Kotlin a spring framework/spring boot

Specifika/výhody/nevýhody použití kotlinu se spring frameworkem

Petr Balat, @petr_balat

Abstract

1. Krátce o frameworku.

2. Podpora kotlinu (jsr305, nepovinné parametry, default metody).

3. Speciality jako JPA/hibernate, spring rest a jdbc template

4. Unit testy a kotlin dsl.

5. Podpora kotlin coroutines a async přístupu

5. srovnání s micro frameworkem ktor/spring fu (pokud zbyde čas)

Spring Framework

https://spring.io/projects/spring-framework

“Lepidlo” pro enterprise aplikace. Snaží se řešit potřeby od dependency injection, přes mvc framework, přístup k databázi, auth, cache až po integraci J2EE(JMS/JCA).
Od verze 5.0 (28.září 2017) je oficiální podpora **Kotlin**u. Framework je ale napsán v javě.

Spring Boot

https://spring.io/projects/spring-boot

Zapouzdřuje a integruje spring.io projekty (jádro spring framework) a externí java knihovny/frameworky do stand-alone řešení (jar/war).
Embedded web server, jednoduchá konfigurace. Výborné pro mikroslužby.
Začít s vy-generováním projektu http://start.spring.io/ - na konci si ukážeme jak.

Specifika

- Spring f. instance komponent vytváří pomocí CGLIB proxy => použít open. Vyřešeno maven/gradle pluginem kotlin-allopen.
- Nad metodami s default parametry volané např. z html šablon (thymeleaf, freemarker) použít anotace @JvmOverloads

```
<a th:href="${filtr.urlPage(pageItems.getNumber())}" alt="show">Hello</a>
<a th:href="${filtr.urlPage(pageItems.getNumber(), null, lists)}" alt="show">Hello</a>
```

```
@JvmOverloads
fun urlPage(page: Int, ignoreValues: Map<String, Any?>? = null, ignoreProperties: Iterable<String>? = null): String {
    val url: String = urlPage(page
            , obj: this
            , ignoreProperties: IGNORE_PROPERTIES + (ignoreProperties ?: emptyList())
            , ignoreValues.orEmpty()
    ).substring( startIndex: 1)
    return "/shop$url"
}
```

Nevýhoda

- Default metody v interface - Řešením je použít extension fun ale pozor, že poté nezafunguje např. anotace cache! nebo jí napsat v javě s anotacemi @NotNull a @Nullable

```
@Repository
interface Test4ShowRepository : JpaRepository<Test4Show, Int> {

    @Cacheable( ...value: "findVisible")
    @Query( value: """select t
from Test4Show t
where t.visible = true""")
    fun findVisible(): Int
}

//zde anotace @Cacheable nefunguje!
fun Test4ShowRepository.findByXXX(limit: Int = 100): List<Test4Show> =
        this.findAll().filter { complicatedMetod() }.take(limit)
```

Features oproti javě

- **Celý spring framework/boot obsahuje anotace @NotNull a @Nullable, takže je podporován kotlin null-safety viz jsr305**
- Z reflexe umí číst např. povinnost request parametru

```kotlin
@RestController
@RequestMapping( ...value: "/api/test")
class Test4ShowApiController(private val repository: Test4ShowRepository,
                             private val service: ConfigService?) {

    @GetMapping
    fun get(@RequestParam() search: String?): List<Test4Show> = if (search.isNullOrBlank()) {
        repository.findAll()
    } else repository.findByText(search)

    fun config(): Config? = service?.config
}
```

```kotlin
compileKotlin {
    kotlinOptions.jvmTarget = "1.8"
    kotlinOptions.freeCompilerArgs = ["-Xjsr305=strict"]
}
compileTestKotlin {
    kotlinOptions.jvmTarget = "1.8"
    kotlinOptions.freeCompilerArgs = ["-Xjsr305=strict"]
}
```

```java
@RestController
@RequestMapping("/api/test")
public class JavaTest4ShowApiController {

    private final Test4ShowRepository repository;
    private final ConfigService service;

    public JavaTest4ShowApiController(Test4ShowRepository repository,
                                      @Autowired(required = false) ConfigService service) {
        this.repository = repository;
        this.service = service;
    }

    @GetMapping
    public List<Test4Show> get(@RequestParam(required = false) String search) {
        if (search == null || kotlin.text.StringsKt.isBlank(search)) {
            return repository.findAll();
        }
        return repository.findByText(search);
    }

    public Config config() {
        if (service == null) {
            return null;
        }

        return service.getConfig();
    }
}
```

Features oproti javě

- nově od verze 5 možnost dsl registrace Bean/component (v bootu se nevyužije - použito ve fu)
- extension metody pro RestTemplate a WebApi
- a další např. https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0

```
beans {
    bean<Foo>()
    bean { Bar(ref()) }
}
```

```
GenericApplicationContext context = new GenericApplicationContext();
context.registerBean(Foo.class);
context.registerBean(Bar.class, () -> new
        Bar(context.getBean(Foo.class))
);
```

