Занятие 6

Экосистема Spring

Экосистема

Основное предназначение Spring - создание и управление жизненным циклом объектов.
Экосистема родилась как множество библиотек, в которых описаны компоненты на любой случай жизни

Экосистема

Spring Framework
Spring Boot
Spring Data
Spring Security
Spring Cloud
Spring Integration
Spring Batch
Spring HATEOAS
Spring Web Services
Spring AMQP
Spring Mobile
Spring WebFlow
Spring Session
Spring Vault
Spring Test

Spring JDBC
Spring MVC
Spring Reactor
Spring Kafka
Spring Redis
Spring WebMVC
Spring Cloud DataFlow
Spring Statemachine
Spring Rest Docs
Spring Auth. Server
Spring Flo
....

Spring Boot

Конфигурации и автоконфигурации

При разработке классического Spring приложения огромную роль приобретают конфигурации, управляющие жизненным циклом объектов.

В определенный момент стало понятно, что некоторые объекты создаются всегда (ну или почти всегда).

Spring Boot - это расширение Spring Framework, позволяющее автоматически создавать объекты, расположенные не только в конфигурации приложения, но и в подключенных библиотеках.

Добавляем зависимость - в приложении создаются и конфигурируются необходимые объекты (плюс добавляются транзитивные зависимости).

Зависимость должна быть оформлена специальным образом. Такая зависимость называется стартером (Spring Boot Starter)

@SpringBootApplication

Ключевая аннотация @SpringBootApplication. Логически является комбинацией следующих аннотаций

`@Configuration` то есть в данном классе можно конфигурировать объекты

`@EnableAutoConfiguration` запускает поиск конфигураций в зависимостях

`@ComponentScan` задает текущий пакет в качестве основного (родительского) для поиска компонентов

SpringApplication

`SpringApplication` - класс со статическим методом run(), который создает ApplicationContext и инициирует его жизненный цикл.

Таким образом, запуск Spring Boot приложения выглядит следующим образом

```
package io.ylab.app;

@SpringBootApplication
public class SpringSandboxApplication {

  public static void main(String[] args) {
    ApplicationContext applicationContext =
                        SpringApplication.run(SpringSandboxApplication.class, args);
  }
}
```

можно выбросить

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
  <version>2.7.6</version>
</dependency>
```

Все компоненты должны находится “внутри” пакета `io.ylab.app`

Настройки

Важная особенность Spring Boot Starter’a - конфигурирование работы с файлами настроек. Для использование настроек из файла необходимо сделать следующее

1. Создать файл `src/main/resources/application.properties (application.yml)`
2. Добавить настройки
3. Использовать с помощью аннотации `@Value`

```
…
password=qwerty
…
```



```
@Component
public class Comp {
  @Value("${password}")
  private String password;
}
```

Spring AMQP

Spring AMQP

Spring AMQP Starter - это набор компонентов для работы с RabbitMQ по протоколу AMQP. Помимо клиента для RabbitMQ предоставляет следующие возможности

- RabbitTemplate - объект, используемый для отправки сообщений
- Процессор @RabbitListener для получения сообщений
- Queue, Exchange, Binding могут быть зарегистрированы как Spring Beans
- Подключение к RabbitMQ можно настраивать с помощью свойств из src/main/resources/application.properties (или любого другого источника настроек)

  Appendix A. Common application properties

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
  <version>2.6.13</version>
</dependency>
```

Spring AMQP

Отправка

```
@Component
public class MessageSender {
  @Autowired
  private RabbitTemplate rabbitTemplate;

  public void send(String m) {
    System.out.println("Sent >>> " + m);
    rabbitTemplate.convertAndSend("test_q", m);
  }
}
```

Получение

```
@Component
public class MessageProcessor {

  @RabbitListener(queues = "test_q")
  public void process(String string) {
    System.out.println("Received <<< " + string);
  }
}
```

Создание очереди

```
@Bean
public Queue testQ() {
  return new Queue("test_q", true);
}
```

Spring JDBC

Spring JDBC

Spring JDBC Starter - это набор компонентов для работы с БД средствами JDBC. Является “надстройкой” над JDBC, позволяет автоматизировать некоторые рутинные вещи

Pure JDBC -> **Spring JDBC** -> Spring Data JDBC -> Spring Data JPA

1. Настройка DataSource с помощью файла конфигурации
2. JdbcTemplate
3. RowMapper

   …

Appendix A. Common application properties

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
  <version>2.6.13</version>
</dependency>
```

Spring JDBC. Выборка

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

```
@Component
public class PersonDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;

  public List<Person> findAll() {
    RowMapper<Person> rowMapper = (resultSet, rowNum) -> {
      Person person = new Person();
      person.setFirstName(resultSet.getString("first_name"));
      person.setMiddleName(resultSet.getString("middle_name"));
      person.setLastName(resultSet.getString("last_name"));
      return person;
    };
    List<Person> res = jdbcTemplate
                          .query("select * from person", rowMapper);
    return res;
  }
}
```

Spring JDBC. Вставка

```
@Component
public class PersonDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;

  public void addPerson(Person person) {
    PreparedStatementSetter preparedStatementSetter = ps -> {
      ps.setString(1, person.getFirstName());
      ps.setString(2, person.getLastName());
      ps.setString(3, person.getMiddleName());
    };
    jdbcTemplate.update("insert into person (first_name, last_name, middle_name)
"
                              + "values (?,?,?)",  preparedStatementSetter);
  }

}
```

HTTP

HTTP

HTTP - протокол транспортного уровня. Раньше использовался для передачи HTML, но в настоящее время используется для передачи любых данных.

HTTP запрос можно воспринимать как обычный текстовый формат, имеющий определенную структуру

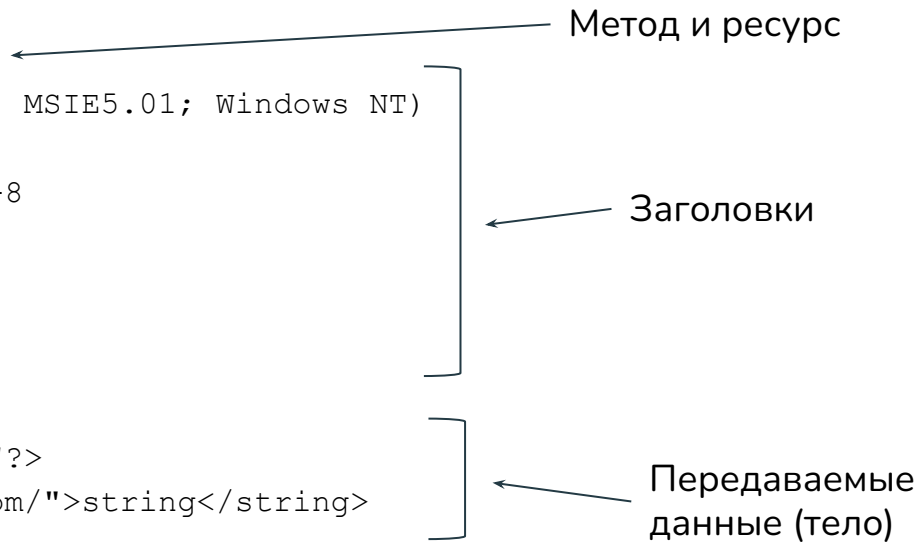
В данный момент протокол HTTP является самым популярным протоколом синхронного взаимодействия между приложениями

HTTPS - только зашифрованное

HTTP Запрос

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```



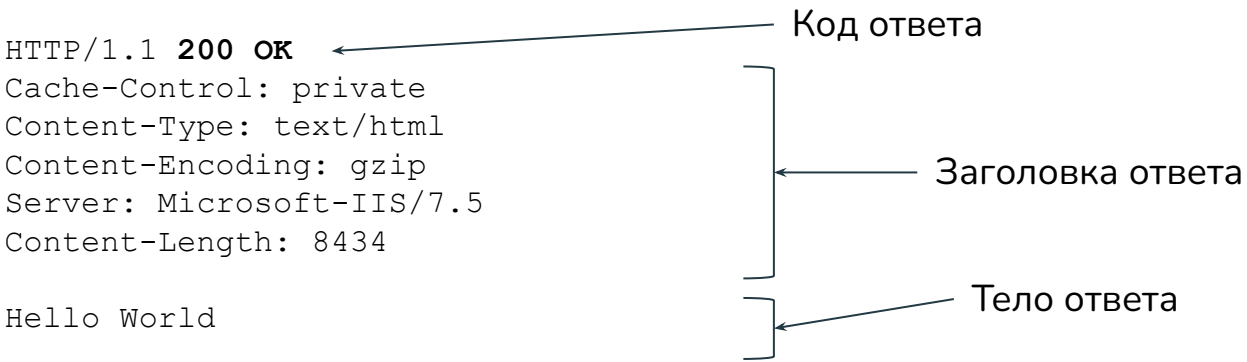
HTTP Методы

- **GET**. Обычно используется для получения данных. Является идемпотентным (многократный вызов имеет тот же эффект, что одиночный вызов). Не имеет тела
- **POST**. Обычно используется для создания новой сущности. Может содержать тело, не является идемпотентным (многократный вызов меняет состояние, создавая новые сущности при каждом вызове)
- **PUT**. Используется для обновления сущности. Имеет тело, является идемпотентным
- **PATCH**. Может использоваться вместе с PUT для частичного обновления сущности. А может вообще отсутствовать.
- **DELETE**. Удаляет ресурс. Не имеет тела, идемпотентный
- ....

Методы HTTP запроса

HTTP Ответ

Ответ на HTTP запрос также является простым текстом, имеющим определенную структуру.



HTTP Коды ответов

Код ответа - это число, сообщающее вызывающей стороне общий статус обработки запроса. Самые распространенные коды ответов:

- **2xx**. 200, 201, 202... Возвращаются, когда запрос успешно обработан
- **4xx**. Возвращается, когда запрос сформирован некорректно. 401 - когда нет данных авторизации для выполнения запроса, 404 - когда не найден запрашиваемый ресурс, 451 - заблокировано по требованию органов власти и т.д. 418 - I'm a teapot, шуточный статус, добавленный 1 апреля 1998 года.
- **5xx**. Возвращается, когда произошла ошибка сервера при обработке запроса. 500 - общая ошибка запроса, 502 - некорректный прокси
- **3xx**. Возвращается, когда запрошенный ресурс был перемещен в другое место.

Некоторые коды могут требовать заполнения определенных заголовков

Статус 301 (Moved Permanently) требует заголовка Location в ответе

HTTP response status codes

Как из всего этого собрать приложение?

1. Средство генерирования запросов. Обычно - фронтенд и/или Postman
2. HTTP сервер. Компонент, который будет принимать запросы от клиентов и отдавать ответы
3. Средство, которое позволит при получении запроса выполнять некоторый код, выполняющий бизнес-логику и формирующий ответ для клиента

Spring MVC

Spring MVC и Spring Web Starter

Spring MVC - это только часть, позволяющая связывать HTTP запросы с кодом. Для того, чтобы приложение могло принимать запросы, необходим так называемый сервлет-контейнер. Раньше эту роль выполнял Tomcat, запущенный на сервере.

С появлением Spring Boot Tomcat может быть частью приложения!

То есть приложение при запуске запускает HTTP сервер и публикует свой код в него

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.7.8</version>
</dependency>
```

Controller

Controller - термин, пришедший из MVC. В контексте Spring обозначает класс, ответственный за прием HTTP запросов

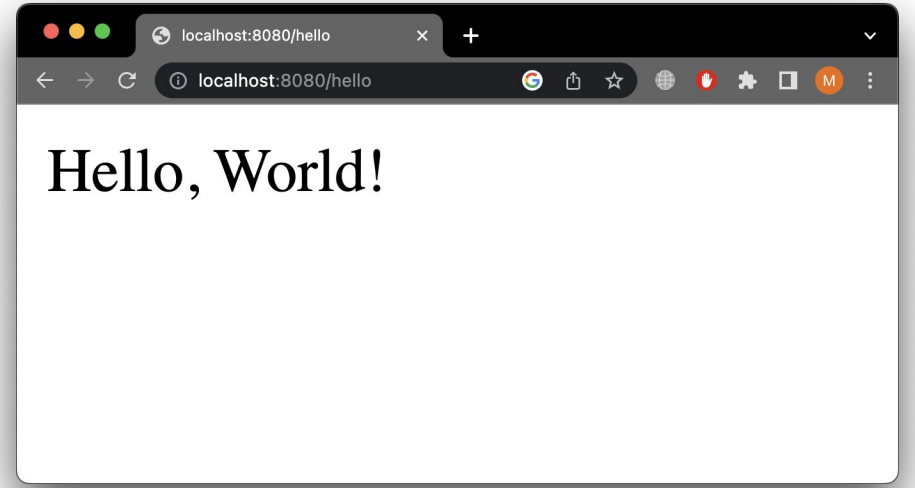
```
@RestController
public class Controller {
  @RequestMapping(
      method = RequestMethod.GET,
      path = "/hello"
  )
  public ResponseEntity<String> hello() {
    return ResponseEntity
              .status(200)
              .body("Hello, World!");
  }
}
```

Аннотация регистрирует не просто бин, а обработчик HTTP запросов

Метод будет обрабатывать запросы метода GET по пути /hello

Формирует ответ, который будет передан вызывающей стороне

Controller



Live Coding Section

Продемонстрировать, как можно передавать данные в методы контроллера используя аннотации

```
@RequestBody
@PathParam
@RequestParam

@ResponseBody
```

Open API Starter

HTTP методы бывают разные. Могут быть разные коды ответов, разная логика и прочее.
Open API - это стандарт документирования HTTP методов. Идея в том, чтобы описать методы в стандартном виде, доступным для автоматизированной обработки.
Модуль Open API для Spring Boot предоставляет возможность визуализации и тестирования HTTP эндпойнтов.

Endpoint - это название для пути, по которому зарегистрирован обработчик. Например /hello из предыдущего примера - это endpoint

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.9</version>
</dependency>
```

http://localhost:8080/swagger-ui/index.html

Юнит тесты

Юнит тесты

Основная идея юнит-тестов - проверка логики работы компонентов. Логически делится на следующие этапы

1. Подготовка данных
2. Запуск бизнес-логики на основе подготовленных данных
3. Сравнение фактического результата и ожидаемого. Если результаты совпадают - юнит тест считается успешным. Если не совпадают - проваленным

Проблемы:

- Подготовка данных. Если данные получаются из внешних источников - это может быть проблемой
- Запуск бизнес-логики. Если бизнес-логика “размазана” по нескольким компонентам, которые между собой соединены при помощи Spring

JUnit

JUnit - одно из самых популярных решений (де-факто - стандарт)

src/test/java

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
```

```
@Test
public void testFibonacciNumber() {
  int n = 10;
  int expected = 34;
  int actual = new FibonacciCounter().get(n);
  Assert.assertEquals("Ошибка вычисления", expected, actual);
}
```

```
java.lang.AssertionError: Ошибка вычисления
Expected :35
Actual   :34
<Click to see difference>
```

Spring Boot JUnit

```
@SpringBootTest(classes = FibonacciCounter.class)
public class FibonacciSpringTest {

  @Autowired
  private FibonacciCounter fibonacciCounter;

  @Test
  public void testFibonacciNumber() {
    int n = 10;
    int expected = 34;
    int actual = fibonacciCounter.get(n);
    Assertions.assertThat(actual).isEqualTo(expected)
        .describedAs("Ошибка вычисления");
  }
}
```

https://www.baeldung.com/spring-boot-testing

Эпилог

Полезные советы. Java

- Пишите юнит тесты. Почитайте про TDD. Не злоупотребляйте Testcontainer’ами
- Используйте анализатор кода: Checkstyle, SonarLint
- Не пишите длинные классы/методы. Идеальная длина методов - не больше 50 строк
- Обрабатывайте исключения. Выводить в лог - только если оно не будет брошено дальше
- Старайтесь не злоупотреблять вложенностью (try/catch, циклы, if-else)
- Старайтесь не злоуопотреблять Stream API
- SOLID! Особенно S
- Учите новые возможности LTS релизов языка. 8, 11, 17
- Постарайтесь разобраться с многопоточностью
- Используйте существующие библиотеки для рутинных операций: guava, *-commons

Полезные советы. Технологии и фреймворки

- MyBatis
- MapStruct
- Lombok
- Системы логирования: log4j, slf4j, logback
- Альтернативные системы обмена сообщений: Kafka, ActiveMQ
- Поинтересуйтесь, что такое BPM и какие BPM движки используются (jBPM, Camunda)
- Ознакомьтесь с основными концепциями Spring Security
- Keycloak, JWT, OAuth, OpenID Connect
- **Maven**: BOM, Parent/effective pom, dependencyManagement
- Изучите, какие бывают Maven* Plugin’ы, что они могут делать
- **Docker**. Что это такое, как упаковать приложение в образ
- Что умеет Spring Boot Actuator
- Liquibase/Flyway. Старайтесь использовать во всех проектах
- Разберитесь как писать свои Spring Boot Starter’ы😅

Полезные советы

- Прочитайте “Чистую архитектуру” и “Чистый код”
- При изучении новой технологии - старайтесь нырнуть как можно глубже. Больше экспериментов!
- Изучите основные методологии и технологии, используемые в работе: Git Flow, Agile, GitLab/GitHub/BitBucket, Jira, Confluence, Trello, Figma
- Освойте средство рисования диаграмм: Diagrams.net (Draw.io)
- УЧИТЕ АНГЛИЙСКИЙ!!!
- Работайте на софт скиллами. Чудаков на букву М никто не любит
- Работайте ответственно, будьте любознательными, принимайте критику, делайте выводы
- Собирайте документацию и литературу

Проекты для портфолио

- Сервис для сокращения ссылок: БД для хранения, API для управления. Все сделано на Spring, покрыто тестами, т.д.
- Телеграмм бот
- ...

Спасибо за то, что были с нами!

TG/IG: @gurin_md
Email: gurin.md.pro@gmail.com