元素的增加而增加，数组在内存中的地址仍然是连续分配的。但是声明一个CArray数组必须是确定数组长度，可以之后随元素增加而增加。

使用CArray要添加一个头文件#include<afxtempl.h>

C++模板支持参数化多态工具，使用模板可以使用户为类或者函数声明一种一般模式，是的类中的某些数据成员或者返回值取得任意类型。模板是一种对类型进行参数化的工具；通常有两种形式：函数模板和类模板；函数模板针对参数类型不同的函数；类模板针对数据成员和成员函数类型不同的类；使用模板的目的久石让程序员能够编写与类型无关的代码。注意：模板的声明或者定义智能在全局，命名空间或者类范围内进行，既不能在局部范围，函数内进行，比如不能再main函数中声明或者定义一个模板。

函数模板的格式：

Template<class 形参名，class 形参名，…>返回类型 函数名（参数列表）

{函数体}；其中template和class是关键字，class可以用typename关键字代替，在这里class和typeclass没区别，<>括号中的参数胶木板形参，模板形参和函数形参很想，模板形参不能为空。一旦声明了模板函数就可以用模板函数的形参名声明类中的成员变量和成员函数，及可以在该函数中使用内置类型的地方都可以使用模板形参名。模板形参需要调用该模板函数是提供的模板实参来初始化模板形参，一旦编译器确定了实际的模板实参类型就称他实例化了函数亩的一个实例。

在STL中都是用了模板类的声明，即template。在模板类的声明中，有两种方式：

**Templae<class T>**

**Template<typename T>**

在这里，class和typename是相同的。即在声明一个template type parameter的时候，class和typename表示完全相同的东西。但在C++中有时候必须使用typename。例：关键字typename被用来作为型别之前的表示符号。

template <class T>

class MyClass{

typename T::SubType \* ptr;

...

};

在这里，typename指出SubType是class T中定义的一个类别，因此ptr是一个指向T::SubType型别的指针。如果没有关键字Typename，SubType会被当成一个static成员，于是T::SubType \* ptr 会被解释为型别T内的数据SubType与ptr的乘积。SubType成为一个型别的条件是，任何一个用来取代T的型别，其内部必须有一个内部型别（inner type）SubType的定义。

1、<>括号中的参数叫模板形参，模板形参和函数形参很像，模板形参不能为空。一但声明了模板函数就可以用模板函数的形参名声明类中的成员变量和成员函数，即可以在该函数中使用内置类型的地方都可以使用模板形参名。模板形参需要调用该模板函数时提供的模板实参来初始化模板形参，一旦编译器确定了实际的模板实参类型就称他实例化了函数目标的一个实例。比如swap的模板函数形式为：

Template<class T>void swap(T&a,T&b){}

当调用这样的模板函数时类型T就会被调用是的类型所替代，比如a b为int型，模板函数swap中的形参T就会被int锁替代，a b为daouble时T会被double 替代。

对函数模板而言不存在h(int,int)这样的调用，不能再函数调用的参数中指定模板形参的类型，对函数模板的调用应使用实参推演来进行，及职能进行h(2,3)这样的调用，或者int a,b;h(a,b)

类模板的格式为：

Template<class 形参名，class形参名，…>class 类名 {…};

类模板和函数模板都是以template开始后接形参列表组成，模板形参不能卫康，一旦声明了类模板就可以用类模板的形参声明类中的成员变量和成员函数，即可以在类中使用内置类型的地方都可以使用模板形参来声明。

1. 类模板对象的创建：比如一个模板类A,则使用类模板创建对象的方法为：A<int> m;

在类A后面跟上一个<>尖括号并在里面填上相应的类型、这样的话类A中凡是用到模板形参的地方都会被int所替代。当类模板有两个模板形参时创建对象的方法为A<int,double > m;类型之间用逗号隔开。

对于类模板、模板形参的类型必须在类名后的尖括号中明确指定。比如A<2> m;用这种方法把模板形参设置为int是错误的，类模板形参不存在实参推演的问题。也就是说不能把整形值2推演为int型传递给模板形参。要把类模板形参调整为int必须这样指定A<int>m.

1. 在类模板外部定义成员函数的方法为：

Template<模板形参刘表>函数返回类型 类名<模板形参名>：：函数名（参数列表）{函数体}，例：

Template<class T1,class T2>void A<T1,T2>::h(){}

当在累外面定义类的成员时template后面的模板形参应与要定义的类的模板形参一致。

1. 模板的声明或定义只能在全局，命名孔家或类范围内进行。即不能在局部范围、函数内进行。

模板的形参：类型形参，非类型形参和模板形参。

1、类型形参：类型形参由关键字class或typename后接说明符构成，如template<class T> void h(T a){};其中T就是一个类型形参，类型形参的名字由用户自己确定。模板形参表示的是一个未知的类型。模板类型形参可作为类型说明符用在模板的任何地方，与内置类型说明符或类类型说明符的使用方式完全仙童，即可以用于指定返回类型，变量声明等。

2、非类型形参：

①非类型模板形参，模板的非类型形参也就是内置类型形参，如template<class T,int a>class B{};其中int a就是非类型模板形参，非类型模板形参在模板定义的内部是常量值，也就是说非类型形参在模板的内部市常量。

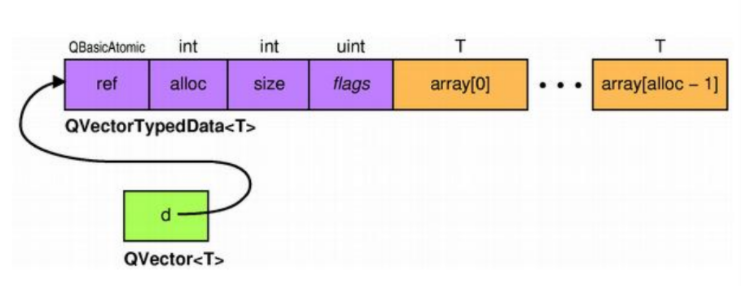
②非类型模板形参只能是整形，指针和引用，下double，string，string\*\*这样的类型是不允许的。但是double&,double\*，对象的应用或指针是正确的。

调用非类型模板形参的实参必须是一个常量表达式，即它必须能在编译时计算出结果。注意：任何局部对象、局部变量、局部对象的地址，局部变量的地址都不是一个常量表达式，都不能用作非类型模板形参的实参。全局指针类型、全局变量，全局对象也不是一个常量表达式，不能用作非类型模板形参的实参。

全局变量的地址或应用，全局对象的地址或应用const类型变量是常量表达式，可以用作非类型模板形参的实参。Sizeof表达式的结果是一个长亮表达式，也能用做非类型模板形参的实参。

QT模板库QVector

QVector是一个提供动态数组的模板类。QVector<T>使用连续内存来保存数据项，分配所需内存的方式与动态内存分配的方式相比，减少了在时间和内存删的开销。



QVector<T>和其他QT容器、大部分值类型的类（包括QString）一样，使用隐式共享。但拷贝一个QVector时，内部只是拷贝了一个指针和递增了引用计数，只偶遇修改数据是，才会进行深拷贝，隐式共享也称写时赋值。

隐式共享是QVector和std：：vector<T>的主要区别。QBasicAtomic类用于保存引用计数，支持院子操作<线程安全>，确保隐式共享可以跨线程使用。QT在不同架构下使用会变语言来实现QBasixAtomic类。

为避免每次容器增长时都分配内存，QVector<T>会一次分配超过需要的内存。如果参数T是可拷贝类型（可以通过memecpy()ormemmove()进行拷贝的对象，例如C++原类型和QT隐式共享类），QVector<T>使用realloc()一次再分配4096字节增量。这是因为现代操作系统当重新分配一个内存时并不拷贝整块数据，只是物理内存页面简单的重新排序，第一页和最后一页需要进行拷贝。[QT开发（二十五）——QT模板库\_生命不息，奋斗不止的技术博客\_51CTO博客](https://blog.51cto.com/quantfabric/1873252)

QVector与std::vector对比

在使用Qt开发Windows软件的情况下，开发者可以选择Qt提供的容器以及STL中的容器，当你更在意效率时，笔者建议你使用vector；当你更在意的是安全是，建议使用QVector。Vector和QVector都通过复制构造函数添加对象，但当执行push\_back时，你会发现vector只调用一次复制构造函数，而QVector调用了两次复制构造函数。

#ifdef DEBUG的理解：在工程设置里有一些设置会对该工程自动产生一系列的宏，用于控制程序的编译和运行。如果把代码加载#ifdef DEBUG和对应的#endif中间，那么这段代码只有在调试（DEBUG）下才会被编译，在RELEASE模式下，这些代码就不会存在于最终的代码里头。可以手动实现DEBUG的定义：#define DEBUG那么你的#ifdef DEBUG就会是真的，夹在中间的代码会进行编译，可以说这些红地阿妈本身是面向编译器使用的，不要用来实现你的业务逻辑代码，这样会带俩很不好的影响。这类宏定义的典型应用就是产生/屏蔽调试信息。

## 自定义类PTestGroupFrame

**QT中setAttrubute()用法：**模态对话框指在子对话框弹出时，焦点被强行集中于该子对话框，子对话框不关闭，用户将无法操作其他的窗口，非模态相反，用户仍然可以操作其他窗口，包括带子对话框的父对话框。

从线程角度来讲，模态对话框实际上是线程阻塞的，也就是子对话框是一个线程，但在创建这个子线程之后，父窗口就阻塞了；而非模态对话框则不是阻塞型的线程模型，父子线程可以并行运行。在Qt中创建模态对话框，主要用到Qdialog的exec函数；Qt中创建非模态对话框，主要用到了QDialog的show函数；

SonDialog \*dlg;

dlg = new SonDialog(this);

dlg->show();

由上面代码，细心的读者可能就会问了，既然new了，如果不delete，那么内存不就存在了泄露的问题了吗？确实如此！所以，我们希望该Qt窗口在退出时自动能够delete掉自己，因此，我们在SonDialog的构造函数里，添加这样的一句代码：

setAttribute (Qt::WA\_DeleteOnClose);

这样，我们的SonDialog就能够在它退出时自动的delete掉自己了，不会再造成内存泄漏问题。

2、Qt给父窗体设置样式不影响子控件样式以及子控件设置透明，

窗体样式不生效：this->setAttribute(Qt::WA\_StyledBackground);

父窗样式影响子控件：this->setStyleSheet(“#objname{backeground-color:rgb(190,190,1900}”);

子控件qlinedit设置透明无边框

ui->btnLoad->setStyleSheet("background:transparent;border-width:0;border-style:outset");

**构造函数PTestGroupFrame():**继承PFrame(),初始化参数testID,对象类型PCT\_FRAME.

设置标题名称及扩展方式，最后调用初始化函数Init（）；

**类PTestGroupFrame中初始化函数Init():**将测试按钮类PBtnListWidget、测试控制类PTestCtrlWidget、多文档类QMdiArea、测试内容PSheetWidget类实例化，并且创建对应的菜单栏、工具栏；使用PFramePackage 类将14项测试类进行封装，在通过模板类PItemContext将对应的14项测试装入对应模板；最后使用函数AddControls将以上模板加入窗口。再通过UpdateUI调用函数AddContext将各测试面板和个按钮关联起来。

其中PTestGroupFrame类中更新界面函数UpdateUI中通过基类Pcontrol中函数GetType获取当前部件的类型m\_UIType，而m\_UIType的值由继承Pcontrol的子类构造函数初始化时赋值，各部件类型使用枚举enum PCONTYPE定义。

**构造函数中先调用SetLayout()用于整体窗口布局及多文档那个窗口布局。**①定义竖直方向布局对象QVBoxLayout \*vLay;实例化按钮列表类m\_itemFrame = new PBtnLIstWidget、测试条件类m\_testCtrlFrame = new PTestCtrlWidget,并将其加入到竖直布局类对象VLay中，再将vLay加入到朱布局对象m\_mainLayout中。②实例化多文档对象m\_mdiArea = new QMdiArea.

将其加入到主布局对象m\_mainLayout中；实例化状态栏列表表格对象m\_sheetFrame = new PsheetWidget(),将其加入到主布局对象m\_mainLayout中.③连接connect多问低昂激活功能到子对象显示槽，并设置标题。

**将子测试框加入模板。**定义容器QVector<PItemContext<PFramePackage>\*> contests,实例化14项测试

**类PTestGroupFrame中增加上下文函数AddContext():**首先通过参数PContext \*context的函数GetCreateBtn获取对应按钮、其中PContext = PItemContext<PFrame>为上下文类PItemContext的模板；再通过connect将该按钮与对应的子测试框连接起来；之后是对该自测试框的属性进行设置。

**类PTestGroupFrame中函数FindContext（）：**找到对应title的按钮。

太盛则折，太弱则泻’。老头儿说我的毛病是 太盛。又说，若对手盛，则以柔化之。可要在化的同时，造成克势。柔不是弱，是 容，是收，是含。含而化之，让对手入你的势。这势要你造，需无为而无不为。无 为即是道，也就是棋运之大不可变，你想变，就不是象棋，输不用说了，连棋边儿 都沾不上。棋运不可悖，但每局的势要自己造。棋运和势既有，那可就无所不为了 。玄是真玄，可细琢磨，是那么个理儿。我说，这么讲是真提气，可这下棋，千变 万化，怎么才能准赢呢？老头儿说这就是造势的学问了。造势妙在契机。谁也不走

A018 QT工具类QACTion、边框QFrame、菜单QMenu梳理

信号槽Connect

一般使用下面的语句来连接signal和slot：

Connect(sender,SIGNAL(valueChange(QString,QString)),receiver,SLOT(updateValue(QString)));

Qt实际上利用SIGNAL和SLOT这两个宏，把气候的函数名转换成一个字符串。随后，moc将会扫描全部文件，将所有的signal和slot提取出来总成一个映射表。QObiect：：connect（）函数则会从这个映射表里找到该字符串，从signal的名字里看就可以找到slot的名字，因此也就知道了在signal emit的时候，该去调用哪一个slot函数。

Qt5之前的语法问题：①没有编译期检查：因为函数名被处理成字符串，所有的检查都是在运行时完成的。导致有时会发生编译通过了，但slot并没有被调用。此时，需要检查consoled的输出，看看有没有什么warning说明 connect并没有成功。

②因为处理的是字符串，所以slot中的类型名字必须用signal的完全一致，而且在头文件中的和实际connect语句中的也必须一致。也就是说，如果用了typedef或者namespace，conenect就可能不成功。

**新语法：使用函数指针。**之前的语法依然可以用，但是现在可用函数指针：

Connect（sender，&Sender：：valueChanged,receiver,&Receiver::updateValue）;其好处有编译器检查，如果把signal或者slot名字编写些错误，或者slot的参数通signal不一致，会在编译期就获得一个错误。

C++11 lambda表达式，eg:

Lambda表达式（通常称为Lamda）是一种在被调用的位置或作为参数传递给函数的位置定义匿名函数对象的简便方法。Lambd通常用于封装传递给算法或者异步函数的少量代码行。

void MyWindow::saveDocumentAs()

{

QFileDialog \*dlg = new QFileDialog();

dlg->open();

QObject::connect(dlg, &QDialog::finished, [=](int result) {

if (result) {

QFile file(dlg->selectedFiles().first());

// ... save document here ...

