Николай Секунов

Программирование на C++ в Linux

+ CD-ROM

- Принципы создания приложений в интегрированной среде KDE
- Программирование в среде KDevelop
- Разработка пользовательского интерфейса
- Реализация многозадачности
- Расширенные возможности языка C++

МАСТЕР_ПРОГРАММ_

Николай Секунов

Программирование на C++ в Linux

Санкт-Петербург
«БХВ-Петербург»
2003

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1
С28

Секунов Н. Ю.

С28 Программирование на C++ в Linux. — СПб.: БХВ-Петербург, 2003. — 368 с.: ил.

ISBN 5-94157-355-3

Книга посвящена созданию приложений, написанных на языке C++, в среде разработки KDevelop. Дано описание способов взаимодействия компонентов приложений. Рассмотрена работа с утилитой Qt Designer и описаны основные элементы управления, используемые в диалоговых окнах, а также классы, созданные для работы с ними. Читатель знакомится с концепцией Документ/Представление и учится создавать элементы пользовательского интерфейса приложения. Кроме того, в отдельных главах разбираются вопросы вывода на экран различной информации, сохранения и восстановления ее из файла, создания текстовых редакторов, работы с шаблонами классов и функций и организации многозадачности в приложении на основе взаимодействующих процессов. В завершение предоставляются рекомендации по созданию справочной системы приложения.

Для программистов

УДК 681.3.068+800.92C++
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Григорий Добин*
Редактор	*Наталья Сержантова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление серии	*Via Design*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 10.06.03.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 29,67.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02
от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-355-3

Содержание

Введение

В настоящее время все большую популярность получает использование свободно распространяемого в исходных текстах программного обеспечения. Причем эта тенденция наблюдается по отношению не только к приложениям, используемым для домашних нужд, но и к полноценным коммерческим приложениям. Одним из наиболее успешных проектов по созданию подобного программного обеспечения (наряду с веб-сервером Apache, языком Perl и набором утилит GNU) является операционная система Linux — полностью 32-разрядная, защищенная, многоплатформенная, многопользовательская и многозадачная UNIX-подобная операционная система. В настоящее время она является одной из основных операционных систем для серверов Интернет и серверов локальных сетей, но в последние годы явно наметилась тенденция ее использования и на домашних компьютерах.

Основным недостатком UNIX-подобных операционных систем до недавнего времени являлась сложность их использования, связанная с необходимостью работы в командной строке. В настоящее время этот недостаток успешно исправляется и работа в интегрированных средах Linux, таких как KDE, уже мало чем отличается от работы в Windows. Однако набор программных средств этих оболочек пока еще уступает по качеству и разнообразию набору программных средств Windows.

Многие программы Linux создаются программистами в свободное время. Можно смело утверждать, что число появляющихся на рынке новых приложений Linux напрямую зависит от качества и удобства используемой ими среды программирования, поскольку в свое свободное время человек хочет получать удовольствие от того, чем он занимается. Поэтому многие программисты отдавали свое предпочтение языку Kylix, являющемуся потомком языка Pascal, созданного в середине 70-х годов для начального обучения школьников программированию.

Ясно, что, имея такое наследство, этот язык не может обеспечивать высокой эффективности создаваемых на нем приложений. По собственному опыту

могу сказать, что программа, написанная на языке Pascal, работает в 4 раза медленнее, чем та же программа, написанная на языке C++. Причем сравнение программ производилось в среде Windows. В среде Linux выигрыш должен быть намного большим, поскольку язык C является внутренним языком операционной системы, и все ее функции оптимизированы для данного языка.

Одним из наиболее распространенных инструментариев разработки приложений на языке C++ для интегрированной среды KDE является KDevelop — полноценная среда разработки, объединяющая в себе все необходимые средства для создания и отладки приложений:

- встроенный транслятор и компилятор языка C++;
- мастер создания приложений, позволяющий создавать работоспособные заготовки различных типов приложений;
- мастер создания классов, позволяющий создавать заготовки новых классов и включать их в существующие приложения;
- систему управления файлами проекта, упрощающую работу с файлами заголовков и реализации классов;
- систему создания документации по приложению с использованием языка SGML, включающую в себя автоматическую трансляцию созданных файлов в формат HTML;
- систему автоматического создания документации по классам и функциям приложения в формате HTML;
- поддержку локализации создаваемых приложений, существенно облегчающую создание версии приложения для нового языка;
- использование для создания пользовательского интерфейса приложения утилиты Qt Designer фирмы TrollTech.
- поддержку совместной работы нескольких разработчиков над одним проектом;
- поддержку, помимо своего внутреннего отладчика, отладчиков ddd и kdbg;
- использование для редактирования значков приложения утилиты KIconEdit.

