УДК 004Dart
ББК 32.973-018.2
Б19

Баккет К.
Б19 Dart в действии. – М.: ДМК Пресс, 2013. – 528 с.: ил.

ISBN 978-5-94074-918-9

Dart – язык программирования для разработки веб-приложений, созданный компанией Google. Он обладает современными объектно-ориентированными средствами, как Java или C#, не отказываясь при этом от свойственных JavaScript динамичности и ориентированности на функциональное программирование. Написанные на Dart приложения транслируются в JavaScript, но могут исполняться и непосредственно в браузерах, поддерживающих Dart. В комплекте с Dart поставляются библиотеки и инструментальные средства промышленного качества. На Dart могут быть написаны как клиентская, так и серверная часть приложения, что упрощает процесс разработки.

В этой книге вы познакомитесь с языком Dart и научитесь использовать его для создания приложений, работающих в браузере, — в настольной или мобильной ОС. Это не просто учебное пособие по языку, довольно быстро автор переходит к техническим аспектам работы с Dart. На большинство вопросов, возникающих при чтении, тут же даются ответы!

Издание предназначено веб-программистам разной квалификации, в том числе малознакомым с объектно-ориентированным программированием.

УДК 004Dart
ББК 32.973-018.2

ISBN 978-1-6172-9086-2 (анг.)

ISBN 978-5-94074-918-9 (рус.)

Содержание

Предисловие

Услышав, что мы приступаем к работе над Dart – структурным, масштабируемым языком с быстрой виртуальной машиной, развитым редактором и компилятором на JavaScript, я поначалу не поверил. «Быть может, именно этот проект сделает веб-программирование проще для таких разработчиков, как я сам?» – задавался я вопросом в надежде получить утвердительный ответ. Имея опыт работы со структурными языками и мощными инструментальными средствами разработки, я ожидал какого-нибудь более продуктивного способа создавать современные крупномасштабные веб-приложения. Проект Dart казался как раз тем, что я искал. Я вырос на таких объектно-ориентированных языках, как C++, Java и Ruby, которые применял при построении своих первых интерактивных веб-сайтов, а позже обогащенных веб-приложений с развитой клиентской частью. Я научился продуктивно работать с классами, объектами и модульным кодом. Я ценил интегрированные среды разработки (IDE) за встроенные в них средства анализа, рефакторинга и навигации, потому что они помогали мне писать большие и сложные приложения. Жизнь была прекрасна. Я искал новые направления приложения своих знаний, и мне повезло получить работу в команде, работавшей над браузером Chrome. Поначалу я осваивал современный браузер и с воодушевлением принялся изучать многочисленные новые возможности HTML5. Ныне сеть веб развивается так быстро и у нее появилось столько пользователей, что находиться в центре событий безумно интересно. Жизнь стала еще прекрасней.

И хотя итеративный процесс веб-разработки с его коротким циклом мне очень нравился, все же не хватало столь любимых мной структурных языков и полезных инструментов. Я хотел, чтобы программы для современных браузеров можно было писать в IDE, умеющих выполнять автозавершение кода, на языках с настоящими классами... перечислять можно долго.

Поэтому, услышав о Dart, я ухватился за представившуюся возможность. Писать для самой интересной платформы, да еще не от-

казываясь от навыков и инструментов, с которыми хорошо знаком и умеешь работать? Да кто же будет против!

И я был не единственным разработчиком, кто с радостью присоединился к проекту. Автор этой книги Крис Бакетт был одним из первых, кто принял Dart. Он завел свой блог Dartwatch в тот самый день, как Google анонсировала Dart, и этот блок по сей день живет и процветает. Крис участвовал в проекте с самого начала, поэтому естественно, что именно он написал одну из первых книг, призванных помочь другим разработчикам в изучении Dart.

Крис – потрясающий автор, потому что сумел написать книгу в условиях, когда язык и его библиотеки претерпевали постоянные изменения. Ему удалось осветить многочисленные аспекты проекта Dart. Особенно мне понравились примеры, в которых иллюстрируется не только функциональность базового языка, но и более сложные вопросы, касающиеся HTML5. Крис рассматривает одностраничные приложения и показывает, как с помощью Dart создаются современные приложения, работающие в браузере. Но и это не все – вы научитесь программировать на Dart даже серверные приложения!

После года напряженной работы, десятков тысяч сохранений в системе управления версиями, исправления тысяч ошибок и непрерывающейся обратной связи с сообществом мечта о структурном языке для веб-программирования стала реальностью. И хотя разработка Dart еще не завершена, сегодня мы – благодаря книге Криса – уже можем создавать замечательные приложения для современного Интернета. Ликуйте!

Сет Лэдд,
разработчик и пропагандист
Google

Вступление

В октябре 2011 года подтвердились слухи о том, что Google разработала новый язык, ориентированный на разработку сложных веб-приложений характерного для Google масштаба. Спустя месяц в сети был обнародован внутренний документ Google под названием «Future of JavaScript» (Будущее JavaScript), из которого следовало, что в Google ведутся работы над языком, более подходящим для веб-разработки, чем JavaScript. Упоминалось даже предположительное название языка – Dash. Идея о создании нового языка родилась из-за медленного развития JavaScript, обусловленного отчасти тем, что в нем заинтересовано слишком много сторон и комитетов. Целью же было показать, каким мог бы стать JavaScript, если бы он был придуман сегодня. Основная задача формулировалась так: «сохранить динамическую природу JavaScript, но повысить производительность и обеспечить поддержку со стороны инструментальных средств при разработке крупных проектов». Кроме того, язык должен был допускать кросс-компиляцию на JavaScript. Апробационная техническая версия языка была представлена миру и получила название Dart.

Я как раз тогда закончил работу над крупным заказным справочным приложением, написанным на GWT и предназначенным для развертывания на устройствах с сенсорными экранами в среде, неблагоприятной для компьютеров. Технология Google Web Toolkit (GWT) была разработана Google для кросс-компиляции Java на JavaScript. GWT позволяет разработчику воспользоваться модульной структурой, типобезопасностью и инструментальной поддержкой Java при написании ориентированных на браузеры программ, не требующих подключения дополнительных модулей, таких как Flash или Silverlight. Потратив предшествующие два года на написание GWT-кода и координацию работы программистов из трех стран, я понимал, насколько ценны средства проверки кода в точках интеграции – то, чего так не хватает языку JavaScript. Меня также привлекала возможность использовать один и тот же код на сто-

(Steve Pretty), Терри Бэрч (Terry Birch) и Вильгельм Леман (Willhelm Lehman).

Также спасибо всем, кто писал в форуме книги, указывая на неизбежные опечатки, и всем читателям предварительных вариантов – участникам программы Manning's Early Access Program (MEAP).

Наконец, спасибо всем разработчикам Dart, в том числе Сету Лэду, который помогал мне и многим другим ранним приверженцам оставаться в курсе изменений по мере эволюции Dart от первоначальной версии к той, что мы имеем сегодня. Отдельная благодарность Сету за любезное согласие написать предисловие к этой книге.

Об этой книге

Данная книга призвана помочь вам изучить язык Dart, понять его экосистему и писать на нем код, работающий в современных браузерах и на сервере. Вы будете использовать последние технологии HTML5, которые позволяют создавать приложения, работающие в браузере без подключения к сети, а также писать на Dart серверный код, поддерживающий двустороннюю связь с браузерами.

Будучи структурным языком, Dart идеален для разработки крупномасштабных приложений силами территориально разнесенной команды. А благодаря наличию инструментов для автоматической проверки кода, написанного всеми членами команды, Dart облегчает жизнь разработчикам.

Предполагаемая аудитория

Эта книга рассчитана на разработчиков, недовольных отсутствием должной языковой структуры и инструментальных средств для создания браузерных приложений. Если вы на рабочем уровне владеете Java, C# или JavaScript, то сможете сразу же приступить к работе с Dart.

И программисты, предпочитающие писать интерактивные пользовательские интерфейсы, и те, кому больше по душе разработка эффективного серверного кода, обнаружат, что Dart в сочетании с современными браузерными технологиями привносит на сторону клиента структуру, характерную для сервера, а на сторону сервера – гибкость, динамичность и скорость разработки, присущую клиентским приложениям.

Эта книга поможет быстро освоить идеи языка Dart как начинающему веб-разработчику, так и поднаторевшему в написании структурного кода. Все новые концепции поясняются на примерах. В тексте отмечаются как черты, роднящие Dart с другими языками, в частности Java и JavaScript, так и различия между ними.

Для Dart, как и для Java, имеются великолепные инструментальные средства. С другой стороны, Dart, как и JavaScript, не требует этапа компиляции, а это значит, что вы сможете очень быстро приступить к созданию клиентских и серверных приложений на Dart.

Структура книги

Эта книга построена так, чтобы читатель как можно скорее начал работать с Dart. Она состоит их четырех частей. Первая часть содержит обзорные главы.

- В главе 1 приводятся общие сведения об идеях и средствах языка и рассказывается о том, почему вообще Dart появился. Вы узнаете о философии, стоящей за Dart, и о том, какого рода крупномасштабные веб-приложения можно разрабатывать с его помощью.
- В главе 2 рассматривается более широкая экосистема, сложившаяся вокруг Dart, в том числе богатейшие инструментальные средства, которые вы получаете, выбирая структурный язык веб-разработки, созданный компанией, являющейся лидером на этом рынке. Обладая необходимыми техническими ресурсами, Google создала не только сам язык, но и IDE, специализированный браузер для быстрой разработки, серверную виртуальную машину и другие инструменты, нужные для создания высококачественного кода.
- В главе 3 мы напишем простенькое веб-приложение, чтобы понять, как Dart взаимодействует с браузером. Мы сконструируем пользовательский интерфейс, будем прослушивать события браузера и создадим автономные тесты, подтверждающие корректность кода.

Во второй части рассматриваются базовые языковые средства.

- Глава 4 посвящена функциям, которые в Dart являются полноценными объектами. Пишущим на JavaScript некоторые приемы функционального программирования покажутся знакомыми, тогда как разработчики на Java и C# найдут много новых для себя идей, типичных для браузерных приложений.
- В главе 5 мы продолжим конструировать приложение, воспользовавшись встроенной в Dart системой библиотек, и покажем, как эта система соотносится с ограничением доступа. Механизм ограничения доступа в Dart, возможно, станет сюрпризом для разработчиков на Java и C# и окажется желанным подарком для программистов на JavaScript.

- В главах 6, 7 и 8 исследуется структура классов и интерфейсов в Dart. Классы образуют костяк любого сколько-нибудь масштабного приложения, поэтому без умения строить эффективные иерархии классов и пользоваться библиотечными классами, написанными другими людьми, никак не обойтись.
- В главе 9 мы вернемся к функциональному программированию и разберемся с асинхронной природой API доступа к веб. Вы узнаете, как работать с будущими значениями, то есть с переменными, которые получат значение в будущем. Это подготовит почву для работы с API, предоставляемыми клиентскими и серверными библиотеками Dart.

В третьей части обсуждается разработка клиентских приложений, работающих в браузере.

- В главе 10 вы узнаете о цикле обработки событий и о создании пользовательских интерфейсов в Dart.
- В главе 11 в структуру приложения добавляются браузерная навигация, сохранение данных на стороне клиента и обработка данных, представленных в формате JSON.
- Добравшись до главы 12, вы уже будете готовы подключить свое приложение к внешним системам, например к внешнему JavaScript-коду и сторонним серверным API. Хотя Dart рассчитан на все современные браузеры, в этой главе мы научимся создавать пакет для развертывания в качестве приложения Chrome в магазине Google Web Store.

В части 4 речь пойдет об интерфейсе с серверным кодом.

- В главе 13 мы напишем на Dart командное приложение, обращающееся к файловой системе и отправляющее данные по протоколу HTTP, то есть разработаем простой файловый сервер.
- В главе 14 мы подключим клиентское приложение к серверной базе данных и организуем двустороннее взаимодействие по технологии Web Sockets для проталкивания данных клиенту.
- В главе 15, уже зная о взаимодействии с сервером, вы сможете понять, как Dart решает проблему параллелизма с помощью системы изоляторов – модели многопоточности на основе передачи сообщений, более безопасной, чем эквивалентные механизмы в Java и C#. Мы также воспользуемся системой изоляторов для динамической загрузки кода на Dart в работающее приложение. Тем самым мы заложим фундамент для разработки расширений и подключаемых модулей.

В приложениях содержится краткое справочное руководство по базовому языку Dart с примерами, иллюстрирующими синтаксические особенности и странности Dart.

Графические выделения и загрузка исходного кода

Весь исходный код в тексте выделяется моноширинным шрифтом. В книге много фрагментов кода и диаграмм, которые представляют собой законченные аннотированные листинги, иллюстрирующие основные идеи. Все это, как правило, тесно связано с окружающим текстом и составляет неотъемлемую часть изучения Dart.

Иногда код приходилось переформатировать, чтобы он поместился на страницу, но в общем случае код уже написан с учетом ограничений по ширине строки. Сами примеры часто упрощены, чтобы продемонстрировать какую-то важную концепцию, не отвлекаясь на детали, но в основном тексте и в аннотациях к коду приводятся дополнительные подробности.

Весь исходный код, представленный в этой книге, можно скачать с сайта издательства по адресу www.manning.com/DartinAction.

Требования к программному обеспечению

Для работы с Dart требуется как минимум Dart SDK, который можно скачать с сайта www.dartlang.org. В состав Dart SDK включены редактор Dart Editor, специализированный браузер Dartium (необходимый для быстрой разработки на Dart) и конвертер Dart в JavaScript. Имеются версии для Windows, Mac и Linux.

Автор в сети

Приобретение книги «Dart в действии» открывает бесплатный доступ к закрытому форуму, организованному издательством Manning Publications, где вы можете оставить свои комментарии к книге, задать технические вопросы и получить помощь от автора и других пользователей. Получить доступ к форуму и подписаться на список рассылки можно на странице www.manning.com/DartinAction. Там же написано, как зайти на форум после регистрации, на какую помощь можно рассчитывать, и изложены правила поведения в форуме.

Издательство Manning обязуется предоставлять читателям площадку для общения с другими читателями и автором. Однако это не означает, что автор обязан как-то участвовать в обсуждениях; его

присутствие на форуме остается чисто добровольным (и не оплачивается). Мы советуем задавать автору какие-нибудь хитроумные вопросы, чтобы его интерес к форуму не угасал!

Форум автора в сети и архивы будут доступны на сайте издательства до тех пор, пока книга не перестанет печататься.

Об авторе

Крис Бакетт работает техническим консультантом по созданию бизнес-приложений масштаба предприятия, доступных через веб. Крис ведет популярный блог Dartwatch.com и активно участвует в списке рассылки dartlang.

ЧАСТЬ I. ВВЕДЕНИЕ В DART

Dart – замечательный язык для разработки веб-приложений. В главе 1 мы расскажем о том, зачем был создан Dart и как с его помощью решаются некоторые задачи, с которыми сталкиваются веб-разработчики. Вы узнаете о возможностях языка и о том, почему одностраничные приложения – подходящая архитектура для применения Dart.

В главе 2 мы начнем изучать развитую экосистему инструментальных средств, окружающую Dart. Dart – не просто язык, это еще и полный набор средств для разработки, включая IDE, специализированный браузер для тестирования и отладки, а также конвертер из Dart в JavaScript.

В главе 3 мы напишем на Dart простое приложение, на примере которого продемонстрируем методику создания одностраничных приложений, работающих в браузере. По ходу дела мы познакомимся с языком, в том числе с классами, функциями и переменными. К концу главы у нас будет готов проект с работоспособным пользовательским интерфейсом и автономными тестами, после чего мы сможем перейти к систематическому изучению базового языка в части II.

Глава 1. Здравствуй, Dart

В этой главе
- Основные сведения о платформе разработки на Dart.
- Взгляд на язык Dart.
- Инструментальные средства для разработки Dart-приложений.

Dart – удивительный язык, открывающий возможность создавать сложные веб-приложения быстрее и с меньшим количеством ошибок, чем раньше. В этой главе мы узнаем, как сочетаются язык Dart и его инструментальная экосистема, рассмотрим основные особенности языка и увидим, как его можно использовать для построения одностраничных веб-приложений.

1.1. Что такое Dart?

Dart – структурный язык программирования с открытым исходным кодом, предназначенный для создания сложных веб-приложений, работающих в браузере. Выполнить приложение, написанное на Dart, можно либо в браузере, который поддерживает Dart напрямую, либо предварительно откомпилировав код на JavaScript. Dart обладает знакомым программистам синтаксисом, в нем есть классы и необязательные типы, он является однопоточным. Реализована также модель параллелизма на основе изоляторов, которая допускает параллельное выполнение, – мы обсудим ее в главе 15. Помимо исполнения в браузере и конвертации в JavaScript, написанный на Dart код можно запускать из командной строки в виртуальной машине Dart, что позволяет писать клиентскую и серверную части приложения на одном языке.

Синтаксис Dart очень похож на синтаксис Java, C# и JavaScript. Это не случайно – при создании Dart ставилась задача разработать знакомый язык. Вот крохотный скрипт на Dart, состоящий из единственной функции `main`:

```
main() {                          ← Функция main() – точка входа, с которой начинается
                                    исполнение полностью загруженного скрипта
  var d = "Dart";                 ← Типизация необязательна (тип не указан)
  String w = "World";             ← Аннотация типа (задан тип String)
  print("Hello ${d} ${w}");       ← Для вывода на консоль браузера или на stdout производится
}                                   подстановка в строку
```

Этот скрипт можно внедрить в HTML-страницу с помощью тега `<script type="application/dart">` и выполнить в браузере Dartium (издание браузера Google Chrome, предназначенное для разработки на Dart). С помощью программы dart2js можно также преобразовать его в JavaScript и затем исполнить в любом современном браузере или запустить из командной строки на сервере, воспользовавшись виртуальной машиной Dart (ВМ Dart).

Но Dart – не просто язык. На рис. 1.1 изображена экосистема инструментов, включающая несколько сред исполнения, редактор, языковые утилиты и полный набор библиотек – все, что нужно для комфортной разработки сложных веб-приложений.

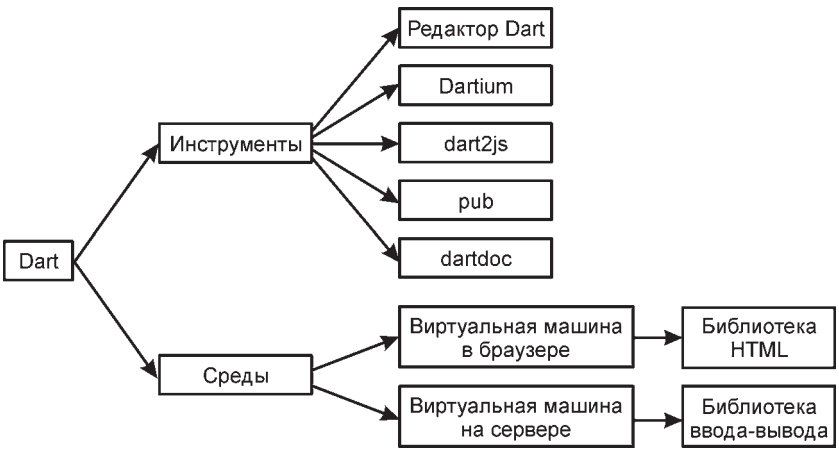

Рис. 1.1. Dart – больше, чем просто язык. Проект Dart включает целую экосистему

Помимо великолепной инструментальной экосистемы, помогающей при создании приложений, Dart спроектирован так, что покажется знакомым разработчикам как серверных приложений на Java или C#, так и клиентских – на JavaScript или ActionScript.

Основным инструментом разработчика на Dart является браузер Dartium, который позволяет писать и редактировать код и наблюдать за его исполнением. Сочетание Dartium с редактором Dart дополнительно позволяет отлаживать трафик между клиентом и сервером.

1.1.1. Знакомый синтаксис помогает в освоении языка

Одним из важнейших проектных решений было стремление сделать Dart знакомым программистам на JavaScript и на Java/C#. Это помогает быстро освоить новый язык. Если вы владеете хотя бы одним из вышеупомянутых языков, то чтение и понимание написанного на Dart кода не вызовет особых затруднений.

Разработчикам на Java и C# привычны системы типов, классы, наследование и связанные с этим идеи. С другой стороны, на JavaScript пишут как дизайнеры пользовательского интерфейса, которые просто копируют чужой код, чтобы сделать страницу интерактивной (и никогда в жизни не пользовались типами), так и профессиональные программисты, понимающие, что такое замыкание и прототипическое наследование. Чтобы удовлетворить потребности столь разнообразной аудитории, в Dart введена факультативная типизация, позволяющая либо не указывать тип нигде (объявляя переменные с помощью ключевого слова `var`, как в JavaScript), либо везде добавлять аннотации типа (`String`, `int`, `Object` и т. п.), либо применять смешанный подход.

Использование информации о типе позволяет программисту документировать свои намерения, что полезно как для автоматизированных инструментальных средств, так и для коллег. Типичный процесс разработки Dart-приложения подразумевает постепенное построение системы типов по мере того, как программа обретает форму. Добавление или удаление информации о типе не влияет на работу программы, зато позволяет виртуальной машине более эффективно проверять правильность кода. Таким образом, система типов в Dart лежит где-то посередине между динамической типизацией в JavaScript и статической в Java и C#.

В табл. 1.1 сравниваются различные характеристики Dart, Java и JavaScript.

Dart – универсальный язык, позволяющий, подобно JavaScript и Java, создавать приложения разных типов. Но по-настоящему он сверкает при разработке сложных веб-приложений.

Таблица 1.1. Сравнение некоторых характеристик Dart, Java и JavaScript

Характеристика	Dart	Java	JavaScript
Система типов	Факультативная, динамическая	Строгая, статическая	Слабая, динамическая
Функции – полноценные объекты	Да	Можно имитировать с помощью анонимных функций	Да
Замыкания	Да	Да, с помощью анонимных классов	Да
Классы	Да, одиночное наследование	Да, одиночное наследование	Прототипические
Интерфейсы	Да, множественное наследование	Да, множественное наследование	Нет
Параллелизм	Да, на основе изоляторов	Да, на основе потоков	Да, на основе рабочих процессов в HTML5

1.1.2. Архитектура одностраничного приложения

Одностраничные приложения типа Google Mail, Google Instant Search и Google Maps – именно то, для чего предназначен Dart. Исходный код всего приложения (или, по крайней мере, наиболее часто используемых его частей) загружается единственной веб-страницей. Этот – работающий в браузере – код отвечает за построение пользовательского интерфейса и загрузку с сервера отображаемых в нем данных (см. рис. 1.2).

В одностраничных приложениях используется быстрая виртуальная машина на стороне клиента, позволяющая перенести часть обработки с сервера на клиент. В результате сервер способен обслужить больше запросов, так как избавлен от необходимости верстать страницу. Благодаря включенным в Dart библиотекам для работы с HTML, поддерживающим современные HTML5-технологии хранения и кэширования данных в браузере, удается еще больше повысить производительность приложения и даже дать пользователю работать без подключения к сети.

У любого Dart-скрипта имеется единственная точка входа – функция `main()`, – с которой начинается исполнение скрипта виртуальной машиной Dart. Можно полагаться на то, что весь код, составляющий

приложение, уже загружен к моменту вызова `main`; в Dart невозможно определить и выполнить новую функцию, как в JavaScript не существует ни функции `eval()`, ни какого-либо иного способа модификации исполняемого кода. Именно эта особенность помогает писать на Dart приложения, отвечающие одностраничной архитектуре, потому что есть уверенность, что исполняемый код заранее известен. ВМ Dart пользуется этой гарантией для ускорения инициализации приложения, применяя мгновенные снимки кучи, – это позволяет загрузить Dart-приложение намного быстрее эквивалентного приложения на JavaScript.

Рис. 1.2. Одностраничное приложение работает в браузере и обращается к серверу только за данными

Памятка

- Dart – язык разработки веб-приложений со знакомым синтаксисом.
- Благодаря наличию инструментальной экосистемы разработка на Dart более продуктивна, чем на эквивалентных динамических языках.
- Факультативная система типов в Dart лежит посередине между динамической типизацией в JavaScript и статической типизацией в Java.
- Аннотации типов могут оказаться очень полезными при коллективной разработке, так как упрощают автоматизированную проверку исходного кода.
- Dart идеален для разработки одностраничных веб-приложений.

Познакомившись с Dart как с платформой разработки, можно приступить к практическому освоению некоторых важнейших особенностей языка.

1.2. Первый взгляд на язык Dart

Dart – полнофункциональный современный язык программирования. Хотя корнями он уходит в Smalltalk, в нем ощущается влияние ряда других языков, в частности Java, C# и JavaScript. В этом разделе мы познакомимся с некоторыми базовыми концепциями языка и кратко упомянем о более сложных механизмах, которые будут подробно рассмотрены впоследствии.

Dart – развивающийся язык

На момент написания этой книги Dart находится на этапе перехода от апробационной технической версии к выпускной версии, которую Google называет «Milestone 1» (Контрольная точка 1). Контрольная точка 1 – это не версия 1, а проведенная на песке черта, от которой можно плясать при разработке и совершенствовании окружающих базовый язык библиотек. Платформа Dart мыслится как полнофункциональная среда разработки типа «все включено», содержащая все необходимое для создания сложных веб-приложений. И компания Google вместе с сообществом Dart сейчас занята разработкой таких библиотек.

Кроме того, контрольная точка 1 определяет состояние, в котором уже можно приступать к созданию приложений в уверенности, что несовместимые изменения в синтаксисе языка будут появляться нечасто. Однако изменения в сопутствующих библиотеках вполне вероятны, поэтому в редактор Dart включена полезная функция наведения порядка (Clean-up), позволяющая применить к коду изменения, внесенные в базовый язык и в библиотеки.

1.2.1. Строковая интерполяция

В веб-приложениях строки используются повсеместно. Dart предлагает несколько способов преобразовать выражение в строку, в том числе функцию `toString()`, встроенную в базовый класс `Object`, и строковую интерполяцию.

Для строковой интерполяции используется знак `$` или выражение `${ }` внутри одиночных или двойных кавычек. Чтобы подставить в строку переменную, необходимо предпослать ей префикс `$`, например: `$name`. Если же требуется подставить результат вычисления выражения, в частности вызова метода, то следует воспользоваться фигурными скобками:

```
"Ответ равен ${5 + 10}"
```

Для создания строки, занимающей несколько строчек, используются три двойные кавычки, а строковые литералы (внутри которых знак $ не имеет специального смысла) создаются с помощью префикса r, например: r'literal string'. Для конкатенации двух строк оператор + не применяется; необходимо пользоваться либо строковой интерполяцией ($forename $surname), либо – если речь идет о заранее известных значениях – просто записать две строки одна за другой. Например, выражение

```
var title = "Dart " "в " "действии";
```

порождает строковую переменную со значением "Dart в действии".

В листинге ниже показаны некоторые операции со строками. Для вывода результатов используется встроенная в Dart функция print, которая печатает на стандартный выход, если выполняется на сервере, или на отладочную консоль браузера – если в браузере.

Листинг 1.1. Строковая интерполяция в Dart

```
void main() {
  var h = "Hello";
  final w = "World";
  print('$h $w');

  print(r'$h $w');

  var helloWorld = "Hello " "World";
  print(helloWorld);

  print("${helloWorld.toUpperCase()}");
  print("Ответ равен ${5 + 10}");

  var multiline = """
<div id='greeting'>
  "Hello World"
</div>""";
  print(multiline);

  var o = new Object();
  print(o.toString());
  print("$o");
}
```

$ используется для вычисления простых переменных

← При наличии префикса r выводится строковый литерал без интерполяции

Соседние строковые константы конкатенируются

Вычисляемое выражение должно быть заключено в фигурные скобки ${ }

← При записи многострочных строк первый символ новой строки после """ игнорируется

Многострочные строки могут содержать одиночные и двойные кавычки

В случае строковой интерполяции автоматически вызывается функция toString()

Эта программа печатает следующее:

```
Hello World
$h $w
Hello World
HELLO WORLD
Ответ равен 15
<div id='greeting'>
  "Hello World"
</div>
Instance of 'Object'
Instance of 'Object'
```

Во время экспериментов с Dart вы часто будете использовать строковую интерполяцию и функцию `print`, чтобы печатать переменные и подставлять значения в HTML-код.

1.2.2. Факультативные типы в действии

Одно из основных отличий JavaScript от Dart – наличие в Dart концепции типов. Однако типы факультативны, и, чтобы воспользоваться строгой типизацией, необходимо включить в код аннотации типов.

Факультативные аннотации типов могут встречаться в объявлениях переменных, параметров функций и возвращаемых значений, а также в определениях классов. Ниже показаны четыре способа объявить строковую переменную `message`. В первых двух аннотации типа нет вообще, а в двух оставшихся есть аннотация `String`, сообщающая программистам и инструментам о том, что в этой переменной будет храниться строковое значение.

```
var messageA;                          | Нет аннотаций типов
var messageB = "Hello Dart";           |

String messageC;                       | Аннотация типа присутствует
String messageD = "Hello Dart";        |
```

В двух из четырех случаев переменная `message` инициализируется в момент объявления. Если известно, что значение не изменится после объявления, то следует добавить ключевое слово `final`:

```
final messageE = "Hello Dart";          ← Использование final без аннотации типа
final String messageF = "Hello Dart";   ← Использование final с аннотацией типа
```

Подробнее о ключевом слове `final` мы будем говорить ниже.

Чтобы продемонстрировать полезность факультативной типизации, рассмотрим приведенный ниже код, в котором функция `trueIfNull()` принимает два параметра и возвращает `true`, если оба равны `null` (и `false` в противном случае). Сейчас в этом коде нет аннотаций типов, но мы объясним, как ими можно было бы воспользоваться для выражения намерений.

```
trueIfNull(a, b) {                    | Функция принимает
  return a == null && b == null;      | два значения
}

main() {
  final nums = trueIfNull(1,2);  ← Сохраняет "false" в динамической переменной nums
  final strings = trueIfNull("Hello ", null);   ← Сохраняет "true" в динамической
                                                  переменной strings
  print("$nums");          | Выводит переменные
  print("$strings");       | nums и strings на консоль
}
```

Поскольку в этом фрагменте аннотаций типов нет, то мы ничего не можем сказать о намерениях разработчика. Имя функции `trueIfNull(a,b)` позволяет предположить, что она принимает два значения типа `int` и возвращает значение типа `bool` (`true` или `false`), но, возможно, разработчик имел в виду что-то другое – например, хотел вернуть строку `"true"` вместо булевского значения. Факультативная типизация в Dart позволяет разработчику документировать свои намерения в виде информации о типах параметрах и возвращаемого значения.

```
bool trueIfNull(int a, int b) {   ← Добавлены типы параметров и возвращаемого значения
  return a == null && b == null;
}

main() {
  final bool nums = trueIfNull(1,2);                  | В объявления переменных
  final bool strings = trueIfNull("Hello ", null);    | добавлена информация о типе
  print("$nums");
  print("$strings");
}
```

Примечание. В примере выше встречается тип `bool`. В Dart, в отличие от JavaScript, существует единственный способ представить значение

истинности «ложь», а именно ключевым словом `false`. Нуль и `null` не вычисляются как «ложь».

Добавление аннотаций типа не изменяет работу Dart-приложения, а только сообщает полезную информацию, которой инструменты и ВМ могут воспользоваться для проверки кода и нахождения ошибок типизации. Любая информация о типе может помочь инструментам в проведении статического анализа кода (в редакторе или в командной утилите непрерывной интеграции), а также на этапе выполнения. Программисты, которым в будущем придется сопровождать ваш код, тоже скажут «спасибо».

Совет. Указывайте конкретные типы (например, `String`, `List` и `int`) там, где это полезно для целей документирования, например при объявлении параметров функций, возвращаемых значений и открытых членов класса. Там же, где это не так, например в теле функции, используйте ключевое слово `var` или `final` без аннотации типа. Такой подход рекомендуется в руководстве по стилю программирования на Dart, доступном на сайте www.dartlang.org. Привыкайте к смешанному стилю, поскольку именно так и предполагается писать на Dart.

Факультативная типизация лежит в основе многих механизмов Dart и многократно упоминается на страницах этой книги там, где синтаксические отличия от Java и JavaScript заслуживают пояснения.

Функции как таковые рассматриваются в главе 4.

1.2.3. Традиционная структура на основе классов

В Dart классы и интерфейсы используются традиционным для объектно-ориентированных языков способом, без каких бы то ни было сюрпризов. Поддерживается наследование одному классу и нескольким интерфейсам. Если вы незнакомы с объектно-ориентированным программированием, то, наверное, имеет смысл почитать что-нибудь на эту тему – ресурсов в веб предостаточно. Пока отметим лишь, что объектно-ориентированная модель, принятая в Dart, похожа на то, что есть в Java и C#, и совсем не похожа на JavaScript. Подробнее классы рассматриваются в главах 6 и 7.

По умолчанию все классы в Dart наследуют классу `Object`. У них могут быть открытые и закрытые члены, а удобный синтаксис ме-

тодов чтения и установки позволяет использовать поля и свойства взаимозаменяемым образом, не затрагивая пользователей класса. Ниже приведен пример простого класса.

Листинг 1.2. Простой класс в Dart

```
class Greeter {    ← Ключевое слово class определяет новый класс
  var greeting;    ← Открытое свойство
  var _name;       ← Имена закрытых свойств начинаются с символа _

  sayHello() {     ← Открытый метод
    return "$greeting ${this.name}";    ← Строковая интерполяция
  }

  get name => _name;                    | Синтаксис сокращенной записи методов чтения
  set name(value) => _name = value;     | и установки
}

main() {
  var greeter = new Greeter();    ← Ключевое слово new создает новый экземпляр класса Greeter
  greeter.greeting = "Hello ";    | Синтаксис присваивания значений полям и свойствам
  greeter.name = "World";         | одинаков
  print(greeter.sayHello());
}
```

В этом простом классе заключено немало интересного. Закрытые члены обозначаются добавлением знака подчеркивания _ в начало имени. Это принятое в Dart соглашение позволяет читателю кода сразу сказать, что доступ к методу или свойству осуществляется в закрытой области видимости.

Синтаксис методов чтения и установки также полезен, потому что позволяет одинаково работать как с полями, так и со свойствами. Поэтому проектировщик класса может сначала раскрыть поле (как в случае `greeting`), а затем преобразовать его в свойство, к которому можно обращаться только с помощью методов чтения и установки – акцессоров (как в случае `name`); при этом никаких изменений в вызывающий код вносить не придется.

Ключевое слово `this`, которое вызывает столько путаницы в JavaScript, тоже используется традиционным для ООП способом. Оно ссылается на конкретный экземпляр самого класса, а не на владельца класса в данный момент времени (как в JavaScript).

Классы факультативны
В отличие от Java и C#, классы в Dart факультативны. Функции могут находиться в области видимости верхнего уровня, не будучи членами классами. Иными словами, объявлять класс, лишь для того чтобы создать функцию, необязательно. Если выясняется, что класс содержит только служебные методы, то, возможно, заводить его не стоило, а лучше было определить функции на верхнем уровне.

1.2.4. Определение подразумеваемого интерфейса

В Dart, как и в Java и C#, имеются интерфейсы, но для их определения применяется структура класса. Предполагается, что любой класс неявно определяет интерфейс, включающий его открытые члены. В листинге 1.3 определены класс `Welcomer` и функция верхнего уровня `sayHello()`, которая принимает в качестве параметра экземпляр `Welcomer`. Помимо ключевого слова `extends` для реализации наследования в смысле Java и C#, можно также использовать определяемый классом интерфейс с помощью ключевого слова `implements`. Класс `Greeter` реализует открытые методы класса `Welcomer`, что дает возможность использовать его вместо `Welcomer`. Таким образом, программист может принимать во внимание только интерфейс класса, а не его конкретную реализацию.

Листинг 1.3. У любого класса есть неявный интерфейс

```
class Welcomer {
  printGreeting() => print("Hello ${name}");
  var name;
}

class Greeter implements Welcomer {
  printGreeting () => print("Greetings ${name}");
  var name;
}

void sayHello(Welcomer welcomer) {
  welcomer.printGreeting();
}

main() {
  var welcomer = new Welcomer();
  welcomer.name = "Tom";
```

Классу Welcomer можно унаследовать...

...но у него имеется также неявный интерфейс, реализуемый классом Greeter

← Ожидается аргумент типа Welcomer

```
  sayHello(welcomer);

  var greeter = new Greeter();
  greeter.name = "Tom";
  sayHello(greeter);      ← Поскольку Greeter реализует интерфейс Welcomer, его можно
}                           использовать вместо Welcomer
```

Способность реализовывать класс, не имеющий явно определенного интерфейса, – весьма полезная особенность Dart. Благодаря ей можно без особого труда создавать подставные классы (mocks) или предоставлять собственную реализацию класса; необязательно явно наследовать общему базовому классу.

1.2.5. Фабричные конструкторы для предоставления реализации по умолчанию

Помимо синтаксиса конструкторов, аналогичного Java и C#, в Dart имеется понятие фабричного конструктора (factory constructor). Можно определить базовый класс, используемый как интерфейс, и включить в него фабричный конструктор, который предоставит конкретный экземпляр по умолчанию. Это особенно полезно, когда в большинстве случаев требуется всего одна реализация некоторого интерфейса.

В листинге 1.4 приведен пример класса `IGreetable`, в котором имеется фабричный конструктор, возвращающий экземпляр класса `Greeter`. Класс `Greeter` реализует интерфейс `IGreetable` и позволяет пользователям этого интерфейса использовать реализацию `Greeter`, предоставляемую по умолчанию, не зная о том, что именно они используют. Таким образом, проектировщик класса `Greeter` может изменить конкретную реализацию, а пользователи интерфейса `IGreetable` об этом не узнают.

Листинг 1.4. Фабричные конструкторы предоставляют реализации по умолчанию

```
abstract class IGreetable {       ← Определяется интерфейс
  String sayHello(String name);   ← Объявляется метод, который необходимо реализовать

  factory IGreetable() {          | Фабричный конструктор
    return new Greeter();         | возвращает
  }                               | экземпляр Greeter
```

```
}

class Greeter implements IGreetable {
  sayHello(name) {
    return "Hello $name";
  }
}

void main() {
  IGreetable myGreetable = new IGreetable();
  var message = myGreetable.sayHello("Dart");
  print(message);
}
```

Greeter реализует интерфейс IGreetable

Создается экземпляр IGreetable, который возвращает реализацию Greeter

Используется реализация Greeter

Важно отметить, что благодаря этой возможности многие встроенные классы на самом деле являются интерфейсами – в частности, `String` и `int`. У них есть конкретные реализации, возвращаемые фабричными конструкторами. Я буду рассматривать классы, интерфейсы и их взаимодействие с факультативной системой типов во второй части книги.

1.2.6. Библиотеки и область видимости

В Dart имеется возможность организовывать логические наборы исходных файлов. Конечно, можно поместить все Dart-приложение в единственный файл с расширением .dart, но это плохо с точки зрения организации кода и навигации по нему. Для решения подобной проблемы в язык Dart включено понятие библиотеки. В Dart библиотекой называется совокупность исходных файлов, которые можно было бы объединить в один файл, но это не сделано, чтобы человеку было проще работать с кодом.

Выше я уже отмечал, что классы в Dart факультативны. Так сделано, чтобы функции можно было определять на верхнем уровне библиотеки. В Dart библиотека представляет собой один или несколько файлов с расширением .dart, сгруппированных исходя из каких-то логических соображений; в каждом файле может быть несколько классов (в том числе нуль) и несколько функций верхнего уровня (в том числе нуль). Библиотека Dart может импортировать другие библиотеки, необходимые для работы включенного в нее кода.

Для определения библиотеки служит ключевое слово `library`, для импорта библиотек – ключевое слово `import`, а для ссылки на другие

исходные файлы – ключевое слово `part`. Все это иллюстрируется в листинге ниже.

Листинг 1.5. Библиотеки и исходные файлы

```
library "my_library";     ← Объявляется, что этот файл – библиотека

import "../lib/my_other_library.dart";   ← Импортируется библиотека из другой папки

part "greeter.dart";  |  Включаются другие исходные файлы (в частности, содержащий класс
part "leaver.dart";   |  Greeter)

greetFunc() {   ← Определяется функция в библиотечной области видимости верхнего уровня
  var g = new Greeter();   ← Используется класс из файла greeter.dart
  sayHello(g);   ← Вызывается функция из области видимости верхнего уровня библиотеки
}                    my_other_library
```

Из этого листинга видно, что можно определить метод в области видимости верхнего уровня, то есть не являющийся частью какого-либо класса. Поэтому класс следует создавать, только если может потребоваться экземпляр объекта, а не просто как контейнер для группы взаимосвязанных функций.

Библиотека может включать группу исходных файлов в одну область видимости. При этом число исходных файлов произвольно (в том числе нуль), и библиотека, включающая несколько отдельных файлов, эквивалентна библиотеке из одного файла, содержащего те же самые файлы внутри себя. Поэтому каждый исходный файл может ссылаться на код, находящийся в другом исходном файле, при условии что оба являются частями одной и той же библиотеки. Кроме того, любой исходный файл может обращаться к коду, который находится в импортированной библиотеке, как показано на примере файла `my_other_library.dart` на рис. 1.3.

Чтобы избежать конфликтов имен, которые могли бы возникнуть, если написанная вами и импортированная библиотека обе содержат класс с именем `Greeter`, можно назначить библиотеке префикс:

```
import "../lib/my_other_library.dart" as other;
```

а затем ссылаться на содержащиеся в ней классы следующим образом:

```
other.otherLibFunction("blah");
```

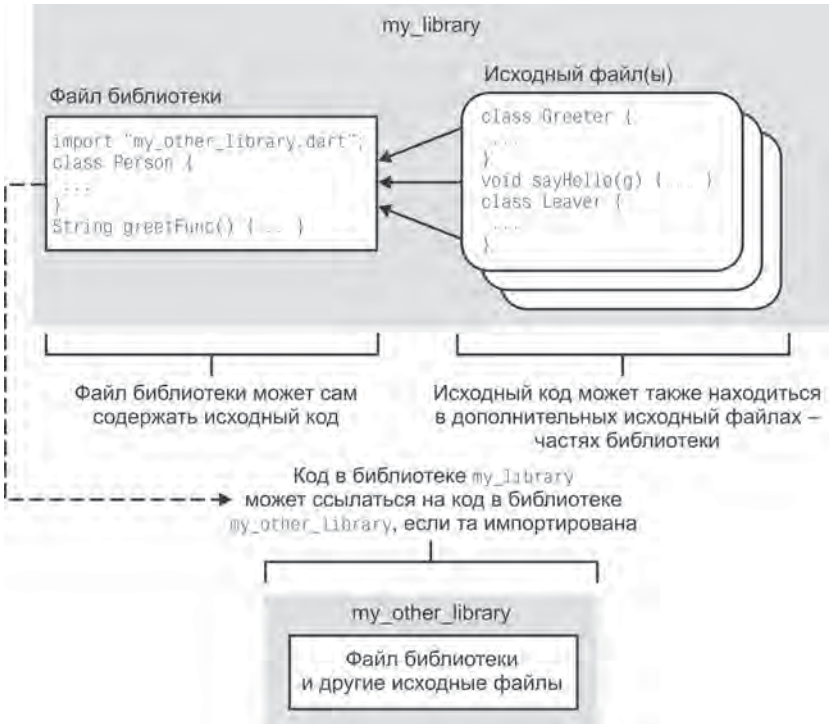

Рис. 1.3. Файл `my_library.dart` состоит из файлов `greeter.dart` и `leaver.dart` и использует другую библиотеку с именем `my_other_library.dart` (которая, в свою очередь, состоит из нескольких исходных файлов)

Это позволяет присваивать классам и методам разумные имена, не опасаясь засорить глобальное пространство имен.

Библиотека может содержать точку входа в приложение, но тогда в ней должна быть функция `main()`. Такая функция может присутствовать в нескольких библиотеках, но выполняется та, что находится в библиотеке, упомянутой в теге `<script>`.

Все функции и классы, содержащиеся в вашей библиотеке, становятся доступны любой импортирующей ее программе, то есть являются открытыми. Чтобы запретить импортирующей программе доступ к отдельным функциям или классам, пометьте их как закрытые.

Закрытость в классах и библиотеках

Хотя библиотеки и классы полезны для обеспечения модульности приложения, рекомендуется скрывать внутренние механизмы их работы. По счастью, в Dart имеется простой способ закрыть внутренности от любопытного взгляда: достаточно поместить в начало имени метода, функции, класса или свойства знак подчеркивания (_). В результате элемент становится доступен только изнутри самой библиотеки.

Закрытость распространяется только на клиентов библиотеки

Вернемся к предыдущему примеру. Если классы `Greeter` и `Leaver` являются частями одной и той же библиотеки, то они могут обращаться к закрытым элементам друг друга (по аналогии с конструкцией `package private` в Java). Это означает также, что свойство или функция `_greeterPrivate()` доступна из любого другого класса в той же библиотеке. Но если класс `Greeter` импортируется в другую библиотеку, то в последней этот метод не виден (рис. 1.4).

Закрытыми могут быть функции верхнего уровня библиотеки, классы, поля, свойства и методы классов (сообща называемые членами), а также конструкторы классов. Внутри библиотеки закрытость игнорируется, то есть из любого файла-части, включенного с помощью ключевого слова `part`, можно обращаться к закрытым элементам в любой другой части той же самой библиотеки `library`. Однако клиенты библиотеки не могут обращаться к ее закрытым элементам (как и к закрытым элементам классов, определенных в этой библиотеке).

1.2.7. Функции как полноценные объекты

В Dart с функциями можно обращаться, как с объектами, – так же, как в JavaScript: разрешается передавать функцию в виде параметра, сохранять функцию в переменной и использовать анонимные (безымянные) функции для обратных вызовов. В следующем листинге приведены примеры разнообразного употребления функций.

Листинг 1.6. Функции как полноценные объекты

```
String sayHello(name) => "Hello $name";   ← ❶ Сокращенный синтаксис объявления функции

main() {
  var myFunc = sayHello;     ← ❷ Функция присваивается переменной
```

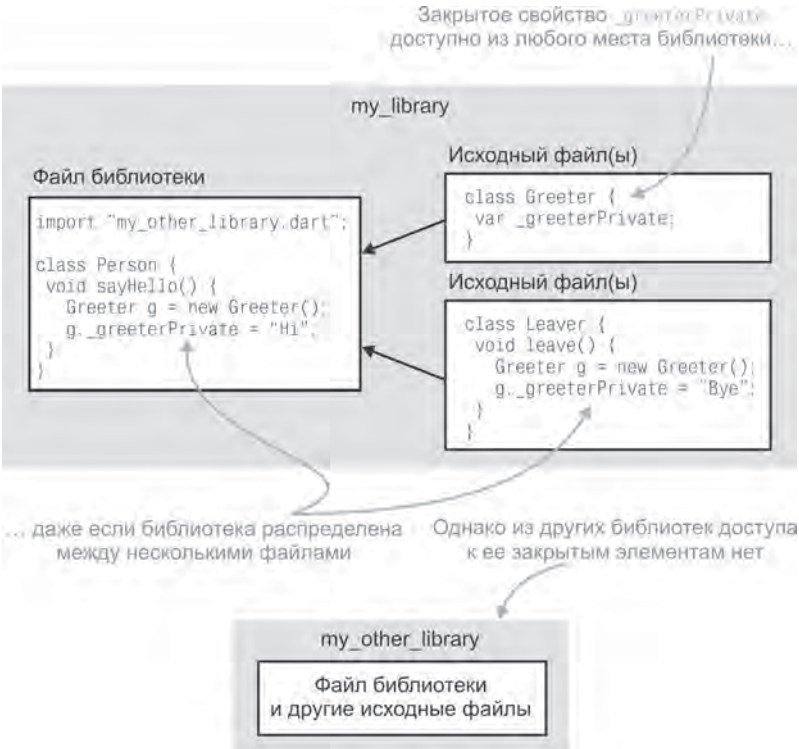

Рис. 1.4. Закрытые элементы – поля, методы, библиотечные функции и классы – скрыты внутри библиотеки. Закрытость обозначается префиксом _. Пользователи библиотеки не имеют доступа к закрытым элементам

```
  print(myFunc("World"));      ← Вызывается функция, хранящаяся в переменной

  var mySumFunc = (a,b) {      ← Определяется анонимная функция
    return a+b;
  };

  var c = mySumFunc(1,2);      ← Вызывается анонимная функция
  print(c);
}
```

❶ – это однострочная функция, возвращающая значение; она определена с помощью сокращенного синтаксиса. Следующие две функции идентичны:

```
String sayHello(name) {
  return "Hello $name";
}

String sayHello(name) => "Hello $name";
```

Определив функцию, мы можем получить ссылку на нее и сохранить ее в переменной ❷. Эту переменную можно передавать, как любое другое значение. Анонимные функции наподобие той, что сохранена в переменной `mySumFunc`, часто применяются в качестве обработчиков событий. Нередко можно встретить такой код:

```
myButton.on.click.add((event) {
  // что-то сделать
});
```

Здесь анонимная функция выделена полужирным шрифтом.

1.2.8. Параллелизм с помощью изоляторов

Dart – однопоточный язык. И хотя такое решение может показаться идущим вразрез с современной аппаратной архитектурой, в которой доступных приложению процессоров становится все больше и больше, оно упрощает написание программ на Dart.

В Dart единицей работы является изолятор (isolate), а не процесс или поток. У каждого изолятора имеется собственная область памяти (разделение памяти между изоляторами не допускается), что облегчает реализацию изолированной модели защиты. Один изолятор может передавать сообщения другому. Когда изолятор получает сообщение (оно может содержать и данные), вызывается обработчик события, который обрабатывает это сообщение точно так же, как сообщение нажатия кнопки. Изолятор-приемник получает копию сообщения, посланного изолятором-отправителем. Изменения полученных данных не видны отправляющей стороне, получатель должен отправить ответное сообщение.

На веб-странице каждый скрипт (содержащий функцию `main()`) исполняется в отдельном изоляторе. Различные части могут быть реализованы разными скриптами; например, один читает новостную ленту, другой занимается синхронизацией автономного хранилища и т. д. Программа на Dart может создать новый изолятор по анало-

гии с тем, как в Java или C# запускается новый поток. Но изолятор отличается от потока тем, что обладает собственной памятью. Не существует никакого способа предоставить общий доступ из нескольких потоков к одной переменной; единственный способ коммуникации между изоляторами – передача сообщений.

Изоляторы используются также для динамической загрузки внешнего кода. Код, не являющийся частью приложения, можно загрузить в защищенную область и исполнять независимо от приложения, организовав коммуникацию с помощью обмена сообщениями. Такое поведение идеально для разработки архитектуры подключаемых модулей.

В реализации виртуальной машины Dart можно при необходимости использовать несколько процессорных ядер для запуска изоляторов. А при трансляции изоляторов на JavaScript они становятся рабочими веб-процессами в смысле HTML5.

Памятка

- В Dart имеется факультативная типизация (для документирования).
- Библиотеки позволяют организовывать код, распределяя его между несколькими исходными файлами.
- Механизм ограничения доступа встроен в язык.
- Функции – полноценные объекты и могут быть определены вне классов.
- В Dart реализован параллелизм с помощью изоляторов, обменивающихся сообщениями.

Бегло познакомившись с основами Dart, мы теперь можем посмотреть, как он применяется в веб-приложениях.

1.3. Веб-программирование на языке Dart

Одна из целей Dart – облегчить жизнь разработчикам. А поскольку Dart – в первую очередь язык программирования для веб, то проектировщики немало постарались, чтобы сделать API манипулирования моделью DOM браузера максимально удобным. В JavaScript доступ к DOM был сущим кошмаром, пока не появилась библиотека jQuery, сделавшая работу с браузером простой и естественной. По аналогии с ней и была написана библиотека `dart:html`.

1.3.1. dart:html: удобная библиотека для работы с моделью DOM браузера

На момент написания этой книги для Dart еще не было библиотеки виджетов пользовательского интерфейса. И хотя команда Dart открыто заявила, что Dart будет языком типа «все включено», пока идет работа над ранними публичными версиями, больше времени приходится уделять совершенствованию самого языка, а не высокоуровневым абстракциям. Однако уже написана библиотека `dart:html`, которая рассматривается как эквивалент ядра jQuery.

Если вам уже доводилось работать с jQuery или другим подобным каркасом, то вы знаете, как CSS-селекторы используются для доступа к элементам DOM, например DIV'у с `id="myDiv"` или ко всем элементам `<p>`. Библиотека `dart:html` упрощает эту задачу. Вместо того чтобы вызывать различные функции получения элементов, например `getElementsById()` или `getElementsByName()`, как то делается при работе со встроенными в браузер API, в `dart:html` используются всего два метода выборки элементов: `query()`, который возвращает один элемент, и `queryAll()`, который возвращает список элементов. А поскольку используются списки Dart, к ним применимы все стандартные функции работы со списками, в частности `contains()` и `isEmpty()`, а также синтаксис доступа к массиву: `element.children[0]`. В следующем листинге приведены примеры работы с DOM с помощью библиотеки `dart:html`.

Листинг 1.7. Взаимодействие с браузером

```
import 'dart:html';    ← Импортируется библиотека dart:html

void main() {
  var button = new Element.tag("button");    ← Создается новый элемент button
  button.text = "Click me";
  button.on.click.add((event) {                    Добавляется анонимная
    List buttonList = queryAll("button");          функция (выделена
    window.alert("There is ${buttonList.length} button");   полужирным курсивом),
  });                                              которая обрабатывает
                                                   событие on.click
  document.body.children.add(button);  ← Кнопка добавлена в тело HTML-документа
}
```

Здесь для создания кнопки используется именованный конструктор. С кнопкой связывается обработчик события (в виде анонимной

функции), после чего она добавляется в тело документа. При запуске этого примера мы наблюдаем картину, показанную на рис. 1.5.

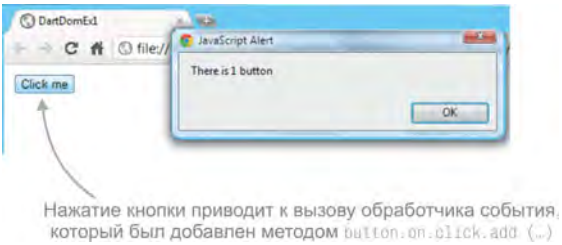

Рис. 1.5. Результат нажатия кнопки, созданной в листинге 1.7

При таком взаимодействии с браузером можно создать сложный пользовательский интерфейс, применяя только Dart и CSS. Мы рассмотрим эту тему в главе 4.

> **Совет.** При написании кода для браузера не забывайте, что функция `print`, использованная в предложении `print("Hello World")`, выводит результат на консоль браузера, а не на страницу. В Chrome для доступа к консоли браузера нужно выбрать пункт **Tools** ⇒ **JavaScript Console** (Инструменты ⇒ Консоль JavaScript) из меню «Настройка и управление».

1.3.2. Dart и HTML5

Библиотека `dart:html` позволяет не только непосредственно взаимодействовать с моделью DOM браузера, но и работать с такими элементами HTML5, как холст, WebGL, события перемещения устройства и данные геолокации. Картинка, изображенная на рис. 1.6, порождена кодом из листинга 1.8, в котором используется HTML5 Canvas API.

В коде рисования на холсте мы сначала добавляем тег HTML5 `<canvas>` в DOM, а затем получаем от него двумерный контекст рисования. После этого в контекст выводятся текст и фигуры.

Листинг 1.8. Рисование на холсте браузера

```
import 'dart:html';
import 'dart:math';

void main() {
  CanvasElement canvas = new Element.tag("canvas");
```

Создается новый элемент CanvasElement

```
  canvas.height = 300;
  canvas.width = 300;
  document.body.children.add(canvas);        ← Холст добавляется в тело документа

  var ctx = canvas.getContext("2d");         ← Получаем от холста контекст рисования

  ctx.fillText("hello canvas", 10, 10);      ← Выводится текст

  ctx.beginPath();
  ctx.arc(50, 50, 20, 0, PI * 2, true);      Рисуется
  ctx.closePath();                           закрашенный круг
  ctx.fill();
}
```

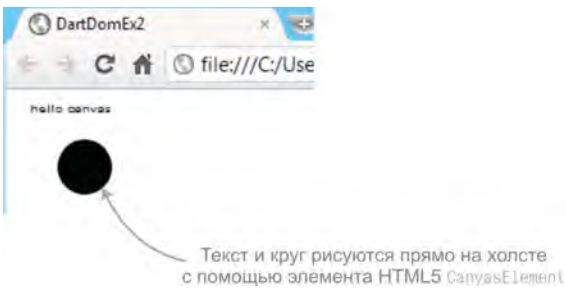

Рис. 1.6. Рисование на холсте браузера

Создав и добавив на страницу элемент `CanvasElement`, мы получаем область, в которой можно рисовать с помощью стандартных методов типа `drawImage`, `fillText` и `lineTo`. Подробнее мы будем рассматривать эту тему в главе 10.

