**QML与c++交互学习笔记(一)**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/包括QT Widget嵌入QML(失败)**

**QT c++非界面类对象在QML中使用**

**C++直接使用QML中的类**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

说明，主要是对QT的文档内例子进行的一些分别解说，希望更容易的理解

C++导出到QML的过程。

1.导出一个简单的类Person

2.具体导出过程

假设我们要导出一个Person类，

**A** 那么就要考虑如何的一个类他才可以导出呢？

     他需要符合一定的条件

    1.继承自QObject

    2.有默认构造函数

**B** 如何导出呢？

通过一个函数

int qmlRegisterType(const char \*uri, int versionMajor, int versionMinor, const char \*qmlName)

int qmlRegisterType()

3.具体的例子

|  |
| --- |
| // person.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef PERSON\_H 2. #define PERSON\_H 3. #include <QObject> 4. class Person : public QObject 5. { 6. Q\_OBJECT 7. public: 8. explicit Person(QObject \*parent = 0); 9. }; 10. #endif // PERSON\_H 12. // person.cpp 13. #include "person.h" 14. Person::Person(QObject \*parent) : 15. QObject(parent) 16. { 17. } |

|  |
| --- |
|  |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include "person.h" 6. **int** main(**int** argc, **char** \*argv[]) 7. { 8. QApplication a(argc, argv); 9. qmlRegisterType<Person>("People",1,0,"Person"); 10. //qmlRegisterType<Person>(); 11. QDeclarativeView qmlView; 12. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 13. qmlView.show(); 14. return a.exec(); 15. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  import People 1.0 //如果是qmlRegisterType<Person>(); 导出就可以注释这条  Rectangle {      width: 640      height: 480      Person{}  } |

说明:我们通过qmlRegisterType<Person>("People",1,0,"Person");

向QML中导出Person类，这个类在People包中，在QML中需要使用Person类的

话就必须包含People包，通过import People 1.0来包含，之后就可以使用Person

创建对象使用来。

**QML与c++交互学习笔记(二)**

1.导出Person类中的成员方法

2.具体导出过程

    导出的方法有

    1.使用Q\_INVOKABLE

    2.使用 槽机制

3.具体代码

|  |
| --- |
| // person.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef PERSON\_H 2. #define PERSON\_H 3. #include <QObject> 4. class Person : public QObject 5. { 6. Q\_OBJECT 7. public: 8. explicit Person(QObject \*parent = 0); 9. Q\_INVOKABLE void FirstEcho(void); 10. public slots: 11. void SecondEcho(void); 12. }; 13. #endif // PERSON\_H |

|  |
| --- |
| // person.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include "person.h" 2. Person::Person(QObject \*parent) : 3. QObject(parent) 4. { 5. } 6. void Person::FirstEcho(void) 7. { 8. // 简简单单打印一句话 9. qDebug("call Person::FirstEcho"); 10. } 11. void Person::SecondEcho(void) 12. { 13. qDebug("call Person::SecondEcho"); 14. } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include "person.h" 6. **int** main(**int** argc, **char** \*argv[]) 7. { 8. QApplication a(argc, argv); 9. qmlRegisterType<Person>("People",1,0,"Person"); 10. //qmlRegisterType<Person>(); 11. QDeclarativeView qmlView; 12. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 13. qmlView.show(); 14. return a.exec(); 15. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  import People 1.0 //如果是qmlRegisterType<Person>(); 导出就可以注释这条  Rectangle {      width: 640      height: 480      Person{ id: per;}      MouseArea{          anchors.fill: parent;          onClicked:{              per.FirstEcho();              per.SecondEcho();          }      }  } |

说明:

    这里导出了两个函数分别是FirstEcho 和SecondEcho 两个函数，这两个函数本别是使用

FirstEcho使用使用 Q\_INVOKABLE导出，SecondEcho直接使用槽。

调用函数在控制台输出一些信息，这里是在鼠标点击界面后出发的。

**QML**与**c++**交互学习笔记**(**三**)**

1.导出Person类中的属性

2.具体导出过程

    1.导出Person一个颜色属性，一个int属性

    注意

    1. 当需要实现属性变化其他引用到此属性的属性也跟着变化的情况的话，需要设置属性相应的信号

    2. 设置属性的时候，使用的类型必须是已经导出到QML中的类型

3.具体代码

|  |
| --- |
| // person.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef PERSON\_H 2. #define PERSON\_H 3. #include <QObject> 4. #include <QColor> 5. class Person : public QObject 6. { 7. Q\_OBJECT 8. // 设置设置属性的名字是 bgcolor 9. // 对应读取函数名字 bgColor 10. // 对应写函数名字 setBgColor 11. // 属性发生改变后发送信号 sendBgColorChange 12. Q\_PROPERTY(QColor bgcolor READ getBgColor WRITE setBgColor NOTIFY sendBgColorChange) 13. // 设置设置属性的名字是 count 14. // 对应读取函数名字 getCount 15. // 对应写函数名字 setCount 16. // 属性发生改变后发送信号 sendCountChange 17. Q\_PROPERTY(**int** count READ getCount WRITE setCount NOTIFY sendCountChange) 18. public: 19. explicit Person(QObject \*parent = 0); 20. QColor getBgColor(void) const; 21. void setBgColor(const QColor& color); 22. **int** getCount(void); 23. void setCount(**int** count); 24. signals: 25. void sendBgColorChange(void); 26. void sendCountChange(void); 27. private: 28. QColor  m\_Color; 29. **int**     m\_Count; 30. }; 31. #endif // PERSON\_H |

|  |
| --- |
| // person.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include "person.h" 2. //--------------------------------- 3. // 4. Person::Person(QObject \*parent) : 5. QObject(parent), m\_Color("blue"), m\_Count(0) 6. { 7. } 8. //--------------------------------- 9. // 10. QColor Person::getBgColor(void) const 11. { 12. return m\_Color; 13. } 14. //--------------------------------- 15. // 16. void Person::setBgColor(const QColor& color) 17. { 18. m\_Color = color; 19. emit sendBgColorChange(); 20. } 21. //--------------------------------- 22. // 23. **int** Person::getCount(void) 24. { 25. return m\_Count; 26. } 27. //--------------------------------- 28. // 29. void Person::setCount(**int** count) 30. { 31. m\_Count = count; 32. emit sendCountChange(); 33. } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include "person.h" 6. **int** main(**int** argc, **char** \*argv[]) 7. { 8. QApplication a(argc, argv); 9. qmlRegisterType<Person>("People",1,0,"Person"); 10. //qmlRegisterType<Person>(); 11. QDeclarativeView qmlView; 12. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 13. qmlView.show(); 14. return a.exec(); 15. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  import People 1.0 //如果是qmlRegisterType<Person>(); 导出就可以注释这条  Rectangle {      width: 640      height: 480      color: per.bgcolor;      Person{ id: per;}      Text {          id: textlabel;          text: "text  " + per.count;      }      MouseArea{          anchors.fill: parent;          onClicked:{              // 当鼠标按下后，由于属性上有信号，当属性发生改变后，              // 所有引用此属性的值的都相应的发生改变              per.bgcolor = "red";              per.count = 20;          }      }  } |