Таким образом, среда разработки KDevelop объединяет в себе все необходимое для создания приложения и позволяет пользователю получить всю требующуюся ему информацию.

Данная книга посвящена созданию приложений, написанных на языке C++, в среде разработки KDevelop. В ней предпринята попытка описания как можно более широкого круга приложений. Поэтому каждое из описываемых типов приложений рассмотрено только в общих чертах, что, впро-

чем, создает хорошую базу для их последующего более детального изучения по другим источникам.

Конечно, в предлагаемой книге освещены далеко не все описанные выше возможности среды разработки, но представленного в ней материала достаточно для написания весьма сложных приложений и довольно полного ознакомления с данной средой.

Для кого предназначена эта книга?

Потенциальными читателями этой книги являются как уже достаточно квалифицированные программисты, имеющие большой опыт работы на языке C++ в среде Windows, так и начинающие. Для тех, кто имеет опыт создания приложений в среде разработки Microsoft Visual Studio, в книгу включены комментарии, демонстрирующие основные отличия KDevelop от этой среды, что существенно облегчит им изучение KDevelop. Начинающие программисты найдут в данной книге исчерпывающий материал по созданию основных типов приложений в среде Linux и организации их взаимодействия.

Поскольку операционная система Linux всегда считалась прибежищем наиболее продвинутых программистов, среди ее пользователей сформировалось мнение, что чем сложнее процесс создания программы, тем круче создавший ее программист. Поэтому некоторые разработчики приложений Linux намеренно отказывались от использования редакторов ресурсов и создавали свои диалоговые окна сразу в исходных текстах программ. Редактирование программ они производили в обычных текстовых редакторах, например в Emacs, а для трансляции и компоновки приложения использовали пакетные файлы.

Конечно, разработка программ указанным выше способом под силу далеко не каждому программисту. Дело в том, что даже очень опытный программист не может создать подобным образом достаточно сложного приложения, или на его создание уйдет столько времени, что на рынке появится уже следующее поколение аналогичных программ. Поэтому даже квалифицированным программистам, придерживающимся подобных взглядов, следует обратить внимание на среду разработки KDevelop и на предоставляемые ею возможности по автоматизации программирования. Тогда они смогут направить свои усилия не на создание простых приложений сложными методами, а на создание сложных приложений простыми методами.

Каждый раздел данной книги иллюстрируется несколькими демонстрационными приложениями, позволяющими читателю на примере работающей программы самостоятельно изучить особенности рассматриваемого раздела программирования в среде KDevelop. Тексты этих примеров будут размещены на прилагаемом к книге CD.

В предлагаемой книге не только описана методика создания различных компонентов приложения, но и указано, какие сложности при этом могут возникнуть, можно ли их обойти и как это сделать. При изложении материала соблюдалась максимальная объективность.

Структура книги

В данной книге рассмотрены принципы создания основных типов приложений, работающих в интегрированной среде KDE. В отличие от Windows, где, по крайней мере, в простейших приложениях разработчик избавлен от необходимости следить за взаимодействием их компонентов, в Linux он должен с самого начала разбираться в способах взаимодействия этих приложений. Поэтому книга начинается с описания способов взаимодействия компонентов приложений.

Одной из областей, где наиболее интенсивно используется взаимодействие компонентов приложения, являются диалоговые окна. Поэтому сразу же после описания способов взаимодействия компонентов их использование демонстрируется на примере специального вида приложений, называемых диалоговыми приложениями. На примере этих приложений рассмотрена работа с утилитой Qt Designer и описаны основные элементы управления, используемые в диалоговых окнах, а также классы, созданные для работы с ними.

Диалоговые приложения являются особым видом приложений, позволяющим с минимальными затратами создать оболочку для работы какого-либо алгоритма. Для создания более сложных приложений разработчику необходимо ознакомиться с концепцией Документ/Представление и научиться создавать элементы пользовательского интерфейса приложения.

