



Бесплатная электронная книга

УЧУСЬ spring-boot

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-
boot

.....	1
1:	2
.....	2
.....	2
Examples.....	2
.....	2
- Spring Spring Gradle as build system.....	4
2: Spring boot + Hibernate + Web UI ().....	6
.....	6
.....	6
Examples.....	6
Maven.....	6
.....	7
.....	8
Tyymeleaf	8
3: Spring Boot + JPA + REST.....	11
.....	11
Examples.....	11
.....	11
.....	11
.....	12
Maven.....	13
4: Spring Boot + Spring Data Elasticsearch.....	15
.....	15
Examples.....	15
Spring Boot Spring Elasticsearch.....	15
.....	15
5: Spring Boot-Hibernate-REST Integration.....	24
Examples.....	24
Hibernate.....	24
REST.....	25

6: Spring-Boot + JDBC	27
.....	27
.....	27
Examples	28
schema.sql	28
JdbcTemplate	29
data.sql	29
7: Spring-Boot Microservice JPA	30
Examples	30
.....	30
.....	30
.....	31
.....	31
.....	32
.....	32
Gradle	33
8: ThreadPoolTaskExecutor:	35
Examples	35
.....	35
9: + JPA + mongoDB	36
Examples	36
CRUD MongoDB JPA	36
.....	37
.....	38
pom.xml	38
: POST	38
URL-	39
:	40
10: + Spring Data JPA	41
.....	41
.....	41
.....	41

.....	41
Examples.....	42
.....	42
.....	42
.....	42
.....	43
DAO.....	44
.....	45
.....	45
.....	46
MySQL.....	47
SQL.....	47
pom.xml.....	48
JAR.....	48
11:	50
.....	50
Examples.....	50
.....	50
12: Redis Spring Boot MongoDB.....	54
Examples.....	54
.....	54
.....	54
13: Spring-Boot MySQL.....	61
.....	61
.....	61
Examples.....	61
Spring-boot MySQL.....	61
14: - - Spring Boot JHipster.....	66
Examples.....	66
Spring Boot jHipster Mac OS.....	66
15: Spring- Amazon Elas.....	70

Examples.....	70
Spring-boot Jar AWS.....	70
16:	77
.....	77
.....	77
Examples.....	78
@SpringBootApplication.....	78
@ComponentScan.....	80
.....	80
17: application.properties.....	82
Examples.....	82
Dev Prod	82
, (maven).....	83
18:	86
Examples.....	86
Spring Spring.....	86
yaml []	89
yaml.....	89
.....	90
19: REST.....	91
.....	91
Examples.....	91
REST-.....	91
JERSEY Spring Boot.....	94
1.	94
2.	94
3.Wiring Jersey	95
4.Done.....	95
API REST RestTemplate (GET).....	96
20: Spring Boot CLI.....	98
.....	98
.....	

Examples.....	99
.....	99
Mac OSX HomeBrew.....	99
Mac OSX MacPorts.....	99
SDKMAN!.....	99
.....	100

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [spring-boot](#)

It is an unofficial and free spring-boot ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official spring-boot.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с весенней загрузкой

замечания

В этом разделе представлен обзор того, что такое Spring-Boot, и почему разработчик может захотеть его использовать.

Следует также упомянуть о любых крупных предметах в весенней загрузке и ссылаться на связанные темы. Поскольку Документация для весенней загрузки является новой, вам может потребоваться создать начальные версии этих связанных тем.

Версии

Версия	Дата выхода
1,5	2017-01-30
1.4	2016-07-28
1,3	2015-11-16
1.2	2014-12-11
1,1	2014-06-10
1,0	2014-04-01

Examples

Установка или настройка

Начало работы с Spring Boot в первый раз довольно быстро благодаря тяжелой работе Spring Community.

Предпосылки:

1. Установлена Java
2. Java IDE Рекомендуется не требовать (IntelliJ, Eclipse, Netbeans и т. Д.).

Вам не нужно устанавливать Maven и / или Gradle. Проекты, созданные [Spring Initializr](#), поставляются с Maven Wrapper (команда `mvnw`) или Gradle Wrapper (команда `gradlew`).

Откройте свой веб-браузер на <https://start.spring.io> Это пусковая панель для создания новых приложений Spring Boot, и теперь мы пойдём с минимальным минимумом.

Не стесняйтесь переключаться с Maven на Gradle, если это ваш предпочтительный инструмент сборки.

Найдите «Веб» в разделе «Поиск зависимостей» и добавьте его.

Нажмите «Создать проект»!

Это загрузит zip-файл под названием demo. Не стесняйтесь извлекать этот файл там, где хотите на своем компьютере.

Если вы выберете maven, перейдите в командную строку в базовый каталог и выполните

Вы должны получить результат успеха сборки:

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.908 s
[INFO] Finished at: 2016-07-21T07:15:06-04:00
[INFO] Final Memory: 27M/331M
[INFO] -----
C:\Users\gsd4tyk\Downloads\demo>
```

Запуск приложения: `mvn spring-boot:run`

Теперь запускается приложение Spring Boot. Перейдите в свой веб-браузер на localhost: 8080

Congrats! Вы только что создали свое первое приложение Spring Boot. Теперь добавим крошечный бит кода, чтобы вы могли видеть его работу.

Поэтому используйте `ctrl + c` для выхода из текущего текущего сервера.

Перейдите к: `src/main/java/com/example/DemoApplication.java` Обновите этот класс, чтобы иметь контроллер

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@SpringBootApplication
public class DemoApplication {

    @RequestMapping("/")
    String home() {
```

```
        return "Hello World!";
    }

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Хорошие вещи теперь позволяют строить и запускать проект снова с `mvn clean install`
`spring-boot:run` !

Теперь перейдите в свой веб-браузер на `localhost: 8080`

Привет, мир!

Congrats! Мы только что закончили создание Spring Boot Application и настроили наш первый контроллер, чтобы вернуть «Hello World!». Добро пожаловать в мир Весенней загрузки!

Простое веб-приложение Spring Spring с использованием Gradle as build system

В этом примере предполагается, что вы уже установили Java и [Gradle](#) .

Используйте следующую структуру проекта:

```
src/
  main/
    java/
      com/
        example/
          Application.java
build.gradle
```

`build.gradle` - это ваш скрипт сборки для системы сборки Gradle со следующим содержимым:

```
buildscript {
    ext {
        //Always replace with latest version available at http://projects.spring.io/spring-
        boot/#quick-start
        springBootVersion = '1.5.6.RELEASE'
    }
    repositories {
        jcenter()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

apply plugin: 'java'
apply plugin: 'org.springframework.boot'
```

```
repositories {  
    jcenter()  
}  
  
dependencies {  
    compile('org.springframework.boot:spring-boot-starter-web')  
}
```

`Application.java` является основным классом веб-приложения Spring Boot:

```
package com.example;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@SpringBootApplication // same as @Configuration @EnableAutoConfiguration @ComponentScan  
@RestController  
public class Application {  
  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class);  
    }  
  
    @RequestMapping("/hello")  
    private String hello() {  
        return "Hello World!";  
    }  
}
```

Теперь вы можете запустить веб-приложение Spring Boot с помощью

```
gradle bootRun
```

и доступ к опубликованной конечной точке HTTP либо с использованием `curl`

```
curl http://localhost:8080/hello
```

или ваш браузер, открыв [localhost: 8080 / hello](http://localhost:8080/hello) .

Прочитайте Начало работы с весенней загрузкой онлайн: <https://riptutorial.com/ru/spring-boot/topic/829/начало-работы-с-весенней-загрузкой>

глава 2: Spring boot + Hibernate + Web UI (Тимелеар)

Вступление

Этот поток посвящен тому, как создать весеннее загрузочное приложение с движком шаблонов hibernate и thymeleaf.

замечания

Также проверьте [документацию Thymeleaf](#)

Examples

Зависимости Maven

Этот пример основан на весенней загрузке 1.5.1.RELEASE. со следующими зависимостями:

```
<!-- Spring -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- Lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<!-- H2 -->
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
</dependency>
<!-- Test -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

В этом примере мы будем использовать Spring Boot JPA, Thymeleaf и веб-стартеры. Я использую Lombok для генерации getters и setters, но это необязательно. H2 будет использоваться как легко настраиваемая база данных в оперативной памяти.

Конфигурация спящего режима

Во-первых, давайте посмотрим, что нам нужно, чтобы правильно настроить Hibernate.

1. `@EnableTransactionManagement` и `@EnableJpaRepositories` - мы хотим, чтобы управление транзакциями и использование репозитория весенних данных.
2. `DataSource` - основной источник данных для приложения. используя в этом случае память h2 в этом примере.
3. `LocalContainerEntityManagerFactoryBean` - фабрика-менеджер весенних сущностей с использованием `HibernateJpaVendorAdapter`.
4. `PlatformTransactionManager` - основной менеджер транзакций для аннотированных компонентов `@Transactional`.

Конфигурационный файл:

```
@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(basePackages = "com.example.repositories")
public class PersistenceJpaConfig {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:mem:testdb;mode=MySQL;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) {
        LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();
        em.setDataSource(dataSource);
        em.setPackagesToScan(new String[] { "com.example.models" });
        JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        em.setJpaVendorAdapter(vendorAdapter);
        em.setJpaProperties(additionalProperties());
        return em;
    }

    @Bean
    public PlatformTransactionManager transactionManager(LocalContainerEntityManagerFactoryBean entityManagerFactory, DataSource dataSource) {
        JpaTransactionManager tm = new JpaTransactionManager();
        tm.setEntityManagerFactory(entityManagerFactory.getObject());
    }
}
```

```

        tm.setDataSource(dataSource);
        return tm;
    }

    Properties additionalProperties() {
        Properties properties = new Properties();
        properties.setProperty("hibernate.hbm2ddl.auto", "update");
        properties.setProperty("hibernate.dialect", "org.hibernate.dialect.H2Dialect");
        return properties;
    }
}

```

Объекты и репозитории

Простой объект: использование аннотаций Lombok `@Getter` и `@Setter` для создания геттеров и сеттеров для нас

```

@Entity
@Getter @Setter
public class Message {

    @Id
    @GeneratedValue(generator = "system-uuid")
    @GenericGenerator(name = "system-uuid", strategy = "uuid")
    private String id;
    private String message;
}

```

Я использую идентификаторы на основе UUID и lombok для генерации getters и setters.

Простой репозиторий для объекта выше:

```

@Transactional
public interface MessageRepository extends CrudRepository<Message, String> {
}

```

Больше о репозиториях: [весенние документы данных](#)

Убедитесь, что объекты находятся в пакете, который отображается в `em.setPackagesToScan` (определенном в компоненте `LocalContainerEntityManagerFactoryBean`) и репозиториях в пакете, отображаемом в `basePackages` (определенном в аннотации `@EnableJpaRepositories`)

Ресурсы Thymeleaf и контроллер весны

Чтобы выявить шаблоны Thymeleaf, нам нужно определить контроллеры.

Пример:

```

@Controller
@RequestMapping("/")

```

```

public class MessageController {

    @Autowired
    private MessageRepository messageRepository;

    @GetMapping
    public ModelAndView index() {
        Iterable<Message> messages = messageRepository.findAll();
        return new ModelAndView("index", "index", messages);
    }

}

```

Этот простой контроллер вводит `MessageRepository` и передает все сообщения в файл шаблона с именем `index.html`, находящийся в `src/main/resources/templates` и, наконец, выставляя его в `/index`.

Точно так же мы можем разместить другие шаблоны в папке шаблонов (по умолчанию весна - `src/main/resources/templates`), передать им модель и передать их клиенту.

Другие статические ресурсы должны быть помещены в одну из следующих папок, выставленных по умолчанию при загрузке весны:

```

/META-INF/resources/
/resources/
/static/
/public/

```

Пример thymeleaf `index.html`:

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head th:fragment="head (title)">
        <title th:text="${title}">Index</title>
        <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}"
href="../../css/bootstrap.min.css" />
    </head>
    <body>
        <nav class="navbar navbar-default navbar-fixed-top">
            <div class="container-fluid">
                <div class="navbar-header">
                    <a class="navbar-brand" href="#">Thymeleaf</a>
                </div>
            </div>
        </nav>
        <div class="container">
            <ul class="nav">
                <li><a th:href="@{/}" href="messages.html"> Messages </a></li>
            </ul>
        </div>
    </body>
</html>

```

- `bootstrap.min.css` находится в папке `src/main/resources/static/css`. Вы можете использовать синтаксис `@{ }` для получения других статических ресурсов с

использованием относительного пути.

Прочитайте Spring boot + Hibernate + Web UI (Тимелеар) онлайн:

<https://riptutorial.com/ru/spring-boot/topic/9200/spring-boot-plus-hibernate-plus-web-ui--тимелеар->

глава 3: Spring Boot + JPA + REST

замечания

В этом примере используются Spring Boot, Spring Data JPA и Spring Data REST, чтобы открыть простой объект домена, управляемый JPA, через REST. В этом примере отвечает формат HAL JSON и предоставляет URL-адрес, доступный для пользователя `/person`. Конфигурация maven включает в себя базу данных H2 в памяти для поддержки быстрого запуска.

Examples

Весенняя загрузка загрузки

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    //main entry point
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

Объект домена

```
package com.example.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

//simple person object with JPA annotations

@Entity
public class Person {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @Column
    private String firstName;

    @Column
```

```

private String lastName;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}
}

```

Интерфейс репозитория

```

package com.example.domain;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

//annotation exposes the
@RepositoryRestResource(path="/person")
public interface PersonRepository extends JpaRepository<Person,Long> {

    //the method below allows us to expose a search query, customize the endpoint name, and
    specify a parameter name
    //the exposed URL is GET /person/search/byLastName?lastname=
    @RestResource(path="/byLastName")
    Iterable<Person> findByLastName(@Param("lastName") String lastName);

    //the methods below are examples on to prevent an operation from being exposed.
    //For example DELETE; the methods are overridden and then annotated with
    RestResource(exported=false) to make sure that no one can DELETE entities via REST
    @Override
    @RestResource(exported=false)
    default void delete(Long id) { }

    @Override
    @RestResource(exported=false)
    default void delete(Person entity) { }

    @Override

```

```

@RestResource(exported=false)
default void delete(Iterable<? extends Person> entities) { }

@Override
@RestResource(exported=false)
default void deleteAll() { }

@Override
@RestResource(exported=false)
default void deleteAllInBatch() { }

@Override
@RestResource(exported=false)
default void deleteInBatch(Iterable<Person> arg0) { }

}

```

Конфигурация Maven

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.4.0.RELEASE</version>
</parent>
<groupId>com.example</groupId>
<artifactId>spring-boot-data-jpa-rest</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>spring-boot-data-jpa-rest</name>
<build>
<plugins>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>

```

```
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
</dependencies>
</project>
```

Прочитайте Spring Boot + JPA + REST онлайн: <https://riptutorial.com/ru/spring-boot/topic/6507/spring-boot-plus-jpa-plus-rest>

глава 4: Spring Boot + Spring Data Elasticsearch

Вступление

[Spring Data Elasticsearch](#) - это реализация [Spring Data](#) для [Elasticsearch](#), которая обеспечивает интеграцию с поисковой системой [Elasticsearch](#).

