Содержание

Предисловие

Мы познакомились с *Марком Нури (Marc Nuri)* много лет тому назад в Red Hat. Он невероятно отзывчивый человек, постоянно готовый прийти на помощь, когда у вас возникают проблемы. Марк всегда открыт для обсуждения любого фрагмента кода, написанного им.

В Red Hat он проделал большую работу по созданию инструментов, облегчающих работу Java-разработчиков, переходящих на контейнеры и Kubernetes. Я не знаю никого, кто был бы более квалифицированным, чем Марк, и способным помочь вам освоить Java и Kubernetes.

Марк также часто выступает на международных конференциях и в своих презентациях разъясняет, почему следует использовать те или иные технологии, показывая простые и понятные примеры использования широкого спектра технологий Java, таких как Maven/Gradle или Spring Boot/Quarkus.

В этой книге вы узнаете, как разрабатывать серверные приложения на Java с помощью Quarkus, нового Java-фреймворка для Kubernetes, который помогает разрабатывать сверхзвуковые (запускающиеся за миллисекунды) и субатомные (требующие минимального объема памяти) Java-приложения. Помимо знакомства с Quarkus, вы также узнаете, как обезопасить серверную часть приложения и повысить ее тестируемость. Кроме того, вы увидите, как разрабатывать пользовательские интерфейсы с помощью React и интегрировать их с Quarkus. Наконец, вы узнаете, как развернуть приложение в Kubernetes, платформе для микросервисов, всего за один шаг (без YAML), что является мечтой разработчика.

Эта книга сделает из вас сверхзвукового и субатомного Java-разработчика.

– Алекс Сото (Alex Soto)
Java Champion, Director of Developer Experience в Red Hat

Об авторе

Марк начинал свою карьеру как внештатный разработчик веб-приложений, создавая и поддерживая программное обеспечение для транспортной отрасли. Затем на протяжении нескольких лет Марк работал в разных компаниях, где занимался созданием масштабируемых веб-приложений для разных отраслей (розничная и оптовая торговля, электронная коммерция и т. д.). В настоящее время разрабатывает программное обеспечение с открытым исходным кодом, специализируясь на создании и обслуживании инструментов для разработчиков Java и Kubernetes.

Я хочу поблагодарить команду Packt и всех, кто помогал и работал над созданием этой книги; это было настоящее командное приключение.

Отдельное спасибо моей семье и всем, кто поддерживал меня и проявлял терпение в этот напряженный период

О рецензентах

Джеймс Брукс (James Brooks) – разработчик с большим опытом. Занимался созданием настольных приложений для Windows и встроенных устройств в General Atomics, корпоративных веб-приложений в J. B. Hunt, разработкой ценообразования в Medicare и контрактами с налоговым судом США в Flexion.

В своей работе использовал несколько фреймворков Java и является большим поклонником Quarkus. В последнее время занимается разработкой приложений React/Node. Желающие могут посетить его личную страницу по адресу https://james.oranbrooks.com. В свободное время руководит агентством цифрового маркетинга Hidden Gems Digital Marketing, расположенным в Фейетвилле, штат Арканзас, США. Страстно увлечен ремеслом разработки программного обеспечения и подходит к нему как истинный инженер.

Маркус Мело (Marcus Melo) – инженер-программист, увлеченный созданием эффективных и надежных решений. Имеет степень магистра информатики и обладает такими сертификатами, как AWS Cloud Practitioner, Java Programmer certification (SCJP 6), Scrum Master Certified и AZ-900 Azure Fundamentals. Занимается программированием на Java уже более 16 лет, имеет опыт использования Java-фреймворков, таких как Quarkus, Spring Boot и Angular. Широко применяет некоторые шаблоны проектирования и методы гибкой разработки (Scrum). Одно время работал разработчиком в BB Tecnologia e Serviços, в подразделении, занимающемся разработкой программных решений для одного из крупнейших банков Бразилии и многих других компаний. В настоящее время он работает в Adentis в Португалии.

Вагнер Рикардо Вагнер (Wagner Ricardo Wagner) – архитектор программного обеспечения. Обладает более чем 15-летним опытом работы на рынке программного обеспечения. Участвовал в нескольких крупных проектах в Бразилии, Уругвае и Парагвае в налоговом и финансовом секторах. Имеет опыт работы с облачными технологиями, Java, Node, Go, базами данных, Google Cloud, K8s, DDD, чистым кодом, архитектурой и дизайном программного обеспечения, сбором измененных данных, NoSQL и т. д.

Я благодарю Бога, мою семью и друзей, которые поддерживали меня на протяжении многих лет. Также я хочу упомянуть Жана Пахла (Jean Pachla) и Николаса Тишлера (Nícolas Tischler), которые всегда ставили передо мной сложные задачи и давали возможность карьерного роста

Вступление

React зарекомендовала себя как одна из самых популярных и широко распространенных библиотек JavaScript благодаря своей простоте и возможности использования для создания масштабируемых приложений. Quarkus – это фантастический фреймворк для разработки серверных приложений, повышающий продуктивность разработчиков благодаря наличию готовых средств интеграции, прикладных служб и т. д., которые привносят новый революционный опыт разработки в Java.

Эта книга описывает практический опыт по созданию и развертыванию комплексного веб-приложения с использованием Quarkus и React. Для простоты книга разделена на три части. В первой части вы найдете базовое введение в Quarkus и его возможности. Узнаете, как запустить проект Quarkus с нуля, как создать надежный и защищенный HTTP-сервер для вашего приложения. Вторая часть посвящена пользовательскому интерфейсу. Здесь вы увидите, как создать пользовательский интерфейс с поддержкой React и как интегрировать его с серверной частью Quarkus. Последняя часть рассказывает, как создавать манифесты конфигурации кластера и как развертывать их в Kubernetes и других альтернативах, таких как Fly.io.

К концу книги вы приобретете навыки владения обеими платформами, необходимые для создания и развертывания автономных полнофункциональных веб-приложений.

Кому адресована эта книга

Эта книга предназначена для разработчиков серверных веб-приложений, имеющих хотя бы небольшой опыт работы с Java и желающих узнать, как с помощью React создавать полнофункциональные приложения путем интеграции их с серверной частью на основе Quarkus. Она также предназначена для разработчиков веб-интерфейсов, имеющих некоторый опыт работы с JavaScript и желающих научиться писать серверные компоненты с использованием Quarkus, интегрировать их со своими интерфейсами и создавать комплексные веб-приложения.

Желательно, чтобы читатель имел хотя бы базовые знания Java и JavaScript, но, вообще говоря, любой разработчик, имеющий опыт работы только с Java или только с JavaScript, сможет без проблем следовать за примерами в этой книге.

О чем рассказывается в книге

Глава 1 «*Создание проекта*» знакомит с фреймворком Quarkus и инструментами и показывает, как создать проект приложения, которое будет разрабатываться на протяжении всей книги.