После ознакомления с этими вопросами читатель переходит к рассмотрению основных видов приложений. Он учится выводить на экран различную информацию, а также сохранять эту информацию в файле и восстанавливать ее из файла. Особая глава посвящена вопросу создания текстовых редакторов, поскольку именно они считаются основным типом приложений в большинстве операционных систем.

После получения базовых знаний по использованию среды разработки KDevelop читатель может перейти к изучению более сложных вопросов программирования в ней. В специальных главах рассмотрена работа с шаблонами классов и функций, работа с классами коллекций и организация многозадачности в приложении на основе взаимодействующих процессов. В завершение предоставляется информация по созданию справочной системы приложения.

В приведенном ниже списке разделов и глав дано краткое их описание, позволяющее пользователю лучше ориентироваться в структуре книги.

Глава 1. Взаимодействие компонентов приложения

Данная глава посвящена принципам организации взаимодействия различных компонентов приложения. В ней дано подробное описание использующихся для этого сигналов и приемников, приведены сведения о системе обработки событий в приложениях KDevelop, а также представлены использующиеся для этого функции и классы.

Глава 2. Диалоговые окна и простейшие элементы управления

В настоящей главе рассмотрено создание диалоговых окон, диалоговых окон с вкладками и мастеров с использованием приложения Qt Designer фирмы TrollTech, описаны основные элементы управления диалогового окна и использующиеся для работы с ними классы. Здесь же продемонстрировано создание диалогового приложения из заготовки приложения KDE с минимальными возможностями.

Глава 3. Классы элементов управления

В данной главе описаны достаточно сложные многофункциональные элементы управления диалогового окна, для работы с которыми используется широкий набор функций поддерживающих их классов.

Глава 4. Классы приложений, документов и представлений

В этой главе рассмотрена концепция Документ/Представление и описаны реализующие ее класс документа, класс представления и класс приложения.

Глава 5. Создание элементов пользовательского интерфейса

В данной главе рассмотрены основные типы элементов пользовательского интерфейса приложений, включая меню, панель инструментов и строку состояния. Поскольку в библиотеке Qt, в отличие от библиотеки MFC, работа с элементами пользовательского интерфейса не скрывается от разработчика, процедура создания пользовательского интерфейса в среде разработки KDevelop требует от него большего внимания, чем в среде Visual Studio.

Глава 6. Вывод информации на экран

В настоящей главе рассмотрены принципы организации графического интерфейса приложений, использующих библиотеку Qt или библиотеку интегрированной среды KDE.

Глава 7. Работа с файлами документов

В данной главе рассмотрено сохранение информации приложения в файле и извлечение ее из файла в рамках концепции Документ/Представление.

Глава 8. Работа с текстовыми документами

Операционная система UNIX, как и большинство других операционных систем, создавалась для работы преимущественно с текстовыми приложениями и оптимизирована для выполнения данной задачи. Поэтому вопросы создания текстовых редакторов вынесены в отдельную главу.

Глава 9. Шаблоны и классы коллекций

В данной главе рассматриваются расширенные возможности языка C++, реализованные в библиотеке Qt. Шаблоны представляют собой метод задания функций обработки для переменных абстрактных типов. Это позволяет минимизировать число практически идентичных по тексту функций, выполняющих одну и ту же обработку для переменных различных типов. Использование шаблонов дает возможность сократить размер исходного текста программ и обеспечивает гарантированное внесение изменений во все функции обработки различных типов данных, выполняющих над ними одну и ту же операцию. Классы коллекций можно рассматривать как практическую реализацию идеи шаблонов, позволяющую разработчику создать простую и эффективную структуру хранения данных приложения.

Глава 10. Реализация многозадачности в приложении

Операционная система UNIX является истинно многозадачной, поскольку допускает одновременное исполнение нескольких независимых процессов. Во многих случаях возникает необходимость разбиения приложения на несколько взаимодействующих процессов, в каждом из которых производится своя обработка. Разбиение приложения на процессы позволяет существенно сократить его простои, связанные с ожиданием затребованных им ресурсов, т. к. в этом случае прекращает работу только один процесс приложения, а не все приложение сразу.

Глава 11. Справка в приложении

Любое серьезное приложение, независимо от того, создается ли оно для внутреннего использования или для продажи, должно содержать обширную справочную систему, позволяющую получить информацию по любому вопросу, связанному с данным приложением. Созданию справочной системы приложения и посвящена эта глава.

Приложение 1. Что на CD

Содержит описание приложений, помещенных на прилагаемый к книге CD.

Приложение 2. Ресурсы Интернета

Содержит адреса сайтов, на которых можно получить дополнительную информацию.

Соглашения, принятые в данной книге

В этой книге использовалось специальное форматирование для выделения некоторых текстов или фрагментов текста. Ниже приведены основные принципы выделения текста.

- Исходные тексты фрагментов программ, представляющие собой одну или более строк текста, выделяются специальным шрифтом. При этом комментарии, в том числе и вставленные мастером создания приложений, приводятся на русском языке:

```
/** Конструктор класса KDEEditView */
KDEEditView::KDEEditView(QWidget *parent, const char *name)
  : KEdit(parent, name)
{
 setBackgroundMode(PaletteBase);
}
```

- Когда на эти фрагменты программ имеются ссылки из различных фрагментов текста, они оформляются листингом, как показано ниже:

Листинг 2.1. Заголовок класса `TestDlgImpl`

```
/** Класс реализации диалогового окна */
class TestDlgImpl : public TestDlg
{
  Q_OBJECT
 public:
  TestDlgImpl(QWidget *parent=0, const char *name=0,
                    bool modal = true, WFlags fl = 0);
  ~TestDlgImpl();

 public slots:
  virtual void copyFlag(bool);
  virtual void boxCopy(const QString&);
  virtual void destination(int);

 private:
  bool copy;
  int  dest;
};
```

- Если в тексте встречается имя класса, функции, имя типа переменной или фрагмент программного кода длиной менее строки, он выделяется специальным шрифтом. Например: `int`, `QWidget`, `slotUpdateSave` и т. д.

- Имена функции обычно даются в полной форме. Это значит, что если функция является членом класса, то перед ней указывается имя класса, членом которого она является. При этом указывается имя того класса в иерархии, в котором впервые появилась данная функция. Например: `KAction::setEnabled`. Если перед именем функции не стоит имя класса, значит это либо глобальная функция, либо рассматриваемая в настоящее время функция пользовательского класса, либо часто встречающаяся функция, полное описание которой было только что приведено.
- Заголовок диалогового окна, имя кнопки или команды меню выделяется специальным жирным шрифтом. Например: кнопка **Open**, команда меню **File | New** и т. д.
- Имена клавиш помещаются в угловые скобки, например <Ctrl>, <F1> и т. д.
- Если требуется одновременно нажать несколько клавиш, они объединяются знаком (+). Например: <Ctrl>+<Alt>+<Del>.
- При первом появлении нового термина он выделяется курсивом. Например: *новый термин*.
- Примечания оформляются специальным стилем, как это показано ниже:

Примечание

Примечания содержат дополнительную информацию, которую можно и пропустить.

- Информация, на которую следует обратить особое внимание, оформляются следующим образом:

Внимание!

На эту информацию следует обратить особое внимание.

- Советы читателю оформляются так:

Совет

Это всего лишь совет.

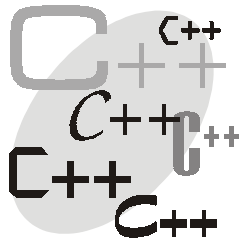
Требования к аппаратным средствам и программному обеспечению

Для работы с описанной в данной книге средой программирования KDevelop 2.1.3 требуется следующее аппаратное и программное обеспечение:

- IBM-совместимый PC с процессором Intel Pentium II/III/Celeron или AMD Athlon/Duron;

- не менее 128 Мбайт оперативной памяти;
- свободное пространство на системном жестком диске не менее 2 Гбайт;
- дисплей SVGA и соответствующий видеоадаптер, обеспечивающие разрешение не менее 800×600 точек и 256 цветов (рекомендовано 65 536 цветов);
- дисковод CD-ROM;
- совместимая с Linux мышь;
- операционная система Linux с KDE 3.0.3 (например, Red Hat 8.0 Mandrake 9.0) и выше.

ГЛАВА 1



Взаимодействие компонентов приложения

Прежде чем перейти к описанию приложений, создаваемых в среде разработки KDevelop, необходимо сначала ознакомиться с тем, каким образом взаимодействуют друг с другом отдельные приложения и отдельные части одного и того же приложения.

