Pokročilé informační systémy

Alternativní tehchnologie a architektury

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

Třívrstvá architektura

- Java EE umožňuje implementovat **monolitický** IS s **třívrstvou architekturou**:
  - Databázová vrstva
    - JPA – definice entit, persistence (`PersistenceManager`)
    - Alternativně: Relační databáze (JDBC), NoSQL (MongoDB), …
  - Logická (business) vrstva
    - Enterprise Java Beans (EJB) nebo CDI beans
    - Dependency injection – volné propojení
  - Prezentační vrstva
    - Webové rozhraní (JSF) nebo API (REST, JAX-RS)

Java Server Faces (JSF)

Serverová prezentační vrstva

Prezentační vrstva

- Webové API + JS tlustý klient
  - Prezentační logika na klientovi
  - Business operace přístupné pomocí API (REST, SOAP, …)
  - Klientská aplikace může využívat klientský framework
    - Angular, React.js, Vue.js, …
- Serverový framework
  - Prezentační logika na serveru
  - Server generuje HTML (CSS, JavaScriptový) kód
  - Komponentově orientované frameworky

Prezentační vrstva na serveru

- Funkčnost zajišťuje webový kontejner
- Pro Jakarta EE standardně 2 vrstvy
  - Facelets
  - Java Server Faces
- Existují další webové frameworky (Struts, Spring, Vaadin, …)

Java Servlet

- Třída implementující chování serveru
- Obecný `jakarta.servlet.GenericServlet`
  - Metody `init()`, `destroy()`, `service()`
- HTTP `jakarta.servlet.http.HttpServlet`
  - Metody `doGet()`, `doPost()`, …
- Vytváření instancí řídí server
- Jedna instance může obsluhovat více požadavků

Java Server Pages

- Dynamické webové stránky
  - HTML (XHTML, XML, …) kód
  - Vložený kód v Javě (*scriptlet*)
  - Definované značky
  - Výrazy – unified expression language (EL)
- Umožňuje definovat **knihovny značek** (tag libraries)
  - Chování každé značky implementováno jako třída
  - XML deskriptor knihovny

Příklad JSP

```
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>

<!DOCTYPE html>
<html>
<head>
    <title>My JSP Page</title>
</head>
<body>
    <h1>Hello World!</h1>
    <p>
    <%
```

Zpracování JSP stránky

- Stránka se překládá na servlet (třídu)
- Překlad zajišťuje kontejner
  - Skripty – vloženy do stránky
  - Tagy – volání metod příslušných tříd
  - Výrazy – volání evaluátoru

Facelets

- Překlad XHTML stránek na interní objektovou reprezentaci – **Facelet**
  - Zpracování šablon
  - Související soubory (resources)
- Možnost definice knihoven značek
  - Vystačíme s existujícími
- Zpracování výrazů jazyka EL
  - Přístup k objektům, jejich vlastnostem a metodám

XML namespaces

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets">
```

```
<ui:composition template="template.xhtml">
    <ui:define name="content">
        <h2>Welcome</h2>
        <p><a href="person_list.xhtml">Simple forms demo</a></p>
    </ui:define>
</ui:composition>
```

Java – Namespaces

- Facelets tags

```
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
```

- JSF Core – obecné definice

