

Содержание

Над книгой работали	11
Предисловие	12
Глава 1. Модель реактивного программирования – обзор и история	17
Событийно-ориентированная модель программирования	18
Событийно-ориентированное программирование в системе X Window.....	19
Событийно-ориентированное программирование в среде Microsoft Windows	20
Событийно-ориентированное программирование в каркасе Qt	22
Событийно-ориентированное программирование средствами библиотеки MFC	23
Прочие модели событийно-управляемого программирования	24
Ограничения классических моделей обработки событий.....	24
Реактивная модель программирования	25
Ключевые интерфейсы реактивной программы.....	26
Методы вталкивания и втягивания данных	28
Дуальность интерфейсов IEnumerable и IObservable	28
Превращение событий в наблюдаемый источник	31
Методологические замечания	36
Итоги	37
Глава 2. Современный язык C++ и его ключевые идиомы	39
Принципы проектирования языка C++.....	40
Абстракция нулевой стоимости	40
Выразительность	40
Взаимозаменяемость	43
Усовершенствования языка, повышающие качество кода.....	44
Автоматический вывод типов	44
Единообразный синтаксис инициализации.....	46
Вариадические шаблоны	46
Ссылки rvalue	48
Семантика перемещения.....	50
Умные указатели	52
Лямбда-функции	54
Функциональные объекты и лямбда-функции	55

Композиция, карринг и частичное применение функций.....	57
Обёртки над функциями.....	60
Операция композиции функций.....	61
Прочие возможности языка.....	63
Выражения-свёртки	63
Сумма типов: тип variant	64
Прочее	65
Циклы по диапазонам и наблюдатели.....	65
Итоги	69

Глава 3. Параллельное и многопоточное

программирование на языке C++	70
Что такое параллельное программирование.....	71
Здравствуй, мир потоков!	72
Управление потоками	74
Запуск потока.....	74
Присоединение к потоку.....	75
Передача аргументов в поток.....	77
Использование лямбда-функций	79
Управление владением	80
Совместный доступ потоков к данным.....	82
Двоичные семафоры	84
Предотвращение тупиков	87
Условные переменные	91
Потокобезопасный стек	93
Итоги	96

Глава 4. Асинхронное программирование

и неблокирующая синхронизация в языке C++	98
Асинхронные задачи в языке C++	99
Фьючерсы и обещания	100
Класс <code>std::packaged_task</code>	102
Функция <code>std::async</code>	104
Модель памяти в языке C++	106
Параллельный доступ к памяти	106
Соглашение о порядке модификации памяти	107
Атомарные операции и типы в языке C++	108
Атомарные типы.....	108
Тип <code>std::atomic_flag</code>	111
Тип <code>std::atomic<bool></code>	113
Тип <code>std::atomic<T*></code> и арифметика указателей	116
Общий случай шаблона <code>std::atomic<></code>	117
Порядок доступа к памяти.....	118
Последовательно согласованный порядок доступа	119

Результат : последовательная согласованность.....	120
Семантика захвата и освобождения	120
Ослабленный порядок доступа к памяти	122
Неблокирующая очередь.....	124
Итоги	126

Глава 5. Знакомство с наблюдаемыми источниками

Шаблон «Наблюдатель»	128
Ограниченность классического шаблона «Наблюдатель»	131
Обобщённый взгляд на шаблоны проектирования	133
Объектно-ориентированная модель программирования и иерархии	135
Обработка выражений с помощью шаблонов «Композит» и «Посетитель»	136
Разглаживание многоуровневых композитов для итеративного доступа	142
Операции отображения и фильтрации списков.....	146
От наблюдателей к наблюдаемым источникам.....	149
Итоги	153

Глава 6. Введение в программирование потоков

событий на языке C++

Что такое программирование потоков данных.....	156
Преимущества модели программирования потоков данных	157
Прикладное программирование с использованием библиотеки Streams	157
Ленивые вычисления	158
Пример программы для обработки потока данных.....	159
Агрегирование значений в парадигме потоков данных	160
Погружение стандартных контейнеров в парадигму потоков данных	160
Несколько слов о библиотеке Streams	161
Программирование потоков событий	162
Преимущества программирования на основе потоков событий.....	162
Библиотека Streamulus и её программная модель	162
Библиотека Spreadsheet для оповещения об изменениях данных	168
Библиотека RaftLib – ещё один инструмент обработки потоков данных	170
Потоки данных и реактивное программирование	172
Итоги	173

Глава 7. Знакомство с моделью маршрутов данных и библиотекой RxCpp

Парадигма маршрутов данных.....	175
Знакомство с библиотекой RxCpp	176
Библиотека RxCpp и её модель программирования	177
Простой пример взаимодействия источника с наблюдателем.....	178
Фильтрация и преобразование потоков данных.....	178

