Инверсия зависимостей

- Инъекции
  - Инъекция через конструктор
  - Инъекция с помощью аннотации @Autowired

Spring Boot — это проект, который построен поверх Spring Framework — платформы для создания корпоративных приложений, куда входят не только веб-сервисы. У Spring Framework много возможностей, но ядро во всем этом многообразии — это контейнер зависимостей.

В этом уроке мы изучим, как Spring собирает приложение воедино с помощью инверсии зависимостей.

Допустим, у нас есть репозиторий `UserRepository` для интеграции модели `User` с базой данных. В простейшем случае репозиторий выглядит так:

```
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> {
}
```

Теперь представим, что на основе этого интерфейса Spring генерирует класс с именем `UserRepositoryImpl`. Как использовать этот класс в своем коде — например, в методе создания пользователя?

```
@PostMapping("/users")
@ResponseStatus(HttpStatus.CREATED)
public User create(@RequestBody User user) {
    // Как использовать репозиторий?
}
```

Если работать в лоб, то получится такой код:

```
var repository = new UserRepositoryImpl();
repository.save(user);
```

Этот код не сработает, потому что у репозитория нет доступа к соединению с базой данных — он не может сохранить данные. Правильный код должен выглядеть как-то так:

```
// Соединяемся с базой данных
var conn = DriverManager.getConnection(/* параметры соединения */);
var repository = new UserRepositoryImpl(conn); // Передаем соединение в репозиторий
```

Придется дублировать этот код везде, потому что репозиторий нужен буквально в каждом контроллере. Но даже если мы вынесем его в метод, мы все равно наткнемся на проблему создания объектов на каждый запрос. С репозиторием это допустимо, но с базой данных соединяться на каждый запрос нельзя — это создаст большую нагрузку на систему.

Пример выше показывает, как возникают зависимости в коде. Так происходит, когда одни классы используют объекты других классов. Причем здесь мы рассмотрели учебный пример с всего одной зависимостью.

В реальных проектах зависимостей может быть значительно больше. Кроме того, они могут быть гораздо глубже — тогда класс может зависеть от класса, который зависит от другого класса, а тот в свою очередь еще от одного класса. Посмотрите, во что превратился бы код:

```
var obj1 = new OneClass();
var obj2 = new TwoClass(obj1);
var obj3 = new ThreeClass();
var obj4 = new FourClass(obj2, obj3);
```

Spring Framework решает эту задачу автоматически с помощью инверсии зависимостей. Программист указывает, как одни классы зависят от других, а фреймворк сам собирает нужные объекты и передает их в код для использования. В таких ситуациях Spring:

- Находит все классы, помеченные его аннотациями
- Вычисляет, как классы зависят друг от друга
- Собирает нужные объекты
- Подставляет их при инициализации объектов этих классов

Эта схема работает только с классами, объекты которых Spring создает сам. Иначе ничего подставить не получится. Например, модели не попадают под эту схему:

```
// Мы работаем с классом напрямую
// Поэтому подстановка зависимостей не сработает
var user = new User();
```

То же самое относится к любым другим классам, объекты которых мы создаем самостоятельно.

Инъекции

Подстановка зависимых объектов в другие объекты называется **инъекцией**. Ниже мы рассмотрим, как это делается в Spring Boot.

Инъекция через конструктор

Классический способ выполнить подстановку зависимостей — это инъекция через конструктор. Изучим пример с контроллером:

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

    private UserRepository userRepository;

    public UserController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    // Где-то здесь в методах используем userRepository
}
```

Spring анализирует конструкторы своих компонентов. Если он видит там указанные зависимости, то использует эти конструкторы для создания объектов и внедрения зависимостей.

Инъекция с помощью аннотации @Autowired

Более простой и широко используемый способ инъекции — это аннотация `@Autowired`:

```
import org.springframework.beans.factory.annotation.Autowired;

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

    @Autowired
    private UserRepository userRepository;
}
```

Это самый короткий и удобный способ указывать зависимости. В большинстве случаев этого подхода достаточно.

Далее →