Приложения операционной системы Linux взаимодействуют с ней посредством обмена событиями. События в объектах могут возникать по различным причинам, как вследствие взаимодействия пользователя с элементами пользовательского интерфейса, так и вследствие внутренних причин, например, завершения какой-то обработки. Единственное, что их объединяет, — это необходимость передать некоторую информацию другому объекту приложения.

Сигналы и приемники

В основе взаимодействия компонентов библиотеки Qt лежит концепция *сигналов* (signal) и *приемников* (slot), являющаяся альтернативой концепции использования для связи между объектами приложения указателей на функции, лежащей в основе X Window. Старая концепция существенно усложняла текст программы и не позволяла производить проверку типов возвращаемых объектов (поскольку эти функции имели тип возвращаемого значения `void*`). В основе концепции сигналов и приемников лежит простой принцип: при возникновении в некотором объекте события, могущего заинтересовать другие объекты, он посылает сигнал, не заботясь о том, будет ли он принят. В заинтересованном объекте устанавливается ловушка на данный сигнал, запускающая при его обнаружении функцию обработки возникшего события, называемую в библиотеке Qt *приемником*.

Сигналы и приемники не поддерживаются языком C++, однако их объявления включаются в структуры классов данного языка. Это кажущееся противоречие объясняется тем, что файлы заголовков и реализации классов, создаваемые разработчиком в среде KDevelop, не являются окончательными,

а будут перед своей компиляцией обработаны препроцессором moc (Meta Object Compiler), устраняющим все противоречия и создающим необходимые структуры для обработки событий.

Для использования механизма сигналов и приемников при обработке событий в своих приложениях разработчик должен:

1. Создавать сигналы и приемники только в классах, производных от класса `QObject`.
2. Добавить в начало заголовка класса макрос `Q_OBJECT` (без последующей точки с запятой).
3. Обработать файл заголовка препроцессором moc для получения компилируемого файла реализации.

Поскольку большинство приложений KDE и Qt используют automake и autoconf (в том числе и все приложения, созданные в среде KDevelop), то вызов препроцессора moc осуществляется в них автоматически, если в этом возникает необходимость. Для этого в заголовок класса и включается макрос `Q_OBJECT`.

Посылка сигналов

Как правило, посылаемые сигналы включаются в заголовок класса, объект которого будет их посылать. Для этого используется фиктивный модификатор прав доступа `signals`, который может использоваться только для функций, не возвращающих значения. Пример заголовка класса, посылающего сигналы, приведен в листинге 1.1.

Листинг 1.1. Заголовок класса, посылающего сигналы

```
class someClass : public QObject
{
 Q_OBJECT

public:
 someClass();

 void someFunction(int);

signals:
 void someSignal();
 void someParameterSignal(int);
};
```

Как видно из листинга 1.1, в качестве сигналов могут выступать как функции без аргументов, так и функции с аргументами, в которых передаются различные параметры посылаемого сигнала.

Посылка сигнала может производиться в любой функции приложения с использованием ключевого слова `emit`. Для транслятора языка C++ это ключевое слово определено как отсутствие какого-либо текста, поэтому для него посылка сигнала интерпретируется как простой вызов функции. Но препроцессор moc при обнаружении данного ключевого слова создает и инициализирует соответствующий метаобъект, реализующий сигнал как функцию члена класса в выходных файлах препроцессора. В листинге 1.2 приведен пример использования ключевого слова `emit` для посылки сигнала `someParameterSignal` из объекта описанного выше класса.

Листинг 1.2. Посылка сигнала

```
void someClass::someFunction(int i)
{
 emit someParameterSignal( i);
}
```

В данном случае функция `someFunction` используется исключительно для посылки сигнала и задания значения его аргумента, однако в этой функции может производиться и любая другая обработка, а посылка сигнала будет только одной из выполняемых ею задач.

Описанный выше метод посылки сигнала широко распространен, однако встречаются случаи, когда класс не может являться потомком класса `QObject`, но из его объекта требуется послать сигнал. В этом случае для посылки сигнала используется объект специального класса `QSignal`. Для использования этого класса в приложении в файл заголовка класса, из которого нужно посылать сигналы, прежде всего, следует включить файл заголовка qsignal.h, содержащий описание класса `QSignal`. После этого в заголовок данного класса добавляется указатель на объект типа `QSignal` и объявление метода `connect`, которое должно иметь следующий вид:

```
void connect(QObject* receiver, const char* member);
```

В конструкторе класса необходимо, используя оператор `new`, создать динамический объект класса `QSignal` и уничтожить его в деструкторе класса оператором `delete`.