Создание потока из контейнера.....	179
Создание собственных наблюдаемых источников	179
Конкатенация потоков	180
Отписка от потока данных.....	180
Визуальное представление потоков данных	181
Операции над потоками данных.....	181
Операция average.....	182
Операция scan.....	182
Соединение операций в конвейер	183
Работа с планировщиками.....	183
Сага о двух операциях: как разглаживать потоки потоков	186
Прочие важные операции.....	191
Беглый взгляд на ещё не изученное.....	192
Итоги	193
Глава 8. Ключевые элементы библиотеки RxCpp	194
Наблюдаемые источники данных	194
Что такое объект-производитель	195
Горячие и холодные источники данных	195
Горячие источники данных	196
Горячие источники данных и механизм повтора	198
Наблюдатели и подписчики.....	199
Единство наблюдаемого и наблюдателя.....	200
Планировщики.....	203
Методы observe_on и subscribe_on.....	206
Планировщик с циклом выполнения run_loop.....	208
Операции над потоками данных.....	209
Операции создания потоков.....	210
Операции преобразования данных	210
Операции фильтрации	211
Операции комбинирования данных	212
Операции обработки ошибок	212
Вспомогательные операции	212
Логические операции.....	213
Математические операции и агрегирование потоков.....	213
Операции для управления подключениями	213
Итоги	214
Глава 9. Реактивное программирование графических интерфейсов на основе каркаса Qt	215
Введение в программирование интерфейсов пользователя на основе каркаса Qt.....	216

Объектная модель библиотеки Qt	217
Сигналы и слоты	218
Подсистема событий	220
Обработчики событий	221
Отправка событий	221
Система метаобъектов	222
Программа «Здравствуй, мир» на основе библиотеки Qt	222
События, сигналы и слоты на примере	225
Создание собственного визуального объекта	225
Создание главного диалогового окна приложения	227
Запуск приложения	231
Интеграция библиотек RxCpp и Qt	232
Реактивная фильтрация событий из каркаса Qt	233
Создание окна и размещение его элементов	235
Наблюдатели для различных типов событий	236
Знакомство с библиотекой RxQt	238
Итоги	241

Глава 10. Шаблоны и идиомы реактивного программирования на языке C++

Объектно-ориентированное программирование и шаблоны проектирования	242
Основные каталоги шаблонов	244
Шаблоны «Банды четырёх»	244
Каталог POA	245
Ещё раз о шаблонах проектирования	246
От шаблонов проектирования к реактивному программированию	248
Разглаживание иерархии и линейный проход	254
От итераторов к наблюдаемым источникам	256
Шаблон «Ячейка»	257
Шаблон «Активный объект»	260
Шаблон «Ресурс взаимны»	262
Шаблон «Шина событий»	263
Итоги	267

Глава 11. Реактивные микросервисы на языке C++

Язык C++ и веб-программирование	269
Модель программирования REST	269
Библиотека REST SDK для языка C++	270
Программирование HTTP-клиента с использованием библиотеки C++ REST SDK	270
Программирование HTTP-сервера	272
Тестирование HTTP-сервера с помощью утилит curl и postman	275

Создание HTTP-клиента с помощью библиотеки libcurl.....	276
Реактивная библиотека-обёртка RxCurl	277
Использование формата JSON с протоколом HTTP	278
Использование библиотеки C++ REST SDK для создания сервера	282
Обращение к REST-сервисам с помощью библиотеки RxCurl.....	290
Несколько слов об архитектуре реактивных микросервисов.....	292
Мелкоблочные сервисы.....	293
Разнородное хранение данных	294
Независимое развёртывание сервисов	294
Оркестровка и хореография сервисов.....	295
Реактивный стиль запросов к веб-сервисам	295
Итоги	295

Глава 12. Особые возможности потоков и обработка

ошибок.....	297
Средства обработки ошибок в библиотеке RxCpp.....	300
Выполнение действия в ответ на ошибку.....	300
Восстановление после ошибки	302
Обработка ошибки путём перезапуска источника данных.....	305
Автоматическое выполнение завершающих действий в случае ошибки	307
Обработка ошибок и планировщики	308
Примеры обработки потоков событий	313
Агрегирование потоков данных.....	313
Событийно-управляемое приложение	315
Итоги	319

Над книгой работали

АВТОРЫ

Прасид Пай работает в индустрии программного обеспечения на протяжении 25 лет. Начинал с системного программирования в среде MS DOS на языке ANSI C. Принимал активное участие в разработке крупных кроссплатформенных систем на языке C++ для систем Windows, GNU Linux и macOS. Обладает опытом применения технологий COM+ и CORBA на языке C++. В последнее десятилетие работает с языком Java и платформой .Net.