```
xmlns:f="http://xmlns.jcp.org/jsf/html"
```

- JSF HTML – obsah stránky

```
xmlns:h="http://xmlns.jcp.org/jsf/core"
```

Java Server Faces

- Komponentový MVC framework nad Facelets
- Značky pro generování základních prvků uživatelského rozhraní
  - Rozšiřitelné knihovny komponent
- Řídicí servlet implementující chování

Nadstavbové knihovny

- PrimeFaces
- Apache MyFaces
- OmniFaces
- RichFaces (JBoss, není dále vyvíjen)
- …

Experssion language

- `#{objekt}` – vyhledá objekt daného jména v kontextu stránky, požadavku, session a aplikace
- Metody: `#{customer.sayHello}`
- Vlastnosti: mapují se na set / get
  - `#{customer.name}`

Logika uživatelského rozhraní

- EJB/CDI služby – model
- *Managed beans*
  - Spravované objekty přístupné přes EL
  - Implementují chování UI
  - Vlastnosti a metody vázané na prvky GUI
- *Validators*
  - Kontrola vstupních dat
- *Converters*
  - Převod dat na řetězec a zpět

Definice chování

- Každá stránka produkuje výsledek ve formě řetězce (outcome)
  - Statický (zadaný konstantou)
  - Dynamický (generovaný metodou)
    - Často po provedení nějaké akce
- Přechody mezi stránkami externě v XML souboru `faces-config.xml`

Demo: Web frontend

Jiné platformy

Alternativy k Jakarta EE

Další platformy – přehled

- Java
  - Existuje mnoho možností kromě „standardní“ J EE
- .NET (Core / Framework)
  - Mnoho řešení na všech vrstvách
- PHP
  - Různé frameworky, důraz na webovou vrstvu
- JavaScript
  - Node.js + frameworky, důraz na web a mikroslužby
- Python, Ruby, … - podobné principy

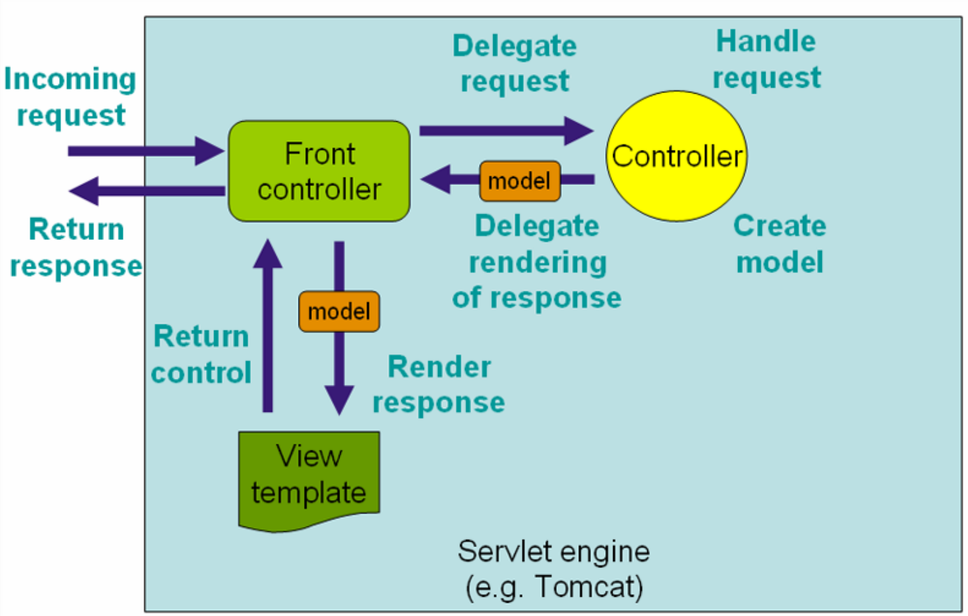
Java – alternativy

- Databázová vrstva
  - Hibernate ORM – implementuje JPA, ale i vlastní API
  - NoSQL databáze – Hibernate OGM, EclipseLink, …
- Business vrstva
  - Spring framework – alternativa EJB pro dependency injection, transakce, správa sezení, …