}

dlg->deleteLater();

});

}

Lambda函数也就是一个函数他的语法定义如下：

[capture](parameters)mutable->return-type{statement}

1、[capture]:捕捉列表。捕捉列表总是出现在Lambda函数的开始处。实际上，[]是Lambda引出符，**他指定捕获哪些变量，以及捕获是通过值还是通过引用进行的；有&前缀的变量通过引用进行访问，没有该前缀的变量通过值进行访问**。编译器根据该引出符判断接下来的代码是否是Lambda函数。并做列表能够捕捉上下文中的变量以供Lambda函数适用；

空capture子句 []指示lambda表达式的主体不访问封闭范围中的变量。可使用默认捕获模式来指示如何捕获Lambda体中引用的任何外部变量：[&]表示通过引用捕获引用的所有变量，而[=]表示通过值捕获他们。

2、（parameters）:参数列表。与普通函数的参数列表一致。如果不需要参数传递，则可以联通括号“（）”一起省略；

3、mutable:mutable修饰符。默认情况下，Lamba函数总是一个const函数，mutable可以取消其相连型。在使用该修饰符时，参列表不可省略（即使参数为空）；

4、->return-type:返回类型。用追踪返回类型形式声明函数的返回类型。不需要返回值的时候可以联通符号“->”一起省略。此外在返回类型明确的情况下，也可以省略该部分，让编译器对返回类型进行推导；

5、{statement}：函数体。内容与普通函数一样，不过除了可以使用参数之外，还可以使用所有捕获的变量。

与普通函数最大的区别是，除了可以使用参数以外，Lambda函数还可以通过捕获列表访问一些上下文中的数据。具体地，捕捉列表描述了上下文中哪些数据可以被Lambda使用，以及使用方式（以值传递的方式或引用传递的方式）。语法上，在“[]”包括起来的是捕捉列表，捕捉列表由多个捕捉项组成，并以逗号分隔。捕捉列表有以下几种形式：

1. [var]表示值传递方式捕捉变量var;

② [=]表示值传递方式捕捉所有父作用域的变量（包括this）;

1. [&var]表示应用传递捕捉变量var；
2. [&]表示引用传递方式布偶卓所有父作用域的变量（包括this）;
3. [this]表示值传递方式捕捉当前的this指针。

其中提到的幅作用域，就是包含Lambda函数的语句块，说通俗点就是报刊lambdade “{}代码块。上面的捕捉列表还可以进行组合,例如：

A019 QT之容器空间QMdiArea

QMdiArea(Multiple Document Interface Area)提供了一个可以同时显示多个文档窗口的区域。本质上，QMdiArea是多文档窗口的窗口管理器。它将它所管理的窗口绘制在它上面，并且以级联或者平铺的方式放置这些窗口。通常情况下，QMdiArea用作QMainWindow的中心空间，一次来创建一个MDI应用程序。但它也可以被放置在任何布局类中。QMdiArea支持所有的串口标志，只要这些窗口标志被当前控件样式支持。

功能详述：QMdiArea中的子窗口是QMdiSubWindow的实例化对象。通过调用addSubWindow来将他们加入MDI区域。通常情况下是传递一个QWidget作为内部控件给这个函数。但是直接传递QMdiSubWindow也是可以的。当一个子窗口获得键盘焦点或者setFocus被调用，这个子窗口便被激活。当激活窗口改变时，subWindowActivated信号会被触发。ActiveSubWindow函数会返回当前激活的窗口。

SubWindowList是一个十分方便的函数，可以用来返回所有子窗口列表。这个信息可以被用在包含窗口列表的弹出菜单上面。

WindowOder用来指定子窗口的排列规则。子串口的排列规则直接影响到subWindowList(),activateNextSubWindow()以及activatePreviousSubWindow()。而且，当窗口使用级联或者平铺时，这个信息也很有用。

QMdiArea为子窗口提供了两个内建的布局策略：cascadeSubWindows()和tileSubWindows()。这两个策略都是槽函数而且很容易与菜单条目连接。

**常用设置：**

**①**添加子窗口使用addSubWindow,移除窗口使用removeSubWindow;

mdiArea->addSubWindow(groupBox\_0);

mdiArea->removeSubWindow(groupBox\_0);

② 设置背景色默认为灰色，可以使用setBackground来设置

mdiArea->setBackground(QBrush(Qt::green))；③设置最大化模式：当活动子窗口最大化时，默认行为是下一个最大化下一个激活的自串口。如果不希望出现这种情况，可以通过下面的方法来设置。

mdiArea->setOption(QMdiArea::DontMaximizeSubWindowOnActivation);

④设置视图模式：有两种视图模式，可以通过setViewMode来设置。

QMdiArea::SubWindowView 0 显示带有窗口框架的子窗口（默认）。

QMdiArea::TabbedView 1 在选项卡栏中显示带有选项卡的子窗口。

1. 设置激活顺序：指定用于对由subWindowList()返回的自串口列表进行排序的条件。排列窗口时，函数cascadeSubWindows（）和tileSubWindows()遵循此顺序。

QMdiArea::CreationOrder 0 窗口按其创建顺序返回。

QMdiArea::StackingOrder 1 窗口按其堆叠顺序返回，最上面的窗口在列表的最后。

QMdiArea::ActivationHistoryOrder 2 窗口将按照它们被激活的顺序返回。

A020 自定义类PApp

类PApp继承基类PControl.

**类PApp构造函数PApp():**继承PControl初始化其类型为PCT\_NONE.

**类PApp中初始化函数Init():使用QCoreApplication(),Q**t是一个开源的C++库，主要用来开发GUI程序，但同时，它也支持控制台程序的开发。并且，这里的控制台程序又分为Qt控制台程序和纯C++控制台程序。Qt控制台程序是和Qt GUI相对的一种程序，他除了可以进行一般打印输出外，也可以像GUI程序一样，支持时间循环、信号和槽的特性。并且，Qt库针对不同的应用程序类型，提供了不同的类来表示，比如，QCoreApplication表示Qt控制台程序，QApplication和QGuiApplication表示GUI程序。他们之间的关系为QCoreApplication继承自最顶层的QObject，QGuiApplication 又继承自QCoreApplication，QApplication又继承自QGuiApplication。今天，我们主要来学习一下QCoreApplication类。QCoreApplication（）类表示控制台程序，它承载了应用程序时间循环，通过这个时间循环对所有来自操作系统和其他时间源，如网络、数据库的事件进行处理和分发。同时，她还实现了应用程序的初始化和退出清理工作，以及系统级和应用级的常用设置。所以，Qt在生成main函数时就自动为我们定义了应用程序对象，其在程序运行过程中，就代表了整个应用程序本身。但由于该对象是main函数的局部对象，不方便我们在其他地方使用，特别是对于GUI程序来说，我们可能需要在某个窗口类中使用到这个应用程序对象，所以，鉴于此，Qt又在全局空间为我们提供了已应用程序指针，即qApp，该全局指针指向的就是main函数中定义的应用程序对象，我们可以在任何地方使用这个指针。当然，我们还可以使用QCoreApplication类中的一个静态成员函数instance()，来获得代表该应用程序的指针，该函数声明如下：

QCoreApplication \*QCoreApplication::instance()

[(16条消息) QCoreApplication\_求道玉的博客-CSDN博客\_qcoreapplication](https://blog.csdn.net/Amnes1a/article/details/64121886)

Init()函数中定义了公共菜单栏及其动作，并且创建主窗口PMainFrame.

**类PApp中创建函数CreateApp():**创建程序App、命名并进行初始化。

**类PApp中获取自定义控件函数GetQComCon():**根据输入参数返回对应自定义部件。

**类PApp中函数Home():**调用框架类PFrame中函数Attach(),该函数将PApp中主界面替换为PFrame类中的主界面。

A022 Qt之QMap基本用法

QMap<Key,T>是Qt的通用容器类之一。它存储<键，值>对，定提供与键相关联的值得快速查找。QMap和QHash提供了非常相似的功能，其差异是：

QHash提供了比QMap更快的查找速度。在QHash上迭代时，项目的顺序是任意的。而QMap总是按键排序。QHash的键类型必须提供operator==()和QHash（key）函数。QMap的键类型必须提供操作符<(),已制定全序殊勋。

* 1. 需要高寒头文件：#include<QMap>

QMap中插入键、值对：

QMap<QString,int>map;

map[“LiMing”] = 14;

map.insert("BaiMing", 12);

map.insert("LiMuu", 20);

map.insert("LiMuu",100);

可用两种方式插入。往同样的键中重复写值时，新的值会将就的值代替，此时LiMuu的值为100.可以用insertMulti()函数重复写同样的键。

map.insert("LiMuu", 20);

map.insertMulti("LiMuu",100);

* 1. 检索键对应的所有值

Qlist<int> values = map.values(“LiMuu”);

for(int i = 0;i< values.size();i++)\_

qDebug()<<values.at(i);

使用函数values()获取一个键对应的所有值，这个函数返回Qlist类模板。

* 1. 遍历数据

/\* 遍历数据 (先随便插入几个)\*/

map.insert("Math", 100);

map.insert("Chinese", 98);

map.insert("physical", 97);

map.insert("chemical", 96);

map.insert("biological", 95);

//便利数据要是用迭代器、QT提供了两种方式的迭代

//第一种是java类型的迭代

QMapIterator<QString,int> iterator(map);

While(iterator.hasNext()){

Iterator.next();

Qdebug()<<iterator.key()<<”:”<<iterator.value();

}

//第二种是STL类型迭代

QMap<QString ,int>::const\_iterator iterator\_1 = map.constBegin();

While(iterator\_1 != map.constEnd()){

Qdebug()<<iterator.key()<<”:”<<iterator.value();

++iterator\_1;

}

打印输出：两种方法输出一样

“Chinese” : 98

“English” : 99

“Math” : 100

“biological” : 95

“chemical” : 96

“physical” : 97

* 1. 查找是否包含某个键

bool isok = map.contains(“math”);

qDebug()<<isok; 打印输出true

* 1. 获取所有的键和键值

QList<QString> allKeys = map.keys();

qDebug()<<allKeys;

QList<int> allValues = map.valuse()

qDebug()<< allValues; 打印输出：(“Chinese”, “English”, “Math”, “biological”, “chemical”, “physical”) (98, 99, 120, 95, 96, 97)

[(16条消息) [QT] QMap使用详解\_yunke120的博客-CSDN博客\_qmap插入数据](https://blog.csdn.net/qq_39295354/article/details/115895547)

A023 Qt之全局变量extern

1. 原理

声明。以int为例，只能extern int myData;一个头文件中只能写一次extern int myData;假如在1.h中写过，别的头文件就不要再重复写。之后用到用到myData的文件包含1.h即可。

1. Qt designer、Qt Quick Designer 和QtCreator应用程序有什么区别。

Qt designer是一个图形工具，可以构建QWidget GUI, Qt Quick Designer与之类似，只适用于构建QML GUI ,两者都内置在Qt Creator中。Qt designer适用于使用Qt小部件文件（.ui）设计和构建图形用户界面（GUI）的Qt工具，可以按所见即所得的方式编写和自定义串口或对话框，并使用不同的样式和分辨率对齐进行测试。

Qt Quick Designer集成在Qt Creator中的。通常说Qt Quick Designer允许编辑QML文件（.qml）,但是现在它已集成在Qt Creator中。

QtCreator是Qt的IDE,功能最强大，它大大简化了Qt开发，可以原生穿件QT应用程序（QT带有C++引擎）。并且允许我们创建和编辑源代码，调试应用程序等。除此之外，可以在QtCreator中打开.ui文件，他将显示嵌入在完全集成的Qt Creator IDE的Qt Designer应用程序。当然，如果文件是.ui，则Qt Creator 将显示Qt Designer工具；如果文件是.qml，则将允许编辑QML.

