

Оглавление

Об авторе	16
О научных редакторах	17
Благодарности	18
Глава 1. Фреймворк Spring 5	19
Немного истории	19
Принципы и паттерны проектирования	20
Фреймворк Spring 5	21
Простое веб-приложение Spring	22
Использование Maven для создания проекта	23
Добавление зависимостей	23
Веб-конфигурация Spring	26
Классы	33
Запуск приложения	38
Использование Java-конфигурации	40
Резюме	44
Глава 2. Введение в Spring Boot	45
Spring Boot	45
Spring Boot спешит на помощь	47
Spring Boot CLI	48

Модель приложения Spring Boot	50
Почему Spring Boot?	55
Резюме	58
Глава 3. Внутреннее устройство и возможности Spring Boot	59
Автоматическая конфигурация	59
Отключение конкретных автоконфигурационных классов	61
Аннотации @EnableAutoConfiguration и @Enable<технология>	63
Возможности Spring Boot	67
Класс SpringApplication	70
Пользовательский баннер	71
Класс SpringApplicationBuilder	75
Аргументы приложения	78
Интерфейсы ApplicationRunner и CommandLineRunner	80
Конфигурация приложения	82
Примеры использования свойств конфигурации	84
Пользовательский префикс для свойств	91
Резюме	95
Глава 4. Создание веб-приложений	96
Spring MVC	96
Автоконфигурация Spring Boot MVC	97
Spring Boot Web: приложение ToDo	99
Приложение ToDo	100
Запуск: приложение ToDo	111
Тестирование: приложение ToDo	112

Spring Boot Web: переопределение настроек по умолчанию	117
Переопределение настроек сервера	117
Формат даты JSON	118
Content-Type: JSON/XML	119
Spring MVC: переопределение настроек по умолчанию	120
Использование другого контейнера приложения	121
Spring Boot Web: клиент.....	122
Клиентское приложение ToDo	122
Резюме	130
Глава 5. Доступ к данным	131
Базы данных SQL	131
Spring Data	132
Spring JDBC.....	133
Работа с JDBC в Spring Boot	134
Приложение ToDo с использованием JDBC	135
Spring Data JPA	142
Использование Spring Data JPA со Spring Boot.....	143
Создание приложения ToDo с использованием Spring Data JPA	144
Spring Data REST	151
Spring Data REST и Spring Boot	152
Приложение ToDo с Spring Data JPA и Spring Data REST	152
Базы данных NoSQL	159
Spring Data MongoDB	159
Использование Spring Data MongoDB со Spring Boot	160
Приложение ToDo с использованием Spring Data MongoDB	162
Приложение ToDo со Spring Data MongoDB REST	165

Spring Data Redis.....	166
Использование Spring Data Redis со Spring Boot	166
Приложение ToDo со Spring Data Redis.....	166
Дополнительные возможности по работе с данными с помощью Spring Boot.....	170
Резюме	171
Глава 6. Работа с WebFlux и Reactive Data	172
Реактивные системы	172
Манифест реактивных систем	173
Project Reactor	174
Создание приложения ToDo с использованием Reactor.....	175
WebFlux	183
WebClient	184
WebFlux и автоконфигурация Spring Boot.....	185
Использование WebFlux со Spring Boot.....	186
Реактивные данные	193
Реактивные потоки данных MongoDB.....	193
Резюме	202
Глава 7. Тестирование	203
Фреймворк тестирования Spring.....	203
Фреймворк тестирования Spring Boot.....	205
Тестирование конечных точек веб-приложения	206
Имитация компонент.....	207
Тестовые срезы Spring Boot.....	208
Резюме	214

