**1 概要设计**

* 1. **开发环境**

示教器界面开发利用的主要工具是QtCreator集成开发环境。选择QtCreator作为主要的开发工具，理由可归纳为以下几点：

1. 具有优良的跨平台特性。Qt支持以下的操作系统Microsoft Windows 95/98，Microsoft Windows NT,Linux,Solaris,SunOS,HP\_UX,Digital UNIX等。
2. Qt是面向对象的。Qt的良好封装机制使得Qt的模块化程度非常高，可重用性非常好，对于用户开发来说非常方便。Qt提供了一种称为信号槽的安全类型来代替回调函数，这使得各个元件之间的协同工作变得十分简单。
3. 丰富的API。Qt包括多达250个以上的C++类，还提供了基于模板的collection、file、I/O输入输出等类，甚至包括正则表达式的处理功能。
4. 支持2D/3D图形渲染，支持OPENGL。
5. 大量的开发文档。
6. XML支持。
   1. **界面总体框架设计**

****图1.1 界面总体设计框图

上图1.1所示为示教器界面的总体设计框图，其中示教器软件的基类为QtCreator的QWidget，基于该类继承了许多许多不同的对象，根据逻辑依次布局在软件界面之上，例如在显示机器人名称、机器人操作模式、机器人状态的功能上，利用了QLable，这个类继承QWidget，还有在显示急停按钮、USB接口、手动/自动开关按钮上，利用了QPushButton，这个类继继承QWidget，而在点击不同的控件做出不同的相应上，则利用了QtCreator特有的信号槽机制，这个机制类似C++的回调函数，根据对不同的控件进行不同的操作，可以让程序调用不同的函数，从而做出不同的逻辑响应。

除了示教器主界面布局之外，通过上图1.1还可看出在主界面还嵌套了若干个子界面，例如用户自定义界面、配置管理界面、变量管理界面、项目管理界面、程序管理界面、位置管理界面、信息报告界面，这7大模块界面均通过QtCreator的QStackedWidget实现，通过将7个界面放入窗口栈中，每个界面窗口都会有不同的索引，在程序根据索引调用相应的成员函数，就可以让程序显示索引所标记的界面，从而在不同的界面中，点击控件，完成不同的工作。

同理，在程序管理模块中，还嵌套了程序编辑界面，指令设置界面、指令修改界面、指令修改界面，这4个界面的实现方法和上述的7大模块界面类似，均通过QStackedWidget实现，具体的实现方式可以查看QtCreator的官方帮助文档。

**1.3 模块交互流程图**

示教器软件主要由七大模块构成，每个模块并不是独立工作的，而是交互工作的，尤其是变量管理模块、项目管理模块和程序管理模块，这三大模块起到了主要的功能，如下图所示为示教器模块交互流程图：



图1.2 模块交互流程图

由上图1.2所示为示教器的模块交互流程图，启动程序时，首先在变量管理模块新建变量，在这里可以新建全局作用域、项目作用域和程序作用域的变量，支持数据类型变量一共有十种，新建完变量之后，可以通过设置按钮或者示教按钮来设置变量的数值，设置完成之后，进入项目管理模块，新建项目和程序，选中程序之后，可以点击打开或者加载按钮读取程序文件的指令信息，并自动跳转到程序管理模块，区别在于打开程序时候，用户只拥有只读权限，而加载程序，用户可拥有对程序的编辑权限。在程序管理模块中，用户可对指令进行复制、粘贴、剪切、注释、取消注释等操作，其中在指令新建和指令修改的过程中，用户可调用在变量管理模块中设置好的变量，通过程序管理模块和变量管理模块的交互，可以大大增加开发的效率。