JPA/Hibernate/jdbc template

- Lze použít pro JPA entity data class - kotlin-jpa plugin. **Po uložení pomocí reflexe změní hodnoty fieldů, takže není imutabilní!**
- Plugin vytvoří contructor bez parametru.
- Pro jdbc/rest template má spring několik extension metod kde využívá kotlin feature: reified inline metody

```kotlin
@Entity
@Table(name = "STORES")
@EntityListeners(value = [AuditingEntityListener::class])
data class Store(

        @Column(name = "KOD")
        val kod: String,

        @Column(name = "FIRM_ID")
        val firmId: Int,

        @Column(name = "NAME")
        val name: String? = null,

        @Column(name = "DELIVERY_DAYS")
        val deliveryDays: Int = 5,
```

```
    classpath("org.jetbrains.kotlin:kotlin-noarg:$kotlinVersion")
  }
}
apply plugin: 'kotlin-jpa'
```

```
noArg {
    invokeInitializers = true
}
```

```kotlin
fun findByFirstName(firstName: String) {
    jdbcTemplate.query("""select id, name from table
        WHERE first_name = :firstName""".trimMargin(), mapOf("firstName" to firstName)) {
        println(it.getString( columnLabel: "name"))
    }
}
```

Testy

- Spring má výbornou podporu pro unit a integrační testy.
- Nemá ale přímou podporu pro Kotlin.
- Napsal jsem si dsl pro spring mvc testy https://github.com/petrbalat/kd4smt

```java
@SpringJUnitWebConfig(locations = "test-servlet-context.xml")
class ExampleTests {

    private MockMvc mockMvc;

    @BeforeEach
    void setup(WebApplicationContext wac) {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(wac).build();
    }

    @Test
    void getAccount() throws Exception {
        this.mockMvc.perform(get("/accounts/1")
                .accept(MediaType.parseMediaType("application/json;charset=UTF-8")))
            .andExpect(status().isOk())
            .andExpect(content().contentType("application/json"))
            .andExpect(jsonPath("$.name").value("Lee"));
    }
}
```

```kotlin
class ExampleTest : AbstractIntegrationTest() {

    @Autowired
    lateinit var mockMvc: MockMvc

    fun testHome() = performGet( url: "/accounts/1") { this: ResultActions
        expectStatus { isOk }
        expectContent { contentTypeCompatibleWith(MediaType.TEXT_HTML) }
        expectJsonPath( expression: "$.name") { this.value( expectedValue: "Lee")}
    }
}
```

Async/Coroutine/Flux

- Od verze spring. f. 5 resp. boot 2 lze psát aplikace asynchronně (netty, reactor)
- Nutnost async ovladačů k externím zdrojům, nefunguje tedy s jdbc. Podpora pro mongo, redis, cassandra
- Podpora ze strany jetbrains s projektem kotlinx-coroutines-reactor a https://github.com/konrad-kaminski/spring-kotlin-coroutine

```
@PutMapping("/tweets/{id}")
public Mono<ResponseEntity<Tweet>> updateTweet(@PathVariable(value = "id") String tweetId,
                                               @Valid @RequestBody Tweet tweet) {
    return tweetRepository.findById(tweetId)
            .flatMap(existingTweet -> {
                existingTweet.setText(tweet.getText());
                return tweetRepository.save(existingTweet);
            })
            .map(updateTweet -> new ResponseEntity<>(updateTweet, HttpStatus.OK))
            .defaultIfEmpty(new ResponseEntity<>(HttpStatus.NOT_FOUND));
}
```

```
kotlin {
    experimental {
        coroutines "enable"
    }
}
```

```
@PutMapping( ...value: "/tweets/{id}")
suspend fun updateTweet(@PathVariable(value = "id") tweetId: String,
                        @Valid @RequestBody tweet: Tweet): Mono<ResponseEntity<Tweet>> {
    var existingTweet: Tweet = tweetRepository.findById(tweetId).awaitFirstOrNull()
            ?: return Mono.just(ResponseEntity(HttpStatus.NOT_FOUND))
    existingTweet.text = tweet.text
    existingTweet = tweetRepository.save(existingTweet).awaitSingle()
    return Mono.just(ResponseEntity(existingTweet, HttpStatus.OK))
}
```

```
Mono<T> findById(ID var1);

Mono<T> findById(Publisher<ID> var1);

Mono<Boolean> existsById(ID var1);

Mono<Boolean> existsById(Publisher<ID> var1);

Flux<T> findAll();
```

Tipy

- ::property.name

```
@GetMapping
fun query(@AuthenticationPrincipal user: User): List<Project> {
    return repository.findAll(Sort(Sort.Direction.DESC,  ...properties: "name"))
}
```

```
@GetMapping
fun query(@AuthenticationPrincipal user: User): List<Project> {
    return repository.findAll(Sort(Sort.Direction.DESC, Project::created.name))
}
```

- https://github.com/spring-projects/spring-fu

Jetbrains ktor

- Pro případ že je pro vás spring boot těžkopádný, velký
- http://ktor.io - pouze podpora pro web app - uvnitř pouze embedded server (netty) - nic dalšího jako DI, jdbc, atd.
- Velmi rychlé startování, celé asynchronní a napsané v kotlinu

```
fun main(args: Array<String>) {
    val server = embeddedServer(Netty, 8080) {
        routing {
            get("/") {
                call.respondText("Hello, world!", ContentType.Text.Html)
            }
        }
    }
    server.start(wait = true)
}
```