**需求：在实际开发中，Repeater，ListView，TableView等等都需要model，如果只是使用数字，列表，或者ListModel很多时候都不能满足我们的需要，因为有时候数据是动态获取的，无法预先知道所以我们需要自定义model，做法如下：在项目中点击右键-》添加新文件，出现下面的对话框，选择qt-》Qt Item Model**

|  |
| --- |
|  |

会出现选择model类型的对话框，

|  |
| --- |
|  |

这里是ListView的扩展所以使用QabstractListModel，填写好类的名字，ok

|  |
| --- |
|  |

在下一步，就会生成两个文件

目录结构如图：

|  |
| --- |
|  |

生成的model类的源文件在有几个没有实现的方法需要自己实现不过有些方法ListView一般用不到可以将其注释或者删除，或者不要管

需要添加一个方法QHash<int,QByteArray> ***roleNames***() const;这个方法是继承自AbstractItemModel的。

需要添加一个Qlist类型的成员变量

此外，还需要一个枚举来实现***roleNames方法***

***在头文件在定义：***

|  |
| --- |
| enum **myRoleNames**{  **Name** = Qt::DisplayRole+1,  **Value**  }; |

为这个类做一个单例模式。

|  |  |
| --- | --- |
| //customlistmodel.h  #ifndef CUSTOMLISTMODEL\_H  #define **CUSTOMLISTMODEL\_H**  #include <QAbstractListModel>  #include<QHash>  class **CustomListModel** : public QAbstractListModel  {  Q\_OBJECT  public:  enum **myRoleNames**{  **Name** = Qt::DisplayRole+1,  **Value**  };  explicit **CustomListModel**(QObject \***parent** = nullptr);  static CustomListModel\* **getInstance**();//单例方法  // Header:  QVariant ***headerData***(int **section**, Qt::Orientation **orientation**, int **role** = Qt::DisplayRole) const override;  // Basic functionality:  int ***rowCount***(const QModelIndex &**parent** = QModelIndex()) const override;  QVariant ***data***(const QModelIndex &**index**, int **role** = Qt::DisplayRole) const override;  QHash<int,QByteArray> ***roleNames***() const override;  private:  QList<QString> **m\_data**;  // QHash<int,QByteArray> m\_roles;  };  #endif // CUSTOMLISTMODEL\_H | //customlistmodel.cpp  #include "customlistmodel.h"  CustomListModel::**CustomListModel**(QObject \***parent**)  : QAbstractListModel(*parent*)  {  }  CustomListModel \*CustomListModel::**getInstance**()  {  static CustomListModel\* **p\_model** = new CustomListModel();  return p\_model;  }  QVariant CustomListModel::***headerData***(int **section**, Qt::Orientation **orientation**, int **role**) const  {  // FIXME: Implement me!  //对于ListView一般不需要这个  return QVariant{};  }  //rowCount其实就是指model的大小  int CustomListModel::***rowCount***(const QModelIndex &**parent**) const  {  // For list models only the root node (an invalid parent) should return the list's size. For all  // other (valid) parents, rowCount() should return 0 so that it does not become a tree model.  if (parent.isValid())  return 0;  // FIXME: Implement me!  //自己的代码  return m\_data.count();  }  QVariant CustomListModel::***data***(const QModelIndex &**index**, int **role**) const  {  if (!index.isValid())  return QVariant();  if(role ==myRoleNames::Name )  {  return m\_data[index.row()];  }  // FIXME: Implement me!  return QVariant();  }  QHash<int, QByteArray> CustomListModel::***roleNames***() const  {  QHash<int, QByteArray> **roles**;  roles.insert(myRoleNames::Name,"name");//将枚举映射为一个字符串供qml使用  roles.insert(myRoleNames::Value,"value");  return roles;  } |

需要在构造函数中为这个自定义模型设置数据

|  |
| --- |
| // customlistmodel.cpp  #include "customlistmodel.h"  CustomListModel::**CustomListModel**(QObject \***parent**)  : QAbstractListModel(*parent*)  {  m\_data.append("java");  m\_data.append("javaScript");  m\_data.append("python");  m\_data.append("c++");  m\_data.append("golang");  }  CustomListModel \*CustomListModel::**getInstance**()  {  static CustomListModel\* **p\_model** = new CustomListModel();  return p\_model;  }  QVariant CustomListModel::***headerData***(int **section**, Qt::Orientation **orientation**, int **role**) const  {  // FIXME: Implement me!  //对于ListView一般不需要这个  return QVariant{};  }  //rowCount其实就是指model的大小  int CustomListModel::***rowCount***(const QModelIndex &**parent**) const  {  // For list models only the root node (an invalid parent) should return the list's size. For all  // other (valid) parents, rowCount() should return 0 so that it does not become a tree model.  if (parent.isValid())  return 0;  // FIXME: Implement me!  //自己的代码  return m\_data.count();  }  QVariant CustomListModel::***data***(const QModelIndex &**index**, int **role**) const  {  if (!index.isValid())  return QVariant();  if(role ==myRoleNames::Name )  {  return m\_data[index.row()];  }  // FIXME: Implement me!  return QVariant();  }  QHash<int, QByteArray> CustomListModel::***roleNames***() const  {  QHash<int, QByteArray> **roles**;  roles.insert(myRoleNames::Name,"name");//将枚举映射为一个字符串供qml使用  roles.insert(myRoleNames::Value,"value");  return roles;  } |

在main.cpp加载qml文件之前注册自定义model类

|  |
| --- |
| #include <QGuiApplication>  #include <QQmlApplicationEngine>  #include<QQmlContext>  #include"customlistmodel.h"  int **main**(int **argc**, char \***argv**[])  {  #if QT\_VERSION < QT\_VERSION\_CHECK(6, 0, 0)  QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);  #endif  QGuiApplication **app**(*argc*, *argv*);  QQmlApplicationEngine **engine**;  //注册类型  QQmlContext\* **pcontext** = engine.rootContext();  pcontext->setContextProperty("CustomListModel",CustomListModel::getInstance());  const QUrl **url**(QStringLiteral("qrc:/main.qml"));  QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,  &app, [url](QObject \***obj**, const QUrl &**objUrl**) {  if (!obj && url == objUrl)  QCoreApplication::exit(-1);  }, Qt::QueuedConnection);  engine.load(url);  return app.exec();  } |

|  |
| --- |
| 效果 |

如果需要一些比较复杂的数据，可以自己将这些数据先封装到一个类，然后将自定义模型的数据类型修改一下，泛型类型使用自己封装的类

修改后的代码：

|  |  |
| --- | --- |
| //customlistmodel.h  #ifndef CUSTOMLISTMODEL\_H  #define **CUSTOMLISTMODEL\_H**  #include <QAbstractListModel>  class **CustomData**{  public:  // CustomData(QString str,int val){  // m\_str = str;  // m\_val =val;  // }  **CustomData**(QString **str**,int **val**):m\_str(str), m\_val(val){}  QString **m\_str**;  int **m\_val**;  };  class **CustomListModel** : public QAbstractListModel  {  Q\_OBJECT  public:  enum **myRoleNames**{  **Name** = Qt::DisplayRole+1,  **Value**  };  explicit **CustomListModel**(QObject \***parent** = nullptr);  static CustomListModel\* **getInstance**();  // Header:  QVariant ***headerData***(int **section**, Qt::Orientation **orientation**, int **role** = Qt::DisplayRole) const override;  // Basic functionality:  int ***rowCount***(const QModelIndex &**parent** = QModelIndex()) const override;  QVariant ***data***(const QModelIndex &**index**, int **role** = Qt::DisplayRole) const override;  QHash<int,QByteArray> ***roleNames***() const override;  private:  QList<CustomData> **m\_data**;  // QHash<int,QByteArray> m\_roles;  };  #endif // CUSTOMLISTMODEL\_H | //customlistmodel.cpp  #include "customlistmodel.h"  CustomListModel::**CustomListModel**(QObject \***parent**)  : QAbstractListModel(*parent*)  {  m\_data.append(CustomData("Java",80));  m\_data.append(CustomData("javaScript",70));  m\_data.append(CustomData("python",60));  m\_data.append(CustomData("c++",90));  m\_data.append(CustomData("golang",65));  }  CustomListModel \*CustomListModel::**getInstance**()  {  static CustomListModel\* **p\_model** = new CustomListModel();  return p\_model;  }  QVariant CustomListModel::***headerData***(int **section**, Qt::Orientation **orientation**, int **role**) const  {  // FIXME: Implement me!  //对于ListView一般不需要这个  return QVariant{};  }  //rowCount其实就是指model的大小  int CustomListModel::***rowCount***(const QModelIndex &**parent**) const  {  // For list models only the root node (an invalid parent) should return the list's size. For all  // other (valid) parents, rowCount() should return 0 so that it does not become a tree model.  if (parent.isValid())  return 0;  // FIXME: Implement me!  //自己的代码  return m\_data.count();  }  QVariant CustomListModel::***data***(const QModelIndex &**index**, int **role**) const  {  if (!index.isValid())  return QVariant();  if(role ==myRoleNames::Name )  {  return m\_data[index.row()].m\_str;  } else if(role == myRoleNames::Value)  {  return m\_data[index.row()].m\_val;  }  // FIXME: Implement me!  return QVariant();  }  QHash<int, QByteArray> CustomListModel::***roleNames***() const  {  QHash<int, QByteArray> **roles**;  roles.insert(myRoleNames::Name,"name");//将枚举映射为一个字符串供qml使用  roles.insert(myRoleNames::Value,"value");  return roles;  } |

main.cpp没有改变，所以不用摘抄

main.qml

|  |  |
| --- | --- |
| import QtQuick 2.15  import QtQuick.Window 2.15  Window {  width: 640  height: 480  visible: true  title: *qsTr*("Hello World")  //ListView,使用自定义的model  ListView{  width: 300  height: 400  model:CustomListModel  delegate :Text{  id:*txt*  text: name +" ::" +value  }  }  } | //效果 |

另外，还可以使用注册单例的方法，这里主要修改main.cpp，这种方法需要import注册的单例

|  |  |
| --- | --- |
| //main.cpp  #include <QGuiApplication>  #include <QQmlApplicationEngine>  #include<QQmlContext>  #include"customlistmodel.h"  int **main**(int **argc**, char \***argv**[])  {  #if QT\_VERSION < QT\_VERSION\_CHECK(6, 0, 0)  QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);  #endif  QGuiApplication **app**(*argc*, *argv*);  QQmlApplicationEngine **engine**;  //注册类型  //engine.rootContext()->setContextProperty("CustomListModel",CustomListModel::getInstance());  qmlRegisterSingletonInstance("CustomModel",1,0,"CustomListModel",CustomListModel::getInstance());  const QUrl **url**(QStringLiteral("qrc:/main.qml"));  QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,  &app, [url](QObject \***obj**, const QUrl &**objUrl**) {  if (!obj && url == objUrl)  QCoreApplication::exit(-1);  }, Qt::QueuedConnection);  engine.load(url);  return app.exec();  } | //main.qml  import QtQuick 2.15  import QtQuick.Window 2.15  import CustomModel 1.0  Window {  width: 640  height: 480  visible: true  title: *qsTr*("使用自定义的model")  //ListView,使用自定义的model  ListView{  width: 300  height: 400  model:CustomListModel  delegate :Text{  id:*txt*  text: name +" ::" +value  }  }  } |

效果是一样的

|  |
| --- |
|  |