Глава 8. Безопасность	215
Spring Security.....	215
Обеспечение безопасности с помощью Spring Boot	216
Приложение ToDo с базовым уровнем безопасности	217
Переопределяем безопасность базового уровня	222
Переопределение используемой по умолчанию страницы входа	224
Пользовательская страница входа	226
Безопасность при использовании JDBC	233
Создание приложения-справочника с использованием средств безопасности JDBC.....	233
Использование приложения Directory в приложении ToDo	241
Безопасность WebFlux.....	246
Создание приложения ToDo с OAuth2.....	247
Создание приложения ToDo в GitHub	250
Резюме	256
Глава 9. Обмен сообщениями	257
Что такое обмен сообщениями.....	257
Использование JMS со Spring Boot	258
Создание приложения ToDo с использованием JMS.....	258
Использование паттерна публикации/подписки JMS	268
Удаленный сервер ActiveMQ.....	269
Использование RabbitMQ со Spring Boot.....	269
Установка RabbitMQ	270
RabbitMQ/AMQP: точки обмена, привязки и очереди	270
Создание приложения ToDo с помощью RabbitMQ.....	272
Удаленный сервер RabbitMQ	282

Обмен сообщениями в Redis с помощью Spring Boot	282
Установка Redis	283
Создание приложения ToDo с использованием Redis.....	283
Удаленный сервер Redis.....	290
Использование WebSockets со Spring Boot.....	290
Создание приложения ToDo с использованием WebSockets	290
Резюме	300
Глава 10. Spring Boot Actuator	301
Spring Boot Actuator	302
Создание приложения ToDo с использованием Actuator	302
/actuator	307
/actuator/conditions	308
/actuator/beans.....	309
/actuator/configprops	310
/actuator/threaddump	310
/actuator/env	311
/actuator/health	312
/actuator/info.....	313
/actuator/loggers.....	313
/actuator/loggers/{name}	314
/actuator/metrics.....	315
/actuator/mappings	316
/actuator/shutdown	317
/actuator/httptrace	319
Изменение идентификатора конечной точки.....	320
Поддержка CORS в Spring Boot Actuator	320

Изменение пути конечных точек управления приложением	321
Обеспечение безопасности конечных точек	321
Настройка конечных точек	322
Реализация пользовательских конечных точек актуатора	322
Создание приложения ToDo с пользовательскими конечными точками актуатора	323
Конечная точка health Spring Boot Actuator	331
Создание приложения ToDo с пользовательским индикатором состояния приложения	334
Метрики Spring Boot Actuator	339
Создание приложения ToDo с Micrometer: Prometheus и Grafana	339
Получение общей статистики для Spring Boot с помощью Grafana	350
Резюме	352
Глава 11. Создание приложений Spring Integration и Spring Cloud Stream	353
Азбука Spring Integration	354
Программирование Spring Integration	357
Использование XML	362
Использование аннотаций	364
Использование JavaConfig	366
Приложение ToDo с интеграцией чтения файлов	367
Spring Cloud Stream	372
Spring Cloud	372
Spring Cloud Stream	374
«Стартовые пакеты» для приложений Spring Cloud Stream	396
Резюме	397

Глава 12. Spring Boot в облаке	398
Облачная и нативная облачная архитектура.....	398
Приложения на основе 12 факторов	399
Микросервисы	401
Подготовка приложения ToDo как микросервиса.....	402
Платформа Pivotal Cloud Foundry.....	404
PAS: сервис приложений Pivotal	406
Возможности PAS.....	407
Использование PWS/PAS	408
Cloud Foundry CLI: интерфейс командной строки	410
Вход в PWS/PAS с помощью утилиты CLI.....	410
Развертывание приложения ToDo в PAS	411
Создание сервисов.....	414
Убираем за собой.....	418
Резюме	419
Глава 13. Расширение возможностей Spring Boot	420
Создание spring-boot-starter	420
Модуль todo-client-spring-boot-starter	422
Модуль todo-client-spring-boot-autoconfigure	423
Создание функциональности @Enable*	431
Сервис REST API приложения ToDo	434
Установка и тестирование	437
Проект Task	437
Запуск приложения Task	440
Резюме	441

Приложение. Интерфейс командной строки Spring Boot	442
Spring Boot CLI.....	442
Команда run.....	444
Команда test.....	446
Команда grab.....	449
Команда jar.....	450
Команда war.....	451
Команда install.....	452
Команда uninstall.....	453
Команда init.....	454
Примеры использования команды init.....	456
Альтернатива команде init.....	458
Команда shell.....	458
Команда help.....	459
Резюме.....	460

Оставшаяся часть приложения уже вам знакома. Если снова запустить приложение, вы увидите бесконечную отправку сообщений. Взгляните во время его работы на консоль RabbitMQ. Можно также вставить цикл `for` для отправки большего числа сообщений за полсекунды.

Удаленный сервер RabbitMQ

Для доступа к удаленному серверу RabbitMQ необходимо добавить в файл `application.properties` следующие свойства¹:

```
spring.rabbitmq.host=mydomain.com
spring.rabbitmq.username=rabbituser
spring.rabbitmq.password=thisissecured
spring.rabbitmq.port=5672
spring.rabbitmq.virtual-host=/production
```

Вы всегда можете узнать обо всех свойствах RabbitMQ в документации Spring Boot по адресу <https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>.

Теперь вы знаете, насколько легко использовать RabbitMQ со Spring Boot. Если хотите узнать больше о RabbitMQ и технологии Spring AMQP, то можете получить дополнительную информацию на основной странице этого проекта: <http://projects.spring.io/spring-amqp/>.

Можете теперь остановить сервер RabbitMQ, нажав `Ctrl+C` в терминале, где запустили его. Существуют и другие возможности использования RabbitMQ, например создание кластера и обеспечение высокой доступности. Больше информации об этом можно найти на сайте www.rabbitmq.com.

Обмен сообщениями в Redis с помощью Spring Boot

Настала очередь Redis. Redis (REmote DIctionary Server) представляет собой базу данных NoSQL — хранилище пар «ключ/значение». Redis написана на языке C и, несмотря на небольшой объем используемой оперативной памяти, отличается высокой надежностью, масштабируемостью, быстродействием и большими возможностями. Основная ее задача — хранение структур данных,

¹ Разумеется, поменяв соответствующим образом их значения.

таких как списки, хеши, строковые значения, множества и отсортированные множества. Одна из основных ее возможностей — обмен сообщениями по типу «публикация/подписка». Поэтому мы и воспользуемся Redis в качестве брокера сообщений.

Установка Redis

Процедура установки Redis очень проста. Если вы используете Mac OS X/Linux, можете воспользоваться командой `brew` и выполнить следующее:

```
$ brew update && brew install redis
```

Если же вы работаете в Unix или Windows-системе, то можете зайти на веб-сайт Redis и скачать один из установочных пакетов Redis¹ по адресу <http://redis.io/download>.

Создание приложения ToDo с использованием Redis

Использовать Redis для обмена сообщениями по типу «публикация/подписка» очень просто, никаких серьезных отличий от других технологий нет. Мы сейчас реализуем отправку и получение запланированных дел на основе паттерна публикации/подписки с помощью Redis.

Для начала откройте свой любимый браузер и перейдите на сайт Spring Initializr. Добавьте следующие значения в соответствующие поля.

- Group (Группа): `com.apress.todo`.
- Artifact (Артефакт): `todo-redis`.
- Name (Название): `todo-redis`.
- Package Name (Название пакета): `com.apress.todo`.
- Dependencies (Зависимости): Redis, Web, Lombok, JPA, REST Repositories, H2, MySQL.

Можете выбрать в качестве типа проекта Maven или Gradle. Затем нажмите кнопку **Generate Project** (Сгенерировать проект) и скачайте ZIP-файл. Разархивируйте его и импортируйте в вашу любимую IDE (рис. 9.6).

¹ Уточнение: на этом сайте доступен только исходный код, который необходимо скомпилировать для получения исполняемых файлов. Впрочем, существуют и бинарные сборки, правда, для Windows они все довольно сильно отстают от текущей версии.

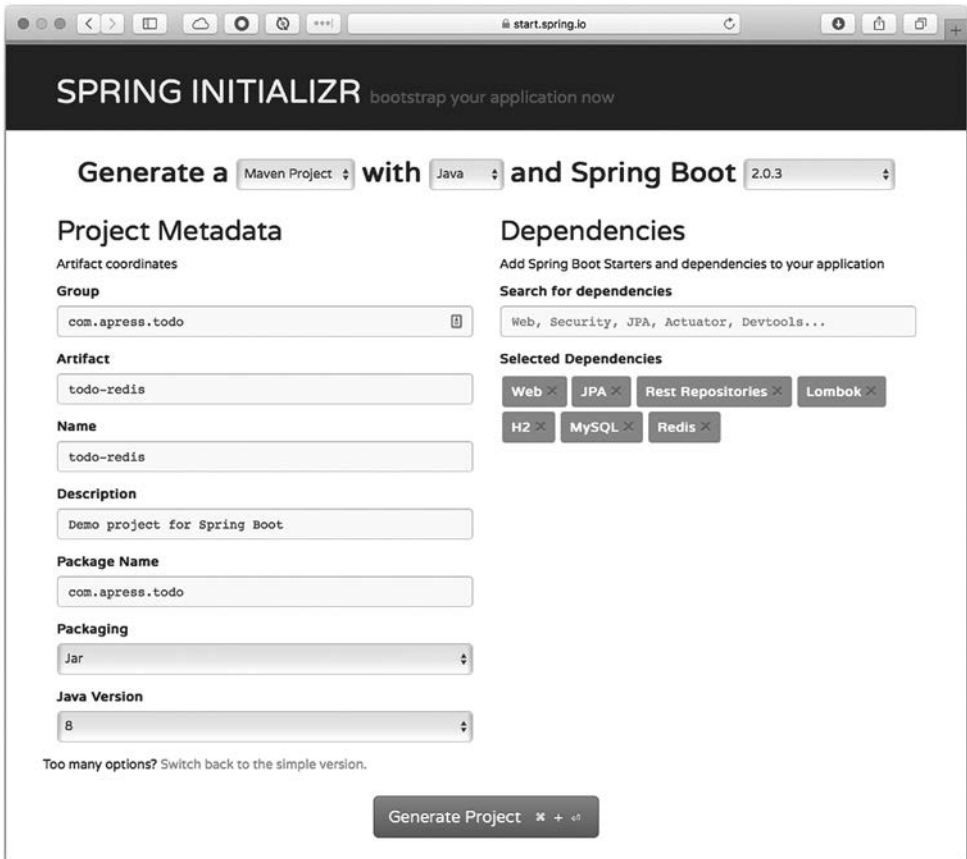


Рис. 9.6. Spring Initializr

Мы воспользуемся классом предметной области `ToDo` и репозиторием из предыдущих глав.

Генератор `ToDo`

Начнем с создания класса генератора, задача которого будет состоять в отправке экземпляра `ToDo` в конкретную тему (листинг 9.12).

Листинг 9.12. `com.apress.todo.redis.ToDoProducer.java`

```
package com.apress.todo.redis;

import com.apress.todo.domain.ToDo;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;

@Component
public class TodoProducer {

    private static final Logger log = LoggerFactory.getLogger(TodoProducer.
        class);
    private RedisTemplate redisTemplate;

    public TodoProducer(RedisTemplate redisTemplate){
        this.redisTemplate = redisTemplate;
    }

    public void sendTo(String topic, Todo todo){
        log.info("Producer> Todo sent");
        this.redisTemplate.convertAndSend(topic, todo);
    }
}
```

В листинге 9.12 приведен класс генератора. Он практически идентичен предыдущим технологиям. В нем используется класс паттерна **Template*; в данном случае *RedisTemplate*, отправляющий экземпляры *ToDo* в конкретную тему.

Потребитель *ToDo*

Далее создадим потребитель для подписки на тему (листинг 9.13).

Листинг 9.13. *com.apress.todo.redis.ToDoConsumer.java*

```
package com.apress.todo.redis;

import com.apress.todo.domain.ToDo;
import com.apress.todo.repository.ToDoRepository;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

@Component
public class TodoConsumer {

    private static final Logger log =
        LoggerFactory.getLogger(TodoConsumer.class);
    private ToDoRepository repository;
```

```
public TodoConsumer(TodoRepository repository){
    this.repository = repository;
}

public void handleMessage(Todo todo) {
    log.info("Consumer> " + todo);
    log.info("Todo created> " + this.repository.save(todo));
}
}
```

В листинге 9.13 приведен потребитель, который подписывается на тему для поступающих сообщений `Todo`. Важно понимать, что для использования прослушвателя обязательно нужен метод с названием `handleMessage` (это ограничение, налагаемое при создании `MessageListenerAdapter`).

Конфигурация приложения `ToDo`

Теперь создадим конфигурацию для нашего приложения `ToDo` (листинг 9.14).

Листинг 9.14. `com.apress.todo.config.ToDoConfig.java`

```
package com.apress.todo.config;

import com.apress.todo.domain.Todo;
import com.apress.todo.redis.TodoConsumer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.listener.PatternTopic;
import org.springframework.data.redis.listener.RedisMessageListenerContainer;
import org.springframework.data.redis.listener.adapter.MessageListenerAdapter;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;

@Configuration
public class ToDoConfig {

    @Bean
    public RedisMessageListenerContainer container(RedisConnectionFactory
        connectionFactory,
                                                MessageListenerAdapter todoListenerAdapter,
                                                @Value("${todo.redis.topic}") String topic) {
        RedisMessageListenerContainer container = new
            RedisMessageListenerContainer();
        container.setConnectionFactory(connectionFactory);
    }
}
```

```
        container.addListener(toDoListenerAdapter, new
            PatternTopic(topic));
        return container;
    }

    @Bean
    MessageListenerAdapter toDoListenerAdapter(ToDoConsumer consumer) {
        MessageListenerAdapter messageListenerAdapter =
            new MessageListenerAdapter(consumer);
        messageListenerAdapter.setSerializer(new
            Jackson2JsonRedisSerializer<>(ToDo.class));
        return messageListenerAdapter;
    }

    @Bean
    RedisTemplate<String, ToDo> redisTemplate(RedisConnectionFactory
        connectionFactory){
        RedisTemplate<String,ToDo> redisTemplate = new
            RedisTemplate<String,ToDo>();
        redisTemplate.setConnectionFactory(connectionFactory);
        redisTemplate.setDefaultSerializer(new Jackson2JsonRedisSerializer<>(
            ToDo.class));
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }
}
```

В листинге 9.14 приведена необходимая для приложения ToDo конфигурация. В этом классе объявлены следующие компоненты Spring.

- Класс `RedisMessageListenerContainer`. Этот класс отвечает за подключение к теме Redis.
- Класс `MessageListenerAdapter`. Этот адаптер использует класс простого Java-объекта в старом стиле (Plain Old Java Object, POJO) для обработки сообщения. Обязательным требованием является название `handleMessage` для обрабатываемого сообщения метода, который получает сообщение из темы в виде экземпляра `ToDo`, вследствие чего требуется также сериализатор.
- Класс `Jackson2JsonRedisSerializer`. Этот сериализатор производит преобразование из экземпляра `ToDo`/в него.
- Класс `RedisTemplate`. Этот класс реализует паттерн `Template` и аналогичен используемым для прочих технологий обмена сообщениями. Для работы с JSON и преобразованием из экземпляров `ToDo`/в них ему требуется сериализатор.

Эти дополнительные настройки необходимы для работы с форматом JSON и корректного преобразования из экземпляров `ToDo`/в них; но вы можете без них обойтись и воспользоваться конфигурацией по умолчанию, для которой требуется сериализуемый объект (например, типа `String`) для отправки. Вместо него можно применить `StringRedisTemplate`.

Добавьте в файл `application.properties` следующее содержимое:

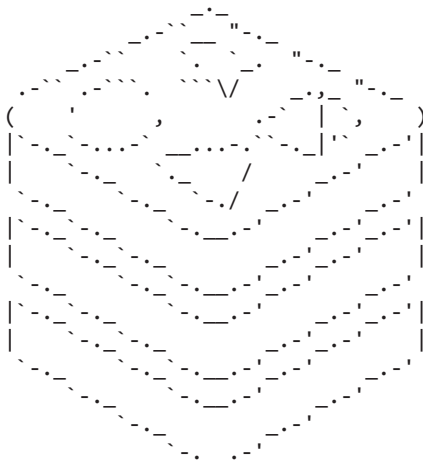
```
# JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=create-drop

# ToDo Redis
todo.redis.topic=todos
```