После этого в классе необходимо реализовать его функцию `connect`. Для класса `someClass`, содержащего указатель на объект класса `QSignal` с именем `pQSignal`, эта реализация должна иметь следующий вид:

```
void connect(QObject* receiver, const char* member)
{
 pQSignal-> connect(receiver, member);
}
```

После этого для посылки сигнала следует вызывать функцию

```
pQSignal->activate();
```

Реализация приемников

В отличие от сигналов, являющихся функциями C++ только по своему синтаксису, приемники представляют собой полноценные функциональные члены класса, которые могут вызываться не только как обработчики сигналов, но и самостоятельно, из других функций приложения. Однако объявление приемника все-таки отличается от объявления любой другой функции класса добавлением в ее модификатор права доступа ключевого слова `slots`. Кроме того, функции приемников, так же как и функции соответствующих им сигналов, не могут возвращать никакого значения. Пример заголовка класса, содержащего приемники, приведен в листинге 1.3.

Листинг 1.3. Заголовок класса, содержащего приемники

```
class otherClass : public QObject
{
 Q_OBJECT

public:
 otherClass();

public slots:
 void someSlotl();
 void someParameterSlot(int);
};
```

При сравнении листингов 1.1 и 1.3 следует обратить внимание на то, что если сигналы не имеют других прав доступа, кроме фиктивного права доступа `signals`, то для приемников обязательно нужно указать модификатор права доступа к ним. Если этот приемник не должен наследоваться потомками данного класса, для него следует указать модификатор права доступа `private`. Для наследуемых приемников достаточно указать модификатор права доступа `protected`. Модификатор доступа `public`, если разобраться, определяет их уже не как приемники, а как обычные функции класса, поскольку разрешает доступ к ним из других классов, что никогда не происходит с приемниками.

Примечание

Хотя все приемники являются полноправными функциями приложения, настоятельно не рекомендуется помещать в этот раздел описания класса другие его функции, не являющиеся приемниками. В тех случаях, когда за описанием сиг-

налов должно следовать описание других членов класса, для них следует в явном виде указать модификатор прав доступа, что будет свидетельствовать о завершении описания приемников.

Реализация соединения

Для связи сигнала с приемником используется статическая функция `QObject::connect`, имеющая следующий синтаксис:

```
bool connect(const QObject* sender, const char* signal, const QObject*
receiver,
const char* member)
```

Первым аргументом данной функции является указатель на объект класса, посылающего сигнал, вторым аргументом — указатель на строку, идентифицирующую сигнал, третьим аргументом — указатель на объект класса, в котором будет производиться обработка данного сигнала, и четвертым — указатель на строку, идентифицирующую функцию обработки данного сигнала (приемник).

Для получения строки, идентифицирующей сигнал, как правило, используется макрос `SIGNAL`, в качестве аргумента которому передается сигнатура функции сигнала (его имя и типы аргументов), а для получения строки, идентифицирующей приемник, используется макрос `SLOT`, в качестве аргумента которому передается сигнатура функции приемника. Поскольку связывание сигнала с приемником можно трактовать как способ обеспечения вызова функции одного класса из объекта другого класса, списки формальных параметров функций сигнала и приемника должны совпадать.

Примечание

Допускается, чтобы список формальных параметров приемника был короче списка формальных параметров сигнала. В этом случае требуется, чтобы список формальных параметров приемника совпадал с началом списка формальных параметров сигнала. Допускается и полное отсутствие формальных параметров у приемника.

Во избежание появления в подобных случаях предупреждений о неиспользованных аргументах функции, в описании сигнала указываются только типы формальных параметров без их имен.

Допускается трансляция сигнала, т. е. использование в качестве приемника сигнала другого сигнала, который должен быть послан объектом-получателем при обнаружении исходного сигнала. В этом случае вместо текстового идентификатора приемника при вызове функции `connect` используется текстовое описание нового сигнала.

Поскольку функция `QObject::connect` полностью характеризует устанавливаемое соединение, она может располагаться в любой точке приложения,