Библиотека `dart:html` дает нам доступ ко всем стандартным элементам браузера. А поскольку библиотека DOM, которая составляет часть `dart:html`, сгенерирована из описания на языке определения интерфейсов WebKit IDL, мы имеем доступ ко всей функциональности современных браузеров, предоставляемой Dart.

Памятка

- Библиотека `dart:html` – это взгляд на модель DOM браузера со стороны Dart.
- Поддержка HTML5 – неотъемлемая часть языка Dart.

Итак, у нас есть некоторое представление о том, как Dart исполняется в браузере. Теперь самое время познакомиться с инструментами, помогающими писать программы на этом языке.

1.4. Инструментальная экосистема Dart

Инструменты Dart считаются частью платформы Dart и потому развиваются так же быстро, как сам язык. Разработчики обычно осваивают язык с помощью имеющихся (или отсутствующих) инструментов, и Google тратит на этот аспект немало сил. Начнем с редактирования кода.

1.4.1. Редактор Dart

Хотя писать программы на Dart можно в любом текстовом редакторе, лучше всего для этой цели приспособлен редактор Dart. Он построен на платформе Eclipse Rich Client Platform (RCP) – каркаса для создания редакторов кода. Редактор Dart располагает такими привычными средствами, как автозавершение кода, навигация, выделение структуры кода, а также умеет производить статический анализ кода, выдавая предупреждения и сообщения об ошибках. Инструмент статического анализа доступен также в виде командной утилиты, которая может включаться в систему непрерывной интеграции для обнаружения ошибок на ранних стадиях. На рис. 1.7 показаны некоторые средства редактора Dart.

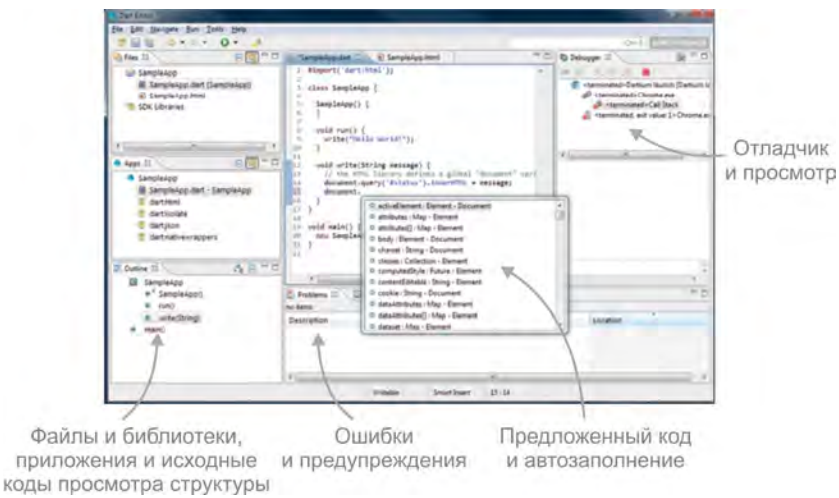

Рис. 1.7. Редактор Dart Editor, показано простое приложение для браузера и окно автозавершения кода

Редактор Dart позволяет писать код, а если этот код ассоциирован с HTML-страницей, то его можно преобразовать в JavaScript и исполнить в браузере по своему выбору с помощью инструмента dart2js. Если используется браузер Dartium – Chrome со встроенной ВМ Dart, – то этап преобразования в JavaScript можно опустить и исполнять Dart-код непосредственно. Dartium также умеет общаться с редактором Dart Editor, что дает возможность сквозной пошаговой отладки.

Если код не связан ни с какой HTML-страницей, то редактор может имитировать его запуск из командной строки с выводом на консоль `stdout`.

1.4.2. Виртуальная машина Dart

ВМ Dart – неотъемлемая часть языка Dart. Одно из ее применений – исполнение Dart-кода из командной строки, например запуск HTTP-сервера или простого скрипта (аналог пакетного файла или скрипта оболочки). Другое – встраивание в объемлющее приложение типа браузера Dartium.

1.4.3. Dartium

Dartium – это специализированная версия браузера Chromium (вариант Google Chrome с открытым исходным кодом), в которую встроена ВМ Dart. Он распознает тип скрипта `application/dart` и исполняет написанный на Dart код непосредственно без промежуточного преобразования в JavaScript. В Dartium включены инструменты разработчика, знакомые многим программистам, которые создают сайты и веб-приложения в Chrome. В сочетании с редактором Dart он позволяет вести пошаговую отладку: вы можете поставить точки останова в редакторе, а затем обновить страницу приложения в Dartium; встроенный в редактор отладчик остановится в достигнутой точке и позволит просмотреть переменные и продолжить исполнение в пошаговом режиме.

Благодаря браузеру Dartium разработка на Dart становится не сложнее, чем на JavaScript. Обновление страницы в браузере – все, что нужно для исполнения Dart-кода.

1.4.4. dart2js: конвертер Dart в JavaScript

Инструмент dart2js служит для компиляции Dart в JavaScript и вызывается из редактора Dart Editor или из командной строки

в виде автономной утилиты. Он компилирует все библиотеки и исходные файлы, составляющие Dart-приложение, в один JavaScript-файл. Результирующий код вполне понятен, хотя в случае, когда разработка на Dart ведется с применением Dartium, читать его вряд ли имеет смысл.

dart2js также включает в JavaScript ссылки на исходный код, позволяющие перейти от порожденного JavaScript-кода к коду на Dart, из которого он получен. Это недавнее дополнение успешно используется и в других языках, транслируемых на JavaScript, в частности CoffeeScript и Google Web Toolkit (GWT).

Примечание. dart2js – третий конвертер Dart в JavaScript. Первый назывался dartc, второй – frog. Эти названия встречаются в старой документации и в блогах.

1.4.5. Управление пакетами с помощью pub

Управление пакетами – важная черта любого языка. В Java для этой цели используется Maven, в .NET – NuGet, а в node.js – npm. В Dart есть собственный менеджер пакетов, который называется pub. Pub позволяет разработчику определить метаданные пакета в файле pubspec и опубликовать библиотеки в таких репозиториях, как GitHub.

Когда вам понадобится какая-то библиотека, вы можете с помощью pub загрузить ее вместе со всеми зависимостями с учетом номеров версий. Мы подробнее обсудим pub и приведем соответствующие примеры в главе 5, когда будем говорить о структуре библиотеки, написанной на Dart.

Памятка

- Инструментальная экосистема составляет неотъемлемую часть проекта Dart.
- Редактор Dart предлагает разработчикам развитые инструментальные средства.
- Благодаря Dartium разработка на Dart оказывается не сложнее обновления страницы в браузере.
- Dart спроектирован так, что легко транслируется на JavaScript.

1.5. Резюме

На первый взгляд может показаться, что Dart – просто очередной язык программирования. Но если рассматривать его в контексте всей окружающей экосистемы, то окажется, что Dart открывает блистательные перспективы в мире веб-разработки. Современные приложения становятся все более сложными, для их создания необходимы крупные коллективы разработчиков, и Dart вместе со своими инструментами и средами обещает внести некую структуру в слишком уж гибкий мир JavaScript.

Создавать одностраничные приложения, работающие в браузере (к примеру, Google Plus), на Dart гораздо легче, потому что сопровождение объемного клиентского кода перестает быть чрезмерно хрупким. Язык Dart – который можно исполнять непосредственно или транслировать на JavaScript – вкупе с HTML5 – идеальное решение для построения веб-приложений, не нуждающихся во внешних подключаемых модулях.

В следующих главах мы ближе познакомимся с экосистемой Dart, изучим во всех подробностях ядро языка и воспользуемся Dart для разработки одностраничных веб-приложений, ориентированных на современные веб-браузеры с поддержкой HTML5. Прочитав эту книгу, вы сможете создавать на Dart приложения, способные работать на стороне клиента без подключения к сети, загружаемые с файлового сервера Dart и подключающиеся к Dart-серверу для сохранения данных в базе.