**2 详细设计**

**2.1 用户自定义模块**

**2.2 配置管理模块**

**2.3变量管理模块**

变量管理模块的作用是监测系统中的全局、项目和程序变量，并且能够设置10种数据类型的变量，从而方便程序的开发和调试，下面将从以下几个方面阐述该模块的技术实现：

1. 变量管理模块总体设计
2. 变量管理模块工作流程

**2.3.1 总体设计**

**** 图2.1 变量管理模块总体设计框图

如上图2.1所示为变量管理模块总体设计框图，通过该图可总结以下几点：

1. 变量管理模块存放与QStackedWidget中，其索引标志Index为2。此后所在该界面上所有的布局，均继承该Widget，模块采用Grid Layout类实现网格状布局。
2. QPushButoon:模块界面中的变量按钮，示教按钮、清除未用变量按钮、设置按钮均为该类的实例。其中变量按钮还在QPushButton中嵌套了QMenu，并在QMenu中放入QAction，从而实现变量的新建、删除、复制、粘贴、剪切、重命名等操作。另外在按钮的布局上，利用了Horizontal Spacer这个类来实现。
3. QLabel:用来显示变量类型。
4. QCombobox：用来实现10中不同数据类型的筛选，其中每种数据类型都会有不同的索引Index来进行标志，便于程序对不同的数据类型进行相应的操作。QLabel和QCombobox通过Horizontal Layout和Horizontal Spacer两个类实现布局。
5. QTreeview、QStandardItemModel、QStandardItem:用来显示全局、项目和程序变量。采用Qt的模型/视图机制，将数据的显示和存储分离开，模型对外提供标准接口存取数据，视图自定义数据的显示方式。当数据发生改变时，模型发出信号通知视图；当用户与视图进行交互时，视图发出信号提供交互信息，从而通过索引访问模型中的具体数据，进行下一步的逻辑操作。

**2.3.2 工作流程**

变量管理模块的工作流程图如下图2.2所示:

图2.2 变量管理模块工作流程图

如上图2.2所示为变量管理模块的工作流程图，接下来结合该流程图进行进一步的阐述和说明：

1. 启动程序，界面初始化:该示教器程序的主程序是一个阻塞函数，在主窗口对象Widget的接口中，通过set\_variable\_charge\_widget()函数来实现对变量管理模块的初始化，初始化的工作大致分为以下几个部分：
2. 为变量按钮添加新建、删除、赋值、粘贴、剪贴和重命名选项。其中利用了QtCreator的QMenu类和QAction类进行开发设计。
3. 为各个控件添加信号和操作，例如变量的新建、删除、复制、粘贴、剪切、删除和重命名等。
4. 创建视图和模型，首先在widget的头文件中创建treemodel\_variable对象来存储变量信息、创建root\_variable为模型treemodel\_variable的根节点，方便后期对变量的访问和操作，另外通过调用treemodel\_variable相应的函数对显示进行设置，例如通过setHeaderData函数来设置模型的标题。
5. 添加信号槽，实现点击视图上不同的item，控制用户对程序按钮的操作权限，降低用户在使用程序时候的误操作。
6. 为变量过滤器添加信号槽，通过为支持的数据类型添加索引标志，从而在筛选不同的数据类型时，作出相应的响应。
7. 点击变量模块按钮，启动变量解析模块。该部分主要解析全局作用域、项目作用域、程序作用域的变量，并且将不同数据类型的变量存储到变量管理模块的模型中，再通过视图显示出来。
8. 对变量进行操作可分为两种情况，即是否有使用变量过滤器的条件下对变量进行新建、复制、粘贴、剪切、删除等操作。
9. 对变量参数的赋值可以通过设置按钮和示教按钮两种方式来实现，其中数值的变化都是实时的在xml和data文件系统中进行更新。
10. 在使用清除未用变量按钮时，程序会先判断变量是否已经在当前加载的程序中使用，若已经使用，程序会警告该变量不可删除；若未使用，变量将成功删除，并且更新data和xml文件系统。