QtCreator只是用来构建QT应用程序的IDE,Qt Widgets和Qt Quick都可以组成。编写Qt Widgets应用程序时，可以在Qt Designer中编辑GUI,对于Qt Quick应用程序，则可以使用Qt Quick Designer，两者都集成到Qt Creator中。

还有一个新工具名称Qt Design Studio,他是一款UI设计和开发工具，它让设计师和开发者可以迅速设计原型，并且开发复杂的可伸缩的UI。简单概括其功能就是让UI设计转换为qml,为工程师所用，并且可以与Photoshop集成。

A024 界面部分工具标定、坐标准值、靶球位置

问题1：ui = new ui CoordTrans();

问题描述：QT生成一个简单的ManinWindow工程后，mainwindows.cpp中实现了Mainwindow的构造函数，代码如下：

MainWindows：：MainWindows(QWidget \*parent):Q MainWindows(parent),ui(new Ui::MainWindow)

冒号后面的两行代码，是C++的列表初始化。其中Q MainWindows(parent)是显式调用基类的构造函数，ui(new Ui::MainWindow)是初始化ui指针。Ui指针在mainwindow.h中定义。当看到ui(new Ui::MainWindow)时，问题来了。MainWindow的构造函数会new一个MainWindow，不是有调用其构造函数了吗，但这岂不是套娃行程死循环，其实是这样的：

QT将两个不同的类都命名成了MainWindow,通过namespace进行了区分。Ui命名空间的变量名，和默认空间的变量名，虽然一样，但实际是两个东西。

问题2：常用部件及含义

Qt：：Widget 是一个窗口或部件，有父窗口就是部件，没有就是窗口

Qt：：Window 是一个窗口，有窗口边框和标题

Qt：：Dialog 是一个对话框窗口

Qt：：Sheet 是一个窗口部件Macintosh表单

Qt：：Drawer 是一个窗口或部件Macintosh周日，去掉串口左上角的图标

Qt：：Popup 是一个弹出式顶层窗口

Qt：：Tool 是一个工具窗口

Qt：：ToolTip 是一个提示窗口，没有标题栏和窗口边框

Qt：：SplashScreen 是一个欢迎窗口，是QSplashScreen构造函数的默认值

Qt：：Desktop 是一个桌面窗口或者部件

Qt：：SubWindow 是以各子窗口

Qt::MSWindowFiredSizeDialogHint: 为Windows系统上的窗口装饰一个窄的对话框边框，通常这个提示用于固定大小的对话框

Qt::MSWindowOwnDC: 为Windows系统上的窗口添加自身的显示上下文菜单

Qt::X11BypassWindowManagerHint: 完全忽视窗口管理器，它的作用是产生一个根本不被管理的无窗口边框的窗口(此时，用户无法使用键盘进行输入，除非手动调用QWidget::activateWindow()函数)

Qt::FramelessWindowHint: 产生一个无窗口边框的窗口，此时用户无法移动该窗口和改变它的大小

Qt::CustomizeWindowHint: 关闭默认的窗口标题提示

Qt::WindowTitleHint： 为窗口装饰一个标题栏

Qt::WindowSystemMenuHint: 为窗口添加一个窗口系统系统菜单，并尽可能地添加一个关闭按钮

Qt::WindowMinimizeButtonHint: 为窗口添加一个“最小化”按钮

Qt::WindowMaximizeButtonHint: 为窗口添加一个“最大化”按钮

Qt::WindowMinMaxButtonHint: 为窗口添加一个“最小化”按钮 和一个“最大化”按钮

Qt::WindowContextHelpButtonHint: 为窗口添加一个“上下文帮助”按钮

Qt::WindowStaysOnTopHint: 置顶窗口

Qt::WindowType\_Mask: 一个用于提示窗口标识的窗口类型部分的掩码

问题3：表格组件类QTableWIdget

在窗体上放置一个QTableWidget组件后，可以在PropertyEditor里对其进行属性设置，双击这个组件可以打开一个编辑器，对齐Colun、Row和Item进行编辑。

一个QTableWidget组件的界面构成如下：



表格的第一行称为行表头，用于设置每一列的标题，第一列称为列表头，可以设置其标题，一般使用缺省的标题，及为行号。行表头、列表头一般是不可编辑的。其余部分是规则的网状，如同一个二维数组，每个网络单元称为单元格，每个单元格都有对应的行号、列号。表格中没哟个单元格时一个QTable WidgetItem对象，可以设置文字内容、字体、前景色、背景色、图标、。每个单元格还可以存储一个QVariant数据，用于设置用户自定义数据。

使用步骤：

* 定义变量指针：QTableWidget \* m\_tableAxis5;

分配内存：m\_tableAxis5 = new QTableWidget;

设置表格名称：m\_tableAxis5->setObjectName(“Axis5”);

* 初始化表格：自定义函数initToolVerTableWidget(QTableWidget\* tw);

1.设置列数：tw->setColumnCount(4);

2.设置行数：tw->setRowCount(1);

tw->setEditTriggers(QAbstractItemView::NoEditTriggers);设置表格不可编辑

//设置占满表格

tw->horizontalHeader()->setStretchLastSection(true);//拉伸最后一行占满空白

tw->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);//平均拉伸每一列

3.隐藏行表头、列表头：

tw->verticalHeader()->hide();

tw->horizontalHeader()->hide();

4.显示栅格：te->setShowGrid(true);

5.使能待增加右键菜单功能的控件，否则在空间上无法弹出右键菜单：

Tw->setContextMenuPolicy(Qt::CustomContextMenu);

a 该函数默认参数是Qt：：DefaultContextMenu,他是利用右键菜单事件contextMenuEvent()来处理，要重写contextMenuEvent(QContextMeuEvent \*event)函数。

B Qt::CustomContextMenu他是触发QWidget::customContextMenuRequested信号，注意仅仅是触发信号，要自己写显示右键菜单的slot。这个信号是QWidget位移与右键菜单有关的信号，

signal：void customContextMenuRequested ( const QPoint & pos )

c Qt::ActionsContextMenu该类型比较简单，直接使用QWidget::actions()函数添加动作即可，点击右键就会自动弹出菜单。Eg:

//添加右键菜

ui.tableView->addAction(ui.action\_add);

ui.tableView->addAction(ui.action\_del);

//右键处理事件

connect(ui.action\_add, SIGNAL(triggered()), this, SLOT(addRecord()));

connect(ui.action\_del, SIGNAL(triggered()), this, SLOT(delRecord()));

6.设置行表头：

7、设置表格内容“居中”

8、设置行表头背景

③将QTableWidget加入到QstackedWidget

使用属性总结：

标签被编辑：QLineEdit::textEdited.

下拉框改变：QComBox::currenTextChagnged.

函数分析：

“轴-5/6”选择框变化后触发函数comBoxAxis\_calTcp\_Changed，该函数调用**GetTableIndexInStackedWidget**(QStackedWidget\* stackedWidget, QString pageName)，该函数输入页面控制类参数QStackedWidget和需要显示的页面名称pagename.第一步该函数通过stackedWidget->count()；得到QWidget的数量，在使用stackedWidget->widget(i)得到索引对应的页面，最后对比该页面的名称是否是是需要的页面名称，若是则返回对应索引。最后在comBoxAxis\_calTcp\_Changed函数中使用stackedWidget->setCurrentIndex();显示当前索引对应页面。

问题4：表格组件类QTableWIdget、QStackedWidget区别

QStackedWidget集成自QFrame，该类提供了多页面切换的布局，一次只能看到一个界面，QStackedWidget可用于创建类似于QTabWidget提供的用户界面。注意：QTableWidget的currentTabText用于修改名称。在“插入页”中插入新的界面。

QStackedWidget属性：

* 接口（方法）

Int addWidget(QWidget \*widget)添加页面，并返回页面对应的索引。

Int count() const 获取页面数量

Int currentIndex（） const 获取当前页面的索引

QWidget \* currentWIdget() const 获取当前页面

Int indexOf(QWidget \*widget) const 获取QWidget页面所对应的索引

Int insertWidget(int index,QWidget \*widget)在索引index位置添加页面

void removeWidget(QWidget \* widget) 移除QWidget页面，并没有被删除，只是从布局中移动，从而被隐藏。

QWidget \* widget(int index) const 获取索引index所对应的页面

* 信号

Void currentChanged(int index)当前页面发生变化时候发射，index为新的索引值

Void widgetRemoved(int index)页面被溢出时候发射，index为页面对应的索引值

* 共有槽函数

Void setCurrentIndex(int index)设置索引index所在的页面为当前页面

VoidsetCurrent Widget(QWidget \*widget)设置QWidget页面为当前页面

一般使用QStackedWidget常用的两种方式：

根据currentWidget（）来判断当前页面，然后通过setCurrentWidget()来设置需要显示的页面。

根据currentindex()来判断当前页面索引，然后通过setCurrentIndex()来设置需要显示的页面。

A025 矩阵变换的左乘与右乘

若绕静坐标系（世界坐标系）旋转，则左乘，即变换矩阵\*坐标矩阵；

若绕动坐标系（自建坐标系）旋转，则又乘，即坐标矩阵\*变换矩阵；

左乘结果是向量旋转之后向量对于原坐标系的位置，右乘是参考坐标系旋转移动之后，向量（并未动）相对于新参考系的坐标。

左乘相当于改变了适量（两个坐标系原点连线构成的矢量），右乘是以某一点为中心进行的变换，这个过程中，矢量没有变。改变矢量与改变坐标系的区别：改变矢量进行的是行变换；改变坐标进行的是列变换，如姿态加位置矩阵的构成，每一行都有n,o,a,每一列都有x,y,z但是只有一个旋转（相机）坐标系中的一个轴坐标，如果绕着世界坐标系旋转，那势必n,o,a作为一个整体进行改变，但是如果绕着相机坐标系旋转，那改变的只是某一轴坐标系。

[(18条消息) 双目立体视觉：一（坐标系变换、左乘右乘、旋转矩阵）\_三眼二郎的博客-CSDN博客\_坐标变换](https://blog.csdn.net/a6333230/article/details/88343282)

A026 如何将欧拉角转换为方向向量

不幸的是，如何定义这些东西有不同的约定（滚动，俯仰，偏航与欧拉角并不完全相同），因此您必须要小心。

如果我们将pitch = 0定义为水平（z = 0），并将偏航定义为相对于x轴的逆时针方向，则方向矢量将为

x = cos(yaw)\*cos(pitch)

y = sin(yaw)\*cos(pitch)

z = sin(pitch)

请注意，我没有使用过roll； 这是方向单位向量，未指定姿态。 编写一个旋转矩阵将旋转物体携带到飞行物体的框架中很容易（如果您想知道，例如，左翼尖指向的位置），但是首先指定约定确实是一个好主意。 您能告诉我们更多有关该问题的信息吗？

编辑：（我一直想回到这个问题两年半。）

对于完整的旋转矩阵，如果我们使用上面的约定，并且我们希望矢量先偏航，然后俯仰，然后滚动，为了获得世界坐标系中的最终坐标，我们必须以相反的顺序应用旋转矩阵。

第一卷：

| 1 0 0 |

| 0 cos(roll) -sin(roll) |

| 0 sin(roll) cos(roll) |

然后音调：

| cos(pitch) 0 -sin(pitch) |

| 0 1 0 |

| sin(pitch) 0 cos(pitch) |

然后偏航：

| cos(yaw) -sin(yaw) 0 |

| sin(yaw) cos(yaw) 0 |

| 0 0 1 |

合并它们，总旋转矩阵为：

| cos(yaw)cos(pitch) -cos(yaw)sin(pitch)sin(roll)-sin(yaw)cos(roll) -cos(yaw)sin(pitch)cos(roll)+sin(yaw)sin(roll)|

| sin(yaw)cos(pitch) -sin(yaw)sin(pitch)sin(roll)+cos(yaw)cos(roll) -sin(yaw)sin(pitch)cos(roll)-cos(yaw)sin(roll)|

| sin(pitch) cos(pitch)sin(roll) cos(pitch)sin(roll)|

因此，对于从x轴开始的单位矢量，最终坐标为：

x = cos(yaw)cos(pitch)

y = sin(yaw)cos(pitch)

z = sin(pitch)

对于从y轴（左翼尖）开始的单位矢量，最终坐标将为：

x = -cos(yaw)sin(pitch)sin(roll)-sin(yaw)cos(roll)

y = -sin(yaw)sin(pitch)sin(roll)+cos(yaw)cos(roll)

z = cos(pitch)sin(roll)

[如何将欧拉角转换为方向向量？ - ITranslater](https://www.itranslater.com/qa/details/2583104079425700864)

A027 Qt中Excel文件导入类Tools

头文件

1. QFileDialog获取文件路径类

选择文件QString QFileDialog::getOpenFileName(QWidget \*pant = 0,const QString &caption = QString(),const QString &dir = QString(),const QString &filter = QString(),QString \*selectedFilter=0,Options options = 0)

参数：

Parent:用于指定父组件，默认参数0。

Caption：是对话框的标题。

Dir:是对话框显示时默认打开的目录，“.”代表程序运行目录，“/”代表当前盘符的根目录。

Filter:是对话框的后缀名过滤器，比如“ImageFiles(\*.jpg \*.png)”,如果使用多个过滤器，使用“；；”分割，比如QObject::tr("EXCEL files (\*.xls);;HTML-Files (\*.txt);;"))。

QStringList QFileDialog::getOpenFileNames():一次选择多个上传。

QString QFileDialog::getSaveFileName();选择文件（保存文件），保存本地文件对话框。

QFileDialog::getSaveFileName();得到保存文件路径、名称。

1. QTextEdit简介

Header：#include<QTextEdit>

Qmake:QT+=widgets

Inherits:QAbstractScrollArea

[(18条消息) Qt类总结（一）QTextEdit\_伴君的博客-CSDN博客\_qtextedit](https://blog.csdn.net/AAAA202012/article/details/120622526)

函数：

Document:QTextDoucument\*:此属性保存文本编辑器的基础文档。

Plaintext:QString:此属性获取并将文本编辑器的内容设置为纯文本。设置属性是，将删除以前的内容并重置撤销、重做历史记录。如果文本编辑有另一种内种类型，则如果调用toPlainText(),则不会将其替换为纯文本。唯一的例外是非中断空间。

toPlainText() const;以纯文本形式返回文本编辑的文本。

Void append(const QString &text):在文本编辑的结尾最佳一个新的段落。注意：附加的新段落与当前段落相同的字符格式和块格式。

1. QTextStream提供了读写文本文件的接口，可以啊哦做QIODevice,QByteArrar,和QString,调用QTextStream的流操作可以方便的读取文字、行、数字等，还提供了文本填充和对齐的相关格式。

[Qt：QTextStream - ShineLe - 博客园 (cnblogs.com)](https://www.cnblogs.com/ShineLeBlog/p/15007369.html)

