
UINF/PAZ1c
epizóda 9
REST

Komunikácia medzi programami po sieti

- Mnohé prístupy
  - od bezstavového volania vzdialených procedúr
  - cez implementáciu zložitejších aplikačných sieťových protokolov
  - až po výpočtové frameworky manažujúce distribúciu dát a výpočov
- Čím jednoduchšie, tým
  - ľahšie použiteľné
  - ľahšie implementovateľné
  - ľahšia multiplatformnosť

Cesta k jednoduchosti

- služba ~ sieťovo prístupné API
  - z mobilnej aplikácie
  - z webovej aplikácie
  - z aplikácie operačného systému
  - z inej služby
- REST ~ webová služba
  - „Representational State Transfer“
  - Roy Fielding, 2000, kapitola v PhD práci
  - využíva existujúci protokol HTTP po svojom
  - jednoduché správy

Filozofia RESTu

- Realizácia CRUD operácií pomocou HTTP príkazov nad kolekciou resource-ov
  - Create ~ POST
  - Read ~ GET
  - Update ~ PUT
  - Delete ~ DELETE
- Resource
  - objekt sveta, najčastejšie entita
  - má identifikátor v tvare URI
    - http://paz1c.ics.upjs.sk/ucitelia/2

Filozofia RESTu

- Resource-y po sieti cestujú v tvare JSON

```
{“meno”: “Jano Pokojný”, “vek”: 23, “deti”: [“Janko”, “Marienka”]}
```

  - legálny JavaScriptový kód
  - parsery/writery v každom jazyku
- stavy ok / chyby / výnimky prichádzajú zo servera cez stavové kódy

200= OK, 404 = Not Found, 201 = Created,...

- filtrovania / triedenia cez request parametre

http://paz1c.ics.upjs.sk/ucitelia?minimalnyVek=35

REST v praxi

- REST API možno jednoducho zverejniť
- nejedna služba má REST API
  - Facebook, Twitter, Instagram
- štandardný spôsob
  - mobilné platformy: Android, iOS
  - web: jQuery, AngularJS, Ember.js
  - knižnice: ElasticSearch, MongoDB

Rest pre našu aplikáciu EntranceManagement

- Nastavíme si, že nový projekt je rozšírením štandardného projektu Spring Boot

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version> 3.1.5</version>
 </parent>
 <dependencies>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
 </dependencies>
```

Rest pre našu aplikáciu EntranceManagement

- SpringBoot má vlastný inicializátor projektov
  - Nemusíme sa hrať s pom.xml
- Podpora pre IntelliJ IDEA Ultimate
- Alebo si viete vytvoriť projekt online a stiahnuť
  - https://start.spring.io/
- Ak chcete používať Eclipse, zvážte plugin Spring Tools
  - https://marketplace.eclipse.org/content/spring-tools-4-aka-spring-tool-suite-4
- Poznámka: SpringBoot je obrovský framework, ktorý robí veľa "mágie". Je fajn mať v IDE na neho špeciálnu podporu

Rest pre našu aplikáciu UserManagement

- náš nový projekt, chce využívať volania metód z pôvodného projektu
- najjednoduchšie bude skopírovať kód
- "správny" spôsob by bol vytiahnuť spoločný kód do samostatnej knižnice použitím Maven modulov
- ešte "správnejší" spôsob by bolo použitie JitPacku aby sme vedeli pridať projekt na Gite ako Maven závislosť

Závislosti v Springu

- Spring má svoj systém na správu závislostí v kóde.
- Zabudnite na klasické Factory
- Privítajte @Configuration, @Bean a @Autowired
  - Implementácia návrhových vzorov **továreň** a **vkladanie závislostí** (dependency injection) pomocou anotácií

Konfigurácia databázy

- @Configuration bude naša továreň
- @Bean je objekt, ktorý chceme vytvárať v továrni
- Na všetky závislosti si vytvoríme beany.

```
@Configuration
public class Config {

    @Bean
    public UserDao userDao() {
        return new MysqlUserDao(jdbc());
    }

}
```

Konfigurácia databázy

- Takže aj na JDBC si vytvoríme Bean
- Ten však potrebuje prihlasovacie údaje do databázy a URL

```
@Configuration
public class Config {
   ...
   @Bean
   public JdbcOperations jdbc() {
      var dataSource = new MysqlDataSource();
      dataSource.setUser(dbUser);
      dataSource.setPassword(dbPassword);
      dataSource.setUrl(dbJdbc);
      return new JdbcTemplate(dataSource);
   }
   ...
}
```

Prihlasovacie údaje do databázy

- Vytvoríme inštančné premenné, ktorým dáme anotáciu @Value
- Hodnoty budú brané zo súboru **application.yaml**, alebo **application.properties** v priečinku resource.
- Pod štandardizovanými kľúčami

```
@Configuration
public class Config {
    @Value("${spring.datasource.url}")
    private String dbJdbc;

    @Value("${spring.datasource.username}")
    private String dbUser;

    @Value("${spring.datasource.password}")
    private String dbPassword;


    ...
}
```

Získanie prihlasovacích údajov do DB

- Kľúče vo @Value anotaciach musia byť v **application.properties**, alebo alternatívne v **application.yaml**