Выступил основным разработчиком компилятора с языка SLANG4.net, первоначально написанного на языке C#, а затем портированного на язык C++ с использованием системы LLVM. Соавтор книги «.NET Design Patterns» («Шаблоны проектирования для платформы .NET»), вышедшей в издательстве Packt Publishing.

Питер Абрахам стал приверженцем языка программирования C++ и пылким борцом за производительность кода со времён своей учёбы в колледже, где он достиг высот в программировании для операционных систем Windows и GNU Linux. Он обогатил свой опыт программированием для архитектуры CUDA, обработкой изображений, компьютерной графикой, работая в таких компаниях, как Quest Global, Siemens и Tektronics.

В своей профессиональной деятельности Питер постоянно использует последние нововведения языка C++ и библиотеку RxCpp. Он обладает большим опытом работы с инструментами разработки кроссплатформенных графических приложений, такими как Qt, WxWidgets и FOX toolkit.

РЕЦЕНЗЕНТ

Сумант Тамбе – разработчик программного обеспечения, исследователь, участник разработки с открытым кодом, блогер, докладчик на различных конференциях, автор многочисленных статей и любитель компьютерных игр. Опытен в применении современного языка C++, брокера сообщений Kafka, различных служб распространения данных, методологии реактивного программирования и потоков данных для решения новых задач, возникающих в области больших данных и интернета вещей.

Автор блога «C++ Truths» («Истины о языке C++») и вики-книги «More C++ Idioms» («Ещё идиомы программирования на языке C++»). Делится своими знаниями в блоге, на конференциях, семинарах и встречах профессионального сообщества. Удостаивался звания Microsoft MVP в области технологий разработки программ на протяжении пяти лет. Имеет докторскую степень по компьютерным наукам в университете Вандербильта.

Предисловие

Эта книга поможет читателю овладеть парадигмой реактивного программирования на языке C++ и создавать асинхронные и многопоточные приложения. Книга включает в себя задачи, взятые из реальной практики, которые читателю предстоит решать с помощью реактивной модели программирования. Здесь освещен долгий путь становления средств обработки событий. Читатель узнает о поддержке параллельного программирования в языке C++ и о функциональном реактивном программировании. Описанные в книге конструкции на базе объектно-ориентированного и функционального программирования позволят читателю создавать эффективные программы. Наконец, читатель узнает о программировании микросервисов на языке C++ и научится создавать собственные операции для библиотеки RxCpp.

Для кого предназначена эта книга

Разработчик, программирующий на языке C++ и интересующийся применением реактивного программирования для создания асинхронных и параллельных приложений, найдёт эту книгу чрезвычайно интересной. От читателя не требуется наличие каких-либо знаний реактивного программирования.

Что охватывает эта книга

В главе 1 «Модель реактивного программирования – обзор и история» вводятся некоторые структуры данных, ключевые для реактивной модели программирования (для краткости – Rx). В ней речь идёт также об обработке событий в графических интерфейсах пользователя, дан общий обзор реактивного программирования, рассказано о реализации графических версий различных видов интерфейса на основе библиотеки классов MFC.

В главе 2 «Современный язык C++ и его ключевые идиомы» рассказано о тонкостях языка C++, о правилах вывода типов, шаблонах с переменным числом аргументов, ссылках rvalue и семантике перемещения, лямбда-функциях, основах функционального программирования, соединении операций в конвейер, а также о том, как реализовать итератор и наблюдателя.

Глава 3 «Параллельное и многопоточное программирование на языке C++» содержит сведения о средствах многопоточного программирования, включённых в стандарт языка C++. Читатель узнает, как запустить поток и управлять им, а также о различных тонкостях стандартной библиотеки, связанных с этим. Эта глава представляет собой хорошее введение в средства поддержки многопоточности, появившиеся в новом стандарте языка C++.

В главе 4 «Асинхронное программирование и неблокирующая синхронизация в языке C++» рассказано о средствах, предоставляемых стандартной библиотекой для организации параллельных вычислений на основе задач. Также в ней говорится о появившейся в современном языке C++ модели памяти для многопоточного программирования.

В главе 5 «Знакомство с наблюдаемыми источниками» говорится об одном из шаблонов проектирования «Банды четырёх» – шаблоне «Наблюдатель» и о его недостатках. Читатель узнает о шаблонах проектирования «Компоновщик» и «Посетитель» в контексте моделирования дерева синтаксического разбора выражения.