Examples

Интеграция с использованием Spring Boot и Spring Elasticsearch

В этом примере мы собираемся реализовать проект `spring-data-elasticsearch` для хранения POJO в `elasticsearch`. Мы увидим пример проекта `maven`, который выполняет следующие действия:

- Вставьте `Greeting(id, username, message)` в поле поиска `elastics`.
- Получите все приветственные элементы, которые были вставлены.
- Обновите элемент Приветствие.
- Удалить элемент Приветствие.
- Получите ярлык по идентификатору.
- Получите все приветствие по имени пользователя.

Интеграция весов и весов

В этом примере мы увидим приложение весенней загрузки, основанное на `maven`, которое объединяет поиск `spring-data-elasticsearch`. Здесь мы сделаем следующее и увидим соответствующие сегменты кода.

- Вставьте `Greeting(id, username, message)` в поле поиска `elastics`.
- Получить все предметы из `elasticsearch`
- Обновите определенный элемент.
- Удалите определенный элемент.
- Получить определенный элемент по `id`.
- Получить определенный элемент по имени пользователя.

Файл конфигурации проекта (`pom.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.springdataes</groupId>
    <artifactId>springdataes</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.6.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-elasticsearch</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <executions>
                    <execution>
                        <goals>
                            <goal>repackage</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>

```

Мы будем использовать [Spring Boot](#) версии 1.5.6.RELEASE и [Spring Data Elasticsearch](#) соответствующей версии. Для этого проекта нам нужно запустить [elasticsearch-2.4.5](#), чтобы проверить наш `apis`.

Файл свойств

Мы поместим файл свойств проекта (named `applications.properties`) в папку `resources` которая содержит:

```

elasticsearch.clustername = elasticsearch
elasticsearch.host = localhost
elasticsearch.port = 9300

```

Мы будем использовать имя кластера по умолчанию, хост и порт. По умолчанию порт 9300 используется в качестве транспортного порта, а порт 9200 известен как порт http. Чтобы увидеть имя кластера по умолчанию, нажмите <http://localhost:9200/>.

Основной класс (Application.java)

```
package org.springdataes;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String []args) {
        SpringApplication.run(Application.class, args);
    }
}
```

`@SpringBootApplication` представляет собой комбинацию `@Configuration`, `@EnableAutoConfiguration`, `@EnableWebMvc` и `@ComponentScan` аннотаций. Метод `main()` использует метод `SpringApplication.run()` Spring Boot для запуска приложения. Там нам не нужна какая-либо конфигурация xml, это приложение является чистым java spring application.

Класс конфигурации Elasticsearch (ElasticsearchConfig.java)

```
package org.springdataes.config;

import org.elasticsearch.client.Client;
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.transport.InetSocketTransportAddress;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.data.elasticsearch.core.ElasticsearchOperations;
import org.springframework.data.elasticsearch.core.ElasticsearchTemplate;
import org.springframework.data.elasticsearch.repository.config.EnableElasticsearchRepositories;
import java.net.InetAddress;

@Configuration
@PropertySource(value = "classpath:applications.properties")
@EnableElasticsearchRepositories(basePackages = "org.springdataes.dao")
public class ElasticsearchConfig {
    @Value("${elasticsearch.host}")
    private String EsHost;

    @Value("${elasticsearch.port}")
    private int EsPort;

    @Value("${elasticsearch.clustername}")
    private String EsClusterName;

    @Bean
```

```

public Client client() throws Exception {
    Settings esSettings = Settings.settingsBuilder()
        .put("cluster.name", EsClusterName)
        .build();

    return TransportClient.builder()
        .settings(esSettings)
        .build()
        .addTransportAddress(new
InetSocketAddress(InetAddress.getByName(EsHost), EsPort));
}

@Bean
public ElasticsearchOperations elasticsearchTemplate() throws Exception {
    return new ElasticsearchTemplate(client());
}
}

```

Класс `ElasticsearchConfig` настраивает `elasticsearch` для этого проекта и устанавливает связь с `elasticsearch`. Здесь `@PropertySource` используется для чтения файла `application.properties` где мы храним имя кластера, хост и порт `elasticsearch`. `@EnableElasticsearchRepositories` используется для включения репозитория `@EnableElasticsearchRepositories` которые по умолчанию сканируют пакеты аннотированного класса конфигурации для хранилищ `Spring Data`. `@Value` используется здесь для чтения свойств из файла `application.properties`.

Метод `Client()` создает транспортное соединение с `elasticsearch`. В приведенной выше конфигурации настраивается сервер `Embedded Elasticsearch`, который используется `ElasticsearchTemplate`. Компонент `ElasticsearchTemplate` использует `Elasticsearch Client` и предоставляет настраиваемый уровень для управления данными в `Elasticsearch`.

Модельный класс (Greeting.java)

```

package org.springdataes.model;

import org.springframework.data.annotation.Id;
import org.springframework.data.elasticsearch.annotations.Document;
import java.io.Serializable;

@Document(indexName = "index", type = "greetings")
public class Greeting implements Serializable{

    @Id
    private String id;

    private String username;

    private String message;

    public Greeting() {
    }

    public String getId() {
        return id;
    }
}

```



```

public void setId(String id) {
    this.id = id;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}
}

```

Здесь мы аннотировали наши объекты данных `Greeting` с аннотацией `@Document`, которую мы также можем использовать для определения параметров индекса, таких как имя, количество осколков или количество реплик. Один из атрибутов класса должен быть `id`, либо путем аннотации его с помощью `@Id` либо с использованием одного из автоматически найденных имен `id` или `documentId`. Здесь значение поля `id` будет автоматически сгенерировано, если мы не установим значение поля `id`.

Класс репозитория Elasticsearch (GreetingRepository.class)

```

package org.springdataes.dao;

import org.springdataes.model.Greeting;
import org.springframework.data.elasticsearch.repository.ElasticsearchRepository;
import java.util.List;

public interface GreetingRepository extends ElasticsearchRepository<Greeting, String> {
    List<Greeting> findByUsername(String username);
}

```

Здесь мы расширили `ElasticsearchRepository` которые предоставляют нам много `apis`, которые нам не нужно определять извне. Это базовый класс репозитория для классов домена, основанных на `elasticsearch`. Поскольку он расширяет классы репозитория `Spring`, мы получаем выгоду от избежания шаблона кода, необходимого для реализации уровней доступа к данным для различных хранилищ сохранения.

Здесь мы объявили метод `findByUsername(String username)` который преобразует в запрос соответствия, совпадающий с именем пользователя, с полем `username` « `Greeting` и возвращает список результатов.

Услуги (GreetingService.java)

```

package org.springdataes.service;

import org.springdataes.model.Greeting;
import java.util.List;

public interface GreetingService {
    List<Greeting> getAll();
    Greeting findOne(String id);
    Greeting create(Greeting greeting);
    Greeting update(Greeting greeting);
    List<Greeting> getGreetingByUsername(String username);
    void delete(String id);
}

```

Сервисный компонент (GreetingServiceBean.java)

```

package org.springdataes.service;

import com.google.common.collect.Lists;
import org.springdataes.dao.GreetingRepository;
import org.springdataes.model.Greeting;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class GreetingServiceBean implements GreetingService {

    @Autowired
    private GreetingRepository repository;

    @Override
    public List<Greeting> getAll() {
        return Lists.newArrayList(repository.findAll());
    }

    @Override
    public Greeting findOne(String id) {
        return repository.findOne(id);
    }

    @Override
    public Greeting create(Greeting greeting) {
        return repository.save(greeting);
    }

    @Override
    public Greeting update(Greeting greeting) {
        Greeting persistedGreeting = repository.findOne(greeting.getId());
        if(persistedGreeting == null) {
            return null;
        }
        return repository.save(greeting);
    }

    @Override
    public List<Greeting> getGreetingByUsername(String username) {
        return repository.findByUsername(username);
    }
}

```

```

@Override
public void delete(String id) {
    repository.delete(id);
}
}

```

В вышеприведенном классе мы `@Autowired GreetingRepository` . Мы можем просто вызвать методы `CRUDRepository` и метод, который мы объявили в классе репозитория, с объектом `GreetingRepository` .

В `getAll()` вы можете найти строку `Lists.newArrayList(repository.findAll())` . Мы сделали это, чтобы преобразовать элемент `repository.findAll()` в `List<>` поскольку он возвращает `Iterable List`.

Класс контроллера (GreetingController.java)

```

package org.springdataes.controller;

import org.springdataes.model.Greeting;
import org.springdataes.service.GreetingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
public class GreetingController {

    @Autowired
    private GreetingService greetingService;

    @ResponseBody
    @RequestMapping(value = "/greetings", method = RequestMethod.GET)
    public ResponseEntity<List<Greeting>> getAll() {
        return new ResponseEntity<List<Greeting>>(greetingService.getAll(), HttpStatus.OK);
    }

    @ResponseBody
    @RequestMapping(value = "/greetings", method = RequestMethod.POST)
    public ResponseEntity<Greeting> insertGreeting(@RequestBody Greeting greeting) {
        return new ResponseEntity<Greeting>(greetingService.create(greeting),
        HttpStatus.CREATED);
    }

    @ResponseBody
    @RequestMapping(value = "/greetings", method = RequestMethod.PUT)
    public ResponseEntity<Greeting> updateGreeting(@RequestBody Greeting greeting) {
        return new ResponseEntity<Greeting>(greetingService.update(greeting),
        HttpStatus.MOVED_PERMANENTLY);
    }

    @ResponseBody
    @RequestMapping(value = "/greetings/{id}", method = RequestMethod.DELETE)

```

```

    public ResponseEntity<Greeting> deleteGreeting(@PathVariable("id") String id) {
        greetingService.delete(id);
        return new ResponseEntity<Greeting> (HttpStatus.NO_CONTENT);
    }

    @ResponseBody
    @RequestMapping(value = "/greetings/{id}", method = RequestMethod.POST)
    public ResponseEntity<Greeting> getOne(@PathVariable("id") String id) {
        return new ResponseEntity<Greeting> (greetingService.findOne(id), HttpStatus.OK);
    }

    @ResponseBody
    @RequestMapping(value = "/greetings/{name}", method = RequestMethod.GET)
    public ResponseEntity<List<Greeting>> getByUserName(@PathVariable("name") String name) {
        return new ResponseEntity<List<Greeting>> (greetingService.getGreetingByUsername(name),
        HttpStatus.OK);
    }
}

```

строить

Чтобы создать это приложение maven

```
mvn clean install
```

Выше команда сначала удалит все файлы в `target` папке и построит проект. После создания проекта мы получим исполняемый файл `.jar`, который называется `springdataes-1.0-SNAPSHOT.jar`. Мы можем запустить основной класс (`Application.java`), чтобы запустить процесс или просто выполнить вышеупомянутую команду, набрав:

```
java -jar springdataes-1.0-SNAPSHOT.jar
```

Проверка API

Чтобы вставить элемент «Приветствие» в поиск elastics, выполните следующую команду:

```
curl -H "Content-Type: application/json" -X POST -d '{"username":"sunkuet02","message": "this is a message"}' http://localhost:8080/api/greetings
```

Вы должны получить следующий результат:

```
{"id":"AV2ddRxBcuirs1TrVgHH","username":"sunkuet02","message":"this is a message"}
```

Вы также можете проверить get api, выполнив:

```
curl -H "Content-Type: application/json" -X GET http://localhost:8080/api/greetings
```

Вы должны получить

```
[{"id":"AV2ddRxBcuirs1TrVgHH","username":"sunkuet02","message":"this is a message"}]
```

Вы можете проверить другой apis, выполнив описанные выше процессы.

Официальные документы:

- <https://projects.spring.io/spring-boot/>
- <https://projects.spring.io/spring-data-elasticsearch/>

Прочитайте Spring Boot + Spring Data Elasticsearch онлайн: <https://riptutorial.com/ru/spring-boot/topic/10928/spring-boot-plus-spring-data-elasticsearch>

глава 5: Spring Boot-Hibernate-REST Integration

Examples

Добавить поддержку Hibernate

1. Добавьте зависимость **spring-boot-starter-data-jpa** к **pom.xml**. Вы можете пропустить тег **версии** , если вы используете **parent-boot-starter-parent** в качестве родителя вашего **pom.xml** . Нижеприведенная зависимость приводит Hibernate и все, что связано с JPA к вашему проекту ([ссылка](#)).

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

2. Добавьте драйвер базы данных в **pom.xml** . Ниже приведена база данных H2 ([ссылка](#)).

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
</dependency>
```

3. Включить ведение журнала отладки для Hibernate в **application.properties**

logging.level.org.hibernate.SQL = debug

или в **application.yml**

```
logging:
  level:
    org.hibernate.SQL: debug
```

4. Добавьте класс сущности в нужный пакет под **\$ {project.home} / src / main / java /** , например, в **com.example.myproject.domain** ([ссылка](#)):

```
package com.example.myproject.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.io.Serializable;

@Entity
```

```
public class City implements Serializable {

    @Id
    @GeneratedValue
    public Long id;

    @Column(nullable = false)
    public String name;

}
```

5. Добавьте **import.sql** в \$ {project.home} / src / main / resources / . Поместите инструкции **INSERT** в файл. Этот файл будет использоваться для совокупности схем базы данных при каждом запуске приложения ([ссылка](#)):

```
insert into city(name) values ('Brisbane');

insert into city(name) values ('Melbourne');
```

6. Добавьте класс репозитория в нужный пакет под \$ {project.home} / src / main / java / , например, в **com.example.myproject.service** ([ссылка](#)):

```
package com.example.myproject.service;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import java.io.Serializable;

import com.example.myproject.domain.City;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.Repository;

interface CityRepository extends Repository<City, Long> {

    Page<City> findAll(Pageable pageable);

    Page<City> findByName(String name);

}
```

В принципе, вот и все! На этом этапе вы уже можете получить доступ к базе данных с помощью методов **com.example.myproject.service.CityRepository** .

Добавить поддержку REST

1. Добавьте зависимость **spring-boot-starter-web** к pom.xml. Вы можете пропустить тег **версии** , если вы используете **parent-boot-starter-parent** как родительский элемент вашего **pom.xml** ([ссылка](#)).

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
```

</dependency>

2. Добавьте контроллер REST в нужный пакет, например, **com.example.myproject.web.rest** ([ссылка](#)):

```
package com.example.myproject.web.rest;

import java.util.Map;
import java.util.HashMap;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;

@RestController
public class VersionController {
    @RequestMapping("/api/version")
    public ResponseEntity get() {
        final Map<String, String> responseParams = new HashMap();
        responseParams.put("requestStatus", "OK");
        responseParams.put("version", "0.1-SNAPSHOT");

        return ResponseEntity.ok().body(responseParams.build());
    }
}
```

3. Запустите приложение Spring Boot ([ссылка](#)).
4. Ваш контроллер доступен по адресу [http: // localhost: 8080 / api / version](http://localhost:8080/api/version) .

Прочитайте [Spring Boot-Hibernate-REST Integration](https://riptutorial.com/ru/spring-boot/topic/6818/spring-boot-hibernate-rest-integration) онлайн: <https://riptutorial.com/ru/spring-boot/topic/6818/spring-boot-hibernate-rest-integration>

глава 6: Spring-Boot + JDBC

Вступление

Spring Boot может использоваться для создания и сохранения SQL Relational DataBase. Вы можете подключиться к базе данных H2 в памяти с использованием Spring Boot или, возможно, выбрать подключение к MySQL DataBase, это ваш выбор. Если вы хотите проводить операции CRUD с вашей БД, вы можете сделать это с помощью JdbcTemplate bean, этот компонент будет автоматически предоставлен Spring Boot. Spring Boot поможет вам, предоставив автоматическую настройку некоторых часто используемых компонентов, связанных с JDBC.

замечания

Чтобы начать работу, в своем затмении sts перейдите к новому -> Spring Starter Project -> заполните ваши координаты Maven -> и добавьте следующие зависимости:

На вкладке SQL -> добавить JDBC + добавить MySQL (если MySQL - ваш выбор).

Для MySQL вам также потребуется добавить Java-коннектор MySQL.

В файле Spring Boot application.properties (файл конфигурации Spring Boot) вам необходимо настроить учетные данные источника данных в MySQL DB:

1. spring.datasource.url
2. spring.datasource.username
3. spring.datasource.password
4. -Имя-класса spring.datasource.driver

например:

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

В папке ресурсов добавьте следующие два файла:

1. schema.sql -> каждый раз, когда вы запускаете приложение Spring Boot запускает этот файл, внутри него вы должны написать свою схему БД, определить таблицы и их отношения.
2. data.sql -> каждый раз при запуске приложения Spring Boot запускает этот файл, внутри него вы можете записать данные, которые будут вставлены в вашу таблицу в

качестве начальной инициализации.

Spring Boot предоставит вам JdbcTemplate bean автоматически, чтобы вы могли мгновенно использовать его следующим образом:

```
@Autowired
private JdbcTemplate template;
```

без каких-либо других конфигураций.

Examples

Файл schema.sql

```
CREATE SCHEMA IF NOT EXISTS `backgammon`;
USE `backgammon`;

DROP TABLE IF EXISTS `user_in_game_room`;
DROP TABLE IF EXISTS `game_users`;
DROP TABLE IF EXISTS `user_in_game_room`;

CREATE TABLE `game_users`
(
  `user_id` BIGINT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(255) NOT NULL,
  `last_name` VARCHAR(255) NOT NULL,
  `email` VARCHAR(255) NOT NULL UNIQUE,
  `user_name` VARCHAR(255) NOT NULL UNIQUE,
  `password` VARCHAR(255) NOT NULL,
  `role` VARCHAR(255) NOT NULL,
  `last_updated_date` DATETIME NOT NULL,
  `last_updated_by` BIGINT NOT NULL,
  `created_date` DATETIME NOT NULL,
  `created_by` BIGINT NOT NULL,
  PRIMARY KEY(`user_id`)
);

DROP TABLE IF EXISTS `game_rooms`;

CREATE TABLE `game_rooms`
(
  `game_room_id` BIGINT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `private` BIT(1) NOT NULL,
  `white` BIGINT DEFAULT NULL,
  `black` BIGINT DEFAULT NULL,
  `opened_by` BIGINT NOT NULL,
  `speed` BIT(3) NOT NULL,
  `last_updated_date` DATETIME NOT NULL,
  `last_updated_by` BIGINT NOT NULL,
  `created_date` DATETIME NOT NULL,
  `created_by` BIGINT NOT NULL,
  `token` VARCHAR(255) AS (SHA1(CONCAT(`name`, "This is a qwe secret 123", `created_by`,
`created_date`))),
  PRIMARY KEY(`game_room_id`)
);
```

```
CREATE TABLE `user_in_game_room`
(
    `user_id` BIGINT NOT NULL,
    `game_room_id` BIGINT NOT NULL,
    `last_updated_date` DATETIME NOT NULL,
    `last_updated_by` BIGINT NOT NULL,
    `created_date` DATETIME NOT NULL,
    `created_by` BIGINT NOT NULL,
    PRIMARY KEY(`user_id`, `game_room_id`),
    FOREIGN KEY (`user_id`) REFERENCES `game_users` (`user_id`),
    FOREIGN KEY (`game_room_id`) REFERENCES `game_rooms` (`game_room_id`)
);
```

Первое загрузочное приложение JdbcTemplate

```
@SpringBootApplication
@RestController
public class SpringBootJdbcApplication {

    @Autowired
    private JdbcTemplate template;

    @RequestMapping("/cars")
    public List<Map<String,Object>> stocks(){
        return template.queryForList("select * from car");
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringBootJdbcApplication.class, args);
    }
}
```

data.sql

```
insert into game_users values(..., ..., ..., ...);
insert into game_users values(..., ..., ..., ...);
insert into game_users values(..., ..., ..., ...);
```

Прочитайте Spring-Boot + JDBC онлайн: <https://riptutorial.com/ru/spring-boot/topic/9834/spring-boot-plus-jdbc>

глава 7: Spring-Boot Microservice с JPA

Examples

Класс применения

```
package com.mcf7.spring;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringDataMicroServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringDataMicroServiceApplication.class, args);
    }
}
```

Модель книги

```
package com.mcf7.spring.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.io.Serializable;

@lombok.Getter
@lombok.Setter
@lombok.EqualsAndHashCode(of = "isbn")
@lombok.ToString(exclude="id")
@Entity
public class Book implements Serializable {

    public Book() {}

    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private long id;

    @NotNull
    @Size(min = 1)
    private String isbn;

    @NotNull
    @Size(min = 1)
    private String title;

    @NotNull
    @Size(min = 1)
```

```

private String author;

@NotNull
@Size(min = 1)
private String description;
}

```

Просто примечание, так как здесь происходит несколько вещей, я хотел быстро разбить их.

Все аннотации с `@lombok` генерируют часть плиты котла нашего класса

```

@lombok.Getter //Creates getter methods for our variables

@lombok.Setter //Creates setter methods four our variables

@lombok.EqualsAndHashCode(of = "isbn") //Creates Equals and Hashcode methods based off of the
isbn variable

@lombok.ToString(exclude="id") //Creates a toString method based off of every variable except
id

```

Мы также задействовали валидацию в этом объекте

```

@NotNull //This specifies that when validation is called this element shouldn't be null

@Size(min = 1) //This specifies that when validation is called this String shouldn't be
smaller than 1

```

Репозиторий книг

```

package com.mcf7.spring.domain;

import org.springframework.data.repository.PagingAndSortingRepository;

public interface BookRepository extends PagingAndSortingRepository<Book, Long> {
}

```

Базовый шаблон репозитория Spring, за исключением того, что мы включили пейджинговый и сортировочный репозиторий для дополнительных функций, таких как ... пейджинг и сортировка :)

Включение проверки

```

package com.mcf7.spring.domain;

import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

public class BeforeCreateBookValidator implements Validator{
    public boolean supports(Class<?> clazz) {

```

```

        return Book.class.equals(clazz);
    }

    public void validate(Object target, Errors errors) {
        errors.reject("rejected");
    }
}

```

Загрузка некоторых тестовых данных

```

package com.mcf7.spring.domain;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DatabaseLoader implements CommandLineRunner {
    private final BookRepository repository;

    @Autowired
    public DatabaseLoader(BookRepository repository) {
        this.repository = repository;
    }

    public void run(String... Strings) throws Exception {
        Book book1 = new Book();
        book1.setIsbn("6515616168418510");
        book1.setTitle("SuperAwesomeTitle");
        book1.setAuthor("MCF7");
        book1.setDescription("This Book is super epic!");
        repository.save(book1);
    }
}

```

Просто загрузив некоторые тестовые данные, в идеале это должно быть добавлено только в профиль разработки.

Добавление валидатора

```

package com.mcf7.spring.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.data.rest.core.event.ValidatingRepositoryEventListener;
import org.springframework.data.rest.webmvc.config.RepositoryRestConfigurerAdapter;
import org.springframework.validation.Validator;
import org.springframework.validation.beanvalidation.LocalValidatorFactoryBean;

@Configuration
public class RestValidationConfiguration extends RepositoryRestConfigurerAdapter {

    @Bean
    @Primary
    /**

```

```

    * Create a validator to use in bean validation - primary to be able to autowire without
    qualifier
    */
    Validator validator() {
        return new LocalValidatorFactoryBean();
    }

    @Override
    public void configureValidatingRepositoryEventListener(ValidatingRepositoryEventListener
    validatingListener) {
        Validator validator = validator();
        //bean validation always before save and create
        validatingListener.addValidator("beforeCreate", validator);
        validatingListener.addValidator("beforeSave", validator);
    }
}

```

Файл сборки Gradle

```

buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'io.spring.gradle:dependency-management-plugin:0.5.4.RELEASE'
    }
}

apply plugin: 'io.spring.dependency-management'
apply plugin: 'idea'
apply plugin: 'java'

dependencyManagement {
    imports {
        mavenBom 'io.spring.platform:platform-bom:2.0.5.RELEASE'
    }
}

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {
    mavenCentral()
}

dependencies {
    compile 'org.springframework.boot:spring-boot-starter-web'
    compile 'org.springframework.boot:spring-boot-starter-data-jpa'
    compile 'org.springframework.boot:spring-boot-starter-data-rest'
    compile 'org.springframework.data:spring-data-rest-hal-browser'
    compile 'org.projectlombok:lombok:1.16.6'
    compile 'org.springframework.boot:spring-boot-starter-validation'
    compile 'org.springframework.boot:spring-boot-actuator'

    runtime 'com.h2database:h2'

    testCompile 'org.springframework.boot:spring-boot-starter-test'
    testCompile 'org.springframework.restdocs:spring-restdocs-mockmvc'
}

```

```
}
```

Прочитайте Spring-Boot Microservice с JPA онлайн: <https://riptutorial.com/ru/spring-boot/topic/6557/spring-boot-microservice-c-jpa>

глава 8: ThreadPoolTaskExecutor: настройка и использование

Examples

конфигурация приложения

```
@Configuration
@EnableAsync
public class ApplicationConfiguration{

    @Bean
    public TaskExecutor getAsyncExecutor() {
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(2);
        executor.setThreadNamePrefix("executor-task-");
        executor.initialize();
        return executor;
    }
}
```

Прочитайте ThreadPoolTaskExecutor: настройка и использование онлайн:

<https://riptutorial.com/ru/spring-boot/topic/7497/threadpooltaskexecutor--настройка-и-использование>

глава 9: Весенняя загрузка + JPA + MongoDB

Examples

Операция CRUD в MongoDB с использованием JPA

Модель клиента

```
package org.bookmytickets.model;

import org.springframework.data.annotation.Id;

public class Customer {

    @Id
    private String id;
    private String firstName;
    private String lastName;

    public Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public Customer(String id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return String.format(
            "Customer[id=%s, firstName='%s', lastName='%s']",
            id, firstName, lastName);
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

```

    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}

```

Клиентский контроллер

```

package org.bookmytickets.controller;

import java.util.List;

import org.bookmytickets.model.Customer;
import org.bookmytickets.repository.CustomerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/customer")
public class CustomerController {

    @Autowired
    private CustomerRepository repository;

    @GetMapping("")
    public List<Customer> selectAll(){
        List<Customer> customerList = repository.findAll();
        return customerList;
    }

    @GetMapping("/{id}")
    public List<Customer> getSpecificCustomer(@PathVariable String id){
        return repository.findById(id);
    }

    @GetMapping("/search/lastName/{lastName}")
    public List<Customer> searchByLastName(@PathVariable String lastName){
        return repository.findByLasttName(lastName);
    }

    @GetMapping("/search/firstName/{firstName}")
    public List<Customer> searchByFirstName(@PathVariable String firstName){
        return repository.findByFirstName(firstName);
    }

    @PostMapping("")
    public void insert(@RequestBody Customer customer) {
        repository.save(customer);
    }
}

```

```

@PatchMapping("/{id}")
public void update(@RequestParam String id, @RequestBody Customer customer) {
    Customer oldCustomer = repository.findById(id);
    if(customer.getFirstName() != null) {
        oldCustomer.setFristName(customer.getFirstName());
    }
    if(customer.getLastName() != null) {
        oldCustomer.setLastName(customer.getLastName());
    }
    repository.save(oldCustomer);
}

@DeleteMapping("/{id}")
public void delete(@RequestParam String id) {
    Customer deleteCustomer = repository.findById(id);
    repository.delete(deleteCustomer);
}
}

```

Репозиторий клиентов

```

package org.bookmytickets.repository;

import java.util.List;

import org.bookmytickets.model.Customer;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface CustomerRepository extends MongoRepository<Customer, String> {
    public Customer findByFirstName(String firstName);
    public List<Customer> findByLastName(String lastName);
}

```

pom.xml

Добавьте ниже зависимости в файл pom.xml:

```

<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>

</dependencies>

```

Вставка данных с использованием клиента для отдыха: метод POST

Для тестирования нашего приложения я использую клиент предварительного отдыха,

который является расширением хром:

Итак, вот снимок для вставки данных:

≡

Request

> http://localhost:8080/customer/insert?firstName=Rakesh&lastName=Shankarnarayan

☐ GET

☒ POST

☐ PUT

☐ DELETE

Other methods

▼ Custom content type ▼

Raw headers

Headers form

Raw payload

Data form

Status: 200: OK ? Loading time: 112 ms

Получить URL-адрес запроса

> http://localhost:8080/customer

☒ GET

☐ POST

☐ PUT

☐ DELETE

Other methods

▼

Raw headers

Headers form

Получить результат запроса:

```
2]
-0: {
  "id": "579372b4a82615cd8b77af49"
  "firstName": "Raghu"
  "lastName": "Shankarnarayan"
}
-1: {
  "id": "5793b008a826191a3c5e9fcf"
  "firstName": "Rakesh"
  "lastName": "Shankarnarayan"
}
```

Прочитайте Весенняя загрузка + JPA + mongoDB онлайн: <https://riptutorial.com/ru/spring-boot/topic/3398/весенняя-загрузка-plus-jpa-plus-mongodb>

глава 10: Весенняя загрузка + Spring Data JPA

Вступление

Spring Boot упрощает создание весенне-летних приложений и услуг с абсолютной минимальной суемой. Он поддерживает соглашение по конфигурации.

Spring Data JPA , часть большого семейства **Spring Data** , упрощает реализацию репозитория на базе JPA. Это упрощает сбор приложений, использующих технологии доступа к данным.

замечания

Аннотации

@Repository : указывает, что аннотированный класс является «репозиторием», механизмом инкапсуляции хранения, поиска и поиска, который эмулирует коллекцию объектов. Команды, реализующие традиционные шаблоны J2EE, такие как «Объект доступа к данным», также могут применять этот стереотип к классам DAO, хотя перед этим следует позаботиться о том, чтобы понять различия между объектом доступа к данным и хранилищами в стиле DDD. Это аннотирование является стереотипом общего назначения, и отдельные команды могут сузить семантику и использовать по мере необходимости.

@RestController : Удобный аннотаций , который сам с аннотацией **@Controller** и **@ResponseBody.Types** , которые несут эту аннотацию трактуются как контроллеры , где **@RequestMapping** методы предполагают **@ResponseBody** семантики по умолчанию.

@Service : указывает, что аннотированный класс - это «Сервис» (например, фасад бизнес-сервиса). Это аннотирование служит специализацией **@Component** , позволяющей автоопределять классы реализации через сканирование классов.

@SpringBootApplication : у многих разработчиков Spring Boot всегда есть свой основной класс, аннотированный с **@Configuration** , **@EnableAutoConfiguration** и **@ComponentScan** . Поскольку эти аннотации часто используются вместе (особенно если вы следуете лучшим практикам выше), Spring Boot предоставляет удобную альтернативу **@SpringBootApplication** .

@Entity : указывает, что класс является сущностью. Эта аннотация применяется к классу сущности.

Официальная документация

Pivotal Software предоставила довольно обширную документацию по Spring Framework, и ее можно найти на

- <https://projects.spring.io/spring-boot/>
- <http://projects.spring.io/spring-data-jpa/>
- <https://spring.io/guides/gs/accessing-data-jpa/>

Examples

Весенняя загрузка и весна.

Мы собираемся создать приложение, которое хранит POJO в базе данных. Приложение использует Spring Data JPA для хранения и извлечения данных в реляционной базе данных. Его наиболее привлекательной особенностью является возможность создания реализаций репозитория автоматически во время выполнения из интерфейса репозитория.

Основной класс

```
package org.springframework.boot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Метод `main()` использует метод `SpringApplication.run()` Spring Boot для запуска приложения. Обратите внимание, что нет ни одной строки XML. Нет файла `web.xml`. Это веб-приложение представляет собой 100% чистую Java, и вам не нужно заниматься настройкой любой сантехники или инфраструктуры.

Класс сущности

```
package org.springframework.boot.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```



```

@Entity
public class Greeting {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String text;

    public Greeting() {
        super();
    }

    public Greeting(String text) {
        super();
        this.text = text;
    }

    /* In this example, the typical getters and setters have been left out for brevity. */
}

```

Здесь у вас есть класс `Greeting` с двумя атрибутами, `id` и `text`. У вас также есть два конструктора. Конструктор по умолчанию существует только для JPA. Вы не будете использовать его напрямую, поэтому его можно назначить `protected`. Другой конструктор - тот, который вы будете использовать для создания экземпляров `Greeting` для сохранения в базе данных.

Класс `Greeting` аннотируется с `@Entity`, указывая, что это объект JPA. Из-за отсутствия аннотации `@Table` предполагается, что этот объект будет сопоставлен с таблицей «Приветствие».

Свойство `id` приветствия аннотируется с `@Id` так что JPA распознает его как идентификатор объекта. Свойство `id` также аннотируется с помощью `@GeneratedValue` чтобы указать, что идентификатор должен генерироваться автоматически.

Другое свойство, `text` остается неаннотированным. Предполагается, что он будет сопоставлен с столбцом, который имеет то же имя, что и само свойство.

Свойства переходного процесса

В классе сущности, аналогичном описанному выше, мы можем иметь свойства, которые мы не хотим сохранять в базе данных или создавать в качестве столбцов в нашей базе данных, возможно, потому что мы просто хотим установить их во время выполнения и использовать их в нашем приложении, поэтому мы можем иметь это свойство, аннотированное аннотацией `@Transient`.

```

package org.springframework.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

```

```
import javax.persistence.Transient;

@Entity
public class Greeting {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String text;
    @Transient
    private String textInSomeLanguage;

    public Greeting() {
        super();
    }

    public Greeting(String text) {
        super();
        this.text = text;
        this.textInSomeLanguage = getTextTranslationInSpecifiedLanguage(text);
    }

    /* In this example, the typical getters and setters have been left out for brevity. */
}
```

Здесь у вас есть тот же класс приветствия, который теперь имеет свойство переходного `textInSomeLanguage` которое можно инициализировать и использовать во время выполнения и не будет сохраняться в базе данных.

Класс DAO

```
package org.springframework.repository;

import org.springframework.model.Greeting;
import org.springframework.data.repository.CrudRepository;

public interface GreetingRepository extends CrudRepository<Greeting, Long> {

    List<Greeting> findByText(String text);
}
```

`GreetingRepository` расширяет интерфейс `CrudRepository`. Тип объекта и идентификатор, с которым он работает, `Greeting` и `Long`, указаны в общих параметрах на `CrudRepository`. Расширяя `CrudRepository`, `GreetingRepository` наследует несколько методов работы с постоянством `Greeting`, включая методы сохранения, удаления и поиска объектов `Greeting`. См. [Это обсуждение](#) для сравнения `CrudRepository`, `PagingAndSortingRepository`, `JpaRepository`.

Spring Data JPA также позволяет вам определять другие методы запроса, просто объявляя свою подпись метода. В случае `GreetingRepository` это показано с помощью `findByText()`.

В типичном приложении Java вы должны написать класс, реализующий `GreetingRepository`. Но это делает Spring Data JPA настолько мощным: вам не нужно писать реализацию интерфейса репозитория. Spring Data JPA создает реализацию «на лету» при запуске

приложения.

Класс обслуживания

```
package org.springframework.service;

import java.util.Collection;
import org.springframework.model.Greeting;

public interface GreetingService {

    Collection<Greeting> findAll();
    Greeting findOne(Long id);
    Greeting create(Greeting greeting);
    Greeting update(Greeting greeting);
    void delete(Long id);

}
```

Сервисный компонент

```
package org.springframework.service;

import java.util.Collection;
import org.springframework.model.Greeting;
import org.springframework.repository.GreetingRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class GreetingServiceBean implements GreetingService {

    @Autowired
    private GreetingRepository greetingRepository;

    @Override
    public Collection<Greeting> findAll() {
        Collection<Greeting> greetings = greetingRepository.findAll();
        return greetings;
    }

    @Override
    public Greeting findOne(Long id) {
        Greeting greeting = greetingRepository.findOne(id);
        return greeting;
    }

    @Override
    public Greeting create(Greeting greeting) {
        if (greeting.getId() != null) {
            //cannot create Greeting with specified Id value
            return null;
        }
        Greeting savedGreeting = greetingRepository.save(greeting);
        return savedGreeting;
    }

}
```

```

@Override
public Greeting update(Greeting greeting) {
    Greeting greetingPersisted = findOne(greeting.getId());
    if (greetingPersisted == null) {
        //cannot find Greeting with specified Id value
        return null;
    }
    Greeting updatedGreeting = greetingRepository.save(greeting);
    return updatedGreeting;
}

@Override
public void delete(Long id) {
    greetingRepository.delete(id);
}
}

```

Класс контроллера

```

package org.springframework.web.api;

import java.util.Collection;
import org.springframework.model.Greeting;
import org.springframework.service.GreetingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/api")
public class GreetingController {

    @Autowired
    private GreetingService greetingService;

    // GET [method = RequestMethod.GET] is a default method for any request.
    // So we do not need to mention explicitly

    @RequestMapping(value = "/greetings", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Collection<Greeting>> getGreetings() {
        Collection<Greeting> greetings = greetingService.findAll();
        return new ResponseEntity<Collection<Greeting>>(greetings, HttpStatus.OK);
    }

    @RequestMapping(value = "/greetings/{id}", produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Greeting> getGreeting(@PathVariable("id") Long id) {
        Greeting greeting = greetingService.findOne(id);
        if(greeting == null) {
            return new ResponseEntity<Greeting>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<Greeting>(greeting, HttpStatus.OK);
    }
}

```

```

    }

    @RequestMapping(value = "/greetings", method = RequestMethod.POST, consumes =
MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Greeting> createGreeting(@RequestBody Greeting greeting) {
        Greeting savedGreeting = greetingService.create(greeting);
        return new ResponseEntity<Greeting>(savedGreeting, HttpStatus.CREATED);
    }

    @RequestMapping(value = "/greetings/{id}", method = RequestMethod.PUT, consumes =
MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<Greeting> updateGreeting(@PathVariable("id") Long id, @RequestBody
Greeting greeting) {
        Greeting updatedGreeting = null;
        if (greeting != null && id == greeting.getId()) {
            updatedGreeting = greetingService.update(greeting);
        }
        if(updatedGreeting == null) {
            return new ResponseEntity<Greeting>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
        return new ResponseEntity<Greeting>(updatedGreeting, HttpStatus.OK);
    }

    @RequestMapping(value = "/greetings/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<Greeting> deleteGreeting(@PathVariable("id") Long id) {
        greetingService.delete(id);
        return new ResponseEntity<Greeting>(HttpStatus.NO_CONTENT);
    }
}

```

Файл свойств приложения для базы данных MySQL

```

#mysql config
spring.datasource.url=jdbc:mysql://localhost:3306/springboot
spring.datasource.username=root
spring.datasource.password=Welcome@123
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto = update

spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.DefaultNamingStrategy

#initialization
spring.datasource.schema=classpath:/data/schema.sql

```

Файл SQL

```

drop table if exists greeting;
create table greeting (
    id bigint not null auto_increment,
    text varchar(100) not null,
    primary key(id)
);

```

Файл pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org</groupId>
<artifactId>springboot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.2.1.RELEASE</version>
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Создание исполняемого JAR

Вы можете запустить приложение из командной строки с помощью Maven. Или вы можете создать один исполняемый JAR-файл, содержащий все необходимые зависимости, классы и ресурсы, и запустите его. Это упрощает отправку, версию и развертывание службы в виде приложения на протяжении всего жизненного цикла разработки, в разных средах и т. Д.

Запустите приложение, используя `./mvnw spring-boot:run`. Или вы можете создать JAR-файл с `./mvnw clean package`. Затем вы можете запустить JAR-файл:

```
java -jar target/springboot-0.0.1-SNAPSHOT.jar
```

Прочитайте [Весенняя загрузка + Spring Data JPA онлайн](https://riptutorial.com/ru/spring-boot/topic/6203/весенняя-загрузка-plus-spring-data-jpa): <https://riptutorial.com/ru/spring-boot/topic/6203/весенняя-загрузка-plus-spring-data-jpa>

глава 11: Контроллеры

Вступление

В этом разделе я добавлю пример для контроллера загрузки Spring Spring с запросом Get и post.

Examples

Контроллер останова пружины.

В этом примере я покажу, как сформулировать контроллер отдыха для получения и отправки данных в базу данных с использованием JPA с наиболее легкостью и наименьшим кодом.

В этом примере мы будем ссылаться на таблицу данных с именем customerRequirement.

BuyingRequirement.java

@Entity @Table (name = "BUYINGREQUIREMENTS") @NamedQueries ({@NamedQuery (name = "BuyingRequirement.findAll", query = "SELECT b FROM BuyingRequirement b")}) Открытый класс BuyingRequirement расширяет Доменные инструменты Serializable {private static final long serialVersionUID = 1L;

```
@Column(name = "PRODUCT_NAME", nullable = false)
private String productname;

@Column(name = "NAME", nullable = false)
private String name;

@Column(name = "MOBILE", nullable = false)
private String mobile;

@Column(name = "EMAIL", nullable = false)
private String email;

@Column(name = "CITY")
private String city;

public BuyingRequirement() {
}

public String getProductname() {
    return productname;
}

public void setProductname(String productname) {
    this.productname = productname;
}
```



```

    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getMobile() {
        return mobile;
    }

    public void setMobile(String mobile) {
        this.mobile = mobile;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }
}

```

Это класс сущности, который включает в себя параметр, относящийся к столбцам в таблице buyingRequirement и их получателях и сеттерах.

IBuyingRequirementsRepository.java (интерфейс JPA)

```

@Repository
@RepositoryRestResource
public interface IBuyingRequirementsRepository extends JpaRepository<BuyingRequirement, UUID>
{
    // Page<BuyingRequirement> findAllByOrderByCreatedDesc(Pageable pageable);
    Page<BuyingRequirement> findAllByOrderByCreatedDesc(Pageable pageable);
    Page<BuyingRequirement> findByNameContainingIgnoreCase(@Param("name") String name,
    Pageable pageable);
}

```

BuyingRequirementController.java

```

@RestController
@RequestMapping("/api/v1")
public class BuyingRequirementController {

```

```

@Autowired
IBuyingRequirementsRepository iBuyingRequirementsRepository;
Email email = new Email();

BuyerRequirementTemplate buyerRequirementTemplate = new BuyerRequirementTemplate();

private String To = "support@pharmerz.com";
// private String To = "amigujarathi@gmail.com";
private String Subject = "Buyer Request From Pharmerz ";

@PostMapping(value = "/buyingRequirement")
public ResponseEntity<BuyingRequirement> CreateBuyingRequirement (@RequestBody
BuyingRequirement buyingRequirements) {

    String productname = buyingRequirements.getProductname();
    String name = buyingRequirements.getName();
    String mobile = buyingRequirements.getMobile();
    String emails = buyingRequirements.getEmail();
    String city = buyingRequirements.getCity();
    if (city == null) {
        city = "-";
    }

    String HTMLBODY = buyerRequirementTemplate.template(productname, name, emails, mobile,
city);

    email.SendMail(To, Subject, HTMLBODY);

    iBuyingRequirementsRepository.save(buyingRequirements);
    return new ResponseEntity<BuyingRequirement>(buyingRequirements, HttpStatus.CREATED);
}

@GetMapping(value = "/buyingRequirements")
public Page<BuyingRequirement> getAllBuyingRequirements(Pageable pageable) {

    Page requirements =
iBuyingRequirementsRepository.findAllByOrderByCreatedDesc(pageable);
    return requirements;
}

@GetMapping(value = "/buyingRequirmentByName/{name}")
public Page<BuyingRequirement> getByName(@PathVariable String name,Pageable pageable){
    Page buyersByName =
iBuyingRequirementsRepository.findByNameContainingIgnoreCase(name,pageable);

    return buyersByName;
}
}

```

Он включает в себя метод

1. Post, который отправляет данные в базу данных.
2. Получить метод, который извлекает все записи из таблицы buyRequirement.
3. Это также метод получения, который найдет требование покупки по имени человека.

Прочитайте Контроллеры онлайн: <https://riptutorial.com/ru/spring-boot/topic/10635/>

глава 12: Кэширование с помощью Redis

Использование Spring Boot для MongoDB

Examples

Почему кэширование?

Сегодня производительность является одной из наиболее важных показателей, которые необходимо оценить при разработке веб-сервиса / приложения. Поддержание вовлеченности клиентов имеет решающее значение для любого продукта, и по этой причине чрезвычайно важно улучшить производительность и сократить время загрузки страницы.

При запуске веб-сервера, который взаимодействует с базой данных, его операции могут стать узким местом. MongoDB здесь не является исключением, и по мере того, как наша база данных MongoDB масштабируется, все может действительно замедляться. Эта проблема может даже ухудшиться, если сервер базы данных отсоединен от веб-сервера. В таких системах связь с базой данных может вызвать большие накладные расходы.

К счастью, мы можем использовать метод `caching` для ускорения работы. В этом примере мы представим этот метод и посмотрим, как мы можем использовать его для повышения производительности нашего приложения с использованием Spring Cache, Spring Data и Redis.

Базовая система

В качестве первого шага мы создадим базовый веб-сервер, который хранит данные в MongoDB. Для этой демонстрации мы назовем ее «быстрой библиотекой». Сервер будет иметь две основные операции:

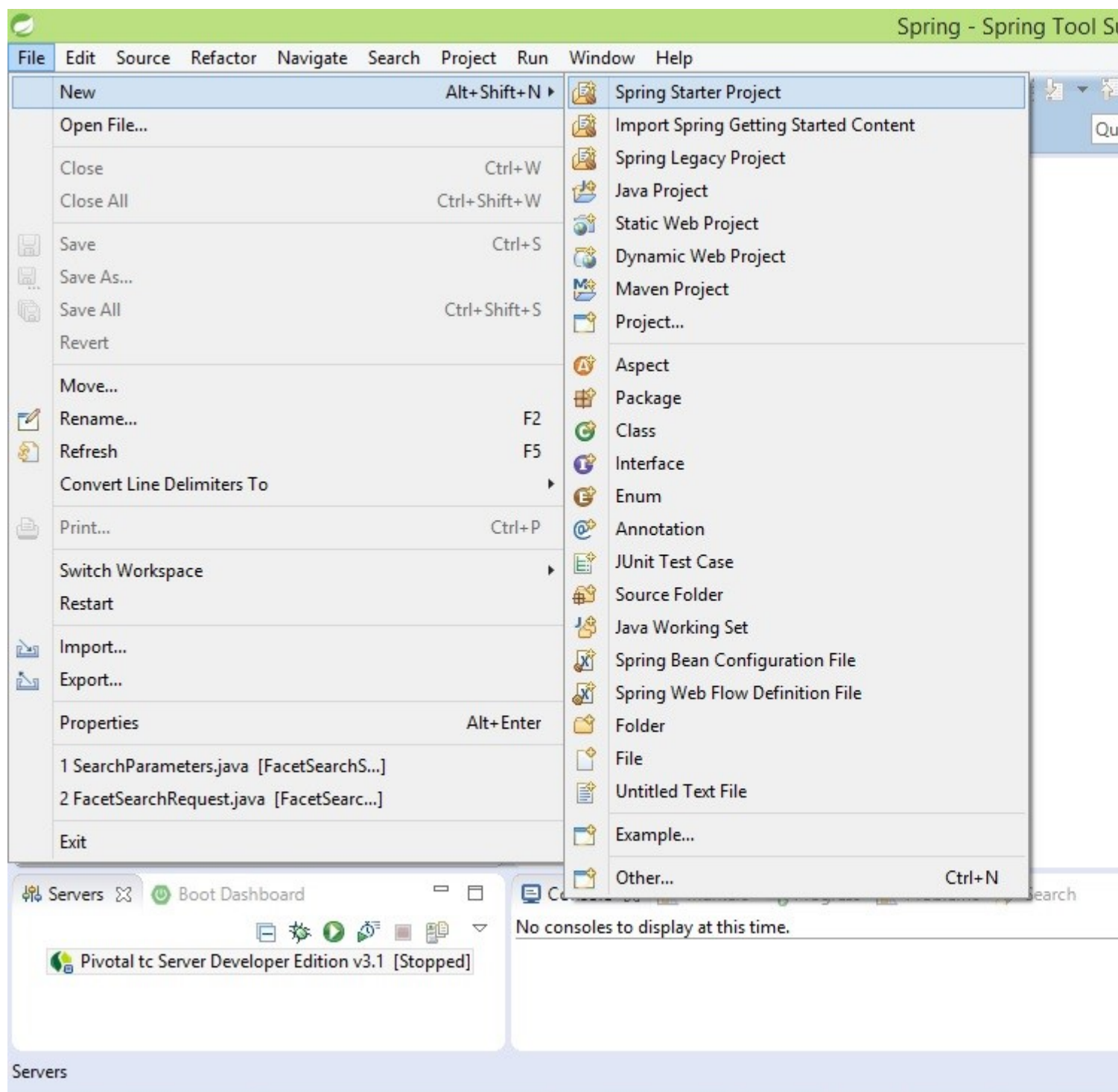
`POST /book` : эта конечная точка получит название, автора и содержание книги и создаст запись в базе данных.

`GET /book/ {title}` : эта конечная точка получит заголовок и вернет его содержимое. Мы предполагаем, что названия однозначно идентифицируют книги (таким образом, не будет двух книг с одним и тем же названием). Лучшей альтернативой было бы, конечно, использовать идентификатор. Однако, чтобы все было просто, мы просто используем заголовок.

Это простая библиотечная система, но позже мы добавим более сложные возможности.

Теперь давайте создадим проект, используя Spring Tool Suite (сборка с использованием

eclipse) и проект Spring Starter



Мы строим наш проект с использованием Java, и для сборки мы используем maven, выбираем значения и нажимаем на следующий

New Spring Starter Project

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:


Package:

Working sets

☐ Add project to working sets


Working sets:

Выберите MongoDB, Redis из NOSQL и Web из веб-модуля и нажмите «Готово». Мы используем Lombok для автоматической генерации Setters и getters значений модели, поэтому нам нужно добавить зависимость Lombok к POM



New Spring Starter Project

<input type="checkbox"/> Security <input type="checkbox"/> Cache <input type="checkbox"/> Retry	<input type="checkbox"/> AOP <input type="checkbox"/> DevTools <input type="checkbox"/> Lombok	<input type="checkbox"/> Atomikos (JTA) <input type="checkbox"/> Validation	<input type="checkbox"/> Bitronix (JTA) <input type="checkbox"/> Session
I/O <input type="checkbox"/> Batch <input type="checkbox"/> JMS (HornetQ)	<input type="checkbox"/> Integration <input type="checkbox"/> AMQP	<input type="checkbox"/> Activiti <input type="checkbox"/> Mail	<input type="checkbox"/> JMS (Artemis)
NoSQL <input checked="" type="checkbox"/> MongoDB <input checked="" type="checkbox"/> Redis	<input type="checkbox"/> Cassandra <input type="checkbox"/> Gemfire	<input type="checkbox"/> Couchbase <input type="checkbox"/> Solr	<input type="checkbox"/> Redis <input type="checkbox"/> Elasticsearch
Ops <input type="checkbox"/> Actuator	<input type="checkbox"/> Actuator Docs	<input type="checkbox"/> Remote Shell	
SQL <input type="checkbox"/> JPA <input type="checkbox"/> HSQLDB	<input type="checkbox"/> JOOQ <input type="checkbox"/> Apache Derby	<input type="checkbox"/> JDBC <input type="checkbox"/> MySQL	<input type="checkbox"/> H2 <input type="checkbox"/> PostgreSQL
Social <input type="checkbox"/> Facebook	<input type="checkbox"/> LinkedIn	<input type="checkbox"/> Twitter	
Template Engines <input type="checkbox"/> Freemarker <input type="checkbox"/> Mustache	<input type="checkbox"/> Velocity	<input type="checkbox"/> Groovy Templates	<input type="checkbox"/> Thymeleaf
Web <input checked="" type="checkbox"/> Web <input type="checkbox"/> Ratpack <input type="checkbox"/> Rest Repositories HAL Browser	<input type="checkbox"/> Websocket <input type="checkbox"/> Vaadin <input type="checkbox"/> Mobile	<input type="checkbox"/> WS <input type="checkbox"/> Rest Repositories <input type="checkbox"/> REST Docs	<input type="checkbox"/> Jersey (JAX-RS) <input type="checkbox"/> HATEOAS



```

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

```

MongoDbRedisCacheApplication.java содержит основной метод, который используется для запуска Spring Boot Application add

Создать класс класса Книга, которая содержит id, название книги, автора, описание и аннотацию с помощью @Data для создания автоматических сеттеров и геттеров из jar project lombok

```

package com.example;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import lombok.Data;
@Data
public class Book {
    @Id
    private String id;
    @Indexed
    private String title;
    private String author;
    private String description;
}

```

Spring Data создает для нас все основные операции CRUD, поэтому давайте создадим BookRepository.Java, который находит книгу по названию и удаляет книгу

```

package com.example;
import org.springframework.data.mongodb.repository.MongoRepository;
public interface BookRepository extends MongoRepository<Book, String>
{
    Book findByTitle(String title);
    void delete(String title);
}

```



```
}
```

Давайте создадим `webservicesController`, который сохраняет данные в MongoDB и извлекает данные по `idTitle` (`@PathVariable String title`).

```
package com.example;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class WebServicesController {
    @Autowired
    BookRepository repository;
    @Autowired
    MongoTemplate mongoTemplate;
    @RequestMapping(value = "/book", method = RequestMethod.POST)
    public Book saveBook(Book book)
    {
        return repository.save(book);
    }
    @RequestMapping(value = "/book/{title}", method = RequestMethod.GET)
    public Book findBookByTitle(@PathVariable String title)
    {
        Book insertedBook = repository.findByTitle(title);
        return insertedBook;
    }
}
```

Добавление кэша До сих пор мы создали базовый веб-сервис библиотеки, но это не удивительно быстро. В этом разделе мы попытаемся оптимизировать метод `findBookByTitle` () путем кэширования результатов.

Чтобы лучше понять, как мы достигнем этой цели, давайте вернемся к примеру людей, сидящих в традиционной библиотеке. Предположим, они хотят найти книгу с определенным названием. Прежде всего, они будут оглядываться вокруг стола, чтобы посмотреть, не привезли ли они туда. Если они есть, это здорово! У них просто был кеш-хит, который находит элемент в кеше. Если они их не нашли, у них был промах кеша, то есть они не нашли элемент в кеше. В случае недостающего элемента им придется искать книгу в библиотеке. Когда они его найдут, они будут держать их на столе или вставить в кеш.

В нашем примере мы будем следовать точно так же алгоритму метода `findBookByTitle` (). Когда его попросят создать книгу с определенным названием, мы будем искать ее в кеше. Если не найти, мы будем искать его в основном хранилище, то есть в нашей базе данных MongoDB.

Использование Redis

Добавление `spring-boot-data-redis` в наш путь к классу позволит весенней загрузке

выполнять свою магию. Он будет создавать все необходимые операции путем автоматической настройки

Теперь давайте теперь аннотировать метод с нижеследующей строкой для кеширования и пусть весенняя загрузка сделает свою магию

```
@Cacheable (value = "book", key = "#title")
```

Чтобы удалить из кеша, когда запись удалена, просто добавьте в нее комментарии в строке ниже в окне BookRepository и дайте нам удалить кеш-память Spring Boot Spring.

```
@CacheEvict (value = "book", key = "#title")
```

Чтобы обновить данные, нам нужно добавить строку ниже к методу, и пусть весенний дескриптор загрузки

```
@CachePut (value = "book", key = "#title")
```

Вы можете найти полный код проекта в [GitHub](#)

Прочитайте [Кэширование с помощью Redis Использование Spring Boot для MongoDB](#)
онлайн: <https://riptutorial.com/ru/spring-boot/topic/6496/кэширование-с-помощью-redis-использование-spring-boot-для-mongodb>

глава 13: Подключение приложения Spring-Boot к MySQL

Вступление

Мы знаем, что весна-загрузка по умолчанию работает с использованием базы данных H2. В этой статье мы увидим, как настроить конфигурацию по умолчанию для работы с базой данных MySQL.

замечания

В качестве предварительного условия убедитесь, что MySQL уже запущен на порт 3306 и создана ваша база данных.

Examples

Пример Spring-boot с использованием MySQL

Мы будем следовать [официальному руководству для весенних ботинок](#) и [spring-data-jpa](#). Мы будем создавать приложение, используя gradle.

1. Создайте файл сборки градиента

build.gradle

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:1.4.3.RELEASE")
    }
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'idea'
apply plugin: 'org.springframework.boot'

jar {
    baseName = 'gs-accessing-data-jpa'
    version = '0.1.0'
}

repositories {
    mavenCentral()
    maven { url "https://repository.jboss.org/nexus/content/repositories/releases" }
```

```

}

sourceCompatibility = 1.8
targetCompatibility = 1.8

dependencies {
    compile("org.springframework.boot:spring-boot-starter-data-jpa")
    runtime('mysql:mysql-connector-java')
    testCompile("junit:junit")
}

```

2. Создание объекта клиента

SRC / Основной / Java / Привет / Customer.java

```

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;

    protected Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return String.format(
            "Customer[id=%d, firstName='%s', lastName='%s']",
            id, firstName, lastName);
    }

}

```

3. Создание репозитория

SRC / Основной / Java / Привет / CustomerRepository.java

```

import java.util.List;
import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {
    List<Customer> findByLastName(String lastName);
}

```

4. Создать файл application.properties

```

##### DataSource Configuration #####
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/your_database_name
jdbc.username=username

```

```

jdbc.password=password

init-db=false

##### Hibernate Configuration #####

hibernate.dialect=org.hibernate.dialect.MySQLDialect
hibernate.show_sql=true
hibernate.hbm2ddl.auto=update

```

5. Создайте файл PersistenceConfig.java

На шаге 5 мы определим, как загрузится источник данных и как наше приложение подключится к MySQL. Вышеприведенный фрагмент - это минимальная конфигурация, необходимая для подключения к MySQL. Здесь мы предлагаем две бобы:

```

@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(basePackages="hello")
public class PersistenceConfig
{
    @Autowired
    private Environment env;

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory()
    {
        LocalContainerEntityManagerFactoryBean factory = new
LocalContainerEntityManagerFactoryBean();

        HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        vendorAdapter.setGenerateDdl(Boolean.TRUE);
        vendorAdapter.setShowSql(Boolean.TRUE);

        factory.setDataSource(dataSource());
        factory.setJpaVendorAdapter(vendorAdapter);
        factory.setPackagesToScan("hello");

        Properties jpaProperties = new Properties();
        jpaProperties.put("hibernate.hbm2ddl.auto",
env.getProperty("hibernate.hbm2ddl.auto"));
        factory.setJpaProperties(jpaProperties);

        factory.afterPropertiesSet();
        factory.setLoadTimeWeaver(new InstrumentationLoadTimeWeaver());
        return factory;
    }

    @Bean
    public DataSource dataSource()
    {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(env.getProperty("jdbc.driverClassName"));
        dataSource.setUrl(env.getProperty("jdbc.url"));
        dataSource.setUsername(env.getProperty("jdbc.username"));
        dataSource.setPassword(env.getProperty("jdbc.password"));
        return dataSource;
    }
}

```

```
}
}
}
```

- **LocalContainerEntityManagerFactoryBean** Это дает нам представление о конфигурациях EntityManagerFactory и позволяет нам выполнять настройки. Это также позволяет нам вводить PersistenceContext в наши компоненты, как показано ниже:

```
@PersistenceContext
private EntityManager em;
```

- **DataSource** Здесь мы возвращаем экземпляр DriverManagerDataSource . Это простая реализация стандартного интерфейса JDBC DataSource, настройка простого старого драйвера JDBC через свойства bean и возврат нового соединения для каждого вызова getConnection. Обратите внимание, что я рекомендую использовать это строго для целей тестирования, так как есть более доступные альтернативы, такие как BasicDataSource . Подробнее см. [Здесь](#)

6. Создание класса приложения

SRC / Основной / Java / Привет / Application.java

```
@SpringBootApplication
public class Application {

    private static final Logger log = LoggerFactory.getLogger(Application.class);

    @Autowired
    private CustomerRepository repository;

    public static void main(String[] args) {
        SpringApplication.run(TestCoreApplication.class, args);
    }

    @Bean
    public CommandLineRunner demo() {
        return (args) -> {
            // save a couple of customers
            repository.save(new Customer("Jack", "Bauer"));
            repository.save(new Customer("Chloe", "O'Brian"));
            repository.save(new Customer("Kim", "Bauer"));
            repository.save(new Customer("David", "Palmer"));
            repository.save(new Customer("Michelle", "Dessler"));

            // fetch all customers
            log.info("Customers found with findAll():");
            log.info("-----");
            for (Customer customer : repository.findAll()) {
                log.info(customer.toString());
            }
            log.info("");

            // fetch an individual customer by ID
```

```

        Customer customer = repository.findOne(1L);
        log.info("Customer found with findOne(1L):");
        log.info("-----");
        log.info(customer.toString());
        log.info("");

        // fetch customers by last name
        log.info("Customer found with findByLastName('Bauer'):");
        log.info("-----");
        for (Customer bauer : repository.findByLastName("Bauer")) {
            log.info(bauer.toString());
        }
        log.info("");
    };
}

```

```

}

```

7. Запуск приложения

Если вы используете IDE, такую как STS, вы можете просто щелкнуть правой кнопкой мыши свой проект -> Запустить как -> Gradle (STS) Build ... В списке задач введите bootRun и Run.

Если вы используете gradle в командной строке, вы можете просто запустить приложение следующим образом:

```
./gradlew bootRun
```

Вы должны увидеть что-то вроде этого:

```

== Customers found with findAll():
Customer[id=1, firstName='Jack', lastName='Bauer']
Customer[id=2, firstName='Chloe', lastName='O'Brian']
Customer[id=3, firstName='Kim', lastName='Bauer']
Customer[id=4, firstName='David', lastName='Palmer']
Customer[id=5, firstName='Michelle', lastName='Dessler']

== Customer found with findOne(1L):
Customer[id=1, firstName='Jack', lastName='Bauer']

== Customer found with findByLastName('Bauer'):
Customer[id=1, firstName='Jack', lastName='Bauer']
Customer[id=3, firstName='Kim', lastName='Bauer']

```

Прочитайте Подключение приложения Spring-Boot к MySQL онлайн:

<https://riptutorial.com/ru/spring-boot/topic/8588/подключение-приложения-spring-boot-к-mysql>

глава 14: Полно-отзывчивое веб-приложение Spring Boot с JHipster

Examples

Создать приложение Spring Boot с помощью jHipster в Mac OS

jHipster позволяет загружать веб-приложение Spring Boot с помощью интерфейса REST API и front-end AngularJS и Twitter Bootstrap.

Подробнее о jHipster здесь: [jHipster Documentation](#)

Установить вареву:

```
/usr/bin/ruby -e "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Посмотрите дополнительную информацию о том, как установить варить здесь: [Установить Brew](#)

Установить Gradle

Gradle - это система управления и сборки зависимостей.

```
brew install gradle
```

Установить Git

Git - это инструмент управления версиями

```
brew install git
```

Установка NodeJS

NodeJS предоставляет вам доступ к npm, менеджеру пакетов узлов, который необходим для установки других инструментов.

```
brew install node
```

Установить Йоман

Йомен - генератор

```
npm install -g yo
```


Установить Bower

Bower - это инструмент управления зависимостями

```
npm install -g bower
```

Установите Gulp

Gulp - это задача

```
npm install -g gulp
```

Установить jHipster Yeoman Generator

Это генератор jHipster

```
npm install -g generator-jhipster
```

Создать приложение

Откройте окно терминала.

Перейдите в корневую директорию, в которой будут храниться ваши проекты. Создайте пустой каталог, в котором вы создадите приложение.

```
mkdir myapplication
```

Перейдите в этот каталог

```
cd myapplication/
```

Чтобы создать приложение, введите

```
yo jhipster
```

Вам будут предложены следующие вопросы:

Какой тип приложения вы бы хотели создать?

Тип вашего приложения зависит от того, хотите ли вы использовать архитектуру микросервисов или нет. Подробное объяснение по микросервисам доступно здесь, если вы не уверены в использовании стандартного «Монолитного приложения».

Выберите *Монолитное приложение* по умолчанию, если вы не уверены

Каково ваше имя пакета Java по умолчанию?

Ваше приложение Java будет использовать это в качестве своего корневого пакета.

Какой тип аутентификации вы бы хотели использовать?

Используйте базовую сессионную *Spring Security* по умолчанию, если вы не уверены

Какой тип базы данных вы бы хотели использовать?

Какую базу данных разработки вы бы хотели использовать?

Это база данных, которую вы будете использовать с профилем «разработки». Вы можете использовать:

Используйте H2 по умолчанию, если вы не уверены

H2, работающий в памяти. Это самый простой способ использования JHipster, но ваши данные будут потеряны при перезагрузке вашего сервера.

Вы хотите использовать кеш второго уровня Hibernate?

Hibernate - это поставщик JPA, используемый JHipster. По соображениям производительности мы настоятельно рекомендуем использовать кеш и настроить его в соответствии с потребностями вашего приложения. Если вы решите это сделать, вы можете использовать либо ehcache (локальный кеш), либо Hazelcast (распределенный кеш для использования в кластерной среде)

Вы хотите использовать поисковую систему в своем приложении? Elasticsearch будет настроен с использованием Spring Data Elasticsearch. Дополнительную информацию вы можете найти в нашем Руководстве по Elasticsearch.

Выберите «нет», если вы не уверены

Вы хотите использовать кластерные сеансы HTTP?

По умолчанию JHipster использует сеанс HTTP только для хранения информации об аутентификации и авторизации Spring Security. Конечно, вы можете поместить больше данных в свои сеансы HTTP. Использование сеансов HTTP вызовет проблемы, если вы работаете в кластере, особенно если вы не используете балансировщик нагрузки с «липкими сеансами». Если вы хотите реплицировать свои сессии внутри своего кластера, выберите этот параметр, чтобы настроить Hazelcast.

Выберите «нет», если вы не уверены

Вы хотите использовать WebSockets? Websockets можно включить с помощью Spring Websocket. Мы также предоставляем полный образец, чтобы показать вам, как эффективно использовать структуру.

Выберите «нет», если вы не уверены

Вы хотите использовать Maven или Gradle? Вы можете создать свое сгенерированное Java-приложение либо с Maven, либо с Gradle. Maven более стабилен и более зрелый. Gradle более гибкая, удобнее для расширения и больше шумихи.

Выберите *Gradle*, если вы не уверены

Вы хотите использовать препроцессор стилей LibSass для вашего CSS? Node-sass - отличное решение для упрощения проектирования CSS. Чтобы эффективно использоваться, вам необходимо запустить сервер Gulp, который будет настроен автоматически.

Выберите «нет», если вы не уверены

Вы хотите включить поддержку перевода с помощью Angular Translate? По умолчанию JHipster обеспечивает отличную поддержку интернационализации, как на стороне клиента с помощью Angular Translate, так и на стороне сервера. Однако интернационализация добавляет немного накладных расходов и немного сложнее управлять, поэтому вы можете не устанавливать эту функцию.

Выберите «нет», если вы не уверены

Какие платформы тестирования вы хотели бы использовать? По умолчанию JHipster предоставляет тестирование модуля Java / интеграции (используя поддержку Spring JUnit) и тестирование модуля JavaScript (используя Karma.js). В качестве опции вы также можете добавить поддержку:

Если вы не уверены, выберите его. По умолчанию у вас будет доступ к junit и Karma.

Прочитайте Полно-отзывчивое веб-приложение Spring Boot с JHipster онлайн:


<https://riptutorial.com/ru/spring-boot/topic/6297/полно-отзывчивое-веб-приложение-spring-boot-с-jhipster>

глава 15: Развертывание примера приложения с использованием Spring-загрузки на Amazon Elastic Beanstalk

Examples

Развертывание примера приложения с использованием Spring-boot в формате Jar на AWS

1. Создайте образец приложения, используя весеннюю загрузку с сайта [инициализатора Spring Boot](#) .
2. Импортируйте код в локальную среду IDE и запустите цель как **чистая установка spring-boot: run -e**
3. Перейдите в целевую папку и проверьте файл jar.
4. Откройте свою учетную запись Amazon или создайте новую учетную запись Amazon и выберите для эластичного бобового стежка, как показано ниже.





AWS


Services

Edit

History

 Elastic Beanstalk

 SQS

 Console Home

All AWS Services

Compute

Storage & Content Delivery

Database

Networking

Developer Tools

Management Tools

Security & Identity

Analytics

Internet of Things


Mobile Services


Application Services


Enterprise Applications


Game Development


>


 API Gateway


 AppStream


 AWS IoT


 Certificate Manager


 CloudFormation


 CloudFront


 CloudSearch


 CloudTrail


 CloudWatch


 CodeCommit


 CodeDeploy

 CodePipeline


 Cognito

 Config

 Data Pipeline


 Device Farm

5. Создайте новую среду веб-сервера, как показано ниже

AWS

Services

Edit

Elastic Beanstalk

My First Elastic Beanstalk Application

New Environment

Environment Type

Application Version

Environment Info

Additional Resources

Configuration Details

Environment Tags

Permissions

Review Information

New Environment


AWS Elastic Beanstalk has two types of environment tiers to support different process HTTP requests, typically over port 80. Workers are specialized applications that process messages in a queue. Worker applications post those messages to your application by using the Amazon SQS API.

Web Server Environment


Provides resources for an AWS Elastic Beanstalk web server in either a single instance or an auto scaling environment. [Learn more.](#)

Worker Environment*

Provides resources for an AWS Elastic Beanstalk worker application in either a single instance or an auto scaling environment. [Learn more.](#)

 * Worker environments require additional permissions to access Amazon SQS.

6. Выберите тип среды как Java для развертывания файла **JAR** для Spring-boot, если вы планируете развертывать его как **WAR**- файл, его следует выбрать как tomcat, как показано ниже

AWS

Services

Edit

Elastic Beanstalk

My First Elastic Beanstalk Application

New Environment

Environment Type

Application Version

Environment Info

Additional Resources

Configuration Details

Environment Tags

Permissions

Review Information


Environment Type

Choose the platform and type of environment to launch.

Predefined configuration: Java ▼ Loc...

AWS Elastic Beanstalk will create an environment

Environment type: Load balancing, auto scaling ▼ Lea...

AWS

Services

Edit

Elastic Beanstalk

My First Elastic Beanstalk Application

New Environment

Environment Type

Application Version

Environment Info

Additional Resources

Configuration Details

Environment Tags

Permissions

Review Information

Environment Type

Choose the platform and type of environment to launch.

Predefined configuration: Tomcat ▼ L...

AWS Elastic Beanstalk will create an environ

Environment type: Load balancing, auto scaling ▼ L...

7. Выберите с настройкой по умолчанию при нажатии следующей следующей ...
8. Как только вы заполните конфигурацию по умолчанию, на обзорном экране JAR-файл можно загрузить и развернуть, как показано на рисунках.

[All Applications](#) > [My First Elastic Beanstalk Application](#) > [Default-Env](#)

[est-2.elasticbeanstalk.com](#))

Dashboard

Configuration

Logs

Health

Monitoring

Alarms

Managed Updates **NEW**

Events

Tags

Overview



Health

Ok

Causes

Recent Events

Time	Type	Details
2016-08-27 03:36:06 UTC+0530	INFO	Environment health has
2016-08-27 03:35:06 UTC+0530	WARN	Environment health has
2016-08-26 13:15:58 UTC+0530	INFO	Deleted log fragments f
2016-08-26 13:12:47 UTC+0530	INFO	Environment health has

Upload and Deploy

To deploy a previous version, go to the [Application Versions](#) page.

Upload application:

Choose File

No file chosen

Version label:

Deployment Preferences

Current number of instances: 1

Cancel

Deploy

9. Как только развертывание будет успешным (5-10 минут в первый раз), вы можете нажать на URL-адрес контекста, как показано на рисунке ниже.

AWS

Services

Edit

Elastic Beanstalk

My First Elastic Beanstalk Application

All Applications > My First Elastic Beanstalk Application > Default-Env

[est-2.elasticbeanstalk.com](#)

Dashboard

Configuration

Logs

Health

Monitoring

Alarms

Managed Updates NEW

Overview

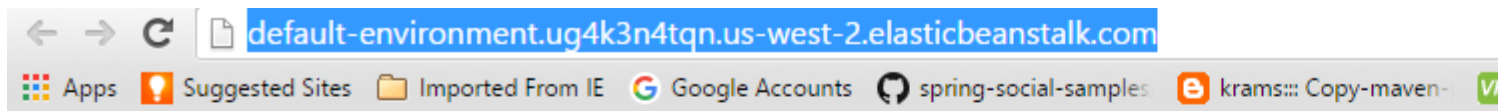
Health

Ok

Causes

Recent Events

10. Результат такой, как показано ниже, он должен работать так же, как с вашим локальным env.



Demo Boot

11. Пожалуйста, найдите мой [URL-адрес Github](#)

Прочитайте Развертывание примера приложения с использованием Spring-загрузки на Amazon Elastic Beanstalk онлайн: <https://riptutorial.com/ru/spring-boot/topic/6117/развертывание-примера-приложения-с-использованием-spring-загрузки-на-amazon-elastic-beanstalk>

глава 16: Сканирование пакетов

Вступление

В этом разделе я расскажу о сканировании весеннего пакета загрузки.

Вы можете найти основную информацию в документах весенней загрузки в следующей ссылке ([используя-boot-structuring-your-code](#)), но я постараюсь предоставить более подробную информацию.

Весенняя загрузка и весна в целом обеспечивают функцию автоматического сканирования пакетов для определенных аннотаций для создания `beans` и `configuration`.

параметры

аннотирование	подробности
@SpringBootApplication	Основная аннотация. используется один раз в приложении, содержит основной метод и действует как основной пакет для сканирования пакетов
@SpringBootConfiguration	Указывает, что класс предоставляет приложение Spring Boot. Должен быть объявлен только один раз в приложении, обычно автоматически, устанавливая @SpringBootApplication
@EnableAutoConfiguration	Включить автоматическую настройку контекста Spring Spring. Должен быть объявлен только один раз в приложении, обычно автоматически, устанавливая @SpringBootApplication
@ComponentScan	Используется для запуска автоматического сканирования пакетов на определенном пакете и его дочерних элементах или для установки пользовательского сканирования пакетов
@Configuration	Используется для объявления одного или нескольких методов @Bean . Можно выбрать автоматическое сканирование пакетов, чтобы объявить один или несколько методов @Bean вместо традиционной конфигурации xml
@Bean	Указывает, что метод создает компонент, который должен управляться контейнером Spring. Обычно аннотированные

аннотирование	подробности
	методы <code>@Bean</code> будут помещаться в аннотированные классы <code>@Configuration</code> которые будут отобраны при сканировании пакетов для создания компонентов, основанных на конфигурации Java.
<code>@Component</code>	Объявляя класс как <code>@Component</code> он становится кандидатом на автоматическое обнаружение при использовании аннотационной конфигурации и сканирования классов. Обычно класс, аннотированный с помощью <code>@Component</code> , станет <code>bean</code> в приложении
<code>@Repository</code>	Первоначально он был определен Domain-Driven Design (Evans, 2003) как «механизм инкапсуляции хранилища. Обычно он используется для указания <code>Repository</code> для <code>spring data</code>
<code>@Service</code>	Очень похоже на <code>@Component</code> . первоначально разработанный Domain-Driven Design (Evans, 2003) как «операция, предлагаемая как интерфейс, который стоит отдельно в модели без инкапсулированного состояния».
<code>@Controller</code>	Указывает, что аннотированный класс является «контроллером» (например, веб-контроллером).
<code>@RestController</code>	Удобная аннотация, которая сама аннотируется с помощью <code>@Controller</code> и <code>@ResponseBody</code> . Будет автоматически выбран по умолчанию, поскольку он содержит аннотацию <code>@Controller</code> по умолчанию.

Examples

`@SpringBootApplication`

Самый простой способ структурирования вашего кода с использованием весенней загрузки для хорошего автоматического сканирования пакетов - `@SpringBootApplication` аннотация `@SpringBootApplication`. Эта аннотация предоставляет в себе 3 другие аннотации, которые помогают с автоматическим сканированием: `@SpringBootConfiguration`, `@EnableAutoConfiguration`, `@ComponentScan` (больше информации об каждой аннотации в разделе «Parameters »).

`@SpringBootApplication` обычно размещается в основном пакете, а все остальные компоненты будут размещены в пакетах в этом файле:

```

com
+- example
  +- myproject
    +- Application.java (annotated with @SpringBootApplication)
    |
    +- domain
    |   +- Customer.java
    |   +- CustomerRepository.java
    |
    +- service
    |   +- CustomerService.java
    |
    +- web
    |   +- CustomerController.java

```

Если не указано иное, пружинная загрузка `@Configuration` обнаруживает `@Configuration`, `@Component`, `@Repository`, `@Service`, `@Controller`, `@RestController` аннотации автоматически под сканированными пакетами (`@Configuration` и `@RestController` выбираются, потому что они аннотируются `@Component` и `@Controller` соответственно).

Пример базового кода:

```

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}

```

Установка пакетов / классов в явном виде

Начиная с версии **1.3** вы также можете сказать весеннюю загрузку для сканирования определенных пакетов, установив `scanBasePackages` или `scanBasePackageClasses` в `@SpringBootApplication` вместо указания `@ComponentScan`.

1. `@SpringBootApplication(scanBasePackages = "com.example.myproject")` - установите `com.example.myproject` в качестве базового пакета для сканирования.
2. `@SpringBootApplication(scanBasePackageClasses = CustomerController.class)` - типа безопасная альтернатива `scanBasePackages` устанавливает пакет `CustomerController.java`, `com.example.myproject.web`, так как базовый пакет для сканирования.

Исключение автоматической настройки

Еще одна важная особенность - возможность исключения определенных классов автоматической конфигурации с использованием `exclude` или `excludeName` (`excludeName` существует с версии **1.3**).

1. `@SpringBootApplication(exclude = DemoConfiguration.class)` - исключает `DemoConfiguration` из автоматического сканирования пакетов.

2. `@SpringBootApplication(excludeName = "DemoConfiguration")` - будет делать то же самое с использованием полностью классифицированного имени класса.

@ComponentScan

Вы можете использовать `@ComponentScan` для настройки более сложного сканирования пакетов. Также есть `@ComponentScans` которые выступают в качестве аннотации к контейнеру, которая объединяет несколько аннотаций `@ComponentScan`.

Примеры базового кода

```
@ComponentScan
public class DemoAutoConfiguration {
}
```

```
@ComponentScans({@ComponentScan("com.example1"), @ComponentScan("com.example2")})
public class DemoAutoConfiguration {
}
```

`@ComponentScan` без конфигурации действует как `@SpringBootApplication` и сканирует все пакеты под классом, аннотированные этой аннотацией.

В этом примере я `@ComponentScan` некоторые полезные атрибуты `@ComponentScan`:

1. **basePackages** - может использоваться для указания определенных пакетов для сканирования.
2. **useDefaultFilters** - установив этот атрибут в `false` (по умолчанию `true`), вы можете убедиться, что весна не сканирует `@Component`, `@Repository`, `@Service` или `@Controller` автоматически.
3. **includeFilters** - может использоваться для *включения* конкретных весенних аннотаций / шаблонов регулярных выражений для включения в сканирование пакетов.
4. **excludeFilters** - может использоваться, чтобы *исключить* определенные шаблоны весенних аннотаций / регулярных выражений для включения в сканирование пакетов.

Есть гораздо больше атрибутов, но они наиболее часто используются для настройки сканирования пакетов.

Создание собственной автоконфигурации

Весенняя загрузка основана на множестве готовых исходных проектов автоматической конфигурации. Вы уже должны быть знакомы с стартовыми проектами весенних ботинок.

Вы можете легко создать свой собственный проект стартера, выполнив следующие простые шаги:

1. Создайте несколько классов `@Configuration` для определения `@Configuration` по умолчанию. Вы должны максимально использовать внешние свойства, чтобы разрешить настройку и пытаться использовать вспомогательные аннотации `@AutoConfigureBefore`, `@AutoConfigureAfter` как `@AutoConfigureBefore`, `@AutoConfigureAfter`, `@ConditionalOnBean`, `@ConditionalOnMissingBean` и т. Д. Более подробную информацию о каждой аннотации можно найти в официальной документации. [Аннотации условий](#)
2. Поместите файл / файлы автоматической конфигурации, который объединяет все классы `@Configuration`.
3. Создайте файл с именем `spring.factories` и поместите его в `src/main/resources/META-INF`.
4. В `spring.factories` установите свойство `org.springframework.boot.autoconfigure.EnableAutoConfiguration` с разделителями, разделенными запятыми, вашими классами `@Configuration`:

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.mycorp.libx.autoconfigure.LibXAutoConfiguration,\
com.mycorp.libx.autoconfigure.LibXWebAutoConfiguration
```

Используя этот метод, вы можете создать свои собственные классы автоматической настройки, которые будут выбраны весной-загрузкой. Spring-boot автоматически сканирует все зависимости maven / gradle для файла `spring.factories`, если он находит его, он добавляет все классы `@Configuration` указанные в нем, в процесс автоматической настройки.

Удостоверьтесь, что ваш проект стартовой `auto-configuration` не содержит `spring boot maven plugin` потому что он упакует проект как исполняемый JAR и не будет загружаться по пути класса, как предполагалось. Весенняя загрузка не сможет найти ваши `spring.factories` и не загрузит вашу конфигурацию

Прочитайте Сканирование пакетов онлайн: <https://riptutorial.com/ru/spring-boot/topic/9354/сканирование-пакетов>

глава 17: Создание и использование нескольких файлов `application.properties`

Examples

Среда Dev и Prod с использованием разных источников данных

После успешной установки приложения Spring-Boot вся конфигурация обрабатывается в файле `application.properties`. Файл находится в файле `src/main/resources/`.

Обычно требуется наличие базы данных позади приложения. Для разработки хорошо иметь настройку `dev` и среды `prod`. Используя несколько файлов `application.properties` вы можете указать Spring-Boot, с какой средой должно запускаться приложение.

Хорошим примером является настройка двух баз данных. Один для `dev` и один для `productive`.

Для среды `dev` вы можете использовать базу данных в памяти, такую как `H2`. Создайте новый файл в каталоге `src/main/resources/` именем `application-dev.properties`. Внутри файла находится конфигурация базы данных в памяти:

```
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
```

Для среды `prod` мы подключимся к «реальной» базе данных, например `postgresql`. Создайте новый файл в каталоге `src/main/resources/` именем `application-prod.properties`. Внутри файла находится конфигурация базы данных `postgresql`:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/yourDB
spring.datasource.username=postgres
spring.datasource.password=secret
```

В вашем файле `application.properties` умолчанию вы можете установить, какой профиль активирован и используется Spring-Boot. Просто установите один атрибут внутри:

```
spring.profiles.active=dev
```

или же

```
spring.profiles.active=prod
```

Важно то, что часть после `-` в `application-dev.properties` идентификатор файла.

Теперь вы можете запустить приложение Spring-Boot в режиме разработки или производства, просто изменив идентификатор. База данных в памяти будет запускаться или подключиться к «реальной» базе данных. Конечно, есть также много вариантов использования нескольких файлов свойств.

Установите правильный профиль пружины, создав приложение автоматически (maven)

Создавая несколько файлов свойств для разных сред или случаев использования, иногда трудно вручную изменить значение `active.profile` вправо. Но есть способ установить `active.profile` в файле `application.properties` при создании приложения с помощью `maven-profiles`.

Допустим, в нашем приложении есть три файла свойств среды:

`application-dev.properties` :

```
spring.profiles.active=dev
server.port=8081
```

`application-test.properties` :

```
spring.profiles.active=test
server.port=8082
```

`application-prod.properties` .

```
spring.profiles.active=prod
server.port=8083
```

Эти три файла просто отличаются портами и активным именем профиля.

В основном файле `application.properties` мы устанавливаем наш профиль пружины с использованием [переменной maven](#) :

`application.properties` .

```
spring.profiles.active=@profileActive@
```

После этого нам просто нужно добавить профили maven в наш `pom.xml` Мы будем устанавливать профили для всех трех сред:

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
```

```
<properties>
  <build.profile.id>dev</build.profile.id>
  <profileActive>dev</profileActive>
</properties>
</profile>
<profile>
  <id>test</id>
  <properties>
    <build.profile.id>test</build.profile.id>
    <profileActive>test</profileActive>
  </properties>
</profile>
<profile>
  <id>prod</id>
  <properties>
    <build.profile.id>prod</build.profile.id>
    <profileActive>prod</profileActive>
  </properties>
</profile>
</profiles>
```

Теперь вы можете создать приложение с помощью maven. Если вы не установили какой-либо профиль maven, его построение по умолчанию (в этом примере это dev). Для указания нужно использовать ключевое слово maven. Ключевое слово для установки профиля в maven - это -P за которым следует имя профиля: `mvn clean install -Ptest`.

Теперь вы также можете создавать пользовательские сборки и сохранять их в своей IDE для более быстрой сборки.

Примеры:

```
mvn clean install -Ptest
```

```

.      _ _ _ _ _
/\ \ / _ _ ' _ _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | | ' _ \ / _ ` | \ \ \ \ \
\ \ / _ _ ) | | _ | | | | | | | ( _ | | ) ) ) )
' | _ _ | . _ | _ | | _ | _ \ _ , | / / / /
=====| _ |=====| _ _ / = / _ / _ / _ /
:: Spring Boot ::                (v1.5.3.RELEASE)

2017-06-06 11:24:44.885 INFO 6328 --- [           main] com.demo.SpringBlobApplicationTests
: Starting SpringApplicationTests on KB242 with PID 6328 (started by me in
C:\DATA\Workspaces\spring-demo)
2017-06-06 11:24:44.886 INFO 6328 --- [           main] com.demo.SpringApplicationTests
: The following profiles are active: test

```

```
mvn clean install -Pprod
```

```

      .      _ _ _ _ _
/\ \ / _ _ ' _ _ _ _ ( _ ) _ _ _ _ \ \ \ \ \
( ( ) \ _ _ | ' _ | ' _ | | ' _ \ / _ ' | \ \ \ \ \
\ \ / _ _ ) | | _ | | | | | | | ( _ | | ) ) ) )
      ' | _ _ | . _ | _ | _ | _ | _ \ _ , | / / / /
=====|_|=====|_|_/ _/_/_/_/
:: Spring Boot ::                (v1.5.3.RELEASE)

```

```
2017-06-06 14:43:31.067 INFO 6932 --- [          main] com.demo.SpringBlobApplicationTests
: Starting SpringApplicationTests on KB242 with PID 6328 (started by me in
C:\DATA\Workspaces\spring-demo)
2017-06-06 14:43:31.069 INFO 6932 --- [          main] com.demo.SpringApplicationTests
: The following profiles are active: prod
```

Прочитайте [Создание и использование нескольких файлов application.properties](https://riptutorial.com/ru/spring-boot/topic/6334/создание-и-использование-нескольких-файлов-application-properties) онлайн:
<https://riptutorial.com/ru/spring-boot/topic/6334/создание-и-использование-нескольких-файлов-application-properties>

глава 18: Тестирование весной

Examples

Как протестировать приложение Spring Spring

У нас есть пример загрузки Spring, который хранит данные пользователя в MongoDB, и мы используем службы Rest для получения данных

Сначала есть класс домена, то есть POJO

```
@Document
public class User{
    @Id
    private String id;

    private String name;
}
```

Соответствующий репозиторий на основе Spring Data MongoDB

```
public interface UserRepository extends MongoRepository<User, String> {
}
```

Затем наш пользовательский контроллер

```
@RestController
class UserController {

    @Autowired
    private UserRepository repository;

    @RequestMapping("/users")
    List<User> users() {
        return repository.findAll();
    }

    @RequestMapping(value = "/Users/{id}", method = RequestMethod.DELETE)
    @ResponseStatus(HttpStatus.NO_CONTENT)
    void delete(@PathVariable("id") String id) {
        repository.delete(id);
    }

    // more controller methods
}
```

И, наконец, наше приложение для загрузки Spring

```
@SpringBootApplication
public class Application {
```

```

public static void main(String args[]){
    SpringApplication.run(Application.class, args);
}
}

```

Если, скажем так, что John Cena, The Rock и TripleH были единственными тремя пользователями в базе данных, запрос / пользователям дал бы следующий ответ:

```

$ curl localhost:8080/users
[{"name":"John Cena","id":"1"}, {"name":"The Rock","id":"2"}, {"name":"TripleH","id":"3"}]

```

Теперь, чтобы проверить код, мы проверим, работает ли приложение

```

@RunWith(SpringJUnit4ClassRunner.class)    // 1
@SpringApplicationConfiguration(classes = Application.class)    // 2
@WebAppConfiguration    // 3
@IntegrationTest("server.port:0")    // 4
public class UserControllerTest {

    @Autowired    // 5
    UserRepository repository;

    User cena;
    User rock;
    User tripleH;

    @Value("${local.server.port}")    // 6
    int port;

    @Before
    public void setUp() {
        // 7
        cena = new User("John Cena");
        rock = new User("The Rock");
        tripleH = new User("TripleH");

        // 8
        repository.deleteAll();
        repository.save(Arrays.asList(cena, rock, tripleH));

        // 9
        RestAssured.port = port;
    }

    // 10
    @Test
    public void testFetchCena() {
        String cenaId = cena.getId();

        when().
            get("/Users/{id}", cenaId).
        then().
            statusCode(HttpStatus.SC_OK).
            body("name", Matchers.is("John Cena")).
            body("id", Matchers.is(cenaId));
    }

    @Test

```

```

public void testFetchAll() {
    when().
        get("/users").
    then().
        statusCode(HttpStatus.SC_OK).
        body("name", Matchers.hasItems("John Cena", "The Rock", "TripleH"));
}

@Test
public void testDeletetripleHHH() {
    String tripleHHHId = tripleHHH.getId();

    when().
        .delete("/Users/{id}", tripleHHHId).
    then().
        statusCode(HttpStatus.SC_NO_CONTENT);
}
}

```

объяснение

1. Как и любой другой тест Spring, нам нужен `SpringJUnit4ClassRunner` чтобы создать контекст приложения.
2. `@SpringApplicationConfiguration` аналогично аннотации `@ContextConfiguration` в том, что оно используется для указания того, какой прикладной контекст (ы), который должен использоваться в тесте. Кроме того, он будет запускать логику для чтения определенных конфигураций Spring Spring, свойств и т. Д.
3. `@WebAppConfiguration` должен присутствовать, чтобы сообщить Spring, что для `WebApplicationContext` должен быть загружен `WebApplicationContext` . Он также предоставляет атрибут для указания пути к корню веб-приложения.
4. `@IntegrationTest` используется, чтобы сообщить Spring Boot, что должен быть запущен встроенный веб-сервер. Предоставляя пары (пары) имени и значения, разделенные двоеточием или равным, любая переменная среды может быть переопределена. В этом примере `"server.port:0"` переопределит настройку порта по умолчанию сервера. Обычно сервер будет использовать указанный номер порта, но значение 0 имеет особое значение. Когда указано как 0, он сообщает Spring Boot сканировать порты в среде хоста и запускать сервер на случайном доступном порту. Это полезно, если у нас есть разные службы, занимающие разные порты на машинах разработки и сервере сборки, которые потенциально могут столкнуться с портом приложения, и в этом случае приложение не запустится. Во-вторых, если мы создадим несколько тестов интеграции с различными контекстами приложений, они могут также столкнуться, если тесты выполняются одновременно.
5. У нас есть доступ к контексту приложения и вы можете использовать `autowiring` для ввода любого компонента Spring.
6. `@Value("${local.server.port}")` будет разрешен фактический номер порта, который используется.
7. Мы создаем некоторые объекты, которые мы можем использовать для проверки.
8. База данных MongoDB очищается и повторно инициализируется для каждого теста,

чтобы мы всегда проверяли достоверное состояние. Поскольку порядок тестов не определен, есть вероятность, что тест `testFetchAll ()` завершится неудачно, если он выполняется после теста `testDeletetripLeHHH ()`.

9. Мы [рекомендуем Rest Assured](#) использовать правильный порт. Это проект с открытым исходным кодом, который предоставляет Java DSL для тестирования остающихся услуг
10. Тесты выполняются с помощью Rest Assured. мы можем реализовать тесты с помощью `TestRestTemplate` или любого другого http-клиента, но я использую Rest Assured, потому что мы можем писать краткую документацию с помощью [RestDocs](#)

Загрузка другого файла `yml` [или свойств] или переопределение некоторых свойств

Когда мы используем `@SpringApplicationConfiguration` он будет использовать конфигурацию из `application.yml` [properties], которая в определенной ситуации не подходит. Поэтому для переопределения свойств мы можем использовать аннотацию `@TestPropertySource`.

```
@TestPropertySource(  
    properties = {  
        "spring.jpa.hibernate.ddl-auto=create-drop",  
        "liquibase.enabled=false"  
    }  
)  
@RunWith(SpringJUnit4ClassRunner.class)  
@SpringApplicationConfiguration(Application.class)  
public class ApplicationTest{  
  
    // ...  
  
}
```

Мы можем использовать атрибут **свойств** `@TestPropertySource` для переопределения конкретных **свойств**, которые мы хотим. В приведенном выше примере мы **переопределяем** свойство `spring.jpa.hibernate.ddl-auto` для `create-drop`. И `liquibase.enabled` to `false`.

Загрузка другого файла `yml`

Если вы хотите полностью загрузить другой файл `yml` для теста, вы можете использовать атрибут **местоположений** на `@TestPropertySource`.

```
@TestPropertySource(locations="classpath:test.yml")  
@RunWith(SpringJUnit4ClassRunner.class)  
@SpringApplicationConfiguration(Application.class)  
public class ApplicationTest{  
  
    // ...  
  
}
```

```
}
```

Альтернативно варианты

Опция 1:

Вы также можете загрузить другой файл **yaml**, поместив **yaml**- файл в `test > resource` КАТАЛОГ `test > resource`

Вариант 2:

Использование аннотации `@ActiveProfiles`

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = Application.class)
@ActiveProfiles("somename")
public class MyIntTest{
}
```

Вы можете видеть, что мы используем аннотацию `@ActiveProfiles` и мы передаем значение **somename** как значение.

Создайте файл под названием `application-somename.yaml` и тест загрузит этот файл.

Прочитайте Тестирование весной онлайн: <https://riptutorial.com/ru/spring-boot/topic/1985/тестирование-весной>

глава 19: Услуги REST

параметры

аннотирование	колонка
@Controller	Указывает, что аннотированный класс является «контроллером» (веб-контроллером).
@RequestMapping	Аннотации для сопоставления веб-запросов на определенные классы обработчиков (если мы использовали с классом) и / или методы обработчика (если мы использовали методы).
метод = RequestMethod.GET	Тип методов HTTP-запроса
ResponseBody	Аннотации, указывающие возвращаемое значение метода, должны быть привязаны к телу веб-ответа
@RestController	@Controller + ResponseBody
@ResponseBody	Расширение HttpEntity, которое добавляет код состояния HttpStatus, мы можем контролировать код HTTP возврата

Examples

Создание REST-сервиса

1. Создайте проект, используя STS (Spring Starter Project) или Spring Initializr (на [странице https://start.spring.io](https://start.spring.io)).
2. Добавьте веб-зависимость в свой pom.xml:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

или введите **веб** - в Search for dependencies окна поиска, добавить веб - зависимость и скачать архивный проект.

3. Создание класса домена (т.е. пользователя)

```
public class User {
```

```
private Long id;

private String userName;

private String password;

private String email;

private String firstName;

private String lastName;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getUsername() {
    return userName;
}

public void setUsername(String userName) {
    this.userName = userName;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

@Override
```

```

        public String toString() {
            return "User [id=" + id + ", userName=" + userName + ", password=" + password + ",
email=" + email
                + ", firstName=" + firstName + ", lastName=" + lastName + "]";
        }

        public User(Long id, String userName, String password, String email, String firstName,
String lastName) {
            super();
            this.id = id;
            this.userName = userName;
            this.password = password;
            this.email = email;
            this.firstName = firstName;
            this.lastName = lastName;
        }

        public User() {}
    }

```

4. Создайте класс UserController и добавьте аннотации @Controller , @RequestMapping

```

@Controller
@RequestMapping(value = "api")
public class UserController {
}

```

5. Определите статическую переменную List List для имитации базы данных и добавьте 2 пользователей в список

```

private static List<User> users = new ArrayList<User>();

public UserController() {
    User u1 = new User(1L, "shijazi", "password", "shijazi88@gmail.com", "Safwan",
"Hijazi");
    User u2 = new User(2L, "test", "password", "test@gmail.com", "test", "test");
    users.add(u1);
    users.add(u2);
}

```

6. Создайте новый метод для возврата всех пользователей в статический список (getAllUsers)

```

@RequestMapping(value = "users", method = RequestMethod.GET)
public @ResponseBody List<User> getAllUsers() {
    return users;
}

```

7. Запустите приложение [по mvn clean install spring-boot:run] и вызовите этот URL http://localhost:8080/api/users

8. Мы можем аннотировать класс с помощью @RestController , и в этом случае мы можем удалить.ResponseBody из всех методов этого класса, (@RestController = @Controller + ResponseBody) , еще одна точка, с которой мы можем управлять обратным кодом http,

если мы используем `ResponseEntity` , мы будем выполнять те же предыдущие функции, но используя `@RestController` И `ResponseEntity`

```
@RestController
@RequestMapping(value = "api2")
public class UserController2 {

    private static List<User> users = new ArrayList<User>();

    public UserController2() {
        User u1 = new User(1L, "shijazi", "password", "shijazi88@gmail.com", "Safwan",
"Hijazi");
        User u2 = new User(2L, "test", "password", "test@gmail.com", "test", "test");
        users.add(u1);
        users.add(u2);
    }

    @RequestMapping(value = "users", method = RequestMethod.GET)
    public ResponseEntity<?> getAllUsers() {
        try {
            return new ResponseEntity<>(users, HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

теперь попробуйте запустить приложение и вызвать этот URL <http://localhost:8080/api2/users>

Создание службы отдыха с помощью JERSEY и Spring Boot

Джерси - одна из многих рамок, доступных для создания служб отдыха. Этот пример покажет вам, как создавать службы отдыха с использованием Jersey и Spring Boot

1.Проект установки

Вы можете создать новый проект с использованием STS или с помощью страницы [Spring Initializr](#) . При создании проекта включите следующие зависимости:

1. Джерси (JAX-RS)
2. Web

2.Создание контроллера

Давайте создадим контроллер для веб-службы Jersey

```
@Path("/Welcome")
@Component
```

```
public class MyController {
    @GET
    public String welcomeUser(@QueryParam("user") String user){
        return "Welcome "+user;
    }
}
```

`@Path("/Welcome")` аннотация указывает на структуру, в которой этот контроллер должен отвечать на путь URI / приветствие

`@QueryParam("user")` указывает на фреймворк, который мы ожидаем от одного параметра запроса с именем `user`

3. Wiring Jersey Конфигурации

Давайте теперь настроим Jersey Framework с Spring Boot: создаем класс, а скорее весенний компонент, который расширяет `org.glassfish.jersey.server.ResourceConfig` :

```
@Component
@ApplicationPath("/MyRestService")
public class JerseyConfig extends ResourceConfig {
    /**
     * Register all the Controller classes in this method
     * to be available for jersey framework
     */
    public JerseyConfig() {
        register(MyController.class);
    }
}
```

`@ApplicationPath("/MyRestService")` указывает на фреймворк, в котором только запросы, направленные на путь `/MyRestService` , должны обрабатываться каркасом джерси, другие запросы должны по-прежнему обрабатываться весной.

Это хорошая идея, чтобы аннотировать класс конфигурации с `@ApplicationPath` , иначе все запросы будут обработаны Джерси, и мы не сможем обойти его, и пусть контроллер пружины обработает его, если потребуется.

4. Done

Запустите приложение и запустите пример URL-адреса (предположим, что вы настроили загрузку весны на порт 8080):

`http://localhost:8080/MyRestService/Welcome?user=User`

В вашем браузере должно появиться сообщение:

Приветственный пользователь

И вы закончили с вашим веб-сервисом Jersey с Spring Boot

Использование API REST с помощью RestTemplate (GET)

Чтобы использовать REST API с помощью `RestTemplate`, создайте проект загрузки Spring с начальным загрузчиком Spring и убедитесь, что добавлена зависимость **Web**:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

После **того**, как **вы создали свой проект**, создайте компонент `RestTemplate`. Вы можете сделать это в основном классе, который уже был сгенерирован, или в отдельном классе конфигурации (класс, аннотированный с помощью `@Configuration`):

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

После этого создайте класс домена, аналогичный тому, как вы должны делать при **создании службы REST**.

```
public class User {
    private Long id;
    private String username;
    private String firstname;
    private String lastname;

    public Long getId() {
        return id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }
}
```

```

    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
}

```

В вашем клиенте, `autowire RestTemplate` :

```

@Autowired
private RestTemplate restTemplate;

```

Чтобы использовать REST API, который возвращает одного пользователя, вы можете теперь использовать:

```

String url = "http://example.org/path/to/api";
User response = restTemplate.getForObject(url, User.class);

```

Используя API REST, который возвращает список или массив пользователей, у вас есть два варианта. Или употребляйте его как массив:

```

String url = "http://example.org/path/to/api";
User[] response = restTemplate.getForObject(url, User[].class);

```

Или используйте его с помощью `ParameterizedTypeReference` :

```

String url = "http://example.org/path/to/api";
ResponseEntity<List<User>> response = restTemplate.exchange(url, HttpMethod.GET, null, new
ParameterizedTypeReference<List<User>>() {});
List<User> data = response.getBody();

```

Помните, что при использовании `ParameterizedTypeReference` вам придется использовать более продвинутый `RestTemplate.exchange()` и вам придется создать его подкласс. В приведенном выше примере используется анонимный класс.

Прочитайте Услуги REST онлайн: <https://riptutorial.com/ru/spring-boot/topic/1920/услуги-rest>

глава 20: Установка Spring Boot CLI

Вступление

Spring CLI позволяет легко создавать и работать с приложениями Spring Boot из командной строки.

замечания

После установки CLI Spring Boot CLI можно запустить с помощью команды `spring` :

Чтобы получить справку из командной строки:

```
$ spring help
```

Чтобы создать и запустить свой первый проект Spring Boot:

```
$ spring init my-app
$ cd my-app
$ spring run my-app
```

Откройте ваш браузер на `localhost:8080` :

```
$ open http://localhost:8080
```

Вы получите страницу с ошибкой `whitelabel`, потому что вы еще не добавили каких-либо ресурсов в свое приложение, но вы готовы к использованию только следующих файлов:

```
my-app/
├─ mvnw
├─ mvnw.cmd
├─ pom.xml
├─ src/
│   └─ main/
│       ├── java/
│       │   └─ com/
│       │       └─ example/
│       │           └─ DemoApplication.java
│       └─ resources/
│           └─ application.properties
└─ test/
    └─ java/
        └─ com/
            └─ example/
                └─ DemoApplicationTests.java
```

- `mvnw` и `mvnw.cmd` - скрипты оболочки Maven, которые будут загружать и устанавливать Maven (при необходимости) при первом использовании.

- `pom.xml` - Определение проекта Maven
- `DemoApplication.java` - основной класс, запускающий приложение Spring Boot.
- `application.properties` - Файл для внешних свойств конфигурации. (Может также быть предоставлено расширение `.yaml`.)
- `DemoApplicationTests.java` - единичный тест, который проверяет инициализацию контекста приложения Spring Boot.

Examples

Ручная установка

См. [Страницу загрузки](#), чтобы вручную загрузить и распаковать последнюю версию, или перейдите по ссылкам ниже:

- [spring-boot-cli-1.5.1.RELEASE-bin.zip](#)
- [весна-загрузка-кли-1.5.1.RELEASE-bin.tar.gz](#)

Установите на Mac OSX с HomeBrew

```
$ brew tap pivotal/tap
$ brew install springboot
```

Установка на Mac OSX с помощью MacPorts

```
$ sudo port install spring-boot-cli
```

Установите на любую ОС SDKMAN!

SDKMAN! является программным обеспечением для разработчиков программного обеспечения для Java. Его можно использовать для установки и управления версиями Spring Boot CLI, а также Java, Maven, Gradle и других.

```
$ sdk install springboot
```

Прочитайте [Установка Spring Boot CLI онлайн](#): <https://riptutorial.com/ru/spring-boot/topic/9031/установка-spring-boot-cli>

кредиты

S. No	Главы	Contributors
1	Начало работы с весенней загрузкой	Andy Wilkinson , Brice Roncace , Community , GVArt , imdzeeshan , ipsi , kodiak , loki2302 , M. Deinum , Marvin Frommhold , Matthew Fontana , MeysaM , Misa Lazovic , Moshiour , Nikem , pinkpanther , rajadilipkolli , RamenChef , Ronnie Wang , Slava Semushin , Szobi , Tom
2	Spring boot + Hibernate + Web UI (Тимелеар)	ppeterka , rajadilipkolli , Tom
3	Spring Boot + JPA + REST	incomplete-co.de
4	Spring Boot + Spring Data Elasticsearch	sunkuet02
5	Spring Boot-Hibernate-REST Integration	Igor Bljahhin
6	Spring-Boot + JDBC	Moshe Arad
7	Spring-Boot Microservice с JPA	Matthew Fontana
8	ThreadPoolTaskExecutor: настройка и использование	Rosteach
9	Весенняя загрузка + JPA + mongoDB	Andy Wilkinson , Matthew Fontana , Rakesh , RubberDuck , Stephen Leppik
10	Весенняя загрузка + Spring Data JPA	dunni , Johir , naXa , Sanket , Sohlowmawn , sunkuet02
11	Контроллеры	Amit Gujarathi
12	Кэширование с помощью Redis Использование Spring Boot для MongoDB	rajadilipkolli
13	Подключение	koder23 , sunkuet02

	приложения Spring-Boot к MySQL	
14	Полно-отзывчивое веб-приложение Spring Boot с JHipster	anataliocs , codependent
15	Развертывание примера приложения с использованием Spring-загрузки на Amazon Elastic Beanstalk	Praveen Kumar K S
16	Сканирование пакетов	Tom
17	Создание и использование нескольких файлов application.properties	Patrick
18	Тестирование весной	Ali Dehghani , Aman Tuladhar , rajadilipkolli , Slava Semushin
19	Услуги REST	Andy Wilkinson , CAPS LOCK , g00glen00b , Johir , Kevin Wittek , Manish Kothari , odedia , Ronnie Wang , Safwan Hijazi , shyam , Soner
20	Установка Spring Boot CLI	Robert Thornton