QTextStream常用语控制台输入输出，它可以自动将标准输入解码为正确的格式。Device、fluch与编码：除了用QTextStream构造器外，也可以通过setDevice()和setString()设置QTextStream的Device和String。可以用seek()跳转到指定位置，如果没有数据可读，那atEnd()会返回true。调用flush（）后，QTextStream就会把所有数据从转移device中，并且调用device的flush（）。QTextStream在内部使用Unicode编码，并且使用QTextCodec来支持不同的字符解码。默认情况下，QTextCodec::codecForLocat()是读写时所用的编码，调用setCodec()设置编码。

1. Qt5 操作Excel需要用到QAxObject类，还要在pro文件中添加QT+=axcontainer.

4.1整页读取

Qvector<QVector<Qstring>> loadExcel(QString strSheetName)

{

If(strSheetName.contains(“.xls”))//兼容老版本

{

strSheetName = strSheetName.left(strSheetName.length()-4);

}

//获取文件路径

QString strPath = QApplication::applicationDirPath() + "/Table.xls";

QFile file(strPath);

if(!file.exists()){

qWarning() << "CExcelTool::loadExcel 路径错误，或文件不存在,路径为"<<strPath;

return;

}

//excel应用成程序

QAxObject \*excel = new QAxObject(“Excel.Application”);

excel->dynamicCall("SetVisible(bool)", false);//true 表示操作文件可见，false不可见

//安路径获取文件

QAxObject \*workbook = workbooks->querySubObject("Open(QString&)", strPath);

//获取所有sheet页

QAxObject \* worksheets = workbook->querySubObject("WorkSheets");

QAxObject \* worksheet = worksheets->querySubObject("Item(QString)", strSheetName);//获取文件sheet页，若参数为（“Item(int)”,1）:获取当前工作表格1，即sheet 1

if(nullptr == worksheet){

qWarning()<<strSheetName<<"Sheet页不存在。";

return;

}

QAxObject \* usedrange = worksheet->querySubObject("UsedRange");//有数据的矩形区域

//获取行数

QAxObject \* rows = usedrange->querySubObject("Rows");

int nRows = rows->property("Count").toInt();

if(nRows <= 1){

qWarning()<<"无数据，跳过该文件";

return;

}

//获取列数

QAxObject \* columns = usedrange->querySubObject("Columns");

int nColumns = columns->property("Count").toInt();

QVariant var = usedrange->dynamicCall("Value");

foreach(QVariant varRow,var.toList()){

QVector<QString> vecDataRow;

foreach(QVariant var,varRow.toList()){

vecDataRow.push\_back(var.toString());

}

vecDatas.push\_back(vecDataRow);

}

//关闭文件

workbook->dynamicCall("Close()");

excel->dynamicCall("Quit()");

if (excel)

{

delete excel;

excel = NULL;

}

return vecDatas;

}

1. Qt QStandardItemModel用法

QStandardItemMode是标准的以项数据（item data）为基础的标准数据模型类，通常

与QtableView组合成Model/View结构，实现通用的二维数据管理功能。主要用到以下3个类：

1. QStandardItemModel：基于项数据的标准数据模型，可以处理二维数据。维护一个二维的项数据数组，每个项是一个QStandardItem类的变量，用于储存项的数据、字体格式、对齐方式等。
2. QTableView:二维数据表视图组件，有多个行和多格列，每个基本现实单元是一个单元格，通过setModal()函数设置一个QStandardItemModel类的数据模型之后，一个单元格显示QStandardItemModel数据模型中的一个项。
3. QItemSelectionModel:一个用于跟踪视图组件的单元格选择状态的类，当在QTableView选择某个单元格，或多个单元格时，通过QStandardItemModel可以获得选中的单元格的模型索引，为单元格的选择操作提供方便。

其关系是：QTableView是界面视图组件，其关联的数据模型是QStandardItemModel,

关联的项选择模型是QStandardItemModel, QStandardItemModel的数据管理的基本单元是QStandardItem.

1. QDesktopServices类提供的函数用于访问巡航简单额桌面服务。

#include <QDesktopServices>

许多桌面环境都会提供一系列服务，可以通过应用程序来执行常见任务，如：以用户应用程序首选项的方式，打开一个网页。

此类包含为服务提供简单接口的函数，返回值表明执行成功或失败。openUrl()函数用于打开位于任意外部应用程序的URL文件。如果URL对应于本地文件系统的资源（URL scheme 是“file”）,将会用一个合适的应用程序带卡文件，否则，将使用一个Web浏览器来获取和显示改文件。

OpenUrl()函数的行为可以定制，对于某些类型的URL,允许应用程序覆盖默认的处理行为。调度机制只允许每个URL scheme使用一个自定义handler,使用setUrlHandler()函数，每个handler为一个槽，只接收一个QUrl参数。

对于每个scheme，现有handle可以用unsetUrlHandler()函数移除，浙江返回给定scheme的默认行为处理方式。

静态共有成员函数：bool openUrl(const QUrl & url) 以用户桌面环境的适当Web浏览器打开指定的url.如果成功，返回TRUE；否则返回false。Qt是跨平台的，windows的文件路径分隔符是\,而linux的分隔符为/,要实现window下编写的程序直接应用到linux下，在构造文件路径时，可以用QDir::toNativeSeparators。

1. 将QTableView中数据保存到txt文本 void Tools::SaveTableViewToFile(QTableView\* table)

解析：①获取保存文件的地址、名称及格式。

QString filePath = QFileDialog::getSaveFileName()。

②取表格的行、列总数。

Int row = table->model()->rowCount();

Int col = talbe->model()->columnCount();

（3）取表格列标题

Table->model(0->headerData(i,Qt::Horizontal,Qt::DisplayRole);i表示第几块区域；Horizontal表示水平方向；有DisplayRple表示列号，没有表示获取整列数据。

（4）获取表格中的所有数据

for (int i = 0; i < row; i++)

{

QString rowStr = "";

for (int j = 0; j < col; j++) {

QModelIndex index = table->model()->index(i, j);

rowStr += table->model()->data(index).toString() + "\t";

}

list.push\_back(rowStr);

}

（5）将list数据存入到QTextEdit textEdit;。

（6）定义文件变量并打开：

QFile file（filePath）；

file.open(QFile::WriteOnly | QIODevice::Text)；

其中，QIODevice:::NotOpen 未打开；QIODevice:::ReadOnly 只读；

IODevice:::WriteOnly只写；QIODevice:::ReadWrite读写；Append已追加的方式打开，新增加的内容将被追加到文件末尾；QIODevice:::Truncate以重写的当时打卡，在写入新的数据时将原有数据全部清除，游标设置在文件开头。QIODevice:::Text在读取时，将行结束符转换成\n；在写入时，将行结束符转换成本地格式；QIODevice:::Unbuffered忽略缓存。

(7) 设置数据输出流格式“utf-8”;

QTextStream ts(&file);

ts.setCodec("utf-8");

(8)将数据输出到数据流，关闭文件。

ts << textEdit.document()->toPlainText();

file.close();

8、将QTableView数据保存到Excel。Void Tools::SaveTableViewToFile(QString& fileName, QTableView\* table, int excCol, int tabRow)

解析：

（1）获取保存文件的地址、名称及格式。

QString filePath = QFileDialog::getSaveFileName()。

（2）定义Eecel对象,设置其操作不可见，不弹出警告窗口。

QAxObject \* excel = new QAxObject(“Excel.Application”);

excel->dynamicCall(“SetVisable(bool)”,false);

excel->setProperty("DisplayAlerts", false);

QAxObject\* workbooks = excel->querySubObject("WorkBooks");//获取Excel工作簿

（3）定义单个工作簿、获取工作区域

QAxObject \* workbook;

Workbooks->dynamicCall(“Add”);

Workbook = excel->querySubObject(“ActiveWorkBook”);

(4)获取工作表、指定页的工作表、工作区域

QAxObject\* worksheets = workbook->querySubObject("WorkSheets");

QAxObject\* worksheet = worksheets->querySubObject("Item(int)", 1);

QAxObject\* usedrange = worksheet->querySubObject("Usedrange");

int iRow = usedrange->property("Row").toInt();

A028 Arts5025-6.0性能测试模块流程

PApp 类进行公共菜单类、工具栏、加密检测、序列化输出等设置。

**PModuleMenuFrame** 创建软件初始界面、此处可添加PApp中创建的各种菜单、工具栏中的对象。初始化函数Init中P175添加性能测试按钮、设置按钮字体、图片、测试ID等。

2022.10.24 本地debug模式调试更改：①papp.cpp中跟踪仪驱动地址P14；②P415页函数Attest()中开放dubug模式下不验证加密狗部分；

2022.11.22

1. **预实验模块**

**1.1总体调用流程**

将机器人的坐标输入到软件中，跟踪仪移动到这些数据代表的空间位姿点，若跟踪仪没能找到这些点，需要手动搜索这些点，再次使用跟踪仪测量，之后将这些跟踪仪测到的点的位姿数据作为预实验的结果。

1. 启动测试按钮连接；ptestctrlwidget.cpp中类PJoinCtrl构造函数中实现连接P37

connect(m\_testBeginBtn, &QPushButton::clicked, g\_app, &PApp:: OnCmdStartTest（）);

停止测试按钮P96

connect(m\_testStopBtn, &QPushButton::clicked, g\_app, &PApp::OnCmdStopTest);

注：预实验测试部分在TestRPT.cpp类数据填充函数FillCalibData()中测试流程完成后通过信号函数m\_pCurItem->signal\_showResultItem();触发预实验中的结果显示表格；通过信号函数

signal\_updateTestStatus(CTRL\_STOP);触发ptestframe.cpp中类PJointModule::showEvent()中链接的状态更新函数OnUpdateTestStatus（）；该函数中获取当前子测试及测试状态->调用控制按钮失能m\_testStopBtn->setEnabled（）该函数触发相应的测试停止函数PApp::OnCmdStopTest()，最后在TestRPT.cpp中启动停止命令CTestRPT::TestUserCmd（USER\_STOP）。

(2)根据界面选择，获取测试ID调用性能测试模块、预实验模块；

主跨框架类void PApp:: OnCmdStartTest（）调用->void PApp::StartRPT()调用->

void PApp::StartRPTCalib()调用->

(3)启动测试基类中控制函数；

BOOL CTestBase::UserCmd()调用->BOOL CTestRPT::TestUserCmd()调用->

（4）启动测试线程；BOOL CTestBase::InitMeasurement(int iRunMode) 调用->

void CTestBase::CreateThread\_Measurement()调用->线程管理类ThreadTracker.cpp中

UINT ThreadMeasurement(LPVOID param) 调用->void CTestRPT::OnStartMeasurement()调用->预实验函数void CTestRPT::StartCalib()调用。在函数void CTestRPT::StartCalib()内获取预实验模块的指令位姿计算跟踪仪测量的目标位置并移动跟踪仪去测量。

（5）在显示预实验模块时获取界面资源：Pretest.cpp类中

P283行显示函数showEvent()中m\_pItem = m\_pTest->GetItemByType(RPT\_PRETEST).

P286 函数InitPoseArr()将待校准的界面位姿数据同步到预实验中。

(6)在显示事件函数中showEvent()将显示信号signal\_showResultItem与预实验的显示函数连接起来。通过InitPoseArr()函数将预实验中的待校准模块的点位数据m\_PoseArr与性能测试中的模块的点位数据连接起来。

例 在位姿模块中PosePage::slot\_refresh()函数在P357行调用m\_pRPTItem->InsertPose(pose);将位姿模块的数据传递到Item的m\_PoseArr中，Item类中P232行使用CRPTItem::InitPoseArr(int iType)//初始化性能测试各模块命令数据点的构造器。

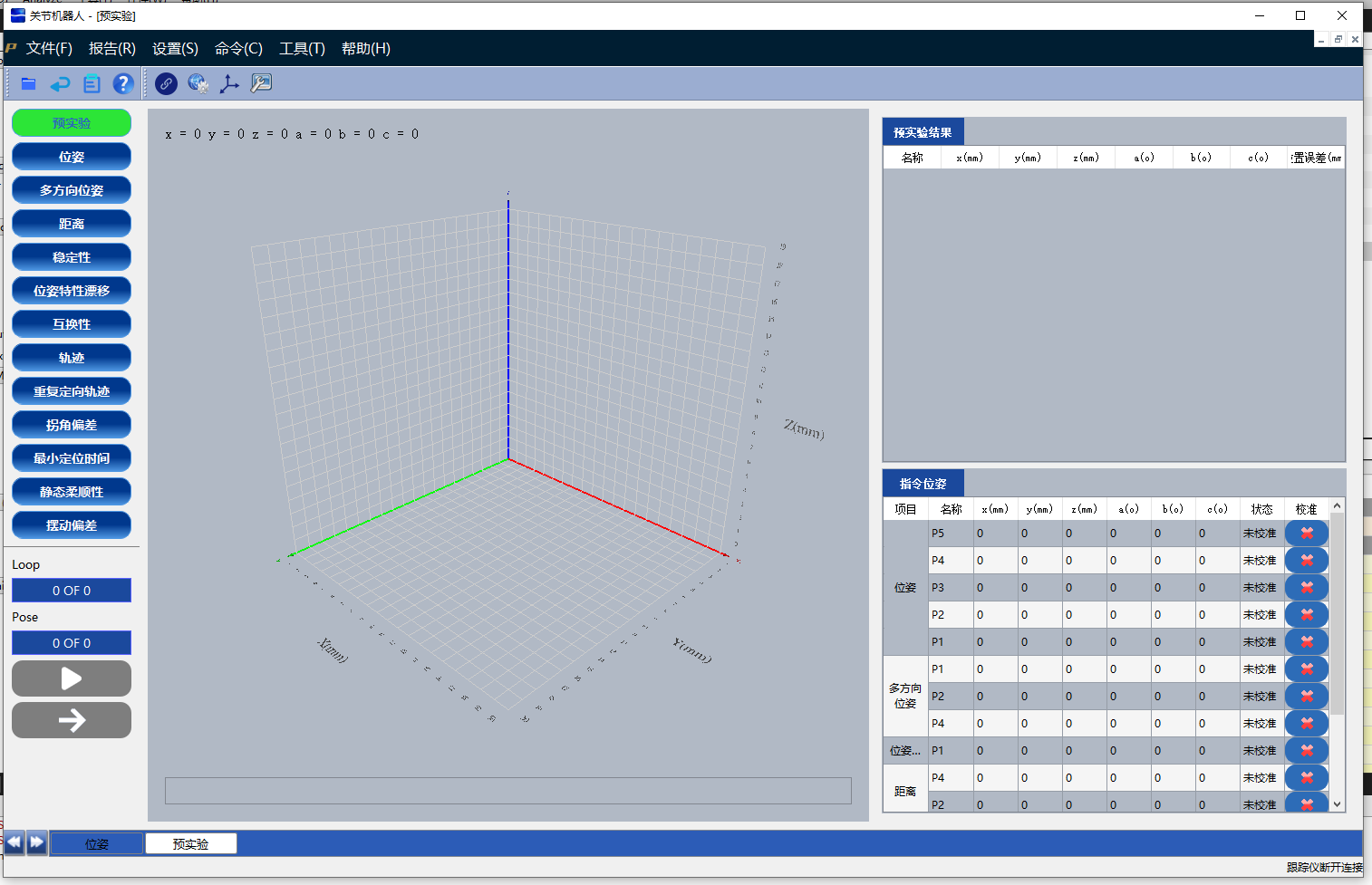
**1.2 预实验类**

（1）界面类：在UI中定义三块内容：3d显示、预实验结果、指令位姿。之后部分在cpp中实现。

H文件中包含RPT界面基类（RPTPageBase）、3D模块类（Viewer3D.h）、数据点姿态类（RPTPose.h）、测试类（TestRPT.h）、参数类（RPTItem.h）。外部已声明类POSition3DWidget;