В главе 6 «Введение в программирование потоков событий на языке C++» внимание сосредоточено на программировании потоков событий. Также рассматривается библиотека `Streamulus`, в которой реализован подход к обработке потоков событий на основе встраиваемых предметно-ориентированных языков (англ. `Domain-Specific Embedded Language, DSEL`). Изложение сопровождается рядом примеров программ.

Глава 7 «Знакомство с моделью потоков данных и библиотекой `RxCpp`» открывается общим обзором вычислительной парадигмы на основе потоков данных, затем показаны основы создания программ с помощью библиотеки `RxCpp`. Читатель изучит набор операций, предоставляемых этой библиотекой.

Глава 8 «Ключевые элементы библиотеки `RxCpp`» даёт представление о том, как средства библиотеки `RxCpp` взаимодействуют между собой. Глава открывается разбором объектов наблюдения (`Observable`), далее описаны механизм подписки и реализация расписания.

Глава 9 «Реактивное программирование графических интерфейсов с помощью библиотеки `Qt`» посвящена применению парадигмы реактивного программирования для создания графических интерфейсов пользователя на основе библиотеки `Qt`. Читатель узнает о концептуальной основе библиотеки `Qt`, об иерархии её классов, системе метаобъектов, а также о механизме сигналов и слотов. В качестве примера создаётся приложение, обрабатывающее события от мыши и фильтрующее их. Затем читателю предстоит знакомство с более сложной темой – как создавать собственные реактивные операции средствами библиотеки `RxCpp`, если имеющихся в ней операций не хватает для той или иной задачи. Знание этих аспектов также поможет понять, как создавать композитные операции, состыковывая уже имеющиеся. Последняя тема в данной книге не освещается, подробную информацию можно найти по адресу https://www.packtpub.com/sites/default/files/downloads/Creating_Custom_Operators_in_RxCpp.pdf.

Глава 10 «Шаблоны и идиомы для реактивного программирования на языке C++» погружает читателя в чудесный мир шаблонов проектирования и идиом языка программирования. Изложение начинается с шаблонов «Банды четырёх», затем разобраны шаблоны, специфические для реактивного программирования.

В главе 11 «Реактивные микросервисы на языке C++» показано, как реактивная модель программирования может использоваться для создания реактив-

ных микросервисов на языке C++. Читатель узнает о наборе средств разработки Microsoft C++ REST SDK и связанной с ним модели программирования.

Глава 12 «Особые возможности потоков и обработка ошибок» посвящена средствам обработки ошибок из библиотеки RxCpp, а также некоторым сложным конструкциям и операциям для обработки потоков событий. Рассказано, как продолжать поток событий после возникновения ошибки, как ожидать, пока источник потока исправляет ошибочное состояние, и возобновлять обработку и как работают обобщённые операции, способные обрабатывать как ошибочные, так и нормальные ситуации.

КАК ИЗВЛЕЧЬ ИЗ ЭТОЙ КНИГИ МАКСИМУМ ПОЛЬЗЫ

Для понимания большинства вопросов, рассматриваемых в этой книге, читателю необходимо владеть программированием на языке C++.

ЗАГРУЗКА ИСХОДНОГО КОДА ПРИМЕРОВ

Файлы с исходным кодом примеров программ можно найти на сайте www.packtpub.com. Покупатель этой книги может посетить страницу www.packtpub.com/support, зарегистрироваться и получить файлы по электронной почте на свой адрес.

Для загрузки файлов нужно выполнить следующие шаги.

1. Войти или зарегистрироваться на сайте www.packtpub.com.
1. Перейти по ссылке **Support**.
2. Щёлкнуть по ссылке **Code Downloads & Errata**.
3. Ввести английское название книги в поле **Search** и выполнить дальнейшие инструкции на сайте.

Когда файлы получены, нужно распаковать их с помощью последних версий архиваторов:

- WinRAR или 7-Zip для ОС Windows;
- Zipreg, iZip или UnRarX для системы Mac;
- 7-Zip или PeaZip для системы Linux.

Исходный код всех примеров к этой книге также можно найти в системе GitHub по адресу <https://github.com/PacktPublishing/Cpp-Reactive-Programming>. Возможные обновления этих примеров будут публиковаться в этом же репозитории.

По адресу <https://github.com/PacktPublishing/> можно также найти обширные репозитории с примерами и к другим книгам и видеолекциям. Рекомендуем читателю ознакомиться с ними.

