Бины и область видимости

- Конфигурация
  - Создание бинов на основе классов
  - Создание бинов на основе методов
- Жизненный цикл бинов
- Область видимости бинов

В Spring бином называется объект, который управляется, создается и настраивается Spring-контейнером. Эти объекты создаются на базе конфигурации, которая задается с помощью аннотаций. Только эти объекты участвуют в инъекции зависимостей при сборке Spring-приложения.

Конфигурация

Далее мы рассмотрим два способа создания бинов.

Создание бинов на основе классов

Если мы просто создадим класс и попытаемся внедрить его объект с помощью аннотации `@Autowired`, то ничего не получится. Spring никак не реагирует на обычные классы. Чтобы превратить этот класс в бин, нужно пометить его аннотацией, например:

- `@Component` — любой класс общего назначения, объект которого мы хотим получить в приложении
- `@Repository` — репозитории
- `@RestController`— контроллеры

Изучим пример с репозиторием:

```
package io.hexlet.spring.repository;

import hexlet.code.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
@Repository
// Во время компиляции этот интерфейс превращается в конкретный класс
public interface UserRepository extends JpaRepository<User, Long> {
}
```

Внедрение происходит так:

```
@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserRepository userRepository;
}
```

Создание бинов на основе методов

В реальных проектах внедряться могут не только объекты классов, реализованные программистом. Еще можно внедрять объекты классов, которые находятся в библиотеках. Например, есть известная библиотека datafaker, которая используется в тестах для генерации данных. Работает она так:

```
import net.datafaker.Faker;

var faker = new Faker();

var name = faker.name().fullName(); // Miss Samanta Schmidt
var firstName = faker.name().firstName(); // Emory
var lastName = faker.name().lastName(); // Barton

var streetAddress = faker.address().streetAddress(); // 60018 Sawayn Brooks Suite 449
```

Существует два основных способа использования этой библиотеки внутри Spring Boot. Первый – это создание объекта напрямую в том месте, где мы хотим его использовать. В примере с Faker мы будем создавать и использовать объект внутри теста:

```
@SpringBootTest
@AutoConfigureMockMvc
public class UsersControllerTest {
```

```
    @BeforeEach
    public void setUp() {
      var faker = new Faker();
      // Тут создаем нужные данные
    }
```

Второй – это создание бина с помощью метода. Для этого нам нужно создать метод внутри любого класса, помеченного аннотацией `@Configuration`. Проще всего это сделать в классе с методом `main`, потому что аннотация `@SpringBootApplication` автоматически добавляет аннотацию `@Configuration`:

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

    @Bean
    public Faker getFaker() { // Имя метода не важно
        return new Faker();
    }
}
```

Теперь Faker можно внедрять как обычную зависимость:

```
@SpringBootTest
@AutoConfigureMockMvc
public class UsersControllerTest {

    @Autowired
    private Faker faker;

    @BeforeEach
    public void setUp() {
      // Тут создаем нужные данные
    }
```

Жизненный цикл бинов

У бинов есть понятие жизненного цикла, что позволяет встраиваться в процесс их создания и уничтожения. Делается это с помощью аннотаций методов `@PostConstruct` и `@PreDestroy` внутри класса нужного бина:

```
import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Component;

@Component
public class MyBean {

    private String message;

    @PostConstruct
    public void init() {
        this.message = "Bean is initialized!";
        System.out.println(message);
    }

    @PreDestroy
    public void cleanup() {
        System.out.println("Cleaning up resources or performing final actions!");
    }

    // ... other methods ...
}
```

Типичные ситуации, когда это бывает нужно:

- Чтение конфигурации и инициализация некоторых свойств
- Установка ресурсов, таких как соединение с базой данных
- Регистрация бинов во внешних системах

Область видимости бинов

Область видимости бинов определяет жизненный цикл и саму видимость бинов внутри контекста приложения. Другими словами, она определяет, сколько объектов создается и как они переиспользуются разными частями приложения. Всего существует шесть областей видимости.

По умолчанию используется область *Singleton*. Бины с такой областью создаются ровно один раз за все время существования приложения. Каждая инъекция такого бина использует один и тот же объект.

Область *Prototype* означает, что новый бин будет создан на каждый запрос (инъекцию):

```
import org.springframework.beans.factory.config.ConfigurableBeanFactory;
import org.springframework.stereotype.Component;

@Scope("prototype")
@Component
public class PrototypeBean {}
```

Область Request означает, что новый бин создается на каждый HTTP-запрос. Актуально только для веб-приложений:

```
import org.springframework.stereotype.Component;
import org.springframework.web.context.annotation.RequestScope;

@RequestScope
@Component
public class RequestScopedBean {}
```

Информацию по остальным областям видимости можно прочитать в официальной документации.

Далее →