使用到的数据表格类QStandardItemModel,主要用到以下3个类：①QStandardItemModel基于项数据的标准数据模型，可以处理二维数据。维护一个二维的项数据数组，每个项是一个QStandardItem类的变量，用于存储数据、字体格式、对齐方式等。②QTableView二维数据表试图组件，有多个行和多格列，每个基本现实单元是一个单元格，通过setModel（）函数设置一个QStandardItemModel类的数据模型之后，一个单元格显示QStandardItemModel数据模型中的一个项。③QItemSelectionModel一个用于跟踪视图组件的单元格选择转台的类，挡在QTableView选择某个单元格，或多个单元格时，通过QItemSelectionModel可以获得选中单元格的模型索引，为单元格的选择操作提供方便。

构造函数CPreTest()：a.定义3D视图垂直布局；b.增加结果数据表格；c.添加表格右键菜单；d.增加位姿数据输入表格



1. QTableWidget水平表头horizontalHeader、数值表头verticalHeader的相关操作方法。

（1）QTableWidget表格部件的表头包括水平表头和竖直表头，水平表头没接对应一列，竖直表头对应表格的一行。setHorizontalHeaderLabels用于一次性顺序设置水平表头对个节显示的文本，调用语法如下：

setHorizontalHeaderLabels（Iterable[str] labels）;

注意：该方法无返回值；该方法设置的多个节是按第0节开始的逻辑顺序设置，不论中间是否出现隐藏节或者交换节或者是否设置标签；如果标签列表超出实际表头节数将忽略多出的，前面的按给定顺序设置；如果标签列表少于实际表头节数也能设置前面对应节数的标签；设置了标签的节自动会创建该节对应的项。

1. 设置竖直表头同上setVerticalHeaderLabels( Iterable[str] labels)。
2. 访问水平节对应项，表头的一个节对应一个项，项的类型与表格部件的项类型相同，

都是QTableWidgetItem实例化对象。水平节对应项可以通过horizontalHeaderItem和setHorizontalHeaderItem方法访问，调用语法如下：

QTableWidgetItem horizontalHeaderItem(int column)

setHorizontalHeaderItem(int column, QTableWidgetItem item)

注意：column参数对应水平表头节的序号，从0开始，其数值必须小于表格部件的总列数，否则设置调用无效，查询返回None；

调用setHorizontalHeaderItem将替换原有节对应的项，标签显示为项的文本；

如果一个节没有设置标签也没有通过setHorizontalHeaderItem设置项，则对应表头节没有对应项。

[PyQt学习随笔：QTableWidget水平表头horizontalHeader、竖直表头verticalHeader的相关操作方法 - 老猿Python - 博客园 (cnblogs.com)](https://www.cnblogs.com/LaoYuanPython/p/12570474.html#:~:text=setHorizontalHeaderLabels%E7%94%A8%E4%BA%8E%E4%B8%80%E6%AC%A1%E6%80%A7%E9%A1%BA%E5%BA%8F%E8%AE%BE%E7%BD%AE%E6%B0%B4%E5%B9%B3%E8%A1%A8%E5%A4%B4%E5%A4%9A%E4%B8%AA%E8%8A%82%E6%98%BE%E7%A4%BA%E7%9A%84%E6%96%87%E6%9C%AC%EF%BC%8C%E8%B0%83%E7%94%A8%E8%AF)

1. 合并单元格效果的实现：tableWidget->setSpan(1,1,2,2),代表从（1,1）单元格开始合并，要合并两行两列。
2. 表格QTableView单元中添加控件。使用函数QTableView::setIndexWidget(const QmodelIndex &index,QWidget \*widget).用法如下：

TableView->setIndexWidget(standItemModel->index(10,1),m\_PushButton); //向表格单元添加一个控件

1. 通过QTableWidget中的item()方法可以获得相应行数和列数的QTableWidgetItem,若要设置该Item的编辑状态，可通过QTableWidgetItem下的setflags函数来设置。Setflags函数需要传入参数为每句类型的，例：

//设置为不可编辑

poseModel->item(i, 1)->setFlags(poseModel->item(i, 1)->flags() & (~Qt::ItemIsEditable));

//设置为不可选择

poseModel->item(i, 1)->setFlags(poseModel->item(i, 1)->flags() & (~Qt::ItemIsSelectable));

A029 Arts6025L位姿测试模块流程

**PModuleMenuframe** 初始化函数中通过PItemContext添加性能测试的启动按钮P175。**Ptestframe.cpp**类PJointModule初始化函数中添加公众工具栏、定义控制（启动测试）类、

通过构造器QVector添加11项测试按钮、再实例化具体测试项设置具体测试项的ID。

直角坐标测试、校准、刚度与上述过程相似。显示事件函数showevent中

绑定测试类CTestRPT中的信号函数signal\_updatePoseLabel与黔东南测试类PjointCtrl中的位置、循环测试及测试流程管理函数OnUpdateTestStatus

**位姿界面类PosePage**

**构造函数PosePage** 界面ui->frame3D中添加3D空间类PositionDWidget、设置测试结果显示表格（添加右键导出功能、链接对应的槽函数）、设置指令位姿填写表格（右键列表、链接对应的槽函数）、链接3D数据保存、姿态测试与否、姿态转换完成指令姿态更新表格。

**表格插入项函数 AppendTableViewWithNo** 通过动态强制转换符dynamic\_cast将当前窗口传递到函数内，然后在获取当前表格的被选中情况，若被选中则获取被选中行的索引，最后在在被选中行下添加新行。

**表格删除项函数 RemoveTableViewRowsWindows**  通过动态强制转换符dynamic\_cast将当前窗口传递到函数内，然后在获取当前表格的被选中情况；若有无选中删除最后一行，若有选中则删除选中行，重置序号。

**显示事件函数 showEvent** 通过界面软件框架显示位姿测试界面;绑定当前测试类CTestRPT;获取当前测试Item,并绑定。

**更新测试参数函数 Testing** 重置测试结果显示列表；获取姿态测试与否、速度、循环次数等参数；

**清空测试结果函数 ResetTestResultTabWidget** 清空数据并重置表头；姿态测试检测函数。

**姿态测试状态监测函数 slot\_cb\_gesture\_stateChanged** 获取姿态测试与否；若姿态不测试，则隐藏指令姿态数据及结果表格数据。并将姿态标志位传递到对应Item中。

**刷新指令位姿列表 slot\_refresh** 获取表格model;初始化Item中指令位姿指针；获取指令位子数据；将数据插入到Item中。**指令位姿数据表格中只要有数据更新就必须刷新列表。**

**跟新姿态数据函数 OnUpdatePosture** 获取姿态列表model；从Item中获取每行的指令位子数据存到表格中；检查姿态测试标志位。

**显示测试结果函数 OnShowResultItem** 定义测试结果类指针CResultRPT->GetResultRPT;获取结果表格，去除表格行表头；设置列表头；设置每一行的结果数据；判断结果数据是否超过设定的“技术指标值”。

**调用的其他参数类PostrureResultDialog ：**

**位姿数据转换函数 postureTrans** 将T6、欧拉角、非标准欧拉角、单位四元数等姿态数据转换到算法坐标系统一的欧拉角下。

**构造函数中** 隐藏窗体帮助、最大、最小按钮，Qt::ApplicationModal标志窗体的模态特性作用在整个应用程序的级别。只要本窗体得不到用户响应，则该窗体所有的其他船体、用户都无法操作，即其他窗体都不响应用户输入操作。Qt::WindowModal标志窗体的模态特性仅仅是作用于本窗体所在的集成层次所在的窗体，当本窗体弹出时，如果用户不响应本窗体，则本窗体的父窗体、上层的多个祖先窗体、兄弟窗体的输入、响应都被阻塞，也就是说QT：：WindowModal标志只波及到本窗体的集成层次上。

**初始化函数Init** 定义T6、单位四元数、欧拉角、关节角表格，定义名称并将表格加入到UI中；将T6赋给中间变量设置其列数、行数，隐藏行列表头，设置第一行名称等；

**T6初始化函数 InitT6** 显示事件中初始化调用该函数，首先获取姿态点的个数m\_pItem->GetPoseNum(),姿态点个数加1为该表格行数。通过循环将获取目标姿态数据、调用欧拉角到T6矩阵的转换函数将指令位姿列表中的姿态欧拉角转化为姿态T6矩阵，最后将T6中姿态的O、A列姿态显示到表格中。

**T6保存函数 SaveT6** 该对话框点击“确定”按钮时调用Save函数，根据m\_Type确定调用函数 SaveT6。SaveT6（）中获取表格中的O/A列的姿态数据，调用NeedTrans()判断T6中后6个数据与初始化中的数据是否偏差过大，若偏差不大则进行后续操作。计算其叉向量在调用函数将T6矩阵转化为标准的欧拉角。最后将转化后的数据赋给指向Item位姿的指针中。

**ZYZ初始化函数InitZYZ** 获取姿态点的个数，设置表格行数比姿态点个数多一行。通过Item->GetPose()得到每一个指令姿态数据，如果为ZYX姿态表示则调用ECN\_MapRPYZyz()将其转换为ZYZ()。

**性能测试界面基类 RPTPageBase**

**构造函数RPTPageBase:**链接界面“帮助”按钮到PApp类中的OnOpenRPTHelp;链接保存OnSaveScreenshot()信号用于保存界面数据图片等,该信号在TestRPT类计算结果函数ComputResult()第2094行调用触发。

**更新图表信号函数OnUpdateChartSignal(int index,int key,double value,QString seriesName) :** 输入参数index 图表位置，即当前测试窗口的第几个图表；key 值位置；value 值；seriesName 线条名称。函数内首先判断图表是否满足条件，满足条件调用二维图表封装类ChartView的函数UpdateSerierData。

**数据交换函数Testing() :** 该函数为虚函数，子类调用时需首先更新该子类的姿态数据；之后重置3D显示模块，若当前测试为姿态测试并且部位互换性测试时，调用3D插件的push\_Point将点位数据加入3D显示模块。若为轨迹测试但不是最小定位时间测试时，根据位姿点个数逐个显示数据。

**序列化输出类 CEconArchive**

**技术指标对话框类 TecIndDialog**

**二维图表封装类ChartView**

A030 AES加解密

密钥 填充 模式

1. 密钥：是一种参数，是在明文转换为密文或者将密文转换为明文的算法中输入的参数。分为对称密钥与非对称密钥。

AES支持三种长度的密钥：128位；192位；256位；

1. 填充

AES的分组加密特性：

全量明文

明文块0

明文块1

明文块3

明文块2

密文块0

密文块1

密文块3

密文块2

AES加密器

密钥128

AES算法对明文加密并不是把整个明文一起加密，而是把明文拆分成一个个独立的明文快，每一个明文快长度128bit。明文快经AES加密器复杂处理，生成一个个独立的密文快，这些密文拼接在一起，就是最终的AES加密结果。

明文块0

但这里涉及到一个问题，假如一段明文长度是196bit，如果按每128bit一个明文块来拆分的话，第二个明文块只有64bit，不足128bit。这时候怎么办呢？就需要对明文块进行填充（Padding） 。

几种典型的填充方式：

NoPadding： 不做任何填充，但是要求明文必须是16字节的整数倍。

PKCS5Pdding(默认)：如果明文快少于16个字节（128bit）,在明文块末尾不租相应的数量的字符，且每个字节的值等于缺少的字符数。比如比如明文：{1,2,3,4,5,a,b,c,d,e},缺少6个字节，则补全为{1,2,3,4,5,a,b,c,d,e,6,6,6,6,6,6 }。

ISO10126Padding：如果明文块少于16个字节（128bit），在明文块末尾补足相应数量的字节，最后一个字符值等于缺少的字符数，其他字符填充随机数。比如明文：{1,2,3,4,5,a,b,c,d,e},缺少6个字节，则可能补全为{1,2,3,4,5,a,b,c,d,e,5,c,3,G,$,6}

PKCS7Padding原理与PKCS5Padding相似，区别是PKCS5Padding的blocksize为8字节，而PKCS7Padding的blocksize可以为1到255字节

需要注意的是，如果在AES加密的时候使用了某一种填充方式，解密的时候也必须采用同样的填充方式。

1. 模式

[AES加解密-CBC ECB - 独孤剑—宇枫 - 博客园 (cnblogs.com)](https://www.cnblogs.com/xzj8023tp/p/12970790.html)

A031 6Probe姿态算法讨论

1. 靶球位置模块只是获取三靶球、机器人在初始位置处的**初始姿态，**
2. ①p9615行void CTestBase::InitTargetPos()中调用

9634行m\_CoordTrans.SetInit3DPoint(p3DPoint,InitPose,PointNum);获取初始化时靶球个数、靶球位姿数据、机器人位置数据。

②调用BOOL CRPTItem::MeasTCPData(CECNCoordTans \*pTans,double\* pTCPData)

计算当前测量点相对于初始位姿的姿态

1. 转换库中m\_pInit3Dpoint为三靶球数据

m\_InitPose为机器人数据

1. 算法验证，代码分析：

R2M[4]//空间坐标系下机器人到测量坐标系下的转换矩阵；

spPose[4]//6prpbe的4个姿态；

robotPose[4]//机器人法兰盘的4个姿态；

SP2M[4]//跟踪仪坐标系到算法坐标系的转换；

[欧拉角，四元数，旋转矩阵之间的关系 - 哔哩哔哩 (bilibili.com)](https://www.bilibili.com/read/cv16181713)

A032 码垛校准模块分析

目的：判断是否可以屏蔽杆长

1. 通用六轴校准算法

在校准库中函数CECNRobotCalib::InitPOEParams（）为POE算法校准参数初始化函数，轴校准开关通过m\_bL从外界传递进来，在函数CECNRobotCalib::InitPOEMatrix()中传递给m\_mtLB，在CECNRobotCalib::POERobotCalib()中传递到校准算法中mtNewA2.Multiply(4,&mt,&m\_mtB,&m\_mtL,&m\_mtLB);

A033 QSharedMemory类详解

1. 概述

QSharedMemory提供了多个线程和进程对共享内存段的访问。它还提供了一种方法，让单个线程或进程锁定内存以进行独占访问。

使用这个共享内存类时，请注意以下平台差异：

Windows：QSharedMemory不“拥有”共享内存段。当所有具有连接到特定共享内存段的QSharedMemory实例的线程或进程已销毁其QSharedMemory实例或退出时，Windows内核会自动释放共享内存段。

在对共享内存进行读写之前用lock()锁住共享内存，完成后记得用unlock()释放。

当QSharedMemory的最后一个实例与段分离时，QSharedMemory会自动销毁共享内存段，并且不会保留对该段的应用。

构造函数：QSharedMemory(const QString &key, QObject \*parent = nullptr)，使用给定的父对象构造一个共享内存对象，并将其键设置为key，因为他的key被设置了，他的create()和attach（）函数就可以调用了。

Void QSharedMemory：：setKey(const QString &key)，为这个共享内存对象设置平台无关的键。如果key与当前key相同，则函数不执行任何操作就返回。可以调用key()来检索平台无关的秘钥。在内部，QSharedMemory将此秘钥转换为特定平台的秘钥。若改用nativeKey()，将获得特定于平台的转换后的秘钥。如果共享内存对象附加到底层共享内存段，他将在设置新秘钥之前与其分离。这个函数不做attach（）。

Bool QSharedMemory::attach(QSharedMemory::AccessMode mode = ReadWrite)尝试键进程关联到由传递给构造函数或调用setKey()或setNativeKey()的密钥标识的共享内存段。访问方式默认为ReadeWrite。也可以是ReadyOnly。附加操作成功返回TRUE，若果返回false，则调用error()来判断发生了那个错误。附加共享内存段后可以通过调用data()获得共享内存的指针。

Bool QSharedMemory：：create(int size, QSharedMemory::AccessMode mode = ReadWrite),使用传递给构造函数的键创建大小为size字节的共享内存段，使用setKey（）或者使用setNativeKey()设置key，然后使用给定的访问模式关联到新的共享内存段并返回true。若果key表示的共享内存段已经存在，则不执行attach操作并返回false，并调用error（）来判断发生了哪个错误。

[(39条消息) 【Qt】QSharedMemory类详解\_沧海一笑-dj的博客-CSDN博客\_qsharedmemory](https://blog.csdn.net/dengjin20104042056/article/details/122305584)

A034 QProcessy类详解

QPross用于启动外部程序，并与之交互通信。

1. 启动外部程序的两种方式
   1. 一体式：void QProcess::start(const QString & program,const QStringList &arguments,OpenMode mode = ReadWrite)外部程序启动后，将随主程序的退出而退出。
   2. 分离式：void QProcess::startDetached(const QString & program,const QStringList & arguments,const QString&workingDirectory=QString(),qint64 \*pid =0)外部程序启动后，当主程序退出时并不退出，而是继续运行。
2. 启动外部程序

启动一个外部程序，需要传递外部程序的路径和执行参数，参数用QStringList来带入。

* 1. 设置路径 void QProcess::setProgram(const QString&program)
  2. 设置参数（可选）void QProcess::setArguments(const QStringList & arguments)
  3. 启动 选择启动函数（两种方式）

1. 启动状态

外部程序未启动时，其状态是NotRunning

当启动时，其状态转变为Starting,正在启动，但此时还未调用起来；

启动之后，继续变为Running,同时发射出started（）信号，此时，可以对QProcess进行读写操作了；

当退出时，其状态改为NotRunning,并发射出finished（）信号。finished（）信号会携带退出码和退出状态，可以分别通过exitCode()和exitStatus()来获得。

当发生错误时，QProcess会发出一个error（）信号，同样的，也可以通过error（）来获得其错误类型，通过state（）获得当前程序的状态。

[Qt之Qprocess - 努力飞的小菜鸟 - 博客园 (cnblogs.com)](https://www.cnblogs.com/wang1994/p/5943154.html)

A035 Polyworks与VS联合使用SDK

自带例程：D:\Program Files\InnovMetric\PolyWorks MS 2021\goodies\PolyWorksSDK

1. **Essential COM Principles and Programming Techniques**

There are two common ways to create COM objects:directly or indirectly.Direct creation is done byasking COM to create an instance of a specific COM class.Then COM return s the default interface ,or the interfae soecified in the creation process.Indirect creation occurs when an object’s

Method creates a new object and returns a specific object’s [interface.](mailto:interface.Chenxw01@126.com)

1. **VS调用COM组件，**试讲com组件定义接口的文件包含到本工程，遮掩操作后才可以在后面之间调用com接库函数。调用com组件前不许调用CoInitialize(NULL);调用完成后需要释CoUninitialize();CoCreateInstace用于创建一个com实例。

参考：<https://www.likecs.com/show-305897934.html#sc=200>

<https://www.lmlphp.com/user/16697/article/item/488236/>

**CComPtr和CComQIPtr用法：**

对于操作原始的接口指针是比较麻烦的，需要我们自己控制引用计数、API调用、异常处理。浴室ATL提供了2个智能指针的模板包装类，CComPtr<>和CComQIPtr<>，这两个类都在<atlbase.h>中声明。

CComPtr和CComQIPtr是智能接口指针类，他们在销毁的时候，不需要手动释放接口指针，在赋值的时候，也不需要手动的AddRef,再出现异常的时候，会自动处理异常，而不是要额外的处理遗产代码。

CComQIPtr集成了CComPtr，使用CComPtr的地方都可以使用CComQIPtr，反过来多数也可以。唯一要注意的是：CComQIPtr<>由于使用了运算符的重载功能，他会自动帮我们调用QueryInterface()函数，因此CComQIPtr<>唯一的缺点是不能定义 IUnknown\*指针。

智能指针定义方法：

//sp表示变量类型

CComPtr<IUnknown> spUnk;//正确

CComPtr<IFun> spFun;//正确

CComQIPtr<IFun> spFun;//正确

CComQIPtr<IFun,&IID\_IFun> spFun;//正确

CComQIPtr<IUnknown> spFun;//错误，CComQIPtr不能定义IUnknown指针

**为什么需要CComPtr?** Com皆苦指针很危险，因为使用过程汇总需要每一个使用者都要严格并且正确的AddRe和Release，一旦出现问题，就会造成兑现不能被正常释放，或者对象重复删除造成程序崩溃。所以使用COM接口必须小心。CComPtr被成为智能指针，是ATL提供的一个模板类，能够从语法上自动完成AddRef和Release。CComPtr的用法很简单，以IHell\*为例，将程序中所有接口指针类型（除了采纳数），都是用CComPtr<IHello>代替即可。及程序中除了参数之外，再也不要用IHell0\*。CComPtr的用法和普通COM指针几乎一样，注意下几点。

1. CComPtr已经保证了AddRef和Release的正确调用，所以不用再调用。
2. 如果要释放一个智能指针，直接给他复制NULL值即可。
3. CComPtr本身析构的时候会释放com指针。
4. 当对CComPtr使用&的时候，要确保CComPtr为NULL.因为通过CComPtr的地址对CComPtr赋值时，不会自动调用AddRef，若不为NULL,则前面的指针真不能释放，CComPtr会使用assert报警。

使用方法：

构造函数 参数1为智能接口的类型，参数2为智能指针的接口ID。

CComPt<IUnknown> pink;

//以下例子完全相同

CComPtr<IXXX> pno;

CComPtr<IXXX,&\_uuidof(IXXX)> pno;

CComPtr<IXXX,&IID\_IXXX> pno;

总结：程序中出参数外，不要直接使用com指针类型，一定用CComPtr来代替。

**CComQIPtr不同之处：**

CComPtr只能创建固定的特定接口指针实例。而CComQIPtr不但实现了CComPtr的所有功能（除了CComQIPtr<IUnknown> spFun不合法的，除此之外都可以代替），而且当我们把一个不同类型的借口指针赋给CComQIPtr的时候，他会自动的调用接口指针的QueryInterface接口，来获得对应的正确的接口指针。

使用方法： CComQIPtr可以用任何类型的接口指针初始化，如果初始化的值与CComQIPtr的类型仙童，那么构造函数简单调用指针的AddRef;但如果类型不同的话，他会县调用指针的QueryInterface来获得相同的类型接口指针，当QueryInterface失败的话，内部指针会被设置为NULL。

<https://blog.csdn.net/weixin_40332490/article/details/105077323>

1. **COM组件之** MULTI\_QI Windows的定义类型，此类型的任何公共static成员都是现成安全的。但不保证所有实例成员都是线程安全的。

**类标识符 CLASSID或CLSID** 始于某一个类对象想联系的唯一标记（UUID）。一个准备创建多个对象的类对象应将其CLSID注册到系统数据库的任务表中，以使客户能够定位并装载于该对象有关的可执行代码。

**Wchar\_t类型字符：**是C/C++字符串数据类型，是一种扩展的字符存储方式。在windows下，wchar占2个字节(Byte）;在Linux下占4个字节。Wchat\_t类型主要用在国际化程序的视线中，但它不等同于Unicode编码。Unicode编码的字符一般以wchat\_t类型存储。Char是8卫子夫类型，最多智能包含256中字符，许多外文字符集所含的字符数据差欧共256个，char无法表示。标准C++中的wprinth()函数以及iostream类库中的类和对象能提供wchat\_t宽字符类型的相关操作。

函数介绍：

wcslen() 类似与char\*类型作参数的strlen()函数 用来获取wchar\_t\*变量的长度（不包含终结符）

wcscpy() 类似与char\*类型作参数的strcpy()函数 用来进行wchar\_t\*变量之间的复制

同样的还有 wcsncpy()

wcscmp() 类似与char\*类型的strcmp()函数 用来对比两个wchar\_t\*变量的大小

wmemset() 类似与char\*类型的memset()函数 用来初始化内存

wprintf() 类似与char\*类型的printf()函数 用来输出wchar\_t\*字符串

简单记忆：

一般把char类型函数中的" str "替换为" wcs "即可

A036 C# 基本概念

1. 命名空间

**using System.Collections.Generic**和System.Collections集合的结构有很多相似之处，不同的是“Generic”提供的都是泛型集合，其实在C# 2.0zhong 新加的命名空间，主要用来创建强制类型集合，以提高类型安全和操作性能，常应用在集合中；而“Collections”中集合并不都支持泛型。

**System.Linq** 命名空间内可对多种数据源和对象进行查询、如数据库、数据集、XML文档、数组等。通过对linq的使用，可以减少代码量并优化检索操作。

From指定数据源和范围变量；

where根据布尔表达式从数据中筛选元素；

Select指定查询结果中的元素所具有的类型或表现形式；

Group对查询结果按照键值进行分组；

Into 提供一个标识符，它可以充当对join group或Select子句结果的引用；

Orderby 对产讯出的元素进行排序；

Join 按照两个指定匹配条件来连接两个数据源；

Let 产生一个用于查询表达式中字表达式查结果的范围变量。

**System.Threading.Takss.Task** Task类表示不反悔值并且通常以一部方式执行的单个操作。Task对象实在.NET Frameworks 4中首次引入的基于任务的几部模型的中心组件之一。可使用Status、IsCanceled、IsComPleted和IsFaulted属性来确定任务状态。通常，lambda表达式用于指定任务要执行的工作。

**System.ComponentModel** 命名空间提供用于实现组件和控件的运行是和设计行为的类。次命名空间包括用于特殊和类型转换器的实现、数据原绑定和组件授权的基类和接口。参考<https://www.jb51.net/article/234528.htm>

**System.Data** 创建vs控制台程序，想要连接数据库Sql Server，最简单的using System

.Data; system.data.sqlclient就是加载不出来。要注意控制台应用程序是.Net Core还是.Net Framworks是第一种的话，使用system.data.sqlclient哭会一直报错，原因就是基于。Net Core需要在NuGet包里手动添加system.data.sqlclient库。

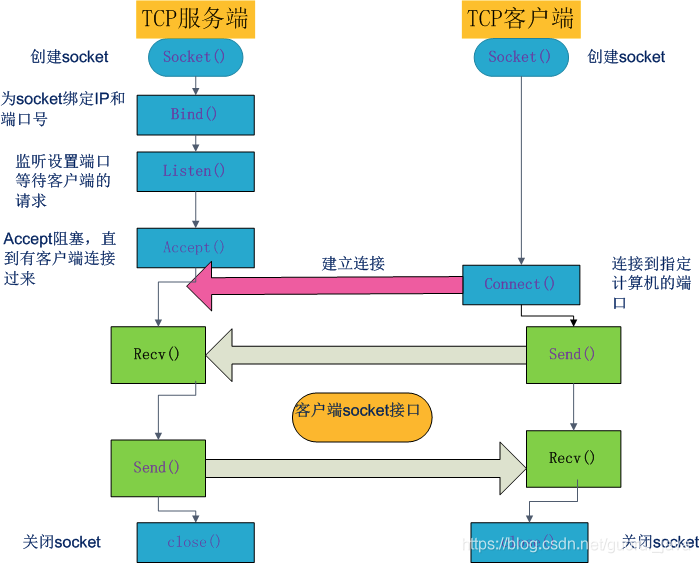
**在C#中用MySql.Data.MySqlClient连接MySql**

在C#中连接MySql数据库其实是件很简单的事情，但对于刚开始学习C#的朋友来说，问题却是不小，主要原因是相对于ACCESS和MSSql 来说，MySql方面的教程文章实在太少，我也是自己摸索好好半天才搞清楚的，呵呵。这篇文章可算不上教程，我主要是担心，怕自己以后会忘记，写在这里， 方便以后查看而已，呵呵。  
　　１、要安装一个　MySQLDriverCS-n-EasyQueryTools-4.0.0-DotNet2.0　这东西可不太好找，只能去MySql的官方网站上下载。  
　　[下载地址 mysql-connector-net-5.0.9.exe](http://mysql.isu.edu.tw/Downloads/Connector-Net/mysql-connector-net-5.0.9.exe)　如果不能下载的话，那就直接到官方网站上去下载。  
　　２、安装完成之后，在C#中添加引用，然后找到MySql.Data，引用到当前项目中，就可以使用了。  
　　３、具体的使用方法：  
　　　　①　using MySql.Data.MySqlClient;

　　　　②　数据库的连接  
MySqlConnection Conn= new MySqlConnection("Database=phpcms2008;Data Source=localhost;User Id=root;Password=");  
③　记录表的读取  
MySqlDataAdapter Da= new MySqlDataAdapter("select \* from phpcms2008\_category where parentid=0", Conn);  
④　　后继操作，这里就不用多说了

**C# 实现socket服务端与客户端通信**

Socket 即“套接字”的意思。



<https://blog.csdn.net/guorui_java/article/details/106142762>

<https://blog.csdn.net/guorui_java/article/details/106142762>

**C# 中delegate的使用方式** 委托的定义不带函数体，切要使用delegate关键字。委托需要致命委托名称一集一个返回类型和一个参数列表，委托定义可以有返回值也可以没有返回值。委托允许讲方法作为参数进行传递；委托可用于定义回调方法；可以把多个方法连接在一起，这样，在事件触发时可同时启动多个时间处理程序；签名不需要与方法精确匹配。

**C# invoke** 为了解决在某个非该空间创建线程中刷新该控件可能会引发异常的问题。

在用.netf framework框架的WinForm构建GUI程序界面时，如果要在控件的时间响应函数中改变控件的状态。原因：控件在主线程中创建，进入空间的事件响应函数时，是在控件所在的响应线程，并不是主线程。在空间的事件响应函数中改变控件状态，可能与主线程发生线程冲突。

正确的写法是在控件响应函数中调用控件的Invoke方法，该方法会顺着控件树向上搜索，知道找到创建空间的那个线程，然后进入该线程改变空间的外观，确保不发生冲突。

Invoke方法需要创建一个委托。可以事先写好函数和与之对应的委托。

A037 QT Pro文件配置项说明

1. QT pro文件配置项说明
2. 注释：#
3. QT+=：这个是添加QT项目需要的模块的，若项目中要排除某个模块，可用QT-=配置项；
4. TEMPLATE=:这个配置项确定qmake为这个应用程序生成那种makefile。有以下五种形式科工选择：

app:建立一个应用程序的makefile,这个是默认值，若模块项未指定，将默认使用此项；

lib:建立一个库的makefile;

vcapp:建立一个应用程序的VisualStudio项目文件；

vclib:建立一个库的VisualStudio项目文件；

subdirs:特殊模板，创建一个可进入特定目录并为一个项目文件生成makefile，此makefile可以调用make.

1. TARGET = :这个配置项用来指定最后生成的目标应用程序的名称。
2. CONFIG +=:用来告诉qmake关于应用程序的配置信息，使用+=表示现有的配置上添加。

qt:告诉qamke此程序是使用qt连编的。及qmake在连接、编译添加所需包含路径时考虑qt的库；

warn\_on:告诉qmake要将编译器设置为输出警告信息形式；

release: 告诉qmake应用程序必须被连编为一个可发布的应用程序。卡法过程中可用debug；

1. UIC\_DIR += :用来指定uic命令，将.ui文件转化为ui\_\*.h文件存放的目录。
2. RCC\_DIR+= :用来指定rcc命令，将.qrc文件转换成qrc\_\*.h文件的存放目录。
3. MOC\_DIR+=:用来指定moc命令，将含有Q\_OBJECT的头文件转换成标准.h文件存放的目录。
4. OBJECTS\_DIR+=:用来指定目标文件obj的存放目录。
5. DEPENDPATH+=:用来指定工程的依赖路径。
6. INCLUDEPATH+=:用来指定工程所需的头文件。
7. CODECFORSRC+=:用来指定源文件的编码格式。
8. FROMS+=:用来指定工程中的ui文件。
9. HEADERS+=:用来指定工程中包含的头文件。
10. SOURCES+=:用来指定工程中包含的源文件。
11. RESOURCES+=:用来指定工程中所包含的资源文件。
12. LIBS+=:用来指定引入的lib文件的路径，一般加上参数-L,根据不同的版本分为两种形式：

Release：LIBS+= -L folder Path

Debug: LIBS+= -L folder Path

1. DEFINES +=:用来定义编译选项。
2. DESTDIR+=:用来指定目标的生成路径。
3. 跨平台处理信息也要写在.pro文件中

A037 QT qInstallMessageHandler（输出详细日志）

安装之前已定义的消息处理程序，返回一个指向抢一个消息处理程序。消息处理程序是一个函数，用于打印调试信息、警告信息、严重错误和致命错误的消息。Qt库 dubug模式包含成百上千的警告信息打印，当发生内部错误时（通常是无效的函数参数）。Qt在release模式下构建还包含这些警告，除非在编译时设置QT\_NO\_WARING\_OUTPUT和或QT\_NO\_WARING\_OUTPUT。如果实现自己的消息处理程序，就可以完全控制这些消息。

缺省的消息处理程序向标准输出打印消息。如果是一个致命的消息，应用程序就会立即终止。仅可以顶一个消息处理程序，因为这通常是在应用范围的基础上进行的，以控制调试输出。调用qInstallMessageHandler(0)可以恢复消息处理程序。

[(60条消息) Qt 之 qInstallMessageHandler（输出详细日志）\_qt日志框架分级别输出\_一去丶二三里的博客-CSDN博客](https://blog.csdn.net/liang19890820/article/details/51838096)

[(60条消息) Qt 之 qInstallMessageHandler（重定向至文件）\_一去丶二三里的博客-CSDN博客](https://blog.csdn.net/liang19890820/article/details/51838379)

A038 C++ initializer\_list详解

Initializer\_list是C++11提供的新类型，定义在头文件中。用于表示某种特定类型的值的数组。和vector一样，initializer\_list也是一种模板类型。

[(60条消息) c++initializer\_list详解\_rotation ㅤ   的博客-CSDN博客](https://blog.csdn.net/fengxinlinux/article/details/72614874)

A039 Qt ApplicationAttribute/WidgetAttribute程序级别属性

使能高清屏显示的缩放。QApplication::setAttribute(Qt::AA\_EnableHighDpiSCaling);

Qt:: ApplicationAttribute指定应用程序范围功能的行为的属性，相关函数有两个setAttribute用来设置属性，testAttribute用来读取是否设置。

Qt::AA\_UseDesktopOpengl = 15 强制使用桌面OpenGL；

Qt::AA\_UseOpenGLES = 16 强制在使用动态加载Opengl实现的平台上使用OpenGL ES 2.0或高版本；

Qt::AA\_UseSoftwareOpenGL = 17强制在使用动态加载的平台上使用基于软件的OpenGL实现；

Qt::AA\_ShareOpenGLContests = 18 允许在属于不同顶级窗口的QOpenGLWidget实例之间共享OpenGL资源。

排名DeskopOpenGL>OpenGLES>Soft

[(60条消息) Qt ApplicationAttribute/WidgetAttribute 程序级别属性\_qt中attribute\_Beyond欣的博客-CSDN博客](https://blog.csdn.net/a15005784320/article/details/114155629#:~:text=enum%20Qt%3A%3AApplicationAttribute%20%E6%8C%87%E5%AE%9A%20%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F%20%E8%8C%83%E5%9B%B4%E5%8A%9F%E8%83%BD%E7%9A%84%E8%A1%8C%E4%B8%BA%E7%9A%84%E5%B1%9E%E6%80%A7%E3%80%82%20%E7%9B%B8%E5%85%B3%E5%87%BD%E6%95%B0%E6%9C%89%E)

A040 C++ std::function详解与实战

简介 std::function是一个函数包装器，该函数包装器模板能包装任何类型的的可调用实体，如普通函数，函数对象、lamda表达式等。包装器可拷贝、移动等。并且包装器类型仅仅依赖于调用特征，而不依赖可调用元素自身的类型。Std::function是C++的新特性，包含在头文件<functional>中。

一个std::function类型对象实力可以包装邪猎这几种可调用实体：函数、函数指针、成员函数、静态函数、lamada表达式和函数对象。std::function对象实例可被拷贝和移动，并且可以使用指定的调用特征来直接调用目标元素。当std::function对象实例未包含任何可调用实体时，调用该对象实例价格抛出std::bad\_function\_call异常。

[C++ std::function详解与实战 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/390883475)

A041 STL unordered\_map

C++ STL标准库中提供有4种无序关联容器。“无序”指的是unordered\_map容器不会像map容器那样对存储的数据进行排序。即map容器中存储的数据是有序的，而unordered\_map容器中是无序的。

Unordered\_map容器和map容器一样，以键值对的形式存储数据，存储的各个键值对的键互不相同且不允许被修改。容器内不会自行对存储的键值对进行排序。Unodered\_map容器在<unordered\_map>头文件中，并位于std命名空间中。因此，如果想使用该容器，代码应包含如下语句：

#include<unordered\_map>

Using namespace std;

其定义如下：

template < class Key, //键值对中键的类型

class T, //键值对中值的类型

class Hash = hash<Key>, //容器内部存储键值对所用的哈希函数

class Pred = equal\_to<Key>, //判断各个键值对键相同的规则

class Alloc = allocator< pair<const Key,T> > // 指定分配器对象的类型

> class unordered\_map;

以上5各参数，必须显式给出2各参数值，并且特殊情况外，最多使用4各参数

创建C++ unordered\_map容器的方法

* 1. 通过调用unordered\_map木板类的默认构造函数，可以创建空的unordered\_map容器，比如：

Std::unordered\_map<std::string,std::string>umap;

* 1. 创建unordered\_map容器的同事，可以完成初始化操作。

Std::unordered\_map<std::string,std::string> umap{

{“python教程”，“hattp://c.”},{“Java教程”，“hattp://c.”},{“Linux教程”，“hattp://c.”}};

穿件umap容器，包涵3个键值对。

* 1. 可以调用unordered\_map模板中提供的赋值（拷贝）构造函数，将现有unordered\_map容器存储的键值对，赋值给新建unordered\_map容器。

如，在第二中法方式创建好umap容器的基础场，再创建并初始化一个umap2容器：

Std::unordered\_map<std::string,std::string> umap2(umap);umap2容器就包含umap容器中所有的键值对。

除此之外，c++标准中还想unordered\_map模板增加了移动构造函数，即以右值引用的方式将临时unordered\_map容器中存储的所有键值对，全部复制给新建容器。例如：

//返回临时 unordered\_map 容器的函数

std::unordered\_map <std::string, std::string > retUmap(){

std::unordered\_map<std::string, std::string>tempUmap{

{"Python教程","http://c.biancheng.net/python/"},

{"Java教程","http://c.biancheng.net/java/"},

{"Linux教程","http://c.biancheng.net/linux/"} };

return tempUmap;

}

//调用移动构造函数，创建 umap2 容器

std::unordered\_map<std::string, std::string> umap2(retUmap());

* 1. 可使用unordered\_map类模板提供的迭代器，在现有unordered\_map容器中选择部分区域内的键值对，为新建unordered\_map容器初始化。例如：

//传入2个迭代器

Std::unordered\_map<std::string,std::string> umap2(++umap.begin().umap.end();此方式出案件的umap2容器，其内部包含umap让其中处第1个键值对外的所有其他键值对。

[C++ STL unordered\_map容器用法详解 (biancheng.net)](http://c.biancheng.net/view/7231.html)

[C++ STL unordered\_map获取元素的4种方法（超级详细） (biancheng.net)](http://c.biancheng.net/view/7237.html)

A042 std::string\_view 使用介绍

C++中字符串有良好总风格，分别是c风格字符串、std::string字符串。C风格字符串性能更高，但不方便操作。如下：

#include<iostream>

#include<string>

Int main()

{

//c风格字符串总是以null结尾

Char cstr1[1] = {} char cstr2[ 'y', 'a', 'n', 'g', NULL];

  strcpy(cstr1, cstr2);

  std::cout << cstr2 << std::endl;

  //C++风格的字符串操作更方便，但是性能不如C风格字符串

  std::string str = "yang";

  std::string str2 = str;

}

C++17中可以使用std::string\_view来获取一个字符串的视图，字符串视图并不真正的创建或者拷贝字符串，只是有个有一个字符串的查看功能。Std::string\_view比std::string的性能要高很多，因为每个std::string都独自拥有一份字符串的拷贝、而std::string\_view只是记录了自己对应的字符串的指针和便宜为孩子。当我们在只是查看字符串的函数中可以直接使用std::string\_view来替代std::string.