Запуск приложения `ToDo`

Перед запуском приложения `ToDo` убедитесь, что сервер `Redis` запущен. Для запуска сервера выполните в терминале следующую команду:

```
$ redis-server
89887:C 11 Feb 20:17:55.320 # Warning: no config file specified, using the
default config. In order to specify a config file use redis-server /path/
to/redis.conf
89887:M 11 Feb 20:17:55.321 * Increased maximum number of open files to
10032 (it was originally set to 256).
```



Redis 4.0.10 64 bit

Standalone mode
Port: 6379
PID: 89887

<http://redis.io>

```
89887:M 11 Feb 20:17:55.323 # Server started, Redis version 3.0.7
89887:M 11 Feb 20:17:55.323 * The server is now ready to accept connections
on port 6379
```


Этот вывод демонстрирует, что сервер Redis готов к использованию и прослушивает на порте 6379. Откройте новое окно терминала и выполните следующую команду:

```
$ redis-cli
```

Это клиент командной оболочки, который умеет подключаться к серверу Redis. Можете подписаться на тему `todos`, выполнив следующую команду.

```
127.0.0.1:6379> SUBSCRIBE todos
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "todos"
3) (integer) 1
```

Теперь можно запустить приложение как обычно (из IDE или с помощью Maven/Gradle). Если вы используете Maven, выполните:

```
$ ./mvnw spring-boot:run
```

После выполнения этой команды вы должны увидеть в журнале примерно следующее:

```
...
Producer> Message Sent
Consumer> ToDo(id=null, description=workout tomorrow morning!,
  created=null, modified=null, completed=false)
ToDo created> ToDo(id=8a808087645bd67001645bd6785b0000, description=workout
  tomorrow morning!, created=2018-07-02T10:32:19.546, modified=2018-07-
  02T10:32:19.547, completed=false)
...
```

Если теперь заглянуть в командную оболочку Redis, вы должны увидеть там что-то вроде следующего:

```
1) "message"
2) "todos"
3) "{\"id\":null,\"description\":\"workout tomorrow morning!\",\"created\":
  null,\"modified\":null,\"completed\":false}"
```

И конечно, вы можете увидеть новое запланированное дело в браузере по адресу <http://localhost:8080/toDos>.

Отлично! Вы создали приложение Spring Boot для обмена сообщениями, использующее Redis. Можете теперь остановить сервер Redis, нажав `Ctrl+C`.

Удаленный сервер Redis

Для доступа к удаленному серверу Redis необходимо добавить в файл `application.properties` следующие свойства:

```
spring.redis.database=0
spring.redis.host=localhost
spring.redis.password=mysecurepassword
spring.redis.port=6379
```

Вы всегда можете узнать обо всех свойствах Redis в документации Spring Boot по адресу <https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>.

Теперь вы знаете, как использовать Redis в качестве брокера сообщений. Если хотите узнать больше о применении хранилищ типа «ключ/значение» со Spring Boot, можете заглянуть на страницу проекта Spring Data Redis: <http://projects.spring.io/spring-data-redis/>.

Использование WebSockets со Spring Boot

Возможно, логичнее было бы поместить обсуждение WebSockets в главу, посвященную веб-приложениям, но мне кажется, что WebSockets относится скорее к обмену сообщениями, поэтому я поместил данный раздел в эту главу.

WebSockets — новый способ связи, заменяющий веб-технологии «клиент/сервер». Он делает возможным длительные односокетные TCP-соединения между клиентом и сервером. Его также называют технологией *проталкивания* (push), поскольку сервер может отправлять данные без необходимости выполнения клиентом долгих опросов для запроса обновлений.

В этом разделе показан пример, включающий отправку сообщения через конечную точку REST (Producer) и получение сообщений (Consumer) с помощью веб-страницы и библиотек JavaScript.

Создание приложения ToDo с использованием WebSockets

Создадим приложение ToDo, использующее репозитории JPA REST. Каждое появляющееся новое запланированное дело выводится на веб-странице.