- Prezentační vrtstva
  - Spring MVC (controllers + JSP / Thymeleaf…)
  - Struts, Play!, …
  - https://www.dailyrazor.com/blog/best-java-web-frameworks/

Spring

- Vznikl jako alternativa k EJB
  - Využití POJO místo (tehdy složitých) EJB
  - Omezení požadavků na infrastrukturu
- Modulární struktura, mnoho součástí
- Dependency injection
  - Podobně jako v J EE, anotace `@Bean`, `@Autowire`, …
  - Opět field, constructor, setter injection
- Spring MVC
  - Tradiční MVC přístup, bližší ostatním frameworkům
  - Ukázková aplikace https://github.com/spring-projects/spring-mvc-showcase

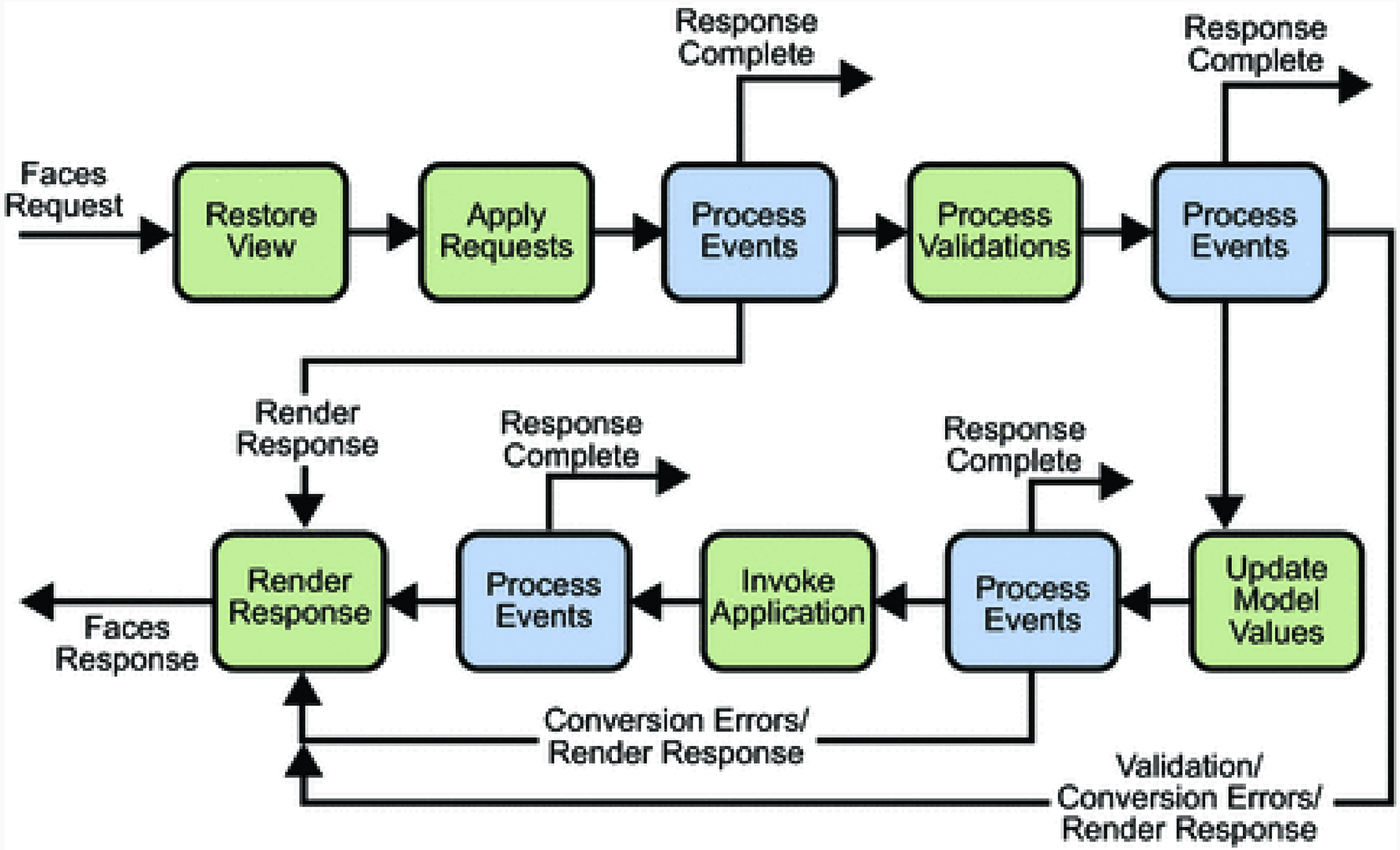
Zpracování požadavku Spring MVC



- Handler matching – anotace v controller třídách, vrací popis view (různé formáty) nebo přímo výsledný obsah
- View matching – výběr view podle výsledku (pokud je)

- https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html

Pro srovnání: JSF



Spring Boot

- Přístup „Vše je v aplikaci“ (včetně serveru)
  - Na rozdíl od Java EE – „Server umí vše“ (thin WARs)
- Usnadňuje vytvoření aplikace a konfiguraci závislostí
  - Maven nebo Gradle šablony
  - Spring moduly (MVC, Security, …), Thymeleaf, JPA,…
- Snadné vytvoření funkční aplikace
  - Třída reprezentující celou aplikaci
  - Konfigurace pomocí anotací
  - Spustitelná main() metoda
- https://www.baeldung.com/spring-boot-start

.NET

- .NET Core / .NET Framework
- Databázová vrstva
  - Entity Framework, (LINQ, Dapper, …)
    https://docs.microsoft.com/cs-cz/ef/core/modeling/
- Business vrstva (služby)
  - ASP.NET Core (dependency injection, middleware)
    https://docs.microsoft.com/en-us/aspnet/core/fundamentals/
- Webová vrstva
  - Razor (MVVM) – two-way data binding, HTML
  - ASP.NET Core MVC – logika+model v C#, view v HTML, REST, …

Entity Framework – entita