КАК ЗАГРУЗИТЬ ЦВЕТНЫЕ ИЛЛЮСТРАЦИИ

Издательство также предоставляет файл в формате PDF, в котором собраны цветные рисунки, снимки экрана, диаграммы и другие иллюстрации к этой

книге. Их можно загрузить по адресу https://www.packtpub.com/sites/default/files/downloads/CppReactiveProgramming_ColorImages.pdf.

ПРИНЯТЫЕ В ТЕКСТЕ СОГЛАШЕНИЯ

На протяжении всей книги текст оформлен с использованием следующих соглашений.

Код_в_тексте: таким шрифтом набраны размещённые в основном тексте элементы исходного кода, имена таблиц баз данных, имена директорий и файлов, расширения имён файлов, пути к файлам, адреса сетевых ресурсов, пользовательский ввод, идентификаторы пользователей в сети Twitter и т. д. Пример использования: «Приведённый выше фрагмент кода инициализирует объект структурного типа `WNDCLASS` (или `WNDCLASSEX` на новых системах) нужным шаблоном окна».

Самостоятельные фрагменты кода, вынесенные в отдельный абзац, оформлены следующим образом:

```
/* закрыть соединение с сервером */
XCloseDisplay(display);

return 0;
}
```

Если нужно обратить внимание читателя на определённую часть кода, она набирается полужирным шрифтом.

```
/* закрыть соединение с сервером */
XCloseDisplay(display);

return 0;
}
```

Всё, что вводится или выводится в консоли, передано в книге следующим образом:

```
$ mkdir css
$ cd css
```

Полужирным начертанием отмечены новые термины, важные слова и текст, отображаемый программой на экране. Например, таким способом показаны пункты меню и сообщения в диалоговых окнах. Пример использования: «На жаргоне оконного программирования это называется циклом обработки **сообщений**».



Так будут оформляться предупреждения и важные примечания.



Так будут оформляться советы или рекомендации.

ОТЗЫВЫ И ПОЖЕЛАНИЯ

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу www.dmkpress.com.

СПИСОК ОПЕЧАТОК

Хотя мы приняли все возможные меры, для того чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу www.dmkpress.com, и мы исправим это в следующих тиражах.

НАРУШЕНИЕ АВТОРСКИХ ПРАВ

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты www.dmkpress.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Модель реактивного программирования – обзор и история

Появление систем X Window System, Microsoft Windows и IBM OS/2 Presentation Manager сделало популярным программирование графических интерфейсов для платформы PC. Это стало большим шагом вперёд по сравнению с преобладавшим ранее интерфейсом командной строки и подходом к программированию, ориентированным на пакетную обработку. Реагирование на событие оказалось в центре внимания программистов по всему миру, тогда как разработчики платформ принялись за создание прикладных программных интерфейсов, с помощью которых программисты могли бы обрабатывать события, – низкоуровневых, в духе языка C, использующих указатели на функции обратного вызова. Модели программирования при этом основывались в основном на модели кооперирующихся потоков, а по мере появления более совершенных микропроцессоров большинство платформ стало поддерживать также и вытесняющую многопоточность. Обработка событий (как и другие асинхронные задачи) становилась всё сложнее, и традиционные подходы к реагированию программ на события всё менее поддавались масштабированию. Несмотря на то что появлялись превосходные инструменты для создания графических интерфейсов, основанные на языке C++, для обработки событий по-прежнему использовались числовые коды сообщений, диспетчеризация через указатели на функции и другие низкоуровневые технологии. Ведущие разработчики компиляторов даже пытались добавлять свои расширения в язык C++, чтобы облегчить программирование в среде Windows. Обработка событий, асинхронные вычисления и другие подобные задачи требовали новых подходов. К счастью, современный стандарт языка C++ поддерживает функциональную парадигму программирования, содержит средства управления потоками вместе с подходящей моделью памяти и улучшенные средства управления памятью, что позволяет программистам работать с асинхронными потоками данных, в частности трактовать события как потоки. Всё это достигается благодаря модели

программирования, называемой реактивным программированием. Чтобы показать общую картину явления, осветим в этой главе следующие вопросы:

- событийно-ориентированная модель программирования и её реализации на различных платформах;
- что собой представляет реактивное программирование;
- различные модели реактивного программирования;
- разбор нескольких простых программ для закрепления понимания основных понятий;
- методология, принятая в данной книге.