**2.3.3 文件系统**

由于变量管理模块的程序设计主要是基于对文件系统的设计，因此对文件系统组成的了解非常必要，下图2.3展示的就是文件系统的组成框图：

图2.3 文件系统组成框图

如上图2.3所示为所设计的变量管理模块文件系统，根目录是一个名为prog\_data\_xml的文件夹，在该文件夹下有三个文件夹，分别名为：data、prog和xml。在data、prog和xml文件夹的下一级目录下，有所创建的以项目命名的文件夹，用来存放项目下所创建的程序文件。

不同之处在于，xml文件夹下一级目录下除了有以项目命名的文件夹，用来存放程序之外，还会有唯一一个名为all的文件，它的作用是：用来存放用户创建的所有全局作用域的变量。同时，在各个项目文件夹下面，还会有一个同样以项目命名的文件夹，它的作用域是：用来存放项目作用域的变量。

如下图所示为xml文件管理系统示意图：

图2.4 xml文件管理系统示意图

**2.4 项目管理模块**

项目管理模块的主要作用是对项目和程序进行管理，并且进行相应程序的操作，例如加载、打开或者终止。接下来我将从以下几个方面来阐述项目管理模块的技术实现：

1. 总体设计
2. 工作流程

**2.4.1 总体设计**

****图2.5 项目管理模块总体设计框图

由图2.5所示为项目管理模块的总体设计框图，可以看出该模块主要由两大部分组成，一个是由QPushButton类所组成的按钮，例如加载按钮、打开按钮、终止按钮、信息按钮和刷新按钮，这些都是QPushButton的对象实例。另外，文件按钮将其传入QMenu类中，并在QMenu中添加QAction,从而实现文件的新建项目、新建程序、删除和重命名等选项。而在关于项目和程序的显示上面，主要是利用了Qt的视图/模型机制，通过该机制实现数据显示和数据存储的分离，从而让程序的开发更加灵活，另外，在项目管理模块界面的布局上，主要是利用了Grid Layout这个工具来实现的。最后，对于按钮的具体实现，比如，点击加载按钮，程序如何加载对于路径下的程序，并且读取其中的内容，显示到程序管理模块中，以上过程的实现方法主要是利用了Qt的信号槽机制，类似于C++的回调函数，但是Qt的信号槽机制更加的灵活和方法，使得开发者很容易的将程序界面上的控件通过自己设定的函数逻辑联系起来，从而达到用户想要的功能。

**2.4.2 工作流程**

了解了项目管理模块的总体设计思路之后，接下来将介绍该模块的工作流程及模块内部控件的逻辑关系，如下图所示为项目管理模块的工作流程图：

图2.6 项目管理模块工作流程图

如上图2.6所示为项目管理模块的工作流程图，接下来针对上图对该模块进行进一步的阐述：

1. 点击项目管理模块按钮，程序将跳转到程序管理模块界面，此时将显示用户曾经新建的项目和程序文件，并且显示在该模块界面中，此时，用户可以通过点击文件按钮下的新建项目或者新建程序选项，继续新建项目和文件。
2. 选中程序，用户才可以获得对加载、打开和终止按钮的点击权限。否则，用户将无法获得加载、打开或者终止按钮的点击权限，因此，用户无法进行下一步的操作。
3. 在正确的选中了程序之后，用户此时已经或者了权限，此时用户如果点击加载按钮，程序将在相应路径读取所选文件的内容，并且将内容显示到程序管理模块当中，读取完毕之后，程序将自动跳转到程序管理模块当中，用户在程序管理模块当中将拥有对所加载程序的读写权限。如果用户点击的是打开按钮，程序也对读取所选中文件的内容，显示到变量管理模块的程序编辑界面当中，但是用户只拥有读权限，无法对打开的程序进行任何编辑和修改操作。
4. 或用户想重新加载另外的程序，首先必须终止当前正在加载的程序之后，才可加载新的程序，否则，程序将报错，作出相应的提醒。

**2.5 程序管理模块**

程序管理模块是七大模块最为重要的一个模块，集合了示教器大部分的工作，包括程序的编辑、指令的插入、删除、复制、剪切、粘贴、变量的示教等等，接下来将从以下几个方面来阐述程序管理模块的技术实现：

1. 总体设计
2. 工作流程

**2.5.1 总体设计**

如下图所示为程序管理模块的总体设计框图：

图2.7 程序管理模块总体设计框图

由上图所示为程序管理模块的总体设计框图，可以看出程序管理模块界面是位于index=4的QStackedWidget中，在该界面中，又分别嵌套了四个界面：程序编辑界面、指令选择界面、指令设置界面还有指令修改界面，每个界面负责不同的工作，共同完成对程序的编辑功能，下面将对四个界面的功能和设计进行进一步的阐述：

1. 程序编辑界面：程序的编辑操作，包括指令的删除、复制、粘贴、剪切、注释和取消注释，其中按钮其中编辑按钮利用了Qt的QPushButton类，指令的删除、复制、粘贴、剪切、注释和取消注释利用了QMenu和QAction类。另外，在该界面中，还有指令新建按钮，该按钮利用QPushButton进行设计，点击该按钮，将跳转到指令选择界面，可以选择不同的指令。还有指令修改界面，该安妮同样利用QPushButton类设计，点击该按钮，将跳转到指令修改界面，完成相应的指令参数的修改。
2. 指令选择界面：该界面的设计主要利用了Qt的视图/模型机制，通过QTreeView、QStandardItemModel和QStandardItem类来实现指令的在界面的显示和选择功能，同时利用信号槽机制，使用户点击不同的指令选项时，将获得对界面按钮不同的操作权限，在获得确定按钮权限，点击之后将进入相应的指令设置界面。
3. 指令设置界面：该界面的设计和指令选择界面的设计类似，在指令参数的显示上，也是利用了QTreeView、QStandardItemModel和QStandardItem三个类，同样利用信号槽机制，使用户点击不同的选项，将获得对界面按钮不同的操作权限，从而对指令或者参数进行相应的操作，设置完成之后，点击确定按钮，将成功插入指令，并且跳转到程序编辑界面。
4. 指令修改界面：在程序编辑界面中，选中某行指令使其高亮，点击指令修改按钮，将进入相应指令的指令修改界面，界面的设计上和指令选择界面、指令设置界面类似，在完成相应的参数修改之后，点击确定按钮，将成功修改指令，并且跳转到程序编辑界面。

**2.5.2 工作流程**

上述阐述了各个指令的设计思路和功能，接下来将综合程序管理模块中各个界面的逻辑关系，如下图2.8所示为程序管理模块的工作流程图：

图2.8 程序管理模块工作流程图

由上图2.8所示为程序管理模块的工作流程图，启动程序的时候，首先在项目管理模块选中某个项目下的某个程序，这时可以对程序进行两种操作，分别是加载和打开。若点击打开按钮，将加载所选的程序，并且程序自动跳转到程序管理模块，但是用户此时只拥有只读权限，不可对程序进行编辑操作。若点击加载按钮，将加载所选的程序，并且程序自动跳转到程序管理模块，此时用户拥有读写权限，用户可对程序进行编辑功能，例如指令的插入、修改、复制、粘贴、剪切等等。加下来将以加载某一程序为例，对程序管理模块的各个界面进行进一步的阐述：

1. 在程序编辑界面，可以看到由编辑按钮、指令修改按钮、指令新建按钮、设置PC按钮和编辑按钮，这些按钮均利用QPushButton类进行设计，在程序指令的显示上，主要利用了QTextEdit类进行设计。
2. 点击程序编辑界面的新建按钮，将进入指令界面，该界面的底部有确定和返回按钮，有QPushButton所设计，在界面的主体部分，可以看到有四大指令组，分别是运动指令组、设置指令组、系统功能指令组和流程控制指令组，这些指令的显示由QTreeView、QStandItemModel、QStandardItem设计。
3. 在指令选择界面选中某个指令，点击确定按钮，将进入到指令对应的指令设置按钮，该界面底部有变量按钮、示教按钮、添加按钮、设置按钮、返回按钮和确定按钮，这些按钮由QPushButton设计，该界面的主体部分由QTreeView、QStandardItemModel、QStandardItem设计。对于界面的使用，这里不再过多阐述，详情请见示教器使用说明书。
4. 在程序编辑界面，选中某一指令，点击指令修改按钮，将进入到指令对应的指令修改界面，该界面和指令设置界面类似，底部有添加按钮、设置按钮、示教按钮、确定按钮和返回按钮，这些按钮由QPushButoon设计，指令修改界面的主体部分由QTreeView、QStandardItemModel、QStandardItem设计。

**2.6 位置管理模块**

位置管理模块的主要作用是实时显示机器人TCP点的位置，包括关节坐标系下的位置信息、世界坐标系下的位置信息以及电机轴的位置信息。接下来将从以下几个方面来阐述该模块的技术实现：

1. 总体设计
2. 工作流程

**2.6.1 总体设计**

如下图2.9所示为位置管理模块的总体设计框图：

图2.9 位置管理模块总体设计框图

由上图所示为位置管理模块的总体设计框图，该模块的主界面位于QStackedWidget，index为5的界面中，接下来将从类的角度阐述该模块的技术实现：

1. QPushButton：利用该类设计了位置管理模块界面底部的伺服按钮、关节按钮、世界按钮、电动速度按钮和电动按钮。其中电动按钮和电动速度按钮下拉的选项通过QMenu类和QAction类来实现。
2. QTableView、QStandardItemModel、QStandaedItem：利用这三个类来实现电机轴、关节坐标系和世界坐标系下的位置信息显示。
3. QLabel:速度、模式和电动速度的显示，可用过成员函数SetText来实现。
4. QComBobox:机器人、参照系和工具的选择，可通过成员函数SetCurrentIndex来显示不同的选项。
5. Grid Layout、Horizontal Spacer:布局管理器的实现。

**2.6.2 工作流程**

了解了位置管理模块的基本实现之后，接下来将对位置管理模块的工作流程作详细的阐述，如下图2.10所示即为位置管理模块的工作流程图：



图2.10 位置管理模块工作流程图

由上2.10所示为位置管理模块的工作流程图，启动程序，点击示教器主界面的切换坐标系按钮可以切换坐标系到关节坐标系、世界坐标系和工具收坐标系，在位置管理模块下由伺服按钮、关节按钮和世界按钮，分别显示点击轴，机器人在关节坐标系和世界坐标系的位置信息。同时还可以在该界面中设置点动度和点动模式。设置完成之后，通过点击+、-按钮来控制机器人点位运动，从而在位置管理模块中便可实时地看到机器人的变化。

**2.7 信息报告模块**

信息报告模块的主要作用是记录用户在使用示教器过程中的一些提示和警告信息，接下来将从以下几个方面来阐述信息报告模块的技术实现：

1. 总体设计
2. 工作流程

**2.7.1 总体设计**

如下图所示为信息报告模块的总体设计框图：



图2.11 信息报告模块总体设计框图

由上图2.11所示为信息报告模块总体设计框图，信息报告模块位于QStackedWidget,索引为6的堆栈窗口中，通过SetCurrentIndex可以显示该界面，在该模块界面中，主体的信息显示部分利用QTableView、QStandardItemMdoel、QStandItem来实现，确认按钮、全部确认按钮、帮助按钮、ID显示按钮则利用QPushButton类来实现，还有一个信息筛选的功能则利用QComBobox来实现，最后在该界面的布局上，主要利用GridLatout和Horizontal Spacer来实现。

**2.7.2 工作流程**