说明：

    在person类中，设置了两个属性bgcolor, count ,他们分别在发送改变后调用自己对应的信号

具体看源代码，这里是设置来矩形框的颜色，文本框中文本。

**QML**与**c++**交互学习笔记**(**四**)**

1.导出Person类，并且一个Job类，Job类包含一个Person的指针

QML中两个类具有包含关系

2.具体导出过程

    1.通过属性来实现，具体的请看代码

3.具体代码

|  |
| --- |
| // person.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef PERSON\_H 2. #define PERSON\_H 3. #include <QObject> 4. #include <QColor> 5. class Person : public QObject 6. { 7. Q\_OBJECT 8. // 设置设置属性的名字是 name 9. // 对应读取函数名字 getName 10. // 对应写函数名字 setName 11. // 属性发生改变后发送信号 sendNameChange 12. Q\_PROPERTY(QString name READ getName WRITE setName NOTIFY sendNameChange) 13. // 设置设置属性的名字是 age 14. // 对应读取函数名字 getAge 15. // 对应写函数名字 setAge 16. // 属性发生改变后发送信号 sendAgeChange 17. Q\_PROPERTY(**int** age READ getAge WRITE setAge NOTIFY sendAgeChange) 18. public: 19. explicit Person(QObject \*parent = 0); 20. QString getName(void) const; 21. void setName(const QString& name); 22. **int** getAge(void); 23. void setAge(**int** age); 24. signals: 25. void sendNameChange(void); 26. void sendAgeChange(void); 27. private: 28. QString     m\_Name; 29. **int**         m\_Age; 30. }; 31. /\* 32. 设想一份工作给予一个人 33. \*/ 34. class Job : public QObject 35. { 36. Q\_OBJECT 37. Q\_PROPERTY(Person \*per READ getPerson WRITE setPerson NOTIFY sendPersonChange) 38. Q\_PROPERTY(QString jn READ getJobName WRITE setJobName NOTIFY sendJobNameChange) 39. public: 40. explicit Job(QObject \*parent = 0); 41. ~Job(); 42. void setPerson(Person \*per); 43. Person \*getPerson(void) const; 44. void setJobName(const QString & jobname); 45. QString getJobName(void) const; 46. signals: 47. void sendPersonChange(); 48. void sendJobNameChange(); 49. private: 50. Person \*m\_Person; 51. QString m\_JobName; 52. }; 53. #endif // PERSON\_H |

|  |
| --- |
| // person.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include "person.h" 2. //--------------------------------- 3. // 4. Person::Person(QObject \*parent) : 5. QObject(parent), m\_Name("unknow"), m\_Age(0) 6. { 7. } 8. //--------------------------------- 9. // 10. QString Person::getName(void) const 11. { 12. return m\_Name; 13. } 14. //--------------------------------- 15. // 16. void Person::setName(const QString& name) 17. { 18. m\_Name = name; 19. emit sendNameChange(); 20. } 21. //--------------------------------- 22. // 23. **int** Person::getAge(void) 24. { 25. return m\_Age; 26. } 27. //--------------------------------- 28. // 29. void Person::setAge(**int** age) 30. { 31. m\_Age = age; 32. emit sendAgeChange(); 33. } 34. //--------------------------------- 35. // 36. Job::Job(QObject \*parent) 37. :QObject(parent), m\_Person(0), m\_JobName("unknown") 38. { 39. } 40. //--------------------------------- 41. // 42. Job::~Job() 43. { 44. } 45. //--------------------------------- 46. // 47. void Job::setPerson(Person \*per) 48. { 49. m\_Person = per; 50. emit sendPersonChange(); 51. } 52. //--------------------------------- 53. // 54. Person \*Job::getPerson(void) const 55. { 56. return m\_Person; 57. } 58. //--------------------------------- 59. // 60. void Job::setJobName(const QString & jobname) 61. { 62. m\_JobName = jobname; 63. emit sendJobNameChange(); 64. } 65. //--------------------------------- 66. // 67. QString Job::getJobName(void) const 68. { 69. return m\_JobName; 70. } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include "person.h" 6. **int** main(**int** argc, **char** \*argv[]) 7. { 8. QApplication a(argc, argv); 9. qmlRegisterType<Person>("People",1,0,"Person"); 10. //qmlRegisterType<Person>(); 11. qmlRegisterType<Job>("People",1,0,"Job"); 12. QDeclarativeView qmlView; 13. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 14. qmlView.show(); 15. return a.exec(); 16. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  import People 1.0 //如果是qmlRegisterType<Person>(); 导出就可以注释这条  Rectangle {      width: 640      height: 480      Job {          id: jobA;          jn: "Learn";          per: Person { id: ps; name: "Luly"; age: 25; }      }      // 显示这份工作的一些信息      Rectangle{          x: 100; y: 100;          width: 100; height: 100;          Text { text: "Job name:" + jobA.jn; }          Text { y: 20; text: "Person name:" + ps.name; }          Text { y: 40; text: "Person age:"  + ps.age; }      }      MouseArea{          anchors.fill: parent;          onClicked:{              // 我要改变工作的名字 工作人的信息              jobA.jn = "Clean House";              ps.name = "Tom";              ps.age = 30;          }      }  } |