Глава 2 «*Добавление поддержки хранилищ*» показывает, как реализовать слой поддержки хранилищ в Quarkus с помощью Hibernate и **Java Persistence API (JPA)**.

Глава 3 «*Создание HTTP API*» объясняет приемы создания слоя служб и использования механизма внедрения зависимостей для реализации HTTP API в Quarkus.

Глава 4 «*Защита приложения*» показывает, как организовать слой безопасности на основе **веб-токенов JSON (JSON Web Tokens, JWT)** для защиты HTTP API.

Глава 5 «*Тестирование серверной части*» представляет среду тестирования Quarkus и Quarkus Dev Services, а также объясняет, как с их помощью реализовать тесты для вашего приложения.

Глава 6 «*Создание двоичного образа*» показывает, как создать выполняемый файл приложения на основе Quarkus.

Глава 7 «*Создание проекта React*» знакомит с React и библиотеками, которые будут использоваться для реализации клиентской части приложения, и показывает, как создать проект.

Глава 8 «*Создание страницы входа*» объясняет порядок реализации инфраструктуры авторизации и демонстрирует приемы создания страницы входа для пользовательского интерфейса приложения.

Глава 9 «*Создание основного приложения*» показывает, как реализовать основные функции приложения и как использовать серверный HTTP API.

Глава 10 «*Тестирование пользовательского интерфейса*» объясняет, как реализовать модульные и интеграционные тесты для пользовательского интерфейса на основе React.

Глава 11 «*Интеграция с Quarkus*» показывает, как интегрировать пользовательский интерфейс с серверной частью на основе Quarkus.

Глава 12 «*Развертывание приложения в Kubernetes*» дает краткий обзор Kubernetes и объясняет, как развернуть приложение в кластере Minikube Kubernetes.

Глава 13 «*Развертывание приложения в Fly.io*» показывает, как развернуть приложение в Fly.io и открыть доступ к нему.

Глава 14 «*Создание конвейера непрерывной интеграции*» представляет механизм GitHub Actions и объясняет, как создать конвейер **непрерывной интеграции (Continuous Integration, CI)**.

Как получить максимум от этой книги

Базовые знания Java и JavaScript желательны, но вообще любой разработчик, имеющий опыт работы только с Java или только с JavaScript, сможет без всяких затруднений прочитать всю книгу.

Вам понадобится последняя версия Java JDK LTS, последняя версия Node.js LTS и рабочее окружение Docker.

Программное обеспечение, используемое в этой книге	**Требования к операционной системе**
Quarkus 2	Windows, macOS или Linux
React 18	

Если вы читаете электронную версию этой книги, то мы советуем вводить код самостоятельно или получить его из репозитория книги на GitHub (ссылка приводится в следующем разделе). Это поможет вам избежать возможных ошибок, связанных с копированием и вставкой кода.

Загрузка цветных изображений

Мы создали дополнительный файл PDF с цветными изображениями снимков экрана и диаграмм, использованных в этой книге. Взять его можно по адресу: https://packt.link/yoqoD.

Типографские соглашения

В этой книге используется ряд соглашений по оформлению текста.

`Код в тексте`: так оформляются фрагменты программного кода в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов, пути к файлам, фиктивные URL, ввод пользователя и ссылки Twitter. Например: «Проект включает каталог `.mvn` и выполняемые файлы `mvnw` и `mvnw.cmd`».

Блоки кода оформляются так:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-reactive</artifactId>
</dependency>
```

Любой ввод или вывод в командной строке будет оформляться так:

```
./mvnw quarkus:dev
```

Жирным шрифтом будут выделяться новые термины, важные определения или слова, которые видны на экране. Например, надписи в меню или в диалоговых окнах будут выделены **жирным шрифтом**. Например: «Для этого нужно создать новую конфигурацию отладки, выбрав в меню пункт **Run > Edit Configurations...** (Пуск | Редактировать конфигурации).

Советы или важные примечания

Будут оформляться так.

Часть I

СОЗДАНИЕ СЕРВЕРНОЙ ЧАСТИ С ПОМОЩЬЮ QUARKUS

В этой части основное внимание уделяется знаниям и навыкам, необходимым для реализации серверной части на Java с использованием фреймворка Quarkus и его расширений. Здесь вы узнаете, как создать проект Quarkus с нуля и получить надежный и защищенный HTTP-сервер для вашего приложения.

Эта часть включает следующие главы:

- главу 1 «Создание проекта»;
- главу 2 «Добавление поддержки хранилищ»;
- главу 3 «Создание HTTP API»;
- главу 4 «Защита приложения»;
- главу 5 «Тестирование серверной части»;
- главу 6 «Создание двоичного образа».

Глава 1

Создание проекта

В этой книге мы создадим полнофункциональное веб-приложение, используя **Quarkus** для серверной части и **React** для пользовательского интерфейса. Quarkus – это новый фреймворк, призванный изменить существующую экосистему Java за счет упрощения разработки **облачных приложений**. ReactJS – одна из самых популярных библиотек на JavaScript для разработки пользовательского интерфейса. К концу книги вы научитесь комбинировать обе платформы для создания и развертывания полнофункционального веб-приложения диспетчера задач.

В этой главе вы познакомитесь с Quarkus и инструментами, которые будут использоваться в этой части. Затем мы создадим проект Quarkus и рассмотрим базовую архитектуру и структуру приложения. К концу этой главы вы научитесь создавать проекты Quarkus и настраивать рабочее окружение разработки для реализации новых функций, а также упаковывать и запускать приложения на своем компьютере.

Эта глава рассматривает следующие темы:

- что такое Quarkus;
- настройка рабочего окружения в IntelliJ IDEA;
- создание приложения Quarkus;
- структура проекта и зависимости.

Технические требования

Вам понадобится последняя версия Java JDK LTS (на момент написания – Java 17). В этой книге мы будем использовать Fedora Linux, но вы можете работать в Windows или macOS.

Исходный код примеров для данной главы доступен в репозитории https://github.com/PacktPublishing/Full-Stack-Quarkus-and-React/tree/main/chapter-01.

Что такое Quarkus?

Java существует уже более 25 лет. Это один из наиболее широко используемых языков программирования, особенно в сфере корпоративных приложений. Его богатая экосистема, обширное и открытое сообщество, а также подход *«написал один раз – работает где угодно»* сделали его стандартным выбором для разработки корпоративного программного обеспечения на десятилетия.

Однако сейчас все меняется. Мы живем в эпоху развития *облачных технологий*, *Kubernetes* и *образов контейнеров*. Такие аспекты, как время запуска или объем используемой памяти, которые раньше не имели большого значения, в наши дни становятся все более актуальными. Java проигрывает в этом отношении другим языкам, специально разработанным для таких новых окружений.

Quarkus – это новый фреймворк для Java, появившийся в 2019 году. Он предлагает практически такой же набор возможностей, что и другие основные фреймворки для Java, такие как Spring Boot или Micronaut, но ускоряет запуск приложений, уменьшает потребление памяти и предлагает дополнительные удобства для разработчиков.

Quarkus создавался с нуля и с самого начала предназначался для работы в *облаке*. Независимо от способа упаковки приложения, Quarkus обеспечивает более короткое время запуска и меньшее потребление памяти, чем альтернативы. Вот почему Quarkus также известен как *сверхзвуковая субатомная Java*.

Quarkus построен на основе проверенных временем **стандартов** и **библиотек**. Команда, разрабатывающая Quarkus, решила положиться на существующие инструменты, платформы и стандарты, а не создавать что-то новое с нуля. Это было сделано для того, чтобы разработчикам не приходилось тратить время на изучение чего-то нового и иметь возможность сосредоточиться на создании более производительных приложений, используя накопленный опыт.

Quarkus использует стандарт **Java Enterprise Edition (EE)**; механизм внедрения зависимостей **Contexts and Dependency Injection (CDI)**; **MicroProfile** – разработанную сообществом спецификацию оптимизации Java EE для микросервисных архитектур, а также настройки и мониторинга; аннотации **Java Persistence API (JPA)** для определения **объектно-реляционных отображений (Object-Relational Mapping, ORM)**; аннотации **Jakarta RESTful Web Services (JAX-RS)** для определения контроллеров REST и многие другие технологии.

Quarkus привносит в Java новые **возможности для разработчиков**. Одним из основных недостатков Java по сравнению с другими языками является традиционно медленный цикл разработки. Простое изменение строки кода обычно влечет перекомпиляцию, переупаковку и перезапуск приложения. Этот процесс может занять от нескольких секунд до нескольких минут, что отрицательно сказывается на продуктивности разработчиков. Quarkus стремится решить эту проблему, предоставляя поддержку «живого кодирования» (live coding), унифицированные настройки, пользовательский интерфейс разработчика и многие другие инструменты, которые вернут разработчикам радость творчества.

Quarkus сочетает традиционный **императивный** и новый **реактивный** стили программирования. Независимо от типа создаваемого приложения, Quarkus предоставляет первоклассную поддержку обеих парадигм. Облачные технологии привнесли в наши системы новые архитектуры, будь то микросервисы, бессерверные функции или службы, управляемые событиями. В этой книге мы рассмотрим новый реактивный и неблокирующий стиль, который значительно улучшает производительность приложений по сравнению с классическим императивным подходом.

Quarkus – это бесплатный проект с открытым исходным кодом, появившийся в марте 2019 года. Это не только быстро развивающийся проект, но и быстро растущее сообщество. Несмотря на свою молодость, он насчитывает уже около 600 участников (на момент написания этих строк) и имеет развитую экосистему расширений (https://github.com/quarkiverse), а в интернете появилось огромное количество публикаций, посвященных ему.

