Pokročilé informační systémy

Business vrtstva a aplikační rozhraní

Doc. Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Implementace business operací

- Implementace business logiky nezávisle na prezentační vrstvě
- Opakovaně použitelné metody
  - Propojení s ostatními vrstvami
- Správa transakcí
- Příp. distribuce na aplikační servery

Enterprise Java Beans (EJB)

- Enterprise Java Beans (EJB)
  - Zapouzdřují business logiku aplikace
  - Poskytují business operace – definované rozhraní (metody)
  - EJB kontejner zajišťuje další služby
    - Dependency injection
    - Transakční zpracování
      - Metoda obvykle tvoří transakci, není-li nastaveno jinak

Vytvoření EJB

- Instance vytváří a spravuje EJB kontejner
- Vytvoření pomocí anotace třídy
  - `@Stateless` – bezstavový bean
    - Efektivnější správa – pool objektů přidělovaných klientům
  - `@Stateful` – udržuje se stav
    - Jedna instance na klienta
  - `@Singleton`
    - Jedna instance na celou aplikaci

Použití EJB

- Lokální
  - Anotace `@EJB` – kontejner dodá instanci EJB
- Vzdálené volání – dané rozhraní
  - Rozhraní definované pomocí `@Remote`

Contexts and Dependency Injection (CDI)

- Obecný mechanismus pro DI mimo EJB
- Omezuje závislosti mezi třídami přímo v kódu
  - Flexibilita (výměna implementace), lepší testování, …
- Injektovatelné objekty
  - Třídy, které nejsou EJB
  - Různé vlastnosti pomocí anotací
- Použití objektu
  - Anotace `@Inject`
  - CDI kontejner zajistí získáni a dodání instance

CDI – Injektovatelné objekty

- Téměř jakákoliv Javovská třída
- Scope
  - `@Dependent` – vzniká pro konkrétní případ, zaniká s vlastníkem (default)
  - `@RequestScoped` – trvá po dobu HTTP požadavku
  - `@SessionScoped` – trvá po dobu HTTP session
  - `@ApplicationScoped` – jedna instance pro aplikaci
  - *Pozor na shodu jmen se staršími anotacemi JSF*
- Pokud má být přístupný z GUI (pomocí EL)
  - Anotace `@Named`

CDI – Dodání instancí

- Anotace `@Inject`
- Vlastnost (field)

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    @Inject
    private NumberGenerator numberGenerator;

}
```

CDI – Dodání instancí

- Konstruktor

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {
    private NumberGenerator numberGenerator;

    @Inject
    public ItemServlet(NumberGenerator numberGenerator) {
        this.numberGenerator = numberGenerator;
    }

    …
}
```

CDI – Dodání instancí

- Setter

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public void setNumberGenerator(NumberGenerator numberGenerator)      {
        this.numberGenerator = numberGenerator;
    }

    …
}
```

Webové API

REST rozhraní pomocí JAX-RS

REST

- Předpokládá CRUD (Create-Retrieve-Update-Delete) operace s *entitami*
  - Ale ve skutečnosti přistupujeme k **business vrstvě**, ne přímo k datům!
  - Tzn. **voláme aplikační logiku**
- Úzká vazba na HTTP
  - Využití HTTP metod a jejich významu
  - Využití stavových kódů v HTTP
- Nedefinuje formát přenosu dat, obvykle JSON, méně XML (často obojí)

Endpointy

- Endpoint = URL, na které lze zaslat požadavek
- Reprezentuje **zdroj** (*resource*), který má nějaký **stav** (*state*)
- Endpointy pro operace se zdroji
  - Kolekce entit, např. `http://obchod.cz/api/objednavky`
  - Jedna entita, např. `http://obchod.cz/api/objednavky/8235`
- Endpointy pro volání funkcí
  - Např. `http://obchod.cz/api/odesli-objednavku`

Metody HTTP – Operace se zdroji

- **GET**
  - Čtení stavu zdroje (read)
- **POST**
  - Přidání podřízeného zdroje (přidání do kolekce, create)
- **PUT**
  - Nahrazení zdroje novým stavem (update)
- **PATCH**
  - Nahrazení části zdroje (update)
- **DELETE**
  - Smazání zdroje (delete)

Metody HTTP – Volání funkcí

- **GET** i **POST**
  - Vykoná operaci vrátí výsledek (serializovaná data)
- Pokud je výsledkem operace nový zdroj, jeho URL se vrátí v hlavičce `Location`.

Stavový kód

- Stavový kód odpovědi HTTP může odpovídat výsledku operace
- Typicky například:
  - 200 Ok
  - 201 Created
  - 400 Bad request
  - 403 Forbidden
  - 404 Not found
  - 500 Internal server error

Formát přenosu dat

- Není specifikován, záleží na službě
  - Obvykle JSON nebo XML (schéma záleží na aplikaci)
- Často více formátu k dispozici
  - Např. http://noviny.cz/clanky.xml http://noviny.cz/clanky.json
  - Využití MIME pro rozlišení typu, HTTP content negotiation

REST a Jakarta EE

- JAX-RS API součástí standardu
- Vytvoření služeb pomocí anotací
- Aplikační server zajistí funkci endpointu (JAX-RS servlet)
  - Mapování URL a HTTP metod na Javovské objekty a metody
  - Serializace a deserializace JSON/XML na objekty
- Různé implementace
  - JAX-RS – Jersey (Glassfish), Apache Axis
  - Serializace – Jackson, gson, MOXy, ...
  - Neřešíme - je vždy součástí aplikačního serveru

REST v Javě

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public User getUser(@PathParam("username") String userName) {
        // volání business operací (aplikační logika)
        return someUser;
```

- Demo: endpointy

Konfigurace

- Třída odvozená od `javax.ws.rs.core.Application`
- Konfigurace pomocí anotací
- Např. ApplicationConfig

Klientská aplikace

- Zasílání REST požadavků
  - Jednotlivé JS frameworky mají vlastní infrastrukturu
- Prezentační logika
  - Navigace
  - Přechody mezi stránkami
  - Výpisy chyb, apod.
- Viz např. jednoduchý klient v jQuery

Návrh REST rozhraní

- Architektura REST je volná, umožňuje „chaoticky“ přidávat endpointy podle potřeby
- **Systematický návrh je nutný**
  - Pevné datové struktury (vycházející z doménového modelu)
    - Včetně reprezentace chybových stavů
  - Mapování business operací na endpointy (vycházející z případů použití)
- Ideálně formální popis rozhraní
  - Mnohem lepší než sdílená tabulka

Popis služeb v REST

- WADL (Web Application Description Language)
  - Založený na XML
  - Podpora v Javě – např. Payara
- OpenAPI https://swagger.io/specification/
  - Používá YAML (alternativně JSON)
  - Např. Payara/Liberty http://localhost:8080/openapi/
  - Generátory rozhraní třetích stran, např. https://github.com/OpenAPITools

Autentizace v REST

- Protokol REST je definován jako **bezstavový**
  - Požadavek musí obsahovat vše, žádné ukládání stavu na serveru
- To teoreticky vylučuje možnost použití sessions pro autentizaci
  - Technicky to ale možné je
  - Problém např. pro mobilní klienty
- Alternativy pro autentizaci:
  - **HTTP Basic** autentizace (nutné HTTPS)
  - Použití tokenu validovatelného na serveru – např. **JWT**
  - Složitější mechanismus, např. OAuth

HTTP Basic

- Standardní mechanismus HTTP, využívá speciální hlavičky
- Požadavek musí obsahovat hlavičku **Authorization**
  - Obsahuje jméno a heslo; nešifrované pouze kódované (base64)
    - **Je nutné použít HTTPS**
- Pro nesprávnou nebo chybějící autentizaci server vrací **401 Auhtorization Required**
  - V hlavičce `WWW-Autenticate` je identifikace oblasti přihlášení
  - Klient tedy zjistí, že je nutná autentizace pro tuto oblast

JSON Web Token (JWT)

- Řetězec složený ze 3 částí
  - Header (hlavička) – účel, použité algoritmy (JSON)
  - Payload (obsah) – JSON data obsahující id uživatele, jehopráva, expiraci apod.
  - Signature (podpis) – pro ověření, že token nebyl podvržen nebo změněn cestou
- Tyto tři části se kódují (base64) a spojí do jednoho řetězce
  - **xxxxx.yyyyy.zzzzz**

Použití JWT

- Klient kontaktuje **autentizační server** a dodá autentizační údaje
  - Stejný server, jaký poskytuje API, nebo i úplně jiný (např. Twitter)
- Autentizační server vygeneruje podepsaný JWT a vrátí klientovi
- Klient předá JWT při každém volání API
  - Nejčastěji opět v hlavičce:

    ```
    Authorization: Bearer xxxxx.yyyyy.zzzzz
    ```

  - API ověří platnost, role uživatele může být přímo v JWT

JWT v Javě

- Součástí standardu Microprofile
  - Ne přímo součást Jakarta EE
  - Ale dostupná na běžných serverech (Payara, Liberty, ...)
- Lze snadno spojit s JAX-RS
- Demo: https://github.com/DIFS-Teaching/rest-auth

Generování tokenu

- Nastavení hodnot *claims*
  - Zejména `iss`, `upn` a `groups`
- Podpis privántím klíčem (RSA, ECDSA), nebo symetrická kryptografie (HMAC)
- Klient uloží token a posílá s každým požadavkem
- Postup generování klíčů např. zde

Autorizace pomocí JWT

- Ověření podpisu pomocí veřejného klíče
- Ověření vydavatele (`iss`)
- Použití uživatelského jména (`upn`) a rolí (`groups`) pro autorizaci.

Autorizace v Javě

- Zapnutí pro celou aplikaci + ověření role u endpointu

```
@ApplicationPath("resources")
@LoginConfig(authMethod = "MP-JWT")
@DeclareRoles({ "admin", "staff", "customer" })
public class JAXRSConfiguration extends Application {

}
```

```
@GET
@Path("/protected")
@RolesAllowed("admin")
public String getProtected() {
    return "ok";
}
```

Konfigurace JWT

- Pomocí Java properties, např. soubor

  `META-INF/microprofile-config.properties`

```
mp.jwt.verify.publickey.location=/publicKey.pem
mp.jwt.verify.issuer=fitdemo
```

(přepokládá veřejný klíč v `/publicKey.pem`)

GraphQL

Typovaná aplikační rozhraní

GraphQL

- https://graphql.org/
- Motivace: klient (klienti) potřebují v různých situacích různá data
  - Např. stránka „seznam osob“ vs. „detail osoby“
  - REST endpoint vrací vždy stejnou strukturu
    - Redundance dat (nevyužijeme všechna data)
    - Více dotazů ((ne)efektivita, složitější logika klienta)
  - Získat některá data může být drahé (pokud je klient nepotřebuje)
- Řešení GraphQL
  - Popis datového modelu API
  - Dotaz na API specifikuje požadovaný tvar odpovědi
  - **Předvídatelný výsledek dotazu**

GraphQL – datový model

- Datový model API (ne nutně serverové aplikace)
- Jednoduché datové typy: Int, Float, String, Boolean, ID, enum
- Uživatelské typy (*types*) = struktury
  - Vlastnosti (parametrizovatelné) – jméno, parametry, typ
  - Typy jednoduché, struktury (vztahy), kolekce
- Speciální typy reprezentující volání API (*root types*)
  - Query – čtení dat
  - Mutation – změna dat
- GraphQL Schema Definition Language (SDL)

GraphQL – definice typů

```
type Person {
  name: String!
  age: Int!
  posts: [Post!]!
}
```

```
type Car {
  type: String!
  reg: String!
  owner: Person!
}
```

```
type Query {
  allPersons: [Person!]!
  findPerson(name: String!): Person!
}
```

```
type Mutation {
  createPerson(name: String!, age: Int!): Person!
}
```

GraphQL – dotazy

```
{
  allPersons {
    name
  }
}
```

```
{
  "allPersons": [
    { "name": "Jan" },
    { "name": "Karolína" },
    { "name": "Alice" }
  ]
}
```

```
{
  findPerson(name: "James") {
    name
    age
    cars {
      type
    }
  }
}
```

```
{
  "findPerson": {
    "name": "James",
    "age": 28,
    "cars": [
        { "type": "Fiat" },
        { "type": "Tesla" }
    ]
  }
}
```

GraphQL – modifikace

```
mutation AddPerson($person: PersonInput)
    updatePerson(p: $person) {
        id
    }
}
```

```
 {
     "person": {
        "born": "1991-06-03T22:00:00Z[UTC]",
        "id": 1154731299,
        "name": "Sylvester",
        "surname": "Stalone"
     }
}
```

```
{
    "data": {
        "updatePerson": {
            "id": 1154731299
        }
    }
}
```

GraphQL přes HTTP

- Jediné endpoint URL
- Odeslání přes GET

  `http://myapi/graphql?query={me{name}}`
- Odeslání přes POST
  - Data `application/json`

    `{"query": "{me{name}}"}`

    `{"query": "{mutation … }", "variables": {name: "Jan"}}`
  - Data `application/graphql`

    `{me{name}}`

Implementace GraphQL

- Klientská strana
  - Žádná speciální podpora – posílání POST požadavků
- Serverová strana
  - Definice datových struktur a dotazů
  - Zpracování SDL nebo generování SDL z kódu
  - Mnoho knihoven
    https://graphql.org/code/

Microprofile GraphQL

- Demo: Open Liberty Guide, demo aplikace
- Závislost v `pom.xml`

```
<dependency>
  <groupId>org.eclipse.microprofile.graphql</groupId>
  <artifactId>microprofile-graphql-api</artifactId>
  <version>2.0</version>
  <scope>provided</scope>
</dependency>
```

- V Open Liberty přidat vlastnost na server

```
<feature>mpGraphQL-2.0</feature>
```

Definice API

```
@GraphQLApi
@RequestScoped
public class Api
{
    @Inject
    private PersonManager personMgr;

    @Query
    @Description("Gets the complete list of people")
    public List<Person> getPeople()
    {
        return personMgr.findAll();
```

Definice datových typů

- Standardní java třídy (POJO)
- Volitelně lze nastavit mapování
  - jmen tříd – `@Type("...")`
  - vlastností – `@Name("...")`
  - a doplnit popis významu – `@Description`.

Autentizace v GraphQL

- Stejný mechanismus, jako u REST
- Stejná konfigurace JWT

```
@GraphQLApi
@LoginConfig(authMethod = "MP-JWT", realmName = "MP-JWT")
@RequestScoped
public class Api
{
    @Inject
    private PersonManager personMgr;

    @Query
    @RolesAllowed("admin")
    @Description("Gets the complete list of people")
    public List<Person> getPeople()
```

A to je vše!

Dotazy?