СОБЫТИЙНО-ОРИЕНТИРОВАННАЯ МОДЕЛЬ ПРОГРАММИРОВАНИЯ

Событийно-ориентированное программирование – это такая модель программирования, в которой ход выполнения программы определяется событиями. Примерами событий могут быть нажатия на кнопку мыши, нажатия клавиш на клавиатуре, жесты на сенсорном экране, сигналы от датчиков, сообщения от других программ и т. д. Событийно-ориентированное приложение основано на механизмах, позволяющих обнаружить события в реальном масштабе времени (или близко к тому) и отвечать, реагировать на них, вызывая подходящие процедуры – обработчики событий. Поскольку большинство ранних программ, обрабатывающих события, было написано на языках C и C++, для организации обработчиков в них применялись низкоуровневые технологии наподобие указателей на функции обратного вызова. Более поздние системы, такие как Visual Basic, Delphi и другие среды быстрой разработки приложений, содержали уже встроенные средства событийно-ориентированного программирования. Чтобы отчетливее показать предмет, сделаем краткий обзор механизмов обработки событий в нескольких различных платформах. Это поможет читателю лучше понять круг проблем, для решения которых предназначены реактивные модели программирования (в контексте разработки графических интерфейсов пользователя).

i В реактивном программировании данные рассматриваются как потоки, а события в системах оконного интерфейса могут рассматриваться как потоки, разнородные элементы которых должны обрабатываться единообразно. Реактивная модель программирования предоставляет средства для сбора событий из разных источников в поток, фильтрации потоков, различных преобразований над потоками, выполнения тех или иных действий над элементами потоков и т. д. Эта модель программирования содержит в своей основе средства асинхронной обработки и управление расписанием асинхронных действий. В этой главе в основном рассматриваются ключевые структуры данных, характерные для реактивного программирования, и то, как создавать простейшие реактивные программы. Реальным реактивным программам внутренне присущ асинхронный принцип работы, тогда как примеры из этой главы работают синхронно. Прежде чем переходить к асинхронному порядку выполнения и управлению расписанием, предстоит сперва объяснить ряд теоретических принципов и соответствующих языковых конструкций, это будет сделано в следующих главах. Примеры из данной главы служат только для первоначального знакомства с предметом и представляют лишь учебный интерес.

Событийно-ориентированное программирование в системе X Window

Система X Window представляет собой кроссплатформенный интерфейс прикладного программирования, поддерживается главным образом в системах, отвечающих стандарту POSIX, а также перенесена в систему Microsoft Windows. Фактически программный интерфейс X представляет собой сетевой протокол оконного графического ввода-вывода, которому требуется оконный менеджер, управляющий совокупностью окон. Клиентское приложение формирует графические образы, а X-сервер отвечает за их отображение на экране конкретной машины. В персональных настольных системах, как правило, графический клиент и сервер работают локально, на одной и той же машине. Следующая программа поможет читателю понять дух модели программирования, лежащей в основе библиотеки XLib, и присущий ей способ обработки событий.

```
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main(void)
{
    Display *display;
    Window window;
    XEvent event;
    char *msg = "Hello, World!";
    int s;
```

В этом фрагменте кода подключены необходимые заголовочные файлы, в которых содержатся прототипы функций, предоставляемых библиотекой XLib для языка C. Создавая программы на основе одной лишь библиотеки XLib, программисту нужно иметь дело с некоторыми специфическими структурами данных. В наше время, впрочем, для создания коммерческих программных продуктов чаще всего используют такие высокоуровневые библиотеки-обёртки, как Qt, WxWidgets, Gtk+ или Fox.

```
/* open connection with the server */
display = XOpenDisplay(NULL);
if (display == NULL){
    fprintf(stderr, "Cannot open display\n");
    exit(1);
}
s = DefaultScreen(display);
/* create window */
window = XCreateSimpleWindow(display,
    RootWindow(display, s), 10, 10, 200, 200, 1,
    BlackPixel(display, s), WhitePixel(display, s));

/* select kind of events we are interested in */
```

```
XSelectInput(display, window, ExposureMask | KeyPressMask);
```

```
/* map (show) the window */
XMapWindow(display, window);
```

В этом фрагменте кода инициализируется соединение с графическим сервером, затем создаётся окно с заданными параметрами. Как правило, программы в среде X Window работают под управлением оконного менеджера, который управляет взаимным расположением окон. Мы выбрали интересные нашей программе типы сообщений, вызвав функцию `XSelectInput`, перед тем как отобразить окно.

```
/* event loop */
for (;;)
{
    XNextEvent(display, &event);

    /* draw or redraw the window */
    if (event.type == Expose)
    {
        XFillRectangle(display, window,
            DefaultGC(display, s), 20, 20, 10, 10);
        XDrawString(display, window,
            DefaultGC(display, s), 50, 50, msg, strlen(msg));
    }
    /* exit on key press */
    if (event.type == KeyPress)
        break;
}
```

Затем программа входит в бесконечный цикл, в котором запрашивает очередное событие, а соответствующая функция из библиотеки `XLib` отображает в окне текстовую строку. На жаргоне оконного программирования это называется циклом обработки **сообщений**. Для получения очередного события используется функция `XNextEvent`.

```
/* close connection to server */
```

```
XCloseDisplay(display);
return 0;
```

```
}
```

Покинув «бесконечный» цикл обработки сообщений, программа закрывает соединение с графическим сервером.