```
[Table("Product")]
public class Product
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public int CategoryId { get; set; }

    [Required, StringLength(50)]
    public string Name { get; set; }
```

PHP

- PHP je rozšiřující modul HTTP serveru
  - Žádný trvale běžící kontejner
  - + stabilita řešení
  - - efektivita, možnost udržovat stav napříč požadavky
- PHP Frameworky
  - Laravel – (MVC) https://laravel.com/
  - Symfony – (MVC) https://symfony.com/
  - Nette – (MVP) https://nette.org/
  - …
- Správa závislostí – composer

Databázová vrstva

- Různé vlastní přístupy
- Laravel
  - Fluent query builder – specifikace SQL dotazů v PHP

    ```
    DB::table('users')->where('name', 'John')->first();
    ```

  - Eloquent ORM
- Nette
  - Nette database – parametrizovatelné SQL dotazy
- **Doctrine** – pokročilé ORM, podobné JPA
  - Integrovatelné do všech frameworků

Doctrine: Entita

```php
<?php
use Doctrine\ORM\Annotation as ORM;

/**
 * @ORM\Entity @ORM\Table(name="products")
**/
class Product {

    /** @ORM\Id
        @ORM\Column(type="integer")
        @ORM\GeneratedValue **/
    protected $id;
```

Doctrine: Uložení objektu

```php
$product = new Product();
$product->setName("Tatranky");
$entityManager->persist($product);
$entityManager->flush();