说明:

    主要是导出了两个类Person和Job, Job 包含一个Person的指针，这样后，可以看到

在QML中，我们需要给予Job对象一个Person来尽心赋值。

**QML**与**c++**交互学习笔记**(**五**)**

1.导出Person类，并且一个PersonGroup类，PersonGroup类是Person的一个组

2.具体导出过程

    1.通过属性来实现，具体的请看代码

3.具体代码

|  |
| --- |
| // person.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef PERSON\_H 2. #define PERSON\_H 3. #include <QObject> 4. #include <QDeclarativeListProperty> 5. #include <QList> 6. class Person : public QObject 7. { 8. Q\_OBJECT 9. Q\_PROPERTY(QString name READ getName WRITE setName NOTIFY sendNameChange) 10. Q\_PROPERTY(**int** age READ getAge WRITE setAge NOTIFY sendAgeChange) 11. public: 12. explicit Person(QObject \*parent = 0); 13. QString getName(void) const; 14. void setName(const QString& name); 15. **int** getAge(void); 16. void setAge(**int** age); 17. signals: 18. void sendNameChange(void); 19. void sendAgeChange(void); 20. private: 21. QString     m\_Name; 22. **int**         m\_Age; 23. }; 24. class PersonGroup : public QObject 25. { 26. Q\_OBJECT 27. Q\_PROPERTY(QDeclarativeListProperty<Person> members READ members) 28. public: 29. explicit PersonGroup(QObject \*parent = 0); 30. QDeclarativeListProperty<Person> members(void); 31. Q\_INVOKABLE **int** membersCount(void) const; 32. Q\_INVOKABLE Person \*member(**int** index) const; 33. private: 34. QList<Person\*> m\_MemberList; 35. }; 36. #endif // PERSON\_H |

|  |
| --- |
| // person.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include "person.h" 2. //--------------------------------- 3. // 4. Person::Person(QObject \*parent) : 5. QObject(parent), m\_Name("unknow"), m\_Age(0) 6. { 7. } 8. //--------------------------------- 9. // 10. QString Person::getName(void) const 11. { 12. return m\_Name; 13. } 14. //--------------------------------- 15. // 16. void Person::setName(const QString& name) 17. { 18. m\_Name = name; 19. emit sendNameChange(); 20. } 21. //--------------------------------- 22. // 23. **int** Person::getAge(void) 24. { 25. return m\_Age; 26. } 27. //--------------------------------- 28. // 29. void Person::setAge(**int** age) 30. { 31. m\_Age = age; 32. emit sendAgeChange(); 33. } 34. //--------------------------------- 35. // 36. PersonGroup::PersonGroup(QObject \*parent) 37. :QObject(parent) 38. { 39. } 40. //--------------------------------- 41. // 42. QDeclarativeListProperty<Person> PersonGroup::members(void) 43. { 44. return QDeclarativeListProperty<Person>(this, m\_MemberList); 45. } 46. //--------------------------------- 47. // 48. **int** PersonGroup::membersCount() const 49. { 50. return m\_MemberList.size(); 51. } 52. //--------------------------------- 53. // 54. Person \*PersonGroup::member(**int** index) const 55. { 56. return m\_MemberList.at(index); 57. } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include "person.h" 6. **int** main(**int** argc, **char** \*argv[]) 7. { 8. QApplication a(argc, argv); 9. qmlRegisterType<Person>("People",1,0,"Person"); 10. //qmlRegisterType<Person>(); 11. qmlRegisterType<PersonGroup>("People",1,0,"PersonGroup"); 12. QDeclarativeView qmlView; 13. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 14. qmlView.show(); 15. return a.exec(); 16. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  import People 1.0 //如果是qmlRegisterType<Person>(); 导出就可以注释这条  Rectangle {      width: 640      height: 480      property int pgcurIndex: 0;      PersonGroup{          id: group;          members: [              Person { name: "A"; age: 20},              Person { name: "B"; age: 21},              Person { name: "C"; age: 22},              Person { name: "D"; age: 23},              Person { name: "E"; age: 24}          ]      }      // 显示这份工作的一些信息      Rectangle{          x: 100; y: 100;          width: 100; height: 100;          Text { id: text1;  text: ""}          Text { id: text2;  y: 20; text: ""}          Text { id: text3;  y: 40; text: ""}      }      MouseArea{          anchors.fill: parent;          onClicked:{              //if (pgcurIndex < group.membersCount() - 1){ // 这里两种方法都可以              if (pgcurIndex < group.members.length - 1){                  pgcurIndex++;              }else{                  pgcurIndex = 0;              }              // 显示信息              text1.text = "PersonGroup index: " + pgcurIndex;              var person = group.member(pgcurIndex);              text2.text = "Person name: " + person.name;              text3.text = "Person age: "  + person.age;          }      }  } |