```
# src/main/resources/application.properties

spring.datasource.url="jdbc:mysql://localhost:3306/entrance"
spring.datasource.username="myuser"
spring.datasource.password="secret"
```

- Často chceme mať možnosť nastaviť parametre externe cez premenné prostredia a nemať ich hard-coded
- Vieme použiť syntax **${**PREMENNA_PROSTREDIA**:**PREDVOLENA_HODNOTA**}**

```
spring.datasource.url=${DB_JDBC:jdbc:mysql://localhost:3306/entrance}
spring.datasource.username=${DB_USER:myuser}
spring.datasource.password=${DB_PASSWORD:secret}
```

main metóda pre Spring Boot

- Pohľadá v triedach, čo má zverejniť
- Spustí webový Servlet kontajner Tomcat

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

@RestController

- Controller je trieda, ktorá obsahuje zverejnené metódy
- Je oanotovaná s @RestController
- Zverejnené metódy sú oanotované s @RequestMapping(časť_url_za_doménou)

```
@RequestMapping("/users")
  public List<User> getAllUsers() {
      ...
  }
```

Použitie DAO v kontroléri

- Stačí vytvoriť inštančnú premennú a konštruktor

```
@RestController
public class UserController {

   private final UserDao userDao;

   public UserController(UserDao userDao) {
      this.userDao = userDao;
   }
   ...
}
```

- Alebo použijeme anotáciu @Autowired a môžeme odstrániť konštruktor

```
@RestController
public class UserController {

   @Autowired
   private UserDao userDao;

   ...
}
```

Získanie hodnoty z URL

http://localhost:8080/users/1

- Všetko medzi dvoma lomkami, alebo od poslednej lomky do konca môžem vytiahnuť do premennej

```
@RequestMapping("/users/{id}")
  public User getUserById(@PathVariable int id) {
     ...
  }
```

Vyhadzujeme vlastné výnimky

```
@RequestMapping("/users/{id}")
  public User getUserById(@PathVariable int id) {
     User user = userDao.findById(id);
     if (user == null) {
        throw new UserNotFoundException();
     }
     return user;
  }
```

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException  extends
                                        RuntimeException {}
```

ResponseEntity = dáta + HTTP status

Alternatíva k vyhadzovaniu výnimiek v controlleri

```
@GetMapping("/users/{id}")
  public ResponseEntity<User> getUserById(
                                          @PathVariable int id) {
     User user = userDao.findById(id);
     if (user == null) {
          return new ResponseEntity<>(HttpStatus.NOT_FOUND);
     }
     return ResponseEntity<>(user, HttpStatus.OK);
  }
```

Posielame dáta do Restu

- HTTP príkaz sa zmení z defaultného GET na POST
- V tele požiadavky posielame dáta vo formáte Json, ktoré chceme spracovať
- alternatíva pre @RequestMapping + method je @PostMapping

```
@RequestMapping(value = "/users",
                        method = RequestMethod.POST)
public void add(@RequestBody User user) {
  /*spracujeme objekt user*/
}
```

O preklad z JSONu sa postará knižnica Jackson

Po úspešnom vytvorení pošleme 201

```
@PostMapping(value = "/users")
@ResponseStatus(HttpStatus.CREATED)
public void add(@RequestBody User user) {
        userDao.addUser(user);
}
```

Analogicky spravíme update a delete

```
@PutMapping("/users")
public void saveUser(@RequestBody User user) { .. }
```

```
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable  Long id) { .. }
```

Nepovinné parametre

- v URL za ? napr. localhost:8080/users?count=10&start=5

```
@GetMapping("/users")
public ResponseEntity<List<User>> getUsers(
      @RequestParam(value = "count", required = false) Optional<Long> count,
      @RequestParam(value = "start", required = false) Optional<Long> start) {
  …
 if (count.isPresent()) {
   resultCount = count.get();
  }
 …
}
```

Mapovač výnimiek na zmysluplné odpovede

- Anotácia @ControllerAdvice je určená na to, aby anotovala triedu, v ktorej budú spoločné odchytávače výnimiek (+iné mapovačky) pre všetky konroléry
- Metóda, ktorá mapuje výnimku na HTTP odpoveď je oanotovaná cez @ExceptionHandler(TriedaVýnimky.class)
- Exception handler okrem čísla HTTP stavu môže posielať aj telo odpovede

Trieda pre telo chybovej odpovede

- Môžeme si sami štandardizovať všetky chybové hlášky
- Klienti dostanú JSON z ktorého ľahko (vždy rovnako) extrahujú čo je problém

```
public class ApiError {

  private int status;
  private String errorMessage;

  public ApiError(int status,
                  String errorMessage) {
    this.status = status;
    this.errorMessage = errorMessage;
  }
  public int getStatus() {
     return status;
  }
  public String getErrorMessage() {
     return errorMessage;
  }
}
```

Príklad mapovača výnimiek

```
@ControllerAdvice
public class UsersAdvice {

   @ExceptionHandler(DaoException.class)
   @ResponseStatus(HttpStatus.NOT_ACCEPTABLE)
   @ResponseBody
   public ApiError handleDaoException(DaoException e) {
      return new
            ApiError(HttpStatus.NOT_ACCEPTABLE.value(),
                  e.getMessage());
   }
…
}
```

Otázky?