Событийно-ориентированное программирование в среде Microsoft Windows

Корпорация Microsoft разработала модель программирования графических интерфейсов пользователя, которую можно считать наиболее успешной оконной системой в мире. Третья версия системы Windows имела ошеломительный

успех в 1990 г., и фирма Microsoft продолжила его развивать в версиях Windows NT, Windows 95, 98, ME. Рассмотрим в общих чертах модель событийно-ориентированного программирования в системе Microsoft Windows (за подробной информацией о том, как работает эта модель, можно обратиться к документации фирмы Microsoft). Следующий пример поможет понять суть программирования в среде Windows на языках С и С++.

```
#include <windows.h>
//----- Prtotype for the Event Handler Function
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
    WPARAM wParam, LPARAM lParam);
//----- Entry point for a Idiomatic Windows API function
int WINAPI WinMain(HINSTANCE hInstance,
    HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    MSG msg = { 0 };
    WNDCLASS wc = { 0 };
    wc.lpfWndProc = WndProc;
    wc.hInstance = hInstance;
    wc.hbrBackground = (HBRUSH)(COLOR_BACKGROUND);
    wc.lpszClassName = "minwindowsapp";
    if (!RegisterClass(&wc))
        return 1;
}
```

Приведённый выше фрагмент кода инициализирует объект структурного типа WNDCLASS (или WNDCLASSEX на новых системах) нужным шаблоном окна. Самое важное в этой структуре – это поле lpfWndProc, в нём находится адрес функции, посредством которой экземпляр окна отвечает на события.

```
if (!CreateWindow(wc.lpszClassName,
    "Minimal Windows Application",
    WS_OVERLAPPEDWINDOW | WS_VISIBLE,
    0, 0, 640, 480, 0, 0, hInstance, NULL))
    return 2;
```

Вызов функции CreateWindow (или CreateWindowEx на новых версиях ОС Windows) создаёт окно на основе оконного класса с именем, взятым из параметра WNDCLASS.lpszClassname.

```
while (GetMessage(&msg, NULL, 0, 0) > 0)
    DispatchMessage(&msg);
return 0;
}
```

Этот блок кода запускает цикл, в который берёт из очереди новые сообщения до тех пор, пока не будет получено сообщение WM_QUIT. Сообщение WM_QUIT завершает цикл. В некоторых случаях их необходимо также подвергнуть некоторой предварительной обработке, перед тем как передавать их функции DispatchMessage. Наконец, системная функция DispatchMessage вызывает оконную функцию обратного вызова, адрес которой был ранее передан через поле lpfWndProc.

```

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
    WPARAM wParam, LPARAM lParam) {
    switch (message) {
    case WM_CLOSE:
        PostQuitMessage(0); break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

```

Показанный выше фрагмент кода представляет собой минимальную функцию обратного вызова (в англоязычной литературе – callback). Обратившись к документации фирмы Microsoft, можно узнать больше о прикладном интерфейсе программирования в среде Windows и о том, как события обрабатываются в программах.

Событийно-ориентированное программирование в каркасе Qt

Каркас Qt представляет собой кроссплатформенный и многоплатформенный инструмент профессиональной разработки программ, включая разработку графических интерфейсов пользователя, который работает в средах Windows, GNU Linux, macOS X и в других системах семейства Mac. Этот инструмент поддерживает также встроенные системы и мобильные устройства. Модель программирования на языке C++ в этом каркасе основана на использовании специального инструмента, называемого **метаобъектным компилятором** (англ. Meta Object Compiler, MOC); он просматривает исходный код в поисках директив (особых макросов и расширений языка), помещённых в исходный код, и определённым образом генерирует вспомогательный исходный код, отвечающий за обработку событий. Таким образом, перед тем как компилятор языка C++ получит исходный код, этот код должен пройти сквозь инструмент MOC, который сгенерирует код, отвечающий стандарту ANSI C++ и не содержащий языковых конструкций, специфических для системы Qt. Более подробные сведения можно почерпнуть из документации по каркасу Qt. Следующая простая программа демонстрирует ключевые моменты программирования в каркасе Qt и присущую ему логику обработки событий.

```

#include <qapplication.h>
#include <qdialog.h>
#include <qmessagebox.h>
#include <qobject.h>
#include <qpushbutton.h>

class MyApp : public QDialog {
    Q_OBJECT
public:
    MyApp(QObject* /*parent*/ = 0):
        button(this)
    {