说明：

    这里导出了两个类Person， PersonGroup， PersonGroup保存来一个Person的组，

我们通过导出的函数来调用类面的成员，获取成员的信息.

**QML**与**c++**交互学习笔记**(**六**)** 关于**qt c++**中创建对象，**QML**获取此对象数据问题

1.假设

    1.在c++中创建一个Person的对象，

    2.在QML中获取并显示数据

    3.在c++中改变数据后，显示的数据能进行相应的改变

    也就是说我们实际是在c++中new一个对象出来，而把这个对象的数据在QML里面进行显示

2.具体代码

|  |
| --- |
| // person.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef PERSON\_H 2. #define PERSON\_H 3. #include <QObject> 4. #include <QDeclarativeListProperty> 5. #include <QList> 6. #include <QColor> 7. class Person : public QObject 8. { 9. Q\_OBJECT 10. Q\_PROPERTY(QString name READ getName WRITE setName NOTIFY sendNameChange) 11. Q\_PROPERTY(**int** age READ getAge WRITE setAge NOTIFY sendAgeChange) 12. public: 13. explicit Person(QObject \*parent = 0); 14. QString getName(void) const; 15. void setName(const QString& name); 16. **int** getAge(void); 17. void setAge(**int** age); 18. // 一个简单的函数, 获取蓝色 19. Q\_INVOKABLE QColor getColor(void) const; 20. Q\_INVOKABLE void changeNameAndAge(void); 21. signals: 22. void sendNameChange(void); 23. void sendAgeChange(void); 24. private: 25. QString     m\_Name; 26. **int**         m\_Age; 27. }; 28. #endif // PERSON\_H |

|  |
| --- |
| // person.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include "person.h" 2. //--------------------------------- 3. // 4. Person::Person(QObject \*parent) : 5. QObject(parent), m\_Name("unknow"), m\_Age(0) 6. { 7. } 8. //--------------------------------- 9. // 10. QString Person::getName(void) const 11. { 12. return m\_Name; 13. } 14. //--------------------------------- 15. // 16. void Person::setName(const QString& name) 17. { 18. m\_Name = name; 19. emit sendNameChange(); 20. } 21. //--------------------------------- 22. // 23. **int** Person::getAge(void) 24. { 25. return m\_Age; 26. } 27. //--------------------------------- 28. // 29. void Person::setAge(**int** age) 30. { 31. m\_Age = age; 32. emit sendAgeChange(); 33. } 34. //--------------------------------- 35. // 36. QColor Person::getColor(void) const 37. { 38. return QColor(Qt::blue); 39. } 40. //--------------------------------- 41. // 42. void Person::changeNameAndAge(void) 43. { 44. setName("Luly"); 45. setAge(31); 46. } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include <QtDeclarative/QDeclarativeContext> 6. #include "person.h" 7. **int** main(**int** argc, **char** \*argv[]) 8. { 9. QApplication a(argc, argv); 10. Person tmpPerson; 11. tmpPerson.setName("Tom"); 12. tmpPerson.setAge(25); 13. QDeclarativeView qmlView; 14. qmlView.rootContext()->setContextProperty("ps",&tmpPerson); 15. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 16. qmlView.show(); 17. return a.exec(); 18. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  Rectangle {      width: 640      height: 480      Text { text: "Person name:" + ps.name; }      Text { y: 20; text: "Person age:" + ps.age; }      Rectangle{ x: 20; y: 40;  width: 20; height: 20; color: ps.getColor();}      MouseArea{          anchors.fill: parent;          // 当鼠标按下后改变名字和年龄          onClicked: { ps.changeNameAndAge(); }      }  } |

说明：

    我们在c++中创建来一个对象，并且在把这个对象导出给QML调用用，我们设置来属性，QML中可以直接使用属性来进行赋值.

**QML**与**c++**交互学习笔记**(**七**)**

1.假设这样一种情况

    我这里由一个Wideget 继承自QWidget上面添加来一个QLabel, 一个QPushButton

    我如何把这个Wideget放到QML中使用，那么我当QPushButton 按下后我怎么在QML中进行处理呢？

    我这里指出一种方法

       让Wideget 继承QGraphicsProxyWidget，对Wideget进行导出，在QML中创建

此对象，在他导出的信中进行处理,具体代码。

    还有就是这个网址上说明来很多QML与c++之间通讯的方法，很悲剧的是我的assistant中却没有者部分，不知道版本低还是怎么的。

<http://doc.qt.nokia.com/4.7-snapshot/qtbinding.html>

2.具体代码

|  |
| --- |
| //widget.h  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #ifndef WIDGET\_H 2. #define WIDGET\_H 3. #include <QWidget> 4. #include <QGraphicsProxyWidget> 5. #include <QPushButton> 6. #include <QLabel> 7. #include <QLineEdit> 8. class Widget : public QGraphicsProxyWidget 9. { 10. Q\_OBJECT 11. public: 12. explicit Widget(QGraphicsItem \*parent = 0); 13. ~Widget(); 14. Q\_INVOKABLE void changeText(const QString& s); 15. signals: 16. void sendOnButton(void); 17. private: 18. QPushButton \*m\_Btn; 19. QLabel      \*m\_Label; 20. QWidget     \*m\_MainWidget; 21. }; 22. #endif // WIDGET\_H |

|  |
| --- |
| //widget.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include "widget.h" 2. Widget::Widget(QGraphicsItem \*parent) : 3. QGraphicsProxyWidget(parent) 4. { 5. m\_MainWidget = new QWidget; 6. m\_Btn = new QPushButton(m\_MainWidget); 7. m\_Label = new QLabel(m\_MainWidget); 8. m\_Btn->setText("PushButton"); 9. m\_Btn->setGeometry(10, 10, 100, 30); 10. m\_Label->setGeometry(10, 40, 200, 30); 11. QObject::connect(m\_Btn, SIGNAL(clicked()), this, SIGNAL(sendOnButton())); 12. setWidget(m\_MainWidget); 13. } 14. Widget::~Widget() 15. { 16. delete m\_MainWidget; 17. } 18. void Widget::changeText(const QString& s) 19. { 20. m\_Label->setText(s); 21. qDebug(" call Widget::changeText"); 22. } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include <QtDeclarative/QDeclarativeContext> 6. #include "widget.h" 7. **int** main(**int** argc, **char** \*argv[]) 8. { 9. QApplication a(argc, argv); 10. qmlRegisterType<Widget>("UIWidget", 1, 0, "Widget"); 11. QDeclarativeView qmlView; 12. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 13. qmlView.show(); 14. return a.exec(); 15. } |

|  |
| --- |
| // UICtest.qml  import Qt 4.7  import UIWidget 1.0  Rectangle {      width: 640      height: 480      color: "black"      Widget { id: uiwidget; x: 100; y: 100; width: 400; height: 100;          // 关键在这里，当一个信号导出后他的相应的名字就是第1个字母大写，前面在加上on          // 例如 clicked -- onClicked   colorchange --onColorchange;          onSendOnButton: { uiwidget.changeText(textinput.text); }      }      Rectangle{          x: 100; y: 20; width: 400; height: 30;  color: "blue"          TextInput {id: textinput; anchors.fill: parent; color: "white" }      }  } |

说明：

    这里实现的是当QPushButton按钮按下后，获取QML中TextInput上的文本，

对QLabel进行设置，关键点在于Widget中的信号函数sendOnButton, 他导出后在QML中

将引发的是onSendOnButton 只要在QML中对这个编写处理就可以实现，具体看代码。

**QML**与**c++**交互学习笔记**(**八**) qt c++**直接调用**QML**中的函数**,** 直接设置属性

1.这里主要是介绍，如何在c++中调用QML中的函数和设置QML中的属性的问题

2.具体代码

|  |
| --- |
| // UICtest.qml  import Qt 4.7  Rectangle {      id: mainWidget;      width: 640      height: 480      function callbyc(v)      {          mainWidget.color = v;          return "finish";      }      Rectangle{          id: secondRect;          x: 100;          y: 20;          width: 400;          height: 300;          Rectangle{              x: 10;              y: 20;              width: 30;              height: 40;              color: "#FF035721"              Text  {                  objectName: "NeedFindObj";                  anchors.fill: parent;                  text: "";              }          }      }  } |

|  |
| --- |
| // main.cpp  **[cpp]** [view plaincopyprint?](http://blog.csdn.net/lvliang2008/article/details/6307362)   1. #include <QtGui/QApplication> 2. #include <QtDeclarative/QDeclarativeView> 3. #include <QtDeclarative/QDeclarativeEngine> 4. #include <QtDeclarative/QDeclarativeComponent> 5. #include <QtDeclarative/QDeclarativeContext> 6. #include <QtDeclarative/QDeclarativeItem> 7. #include <QMetaObject> 8. **int** main(**int** argc, **char** \*argv[]) 9. { 10. QApplication a(argc, argv); 11. QDeclarativeView qmlView; 12. qmlView.setSource(QUrl::fromLocalFile("../UICtest/UICtest.qml")); 13. qmlView.show(); 14. // 获取根节点，就是 QML中 id是mainWidget的节点 15. QDeclarativeItem \*item = qobject\_cast<QDeclarativeItem\*>(qmlView.rootObject()); 16. item->setProperty("color", QVariant("blue")); 17. // 查找到我们需要的节点根均objectname NeedFindObj 来获得，并设置他的文本属性 18. QDeclarativeItem \*item1 = item->findChild<QDeclarativeItem \*>("NeedFindObj"); 19. if (item1) 20. { 21. item1->setProperty("text", QVariant("OK")); 22. } 23. // 调用QML中的函数, 分别是 函数所在的对象， 函数名，返回值， 参数 24. QVariant returnVar; 25. QVariant arg1 = "blue"; 26. QMetaObject::invokeMethod(item, "callbyc", 27. Q\_RETURN\_ARG(QVariant, returnVar),Q\_ARG(QVariant, arg1)); 28. qDebug(" %s",returnVar.toString().toLocal8Bit().data()); 29. return a.exec(); 30. } |

说明：

    这里的根节点是id为mainWidget的矩形元素，那么在C++中获取根节点后就可以，直接的设置他的属性了。其他属性也可以同样,调用指定节点内的函数是通过QMetaObject中的invokeMethod 来进行调用的。

最后所有关于QML和c++交互部分就基本写完，如果想要更多的东西，或者一些其他方法，强烈看看

http://doc.qt.nokia.com/4.7-snapshot/qtbinding.html，或者帮助文档，（究竟是不是我的文档里面没有还是怎么的）