Quarkus позволяет с легкостью создавать выполняемые файлы для вашей платформы, обеспечивает почти прозрачную интеграцию с *GraalVM*, высокопроизводительным дистрибутивом **Java Development Kit (JDK)**, позволяющим компилировать код на Java в автономный двоичный выполняемый файл.

Настройка рабочего окружения в IntelliJ IDEA

В этой книге мы будем работать над созданием полнофункционального веб-приложения, используя Quarkus и React. Поэтому нам придется писать код. Для этого будет использоваться **интегрированная среда разработки IntelliJ IDEA**; однако вы можете использовать любую другую IDE или редактор по своему выбору. IntelliJ IDEA – хороший выбор, потому что последние версии прекрасно подходят для разработки и серверной части, и пользовательского интерфейса.

IntelliJ IDEA – это интегрированная среда разработки (Integrated Development Environment, IDE), созданная в *JetBrains* и основанная на платформе с открытым исходным кодом IntelliJ. Она доступна в двух вариантах: бесплатная версия для сообщества и Ultimate Edition, требующая оформления платной подписки. Впрочем, существует «программа раннего доступа», дающая бесплатный доступ к ранним сборкам версии Ultimate.

Чтобы получить IntelliJ IDEA, перейдите на страницу продукта и загрузите соответствующий пакет для своей системы: https://www.jetbrains.com/idea/download.

Создание приложения Quarkus

Quarkus предоставляет несколько способов создания приложения. Самый простой – использовать страницу https://code.quarkus.io. Также создать приложение можно с помощью **интерфейса командной строки (Command-**

Line Interface, CLI) или запустив цель Maven. Мы будем использовать веб-интерфейс, так как это самый простой подход и очень простой мастер для настройки параметров и зависимостей расширений. Тем не менее вы можете изучить другие альтернативы.

Цель Maven

Maven – это инструмент автоматизации сборки и управления проектами, широко используемый в разработке на Java. Цель – это конкретная задача, которую можно выполнить с помощью Maven. Такие задачи, как компиляция, упаковка и тестирование в проектах Maven, выполняются посредством запуска целей.

Нам предстоит создать проект Java 17 с помощью Maven и последней доступной версии Quarkus. Quarkus – это веб-инструмент, позволяющий создавать и настраивать проекты. Давайте воспользуемся этим, выполнив следующие шаги.

1. Введите в веб-браузере адрес https://code.quarkus.io.
   Перед вами откроется начальная страница мастера, как показано на рис. 1.1.

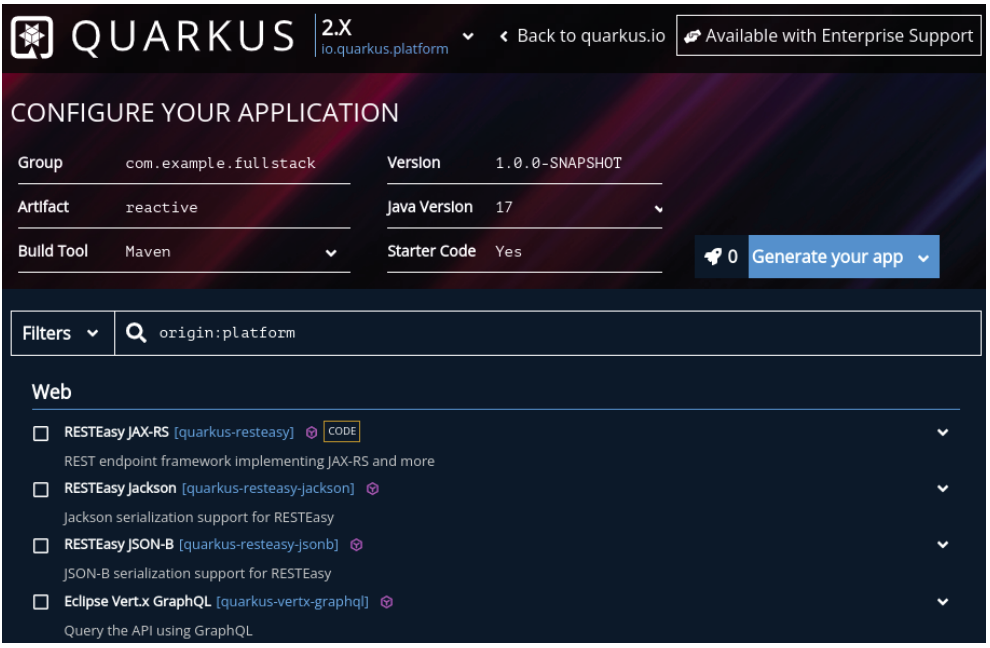

Рис 1.1 ❖ Веб-интерфейс code.quarkus.io

2. В поле **Group** (Группа) введите идентификатор группы проекта Maven. Сгенерированный проект будет включать пакет с этим именем, содержащий начальный фрагмент кода. Я буду использовать имя `com.example.fullstack`, но вы можете использовать любое другое имя по своему усмотрению.

3. В поле **Artifact** (Артефакт) укажите идентификатор артефакта Maven. Я буду использовать идентификатор `reactive`.
4. Выберите базовые зависимости проекта. На данный момент мы добавим только зависимость, позволяющую реализовать конечные точки HTTP. Мы будем использовать реактивные функции Quarkus, поэтому добавим зависимость **RESTEasy Reactive**, установив флажок напротив нее. Кстати, список зависимостей можно отфильтровать, введя требуемый текст в текстовое поле **Filters** (Фильтры), как показано на рис. 1.2.

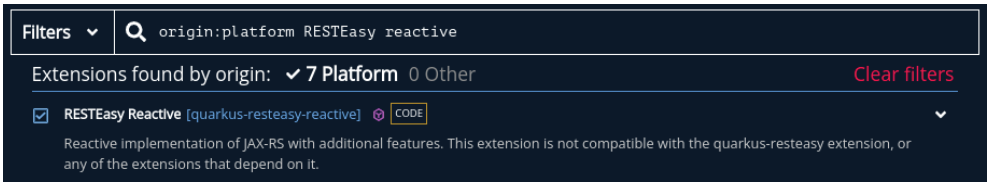

Рис. 1.2 ❖ Выбор зависимостей в веб-интерфейсе code.quarkus.io

   В следующих главах мы продолжим добавлять зависимости по мере расширения функциональности проекта.
5. Щелкните на кнопке **Generate your app** (Создать приложение) и загрузите ZIP-архив с заготовкой проекта.

Создав проект приложения Quarkus, познакомимся с его структурой и посмотрим, какие функции предоставляют зависимости и плагины, подключенные к проекту.

Структура проекта и зависимости

Чтобы начать исследование структуры проекта, выполните следующие шаги.

1. Разархивируйте ZIP-файл, загруженный на предыдущем этапе в папку, где будет находиться исходный код проекта. Я распаковал его в каталог, специально выделенный для проектов, – вы можете распаковать его в любую папку у себя, но обязательно запомните путь к этой папке, потому что над этим проектом мы будем работать на протяжении всей книги.
2. Откройте проект в IntelliJ IDEA или IDE по своему выбору.
   Среда разработки IntelliJ должна автоматически проанализировать ваш проект Maven и загрузить его зависимости. Если этого не случилось, то выполните этот шаг вручную, щелкнув правой кнопкой мыши на файле `pom.xml` и выбрав в меню пункт **Add as Maven Project** (Добавить как проект Maven), как показано на рис. 1.3.

Теперь исследуем содержимое и структуру проекта, а также сгенерированный пример кода.

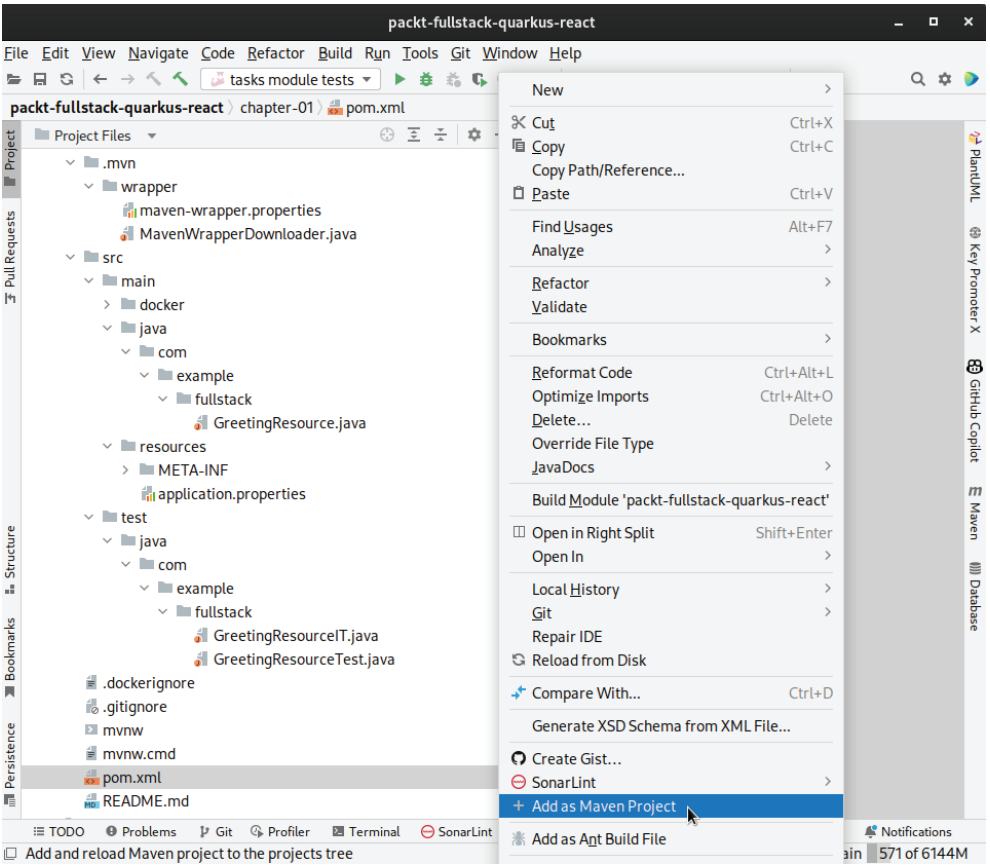

Рис. 1.3 ❖ IntelliJ IDEA и пункт **Add as Maven Project** (Добавить как проект Maven) в контекстном меню

Maven Wrapper

Проект включает **Maven Wrapper** – инструмент, позволяющий запускать согласованную версию Maven в разных окружениях сборки. Инструмент также позволяет запускать Maven без глобальной установки Maven. Проект включает каталог `.mvn` и выполняемые файлы `mvnw` и `mvnw.cmd`.

Цели Maven можно вызывать из терминала, находясь в корневой папке проекта. В Linux или macOS оболочку Maven Wrapper можно запустить командой `./mvnw`. В Windows то же самое можно сделать, выполнив команду `./mvnw` в терминале PowerShell или команду `mvnw` в стандартном терминале **cmd.exe**. Теперь сосредоточимся на изучении конфигурации проекта Maven и исследуем каждый ее раздел.

Проект Maven (pom.xml)

Настройки проекта Maven определены в файле `pom.xml`, содержащем **объектную модель проекта**. Этот **XML**-файл является основной единицей работы для Maven и хранит всю информацию о конфигурации, которая будет использоваться Maven для сборки проекта.

Рассмотрим некоторые разделы файла `pom.xml`, загруженного с веб-сайта Quarkus.

Координаты Maven (GAV)

Координаты Maven, также известные как GAV, – это минимально необходимые ссылки для проекта. Сюда входят поля `groupId`, `artifactId` и `version`, которые мы определили в веб-мастере при создании заготовки проекта:

```
<groupId>com.example.fullstack</groupId>
<artifactId>reactive</artifactId>
<version>1.0.0-SNAPSHOT</version>
```

Эти поля играют роль уникального идентификатора проекта и позволяют ссылаться на него из других проектов точно так же, как на систему координат.

Свойства Maven

Проект поставляется с набором предопределенных свойств в блоке `<properties>`. Наиболее важными являются следующие:

- `maven.compiler.release`

  ```
  <maven.compiler.release>17</maven.compiler.release>
  ```

  Это свойство определяет версию Java для проекта. В данном случае для исходного кода и для целевых классов потребуется версия Java 17. Это свойство используется плагином компилятора Maven (Maven Compiler Plugin) и было добавлено в версии 3.6 плагина. Оно зависит от другого свойства `compiler-plugin.version`, которое вы не должны изменять, по крайней мере убедитесь, что оно всегда выше 3.6;

- `quarkus.platform.version`

  ```
  <quarkus.platform.version>2.10.2.Final</quarkus.platform.version>
  ```

  Это свойство определяет используемую версию Quarkus. Всякий раз, когда выходит новая версия Quarkus, это свойство следует обновлять для обновления вашего проекта. Этого изменения должно быть достаточно для версий с исправлениями и выпусков, не нарушающих обратную совместимость. Для других обновлений может потребоваться изменить некоторые части кода.

Управление зависимостями

Файл pom.xml содержит блок **управления зависимостями** со следующим содержимым:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>${quarkus.platform.artifact-id}
      </artifactId>
      <version>${quarkus.platform.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Этот блок важен для настройки версии расширения Quarkus. Он использует заполнители для следующих свойств Maven, находящихся в разделе `properties`, для ссылки на действующую зависимость:

```
<quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
<quarkus.platform.group-id>io.quarkus.platform</quarkus.platform.group-id>
```

За кулисами Maven копирует раздел управления зависимостями из артефакта Quarkus `io.quarkus.platform:quarkus-bom` (Bill of Materials, BOM) в текущий проект. Это обеспечивает использование согласованной версии для всех расширений Quarkus, которые мы увидим в следующем разделе «Зависимости».

Зависимости

Следующий блок в объектной модели проекта – это определение фактических **зависимостей** библиотеки нашего проекта. Давайте рассмотрим каждую из загруженных зависимостей.

RESTEasy Reactive

В этой книге исследуем новые реактивные возможности Quarkus. RESTEasy Reactive – это реализация спецификации **JAX-RS** для Quarkus, основанная на **Vert.x**. Она в полной мере использует реактивные неблокирующие возможности Quarkus, повышающие общую производительность приложения. Следующий фрагмент кода определяет зависимость для этой библиотеки:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-reactive</artifactId>
</dependency>
```

JAX-RS – это спецификация Java EE или Jakarta EE API, позволяющая реализовать веб-службы REST. Она предоставляет общие аннотации, такие как

`@Path`, `@GET` и `@POST`, которые можно использовать для аннотирования классов и методов, реализующих конечные точки HTTP. Если вы уже имели дело с J2EE, Java EE или Jakarta EE, то, возможно, знакомы с этими аннотациями.

В этом заключается одно из основных преимуществ Quarkus. Кривая обучения очень пологая, потому что многое основано на проверенных стандартах и библиотеках.

Quarkus ArC

Quarkus ArC – это решение для внедрения зависимостей, предоставляемое фреймворком Quarkus. Оно основано на спецификации Java EE CDI 2.0 – проверенном и давнем стандарте. Следующий фрагмент кода определяет эту зависимость:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-arc</artifactId>
</dependency>
```

Одно из преимуществ ArC и большинства расширений Quarkus в целом – ориентированность на время сборки. Большинство оптимизаций выполняются во время сборки, что помогает исключить анализ и оптимизацию во время запуска приложения. В результате приложение запускается практически мгновенно.

Quarkus JUnit5

Quarkus JUnit5 – это основная зависимость фреймворка тестирования в Quarkus. Она предоставляет аннотацию `@QuarkusTest`, которая служит основной точкой входа для окружения тестирования. Следующий фрагмент кода настраивает эту зависимость:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
  <scope>test</scope>
</dependency>
```

Более подробно эту зависимость и ее особенности мы рассмотрим в главе 5 «Тестирование серверной части».

Rest Assured

Rest Assured – это последняя зависимость окружения тестирования, загруженная в проект. Она не входит в состав фреймворка Quarkus, но рекомендуется для проверки его конечных точек. Следующий фрагмент кода определяет эту зависимость; обратите внимание, что значение `groupId` отличается от `io.quarkus`:

```
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
```

```
  <scope>test</scope>
</dependency>
```

Мы будем использовать ее для создания интеграционных тестов нашего приложения.

Плагины

Наряду с более распространенными плагинами Maven раздел плагинов сборки содержит также упоминание плагина **Quarkus Maven**. Этот плагин обеспечивает поддержку целей Maven для доступа к большинству функций Quarkus. Всякий раз, когда вызывается любая команда Maven с префиксом `quarkus:`, за ее выполнение будет отвечать этот плагин.

Профили

Последний раздел в файле `pom.xml` описывает профили. Начальный проект содержит один профиль с идентификатором `native` (рис. 1.4). Активировать этот профиль можно либо с помощью флага выбора профиля `-Pnative`, либо с помощью системного свойства `-Dnative` (см. конфигурацию активации):

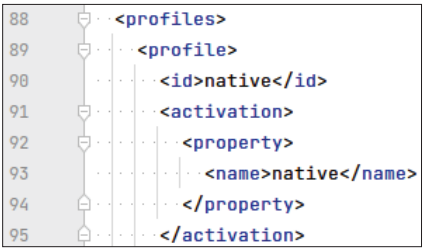

Рис. 1.4 ❖ Начало раздела profiles в файле pom.xml

Профиль определяет некоторые настройки, необходимые для запуска тестов, которые частично переопределяют настройки, указанные в разделе сборки или плагинов. Наиболее важным является свойство `quarkus.package.type`. Оно сообщает фреймворку Quarkus о необходимости создания двоичного файла для конкретной платформы. При упаковке приложения с этим профилем (`./mvnw clean package -Pnative`) мы получим двоичный файл вместо стандартного пакета **архива Java (JAR)**.

Подробнее о профилях мы поговорим в главе 6 «Создание двоичного образа».

Исходные файлы

Начальный проект имеет структуру, характерную для типичного проекта Java. Помимо файла проекта `pom.xml`, в корневом каталоге находится подкаталог `src`, содержащий исходный код проекта.

Свойства приложения

В каталоге `src/main/resources` находится файл `application.properties`. Он содержит основную конфигурацию проекта. Мы будем изменять конфигурацию и поведение приложения, добавляя и изменяя записи в этом файле.

За кулисами Quarkus использует **SmallRye Config** – реализацию спецификации **Eclipse MicroProfile Configuration**. Это еще один проверенный временем стандарт, на котором основан Quarkus.

Это стандартный файл свойств. Каждая запись размещается в отдельной строке. Ключ (имя конфигурационного параметра) и его значение разделены знаком `=`.

Например, вот как выглядит настройка порта сервера приложений:

```
quarkus.http.port=8082
```

Файл `application.properties` также можно использовать для определения значений, внедряемых в приложение.

Допустим, вы определили следующее свойство:

```
publisher.name=Packt
```

Его можно внедрить в приложение, добавив следующий фрагмент кода:

```
@ConfigProperty(name = "publisher.name")
String publisherName;
```

Профили

Quarkus поддерживает возможность сборки и запуска приложения с использованием разных конфигурационных **профилей**. В зависимости от целевого **окружения** может потребоваться выбрать конкретный профиль, определяющий требуемую конфигурацию.

В Quarkus есть три профиля: `dev`, активирующийся в режиме разработки; `test`, активирующийся при выполнении тестов, и `prod`, используемый по умолчанию, когда не применяются другие профили.

Все профили определяются в одном и том же файле; чтобы настроить параметр для определенного профиля, необходимо добавить к его имени префикс `%` и имя профиля, но это не относится к профилю `prod`, который является профилем по умолчанию и не требует указывать префикс.

Продолжая предыдущий пример, мы можем настроить порт сервера в режиме разработки следующим образом:

```
%dev.quarkus.http.port=8082
```

В общем случае принято определять конфигурацию для профиля `prod` и при необходимости добавлять параметры для профиля `dev`.

Статические ресурсы

Проект содержит файл `index.html` в каталоге `src/main/resources/META-INF/resources`. Этот файл будет автоматически обслуживаться HTTP-сервером ба-

зового приложения. После ввода в браузере корневого пути к приложению (`http://localhost:8080`) вас встретит эта целевая страница (рис. 1.5).

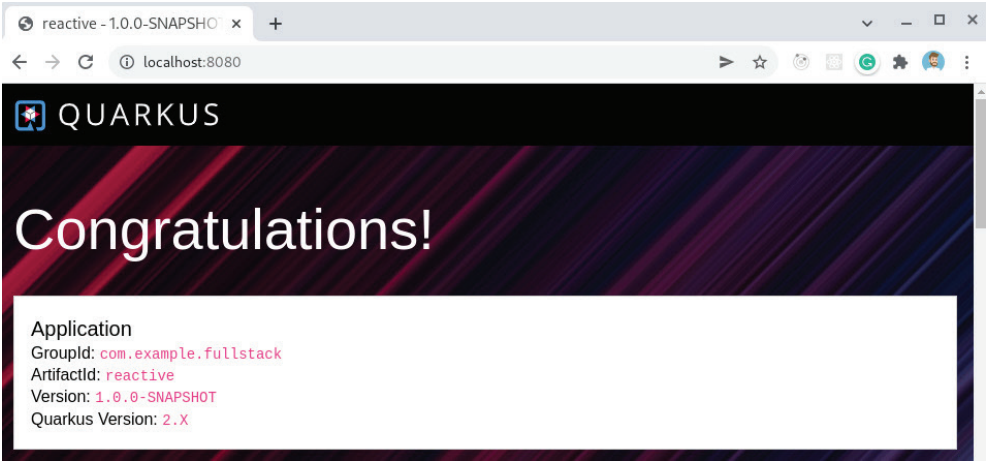

Рис. 1.5 ❖ Начальная страница приложения в браузере (http://localhost:8080)

По умолчанию Quarkus будет обслуживать любые статические файлы, находящиеся в этом каталоге. Однако в нашем приложении мы будем использовать альтернативный метод, потому что текущий подход не совместим с внешней маршрутизацией. В главе 11 «Интеграция с Quarkus» я покажу, как реализовать шлюз API, который будет использоваться как альтернативное решение для обслуживания статических ресурсов.

Java-код

Начальный проект содержит пример кода. Класс `GreetingResource` находится в стандартном каталоге `src/main/java` в пакете `com.example.fullstack`. Для этого класса также имеются два теста в каталоге `src/main/test` в том же пакете: `GreetingResourceTest` и `GreetingResourceIT`. Новый код, сгруппированный по функциям, мы будем размещать в том же корневом пакете.

Файлы Docker

Проект содержит несколько примеров файлов **Docker** в каталоге `src/main/docker`. Эти файлы можно использовать для создания **образов контейнеров** приложения. В главе 12 «Развертывание приложения в Kubernetes» я покажу, как создавать образы контейнеров. Однако мы будем использовать Eclipse JKube, требующий более простой конфигурации и не нуждающийся в файлах Docker. JKube – это плагин Maven, генерирующий все необходимые конфигурации для приложения, чтобы развернуть его в Kubernetes; по этой причине нет необходимости хранить дополнительные файлы конфигурации, такие как YAML-файлы Docker или Kubernetes.

Теперь, познакомившись со структурой файлов и каталогов начального проекта, посмотрим, как выполнять основные задачи, связанные с разработкой новых функций, а также развертыванием и запуском приложения.

Режим разработки

В течение многих лет одной из основных проблем для разработчиков Java было отсутствие или очень небольшая поддержка **горячей перезагрузки,** или **перезагрузки в реальном времени**. Традиционно, когда вносятся некоторые изменения в код, приходится перекомпилировать его, упаковывать и повторно развертывать. Этот процесс может занимать от нескольких секунд до нескольких минут или даже часов. Обычно это один из недостатков, который упоминается, когда люди сравнивают Java с другими языками программирования.

Одна из главных целей Quarkus – вернуть радость творчества разработчикам, поэтому, естественно, это был один из приоритетных вопросов. **Режим разработки** в Quarkus запускает приложение и следит за изменениями в коде. Всякий раз, когда меняется исходный код или файлы ресурсов приложения Java, Quarkus обнаруживает эти изменения и выполняет горячее развертывание. Вам остается только обновить страницу в браузере, чтобы изменения вступили в силу.

Запустить режим разработки можно запуском цели `quarkus:dev` Maven из корневого каталога проекта:

```
./mvnw quarkus:dev
```

Вы увидите результат, изображенный на рис. 1.6.

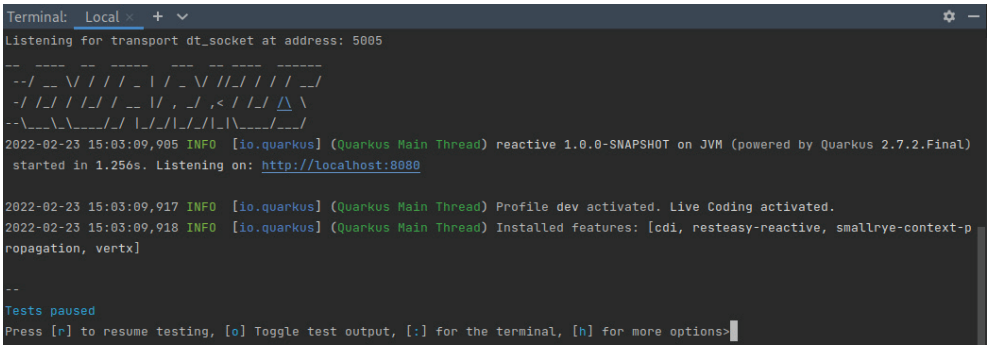

Рис. 1.6 ❖ Терминал IntelliJ, работающий в режиме разработки Quarkus

Прочитав предыдущие сообщения, можно заметить, что Quarkus автоматически выбрал профиль `dev` и запустил режим «живого кодирования» (live coding).

Теперь можно ввести в браузере URL начальной конечной точки (`http://localhost:8080/hello`). Если все пройдет успешно, браузер покажет сообщение **Hello RESTEasy Reactive**, как видно на рис. 1.7.

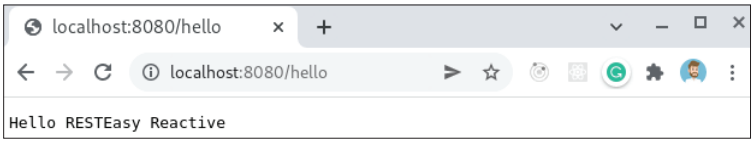

Рис. 1.7 ❖ Конечная точка hello

Определение этой конечной точки можно увидеть в классе `GreetingResource`. Мы можем изменить приветственное сообщение на какое-нибудь другое:

```
@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello() {
    return "Hello Quarkus live coding!";
}
```

В традиционном мире Java теперь нужно было бы перекомпилировать и повторно развернуть приложение, чтобы увидеть изменения. Однако в данный момент достаточно обновить страницу в браузере – и наше измененное сообщение должно появиться.

Отладка в режиме разработки

Внимательно изучив сообщения на рис. 1.6, можно заметить, что Quarkus также открыл порт удаленной отладки:

```
Listening for transport dt_socket at address: 5005
```

Благодаря этому появляется возможность запустить сеанс отладки из IntelliJ IDEA. Для этого нужно создать новую конфигурацию отладки, выбрав в меню пункт **Run > Edit Configurations...** (Пуск > Редактировать конфигурации...), как показано на рис. 1.8.

В диалоге **Run/Debug Configurations** (Пуск/Конфигурации отладки) нужно создать новую конфигурацию **Remote JVM Debug** (Отладка на удаленной JVM). Параметры по умолчанию вполне пригодны для использования с Quarkus, поэтому просто укажите имя для этой конфигурации, как показано на рис. 1.9.

Сохранив конфигурацию, можно запустить отладку и установить точку останова в определении нашей конечной точки. Если после этого обновить окно браузера, то отладчик должен остановиться в точке останова.

В сочетании с простотой отладки механизм автоматического развертывания изменений в реальном времени поможет вам существенно повысить свою производительность. Теперь, когда мы знаем, как использовать режим разработки Quarkus для реализации и отладки кода, давайте посмотрим, как запускать тесты для проверки приложения.

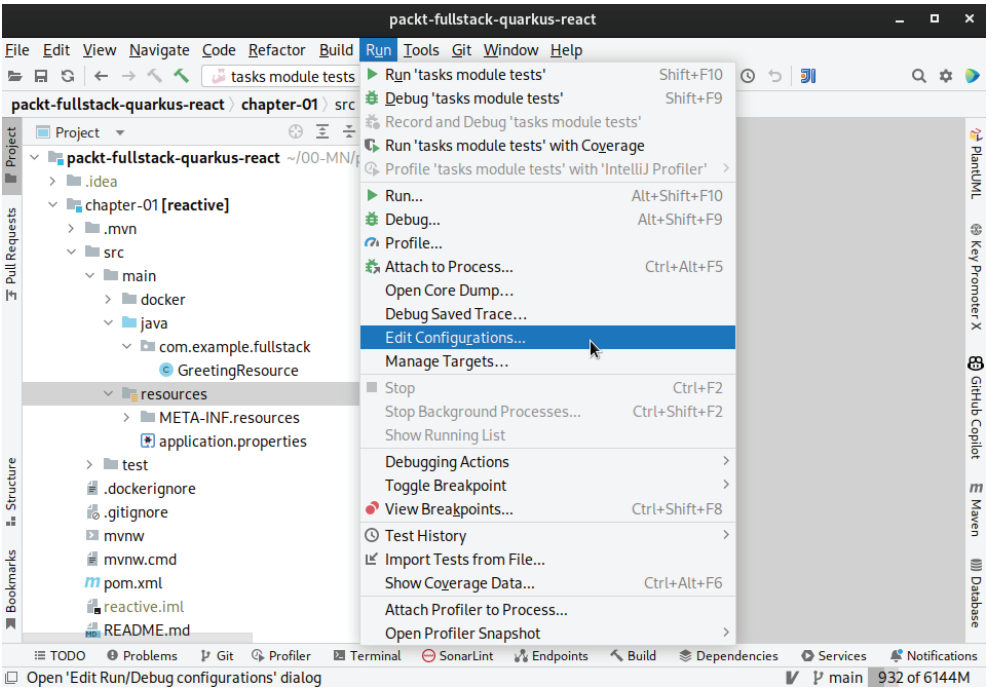

Рис. 1.8 ❖ Меню **Run** (Пуск) в IntelliJ IDEA

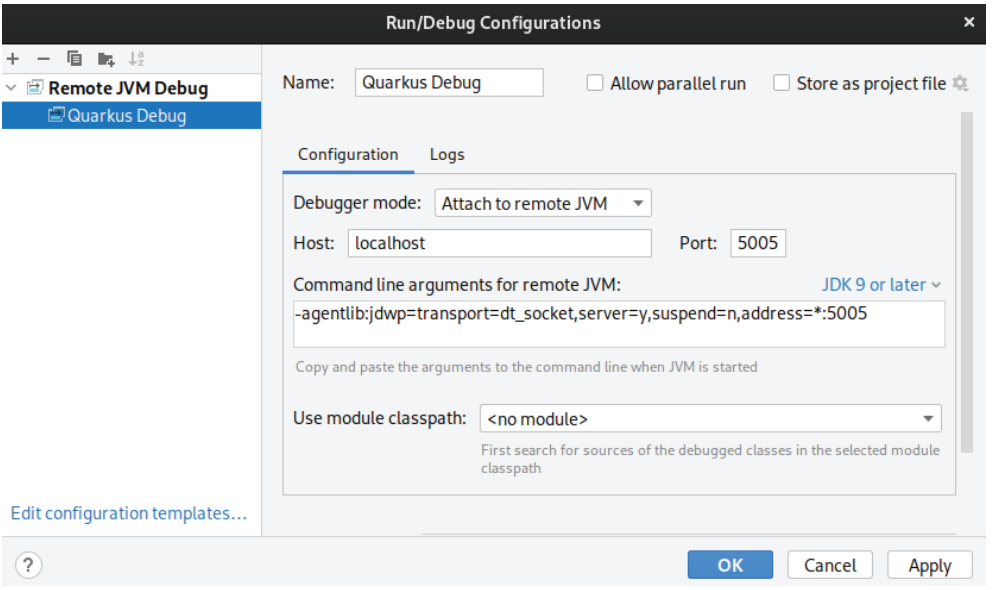

Рис. 1.9 ❖ Конфигурации отладки в IntelliJ IDEA для Quarkus

Непрерывное тестирование

Одна из замечательных особенностей Quarkus 2.X – возможность непрерывного тестирования. Эта функция заимствована из других языков программирования, таких как Ruby, которые уже давно поддерживают ее. Кроме того, это еще один шаг к достижению главной цели фреймворка Quarkus – вернуть Java-разработчикам радость творчества.

Для пользователей, практикующих **разработку через тестирование (Test-Driven Development, TDD)**, эта функция поможет еще больше повысить их продуктивность. В обычном процессе TDD разработчики сначала пишут тест для функции, а затем реализуют код, который обеспечит прохождение этого теста. Этот процесс повторяется для каждой новой функции и после каждого рефакторинга кода. Непрерывное тестирование обеспечивает мгновенную обратную связь и позволяет разработчику сосредоточиться на реализации, а не на процессе.

Когда Quarkus работает в режиме непрерывного тестирования, он обнаруживает изменения как в коде, так и в тестах. Для каждого изменения он повторно запускает соответствующие тесты.

Запустить режим непрерывного тестирования можно с помощью команды Maven:

```
./mvnw quarkus:test
```

Как вы наверняка помните, в разделе «Режим разработки» мы изменили текст приветствия в классе `GreetingResource`, но не изменили тест. Первое, что мы увидим, когда запустим Maven-цель `quarkus:test`, – ошибку теста (рис. 1.10).

Рис. 1.10 ❖ Ошибка после запуска цели quarkus:test

Чтобы исправить ошибку, откроем `GreetingResourceTest` и обновим текст ожидаемого ответа: «Hello Quarkus live coding!»:

```
@Test
public void testHelloEndpoint() {
    given()
      .when().get("/hello")
      .then()
         .statusCode(200)
         .body(is("Hello Quarkus live coding!"));
}
```

После сохранения изменений тест должен автоматически перезапуститься и стать зеленым (рис. 1.11).

Рис. 1.11 ❖ Тест после запуска цели quarkus:test выполнился успешно

Следуя практике TDD, если бы возникла потребность создать конечную точку `/hello/world`, то первым шагом было бы добавление нового теста:

```
@Test
public void testHelloWorldEndpoint() {
    given()
      .when().get("/hello/world")
      .then()
      .statusCode(200)
      .body(is("Hello world!"));
}
```

Реализации этой конечной точки пока нет, поэтому тестирование завершится ошибкой. Далее можно реализовать саму конечную точку, чтобы удовлетворить требование теста:

```
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/world")
public String helloWorld() {
    return "Hello world!";
}
```

Следующим шагом после успешного прохождения тестов может быть получение значения конечной точки из внешней службы. Итак, мы изменили сначала тест, а затем реализацию и снова запустили цикл. Теперь должно быть понятно, как непрерывное тестирование заметно улучшает процесс разработки.

Методика TDD гарантирует, что функции, определенные в предоставленных спецификациях, успешно проходят тестирование, что помогает писать код с меньшим количеством ошибок и тратить меньше времени на длительные сеансы отладки. Теперь, познакомившись с поддержкой методики TDD в Quarkus, давайте посмотрим, как упаковать приложение для его распространения.

Упаковка приложения

Последний шаг перед распространением и запуском приложения – его упаковка. Помимо основного режима упаковки в двоичном формате, который мы анализировали в разделе «Профили», Quarkus предлагает следующие варианты упаковки:

- **fast-jar**: вариант упаковки по умолчанию. Создает оптимизированный выполняемый JAR-пакет вместе с каталогом и его зависимостями;

- **uber-jar**: в этом варианте создается большой JAR-файл со всеми необходимыми зависимостями. Этот вариант лучше всего подходит для распространения автономных приложений;
- **native**: в этом варианте GraalVM упаковывает приложение в один двоичный выполняемый файл для вашей платформы;
- **native-sources**: этот вариант упаковки предназначен для опытных пользователей. Он генерирует файлы, которые потребуются GraalVM для создания двоичного образа, и похож на вариант **native**, но прерывает упаковку перед фактическим вызовом GraalVM, что позволяет выполнить вызов GraalVM отдельно, например в конвейере CI/CD.

Выбрать вариант упаковки можно с помощью Maven-свойства `quarkus.package.type`, в разделе `properties` в файле `pom.xml` или через командную строку при запуске команд Maven:

```
./mvnw -D"quarkus.package.type=uber-jar" clean package
```

В данный момент мы используем вариант упаковки по умолчанию. Упаковать приложение можно командой

```
./mvnw clean package
```

Если все прошло успешно, вы сможете запустить приложение:

```
java -jar target/quarkus-app/quarkus-run.jar
```

И открыть в браузере страницу `http://localhost:8080` или любую из конечных точек HTTP, созданных на предыдущих шагах.

Итоги

В этой главе мы кратко познакомились с Quarkus и увидели его основные преимущества и достоинства. Мы создали новый проект Quarkus с помощью веб-сайта Quarkus и открыли его в IntelliJ IDEA. Рассмотрели содержимое и структуру начального проекта. Проверили зависимости, плагины и сгенерированный исходный код и узнали, как упаковать и запустить пример приложения.

Теперь у вас должно сложиться общее представление о Quarkus, достаточное, чтобы приступить к реализации полнофункционального веб-приложения, которое мы будем разрабатывать на протяжении всей книги. В следующей главе вы увидите, как добавить в приложение диспетчера задач слой хранения данных, как определить классы сущностей и подключить их к базе данных.

Вопросы

1. Что такое Quarkus?
2. Как создать проект Quarkus с нуля?
3. Как запустить проект Quarkus в режиме разработки?
4. Что такое TDD?
5. Как упаковать проект Quarkus?