```

```

button.setText("Hello world!");
button.resize(100, 30);
// When the button is clicked, run button_clicked
connect(&button,
        &QPushButton::clicked, this, &MyApp::button_clicked);
}

```

Макрос `Q_OBJECT` указывает метаобъектному компилятору сгенерировать таблицу диспетчеризации события (Event Dispatch). Когда источник событий подключается к приёмнику, в эту таблицу вставляется новая строка. Сгенерированный код обрабатывается компилятором наряду с исходным кодом, написанным программистом, в результате чего строится исполняемая программа.

```

public slots:
    void button_clicked() {
        QMessageBox box;
        box.setWindowTitle("Howdy");
        box.setText("You clicked the button");
        box.show();
        box.exec();
    }
protected:
    QPushButton button;
};

```

Слово `slots` как расширение языка особым образом обрабатывается метаобъектным компилятором при генерации вспомогательного кода, но для компилятора языка C++ прозрачно, так как представляет собой обычный макрос.

```

int main(int argc, char** argv) {
    QApplication app(argc, argv);
    MyApp myapp;
    myapp.show();
    return app.exec();
}

```

Этот последний фрагмент кода инициализирует объект, играющий роль обёртки над приложением, и отображает главное окно. В целом Qt можно назвать наилучшим из всех каркасов для разработки приложений, разработанных для языка C++. Кроме того, в нём имеются хорошие средства интеграции с популярным языком программирования Python.

Событийно-ориентированное программирование средствами библиотеки MFC

Библиотека классов MFC (Microsoft Foundation Classes) по сей день остаётся довольно популярным средством для создания приложений в среде Microsoft Windows. Если прибавить к ней ещё и библиотеку **ATL (ActiveX Template Library)**, можно получить также некоторую поддержку веб-программирования. Для обработки событий в библиотеке MFC используется механизм, называемый схемой сообщений, или таблицей сообщений (message map). Таблица

сообщений, оформленная с помощью специальных макросов, как показано в следующем примере, присутствует в каждой программе, созданной на основе библиотеки MFC.

```
BEGIN_MESSAGE_MAP(CCLockFrame, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_PAINT()
    ON_WM_TIMER()
END_MESSAGE_MAP()
```

Эта таблица определяет реакцию на стандартные сообщения системы Windows: сообщения WM_CREATE, WM_PAINT и WM_TIMER должны обрабатываться, соответственно, функциями OnCreate, OnPaint и OnTimer. На уровне внутренних механизмов реализации, глубоко скрытых от пользователя, эти таблицы представляют собой массивы, в которых целочисленный код сообщения используется для поиска нужной строки, содержащей указатель на функцию-обработчик. Таким образом, отличие данного подхода от модели обработки сообщений на основе системных вызовов Windows оказывается при внимательном рассмотрении не слишком значительным.



Мы не приводим здесь пример исходного кода, поскольку среди доступных для скачивания примеров программ к этой книге имеется полная реализация одного характерного графического приложения в духе модели реактивного программирования на основе библиотеки MFC. Читатель может изучить исходный код и комментарии к нему, чтобы разобраться в неочевидных аспектах обработки событий в библиотеке MFC.

Прочие модели событийно-управляемого программирования

Системы распределённой обработки объектов, такие как COM+ или CORBA, обладают собственными подсистемами обработки событий. Модель событий, принятая в технологии COM+, основана на понятии «точки соединения» (англ. connection point), представленном интерфейсами IConnectionPointContainer и IConnectionPoint, а в технологии CORBA реализована модель на основе так называемого сервиса событий¹ (event service). Спецификация CORBA поддерживает оба механизма оповещения о событиях: «втягивание» и «вталкивание». Разбор технологий COM+ и CORBA выходит за рамки этой книги, читатель может обратиться к соответствующей документации.

Ограничения классических моделей обработки событий

Цель этого краткого обзора моделей обработки событий, присущих различным платформам, состоит в том, чтобы показать предмет с нужной точки зрения.

¹ Следует отметить, что сервис событий в технологии CORBA подвергается критике из-за ряда серьёзных недостатков (схема доставки сообщений, при которой каждый клиент получает сообщения обо всех событиях от сервера, перегружает сеть; отсутствует фильтрация сообщений по каким-либо критериям, синхронный режим отправки сообщений) – см, например, http://www.k-press.ru/cs/2000/3/corba/corba_callback.asp. – Прим. перев.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru