Spring Framework: Подробный обзор

1. Core: IoC и Dependency Injection

1.1. Inversion of Control (IoC)

IoC — принцип, при котором управление объектами передается контейнеру Spring. Вместо явного создания объектов (`new MyService()`) Spring управляет их жизненным циклом.

Как работает IoC?

1. **ApplicationContext** — центральный интерфейс Spring IoC.
2. **Bean Definition** — метаданные о классе (например, `@Component`, `@Service`).
3. **Bean Factory** — создает и управляет бинами.

1.2. Dependency Injection (DI)

DI — механизм внедрения зависимостей (через конструктор, сеттеры или поля).

Способы DI:

1. **Через конструктор (рекомендуется)**:

```
@Service
public class UserService {
    private final UserRepository userRepository;

    @Autowired // Опционально (Spring 4.3+ автоматически внедряет)
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}
```

2. **Через сеттеры**:

```
@Service
public class UserService {
    private UserRepository userRepository;

    @Autowired
    public void setUserRepository(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}
```

3. **Через поле (не рекомендуется для тестирования)**:

```
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;
}
```

Аннотации для DI:

- `@Autowired` — автоматическое связывание.
- `@Qualifier` — уточнение бина (если несколько реализаций).
- `@Primary` — приоритетная реализация.
- `@Resource` (JSR-250) — аналог `@Autowired` от Java EE.

---

2. Spring Boot

2.1. Автоконфигурация

Spring Boot автоматически настраивает приложение на основе зависимостей (`spring-boot-starter-web`, `spring-boot-starter-data-jpa` и др.).

Как работает:

1. **`@EnableAutoConfiguration`** — включает автоконфигурацию.
2. **`META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports`** — список автоматически подключаемых конфигураций.

2.2. `@SpringBootApplication`

Комбинированная аннотация, включающая:

1. **`@Configuration`** — класс с конфигурацией бинов.
2. **`@EnableAutoConfiguration`** — автоконфигурация.
3. **`@ComponentScan`** — сканирование компонентов в текущем пакете и подпакетах.

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

Основные фичи Spring Boot:

- **Встроенный сервер** (Tomcat, Jetty, Undertow).
- **`application.properties`/`application.yml`** — конфигурация.
- **Actuator** (`/health`, `/metrics`, `/beans`).
- **DevTools** — горячая перезагрузка.

---

3. Spring MVC

3.1. `@Controller` vs `@RestController`

Аннотация	Описание
`@Controller`	Возвращает имя представления (View), используется с Thymeleaf, JSP.
`@RestController`	Возвращает JSON/XML (по умолчанию), аналог `@Controller` + `@ResponseBody`.

3.2. `@RequestMapping` и производные

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

    @GetMapping("/{id}") // GET /api/users/1
    public User getUser(@PathVariable Long id) {
        return userService.findById(id);
    }

    @PostMapping // POST /api/users
    public User createUser(@RequestBody User user) {
        return userService.save(user);
    }

    @PutMapping("/{id}") // PUT /api/users/1
    public User updateUser(@PathVariable Long id, @RequestBody User user) {
        return userService.update(id, user);
    }

    @DeleteMapping("/{id}") // DELETE /api/users/1
    public void deleteUser(@PathVariable Long id) {
        userService.delete(id);
    }
}
```

Основные аннотации:

- `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping` — HTTP-методы.
- `@PathVariable` — извлечение переменных из URL (`/users/{id}`).
- `@RequestParam` — параметры запроса (`/users?name=John`).
- `@RequestBody` — десериализация JSON/XML в объект.
- `@ResponseBody` — сериализация объекта в JSON/XML.

---

4. Spring Data JPA

4.1. `@Repository`

Аннотация для DAO-слоя. Автоматически преобразует исключения в `DataAccessException`.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // Автоматически реализует CRUD
}
```

4.2. `JpaRepository`

Интерфейс, предоставляющий:

- **CRUD-операции** (`save()`, `findById()`, `delete()`).
- **Пагинацию** (`Pageable`).
- **Методы по соглашению** (Query Methods):

```
    List<User> findByName(String name);
    List<User> findByAgeGreaterThan(int age);
```

Пример использования:

```
@Service
public class UserService {
    private final UserRepository userRepository;

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

    public User findById(Long id) {
        return userRepository.findById(id).orElseThrow();
    }
}
```

Кастомные запросы через `@Query`:

```
@Query("SELECT u FROM User u WHERE u.email LIKE %:email%")
List<User> findByEmailContaining(@Param("email") String email);
```

---

5. Spring Security

5.1. Аутентификация

Основные механизмы:

1. **In-Memory Auth**:

```
    @Bean
    public UserDetailsService users() {
        UserDetails user = User.builder()
            .username("user")
            .password("{bcrypt}$2a$10$...")
            .roles("USER")
            .build();
        return new InMemoryUserDetailsManager(user);
    }
```

2. **Database Auth**:

```
@Service
public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) {
        User user = userRepository.findByUsername(username);
        return new org.springframework.security.core.userdetails.User(
            user.getUsername(),
            user.getPassword(),
            getAuthorities(user)
        );
    }
}
```

5.2. Авторизация

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .requestMatchers("/user/**").hasAnyRole("USER", "ADMIN")
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login")
                .permitAll()
            );
        return http.build();
    }
}
```

5.3. JWT (JSON Web Token)

3. **Добавляем зависимость**:

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

4. **Создаем фильтр**:

```
public class JwtTokenFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain) {
        String token = extractToken(request);
        if (token != null && jwtProvider.validateToken(token)) {
            Authentication auth = jwtProvider.getAuthentication(token);
            SecurityContextHolder.getContext().setAuthentication(auth);
```

```
            }
            chain.doFilter(request, response);
        }
    }
```

---

6. Spring Cloud (Микросервисы)

6.1. Service Discovery (Eureka)

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

Конфигурация клиента (`application.yml`):

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

6.2. API Gateway (Zuul)

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

Маршрутизация (`application.yml`):

```
zuul:
  routes:
    users:
      path: /users/**
      serviceId: user-service
```

6.3. Feign (REST-клиент)

```
@FeignClient(name = "user-service")
public interface UserClient {
    @GetMapping("/users/{id}")
    User getUser(@PathVariable Long id);
}
```

Итог

- **IoC/DI** — основа Spring (управление зависимостями).

- **Spring Boot** — быстрый старт + автоконфигурация.
- **Spring MVC** — REST API (`@RestController`, `@GetMapping`).
- **Spring Data JPA** — работа с БД (`JpaRepository`).
- **Spring Security** — аутентификация (JWT) и авторизация.
- **Spring Cloud** — микросервисы (Eureka, Zuul, Feign).