[C++ 17 std::string\_view使用介绍 - xnuwu - 博客园 (cnblogs.com)](https://www.cnblogs.com/yangxunwu1992/p/14018837.html)

要使用C++求解两个坐标系之间的转换矩阵，首先要明确一组点在两个坐标系下的位置数据。假设在A坐标系下的点是P\_A，B坐标系下的点是P\_B，我们需要找到一个3x3的矩阵R和一个3x1的向量T，使得以下关系式成立：

P\_B = R \* P\_A + T

可以使用以下步骤来求解R和T：

将数据集分别减去两个坐标系中心（重心）的坐标，即 P\_A\_centered 和 P\_B\_centered。

计算协方差矩阵 Covariance = P\_A\_centered^T \* P\_B\_centered。

对协方差矩阵进行奇异值分解（SVD）：Covariance = U \* S \* V^T。

计算旋转矩阵 R = V \* U^T。如果 det(R) < 0, 则令 U[:,2] = -U(:,2) 并重新计算 R。

计算平移矩阵 T = P\_B\_mean - R \* P\_A\_mean。

A043 Python库 concurrent.futures—启动并行任务

1. concurrent.futures模块提供一部执行可调用调用对象高层接口。

异步执行可以有ThreadPoolExecutor使用线程或有ProcessPoolExecutor使用单独的进程来实现。两者都是实现抽象类Execute定义的接口。

可用性：非Emscripten,非WASI

此模块在WebAssemby平台wasm32-emscripten和wasm32-wasi上不适用或不可用。

Executor对象

Class concurrent.futures.Executor

抽象类提供异步执行调用方法。要通过它的子类掉用而不是直接调用。

Submit(fn,/,\*args,\*\*kwargs)

调度可调度对象fn,以fn(\*args,\*\*kwargs)方式执行并返回一个代表该可调用对象的执行的Future对象

**with** ThreadPoolExecutor(max\_workers=1) **as** executor:

future = executor.submit(pow, 323, 1235)

print(future.result())

map(fn,\*iterables,timeout=None,hunksize=1)

类似于map(fn,\*iterables)担忧一下差异：

Iterables是立即执行而不是延迟执行的；

Fn是异步执行的并且可以并发对fn的多个调用。

如果\_next\_()被调用且从对Exector.map()原始调用timeout秒之后其结果还补课用则已返回的迭代器将引发TimeoutError。Timeout可以是郑虎或浮点数。如果timeout未指定或为None,则不限制等待时间。

[concurrent.futures --- 启动并行任务 — Python 3.12.4 文档](https://docs.python.org/zh-cn/3.12/library/concurrent.futures.html)

《三生万物》：第一天到大办公室 你该做什么？

2005年1月，我赴中粮集团任董事长。记得在香港到北京的飞机上，我拿着一本世界粮食试产的研究报告，头也没抬地看了一路。可惜郭队粮食这个行业一头雾水，虽然我急迫地想了解，但短期内不可能形成清晰的概念。

到了北京，于中良的高层团队开第一次会，我就给大家出了一个题目，也算布置了份作业，叫“假如你是董事长”。既然我没有清晰的概念，就让大家戴上我的帽子，站在我这个刚上任董事长的角度，对中粮发展提看法。后来有人说你这个办法好，一下子就把大家调动起来了，我说当时我真的没想那么多，饿哦当时就真的不知道从哪里下手。我相信高层团队的每个人或多或少都会想过这个题目“如果我说了算，我觉得应该怎么怎么样”的心气。

接到这个题目后，团队里的任反应不一。大部分人认真地写了看法，虽然有深有浅，有宽有窄。有人在两周时间里就找了咨询公司写了几百页的打报告给我。也有人表现出为难，写了两页纸的简单护肤，主要是表态会努力工作、听从两道，让干啥就干啥。应为我出的题目是宽泛的，大家也不可能有很系统的回答，何况是刚见面。但这个做法还是有很好的效果，大家觉得我愿意听别人建议。我也一下没经过更多的层级，就直接进入了发现问题并给出解决方法的阶段，实在太好了！同时，我也洁身了对高层团队每个人的了解**。也可以说，通过这次回答，每位团队成员表现出的团队特质，再后来的许多年都得到多次重复印证。**我后来给每个人都单独手写了回信，简要感谢和评论了他的建议。我好像和大家占有的信息量突然接近了，成为了团队一员。

再后来几个月，当我真正意识到从外面突然派一位。。。我们也遇到这个情况了！

回头看，现在可以以很坑定地说，企业中的职务不该啊小，新商人的开头是很重要的。有管理学家说，如果你在商人一个新职务的头一年没有取得好成绩，没有简历起令人信服的领导力，以后你得工作会很宽男。第一年的工作效能是有乘数效应的。我想这说的不是新官上任三把火，而是说你以什么样的姿态和方法融入到新的组织中，对未来影响是很大的。

我到中粮几天后，接待了来访的渣打银行董事长，应为与他在香港是就认识，所以说话也直接。他说你来到新公司最大的优势是你带来了经验，但没有带来包袱，你最奥在四个月内做许多决策，因为这时你是客观的，时间长了，你就会对自己前面的决策产生偏爱，就不可观了，而且现在你的行为给团队方向性的信息也是有力量的。

企业中天天面对新变化，如果你哪天悟到新的思路，都可以有个新的开始，但时间一长，面都每日繁杂，想停下来跳出去比较难。新的植物则是天赐良机。因为这时你没有自我否定的包否，有面对万众期待，再加上过去的经验和反思，你脑子里可有已经有了理想公司的雏形。这犹如你画了一幅画，无论多用心，总有不满意的地方，现在可以给你另外一个机会再画一遍，这一定是令人兴奋的机会。

第7章：三标志 一个图形可以代表世界吗

华润、中粮、中化三家公司的标志后面都有故事，我也花了心思。因为构思产生公司标识的过程是个很纠结、很考验人的过程。他要用一个符号、几个字，把公司彻底深刻地说明白，甚至要把公司过去的几点、现在的能力、未来的向往，都表达清楚。要大家认为代表了公司，还要经得住时间考验；既要有方向又要有动力，既要对内有i要对外，既要对公司有深刻理解，又要有深刻感情，还要让大家过目不忘，还要有视觉美感。所以司标不仅是个艺术符号，他是公司整体有形无形多因素聚练额表达，说一家公司的司标象征公司的命运也不为过 。

。。。

当三股势力一样大的时候根本无法调和，还是要先角逐出来一个绝对老大的。这个过程不完成，就形成不了对外的合理；而且这过程时间越短，对三方势力总消耗越小。任何做大的组织和联盟，都有一个舵手领航的。

对面三方势力都是朝廷的人，如果希望他们同归于尽，就应该复制那个最弱的，打压那个最强的。

第9章 三场会

我的经验是战略回顾反思的、星动学习团队研讨的会，是公司自我审视、自我更新、自我再造的国臣给，是学习型 组织的基础。还是那句话，形式可以创造内容，所有新的理念，新的发展方向包括新的组织架构、新的投资都可能是在战略会上提出并得到大家热烈讨论，形成对公司的长远重大影响的。我经历的华润、中粮和中化三个公司，他们很重要的特点就是把战略研讨会当哼团队的重要工作方法。这个会是集体参与、群策群力、民主研讨的会，虽然领导也起到重要作用，但结论不是自己定的，是大家讨论形成的，是有共识的。团队的积极参与也是这些企业能够避免明显战略失误，能够调动大家积极性的重要原因。每个人都觉得自己是决策参与者，对公司决策有用有感，也使集体智慧得到了提升。

。。。

战略研讨会虽是开会，但不是一个人讲话大家听。它也可以是培训，但不是有谁要给大家上课。战略研讨会的基本点就是承认大会在大家中间。要通过研讨则会这种方式把大家的智慧激发出来。协调器俩，形成共识。这也是行动学习的和核心思想。比如说在企业里的组织结构权责划分是一个经常出现摩擦，并造成效率低下、决策水平降低的顽固问题。如何解决呢？理论上有很多种方法，有直线式、有矩阵式、由运营管理型、有战略管控型，还有衍射出的多种类型。这是大企业或者控股公司，特别是多元化公司总部来协调属下不同业务单元的方法。所有这些模式都有其长处和弊端，都可能会引起矛盾。

第11章 三上市 上市公司的益处

上市公司还有一个很大的优势，就是可以用资本市场给与的估值，用各种形式的股票来评价、激励员工。无论是期权、认股权、限制性股票还是奖励股票。或者员工直接持股，公司的整体经营水平集中反映在股票价格上，与员工的表现和收入直接相关，我觉是得最好的长期激励、综合激励。他的评价来自市场，奖励的资金也来子市场，员工彻底于企业整体利益绑在一，虽然是很小的股权，但性质有变。

。。。

前面说到的中粮包装也是，它从一家几乎要倒闭的小工厂逐步发展成金属包装行业的领先企业，他在设立扭亏为盈目标的初期就有上市的目标，授予了撒谎概念是前虚拟股权，在上市后有期权激励，在期权转为股权后有员工持股。这一切都在上市公司和相对规范的环境下进行，是很有力量的。对国有企业可以解决所有却为问题；对民营企业，可以解决员工和老板一条心的问题。公司上市有什么作用？ 不仅是集资，更不仅是流动性和套现，也不仅是有个市场价值和地位，上市后同时有了规则和管理升级的要求，也可以更好、更有效地市场化平啊、激励员工，这些都是管理经营好一家公司必须的条件。

第13章 三个字

人的思维很有意思，不读书不行，不读书没有逻辑框架，也不知道其实大部分的事情别人都说过了。但学习了不思考也不行，所谓“学而不思则罔”。不过，在企业更加如此，这些还不够，要有学，有思，还要有做，做了还要反思，反思还要总结、提炼、分享，在这个基础上，思维才能丰富进步。所有人在新的领域，开始的思维都是感性的、发散的、表层的。这就好像你到了一给新的岗位，虽然接受了很多信息，也提出对这些信息的观察和疑问，但这个过程必须要不断地重复，不断地刺激你的大脑，这样我们此逐步形成了一些因果的规律。我们才能够在因果规律不断重现中发现其互动关系，而后把他们分类并建立他们全域的、相互作用的公式体系。这在大脑里可能会形成一个环路，一个有效的处理器。这个过程通常是不自觉地，所谓孰能生桥，是一个缓慢的学习的过程。这些体会让我更好地理解哲学中认识论的概念，认识毛泽东主席写的《实践论》《矛盾论》。

如果你有了学习的方法和自己形成的思维体系，而这个体系是你自己在思考中获得的，不是你在书上读来的。其实，真的融会贯通的思维方法和思维习惯在书上是学不来的。而形成这个思维体系时很重的环节，就是你要用简洁、准确的语言把思考表达出来，形成文字。读是学习思考，写出来是更深入的斯奥。理清思路的过程就是深度思考的过程。夜深人静，辗转反侧，来回踱步，吸烟喝茶，阔约你这个思维上的临界点进入新的认识空间，有恍然大悟的愉悦。

《两晋悲歌》：慕容垂之小段后

废掉一个太子的政治布局：

1. 首先，需要策划太子去做哪些根本做不好的事，比如军事出征的国之司命，比如治水赈灾等复杂且影响巨大的民生工作，通过做这些事把太子的无能暴露出来。
2. 其次，需要与慕容农、慕容隆同期，分别衡量各自的政治筹码和能量，考虑是否真有实力扳倒太子；要确定这两位是否有胆子和水平一起做这玩命的事；还要确定这两位是否会感恩，否则就是冒着卖白粉的危险却在干卖白菜的活。
3. 再次，需要去衡量梳理太子党的权力网络，看看力挺太子的都是谁，有没有弱点，哪些是需要攻击打掉的，那些人是需要售卖当卧底的。
4. 此外，还需要哪些慕容垂的近臣，持之以恒地说太子弱智，以达到潜移默化的效果。但事实上慕容包做的非常好，他老爹身边的人都一致称赞她。

作为皇后，其态度是具有一锤定音的效果的，是应该在最后关头出来表态的，而不是开局就把王炸砸出去。

因为只要表态了，就一定会被知道，则就属于撕破脸，从此不再有调和的余地。

**精明这东西，不可外露。一是易遭嫉妒，二是易触碰他人利益。即什么都能看出来，又什么都改变不了，还满世界告诉别人自己能看出来的人，是最傻的。**

政治算法的第一考量，是远期利益和局面稳定。一旦个人感情和毫无穿插了尽力啊，就一定不是最优解。感情的补偿和结算，可以拿钱去处理，千万不要拿权力和资源去换。

基于底层逻辑总结出来的推论是用来约束搞不懂低层逻辑的人的，不是用来约束搞得懂底层逻辑的人的。

一视同仁只是口号 驰名双标才是人间正道

《中国哲学简史》冯友兰

道家主张无为，并不是叫人完全不动，或不做任何事情。他的用意是叫人不要以多为胜，“少”就是要抓住要害；也意味着，行事为人不要矫揉造作，不要恣肆放荡。

《庄子》第一章《逍遥游》文字简单，却充满了右的故事。这些故事蕴含的思想是说，人们所说的快乐，其实其中有不同的层次。自由发展人的本性，可以带来相对的快乐，但要达到“至乐”，必须对事物本性右更高的了解。

为达到快乐，第一步便是充分发展人的本性，为此人要有能自由发挥台南府的才能。

万物的本性而天赋的能力各有不同。他们之间的共同特点是：当他们充分并自由发挥天赋才能时，便同样感到快乐。

《黑客与画家》

保罗.格雷厄创业公式：  
（1）搭建原型

（2）上线运营（别管bug）

（3）收集反馈

（4）调整产品

（5）成长壮大

首先，他鼓励创业公司快速发布产品，因为这样可以尽早知道一个创意是否可行。其次，他认为一定要特别关注用户需要什么，这样才有办法将一个坏项目转变成好项目。