echo "Created Product with ID "
        . $product->getId() . "\n";
```

Business vrstva v PHP

- Obvykle v podobě služeb – services
- Framework poskytuje DI kontejner, který registruje služby
  - Procedurálně v PHP nebo externí konfigurační soubor
- Při vytváření controlleru framework dodá závislosti
  - Obvykle *constructor injection*
- Příklady
  - Laravel https://laravel.com/docs/5.8/container#resolving
  - Symfony https://symfony.com/doc/current/components/dependency_injection.html
  - Nette https://doc.nette.org/cs/2.4/dependency-injection

Zpracování požadavku v PHP

- Požadavek na kořenový dokument (index.php)
- Bootstrapping frameworku
  - Načtení konfigurace
  - Inicializace součástí, rozšíření, služeb (DI)
  - Obnova session
- Dekódování parametrů požadavku
  - Směrování požadavku – routing
- Volání aplikační logiky
  - Vytvoření instance controlleru
  - Volání metody podle požadavku (handler)
- Vytvoření odpovědi (view rendering)

Zpracování požadavku v PHP – Laravel

- Přiřazení controllerů k URL je definováno odděleně
  `Route::get('user/{id}', 'UserController@show‘)`
- Controller konfiguruje a vrací view
- https://laravel.com/docs/5.8/controllers

Zpracování požadavku v PHP – Symfony

- Controller je přiřazen k URL pomocí anotací
- Vrací objekt Response
  - Různé druhy, případně včetně obsahu, přesměrování, …
  - Případně sám zajišťuje použití šablon
- https://symfony.com/doc/current/controller.html

Zpracování požadavku v PHP – Nette

- Požadavek vyřizuje *presenter* (metoda `renderXyz(params)`)
- Presenter si sám spravuje model (není formalizováno)
- Předává data do view (template) nebo přímo odesílá odpověď (`sendResponse()`)
- https://doc.nette.org/cs/2.4/presenters

JavaScript – node.js

- Standardní řešení pro JS na serveru
- V8 JavaScript Engine + knihovny
- Procedurální implementace zpracování HTTP požadavků
  - Obdobně jako servlety
- Ukázka: https://nodejs.org/en/docs/guides/getting-started-guide/
- Správce balíků npm
  - Jednoduchá instalace závislostí (knihoven)

Databázová vrstva

- Knihovny pro podporu relačních DB serverů k dispozici v rámci platformy node.js
  - Např. MySQL https://expressjs.com/en/guide/database-integration.html#mysql
- Existují i ORM řešení
- Např. Sequelize
  - Podpora MySQL, SQLite, PostgreSQL, MSSQL
  - https://github.com/sequelize/express-example

Sequelize

```
const User = sequelize.define('user', {
  firstName: {
    type: Sequelize.STRING
  },
  lastName: {
    type: Sequelize.STRING
  }
});
```

```
// Vytvoří tabulku
User.sync({force: true}).then(() => {
  return User.create({
    firstName: 'John',
    lastName: 'Hancock'
  });
});
```

Business vrstva

- Implementace v JS, žádné standardní řešení
- Modularizace řešena na úrovni node.
- Případné DI řešení
  - https://www.npmjs.com/package/node-dependency-injection

Webová vrstva

- Velké množství frameworků s různými přístupy
  - http://nodeframework.com
- Express
  - Mapování HTTP požadavků na funkce v JS
    http://expressjs.com/en/guide/routing.html
  - Views pomocí několika template engines
    http://expressjs.com/en/guide/using-template-engines.html
- Full stack frameworky
  - Těsnější integrace s frontendem, např. Meteor

Další architektury API

Alternativy k REST a GraphQL

Standardizace API

- Předchůdci REST
  - Snaha o maximální standardizaci volání serverových služeb přes HTTP
  - Vznik komplikovaných standardů webových služeb (SOAP atd.)
  - Obtížně použitelné bez podpůrných technologií – omezení na konkrétní implementační platformy
- REST – zjednodušení v reakci na komplikovanost WS
  - Flexibilita, ale žádný standard – mnoho ad hoc řešení
- GraphQL

XML-RPC

- Předchůdce webových služeb
- Jednodušší – stále používané
- Definované XML zprávy pro předání parametrů i výsledku
- Podpora datových typů včetně polí, seznamů a struktur

XML-RPC volání

```
<?xml version="1.0"?>
<methodCall>
    <methodName>trida.jePrvocislo</methodName>
    <params>
        <param>
            <value><int>1345</int></value>
        </param>
    </params>
</methodCall>
```

XML-RPC výsledek

```
<?xml version="1.0"?>
<methodResponse>
    <params>
        <param>
            <value><boolean>0</boolean></value>
        </param>
    </params>
</methodResponse>
```

Volání v PHP

```
function xmlrpc($url, $method, $params, $types = array(), $encoding = 'utf-8
    foreach ($types as $key => $val) {
        xmlrpc_set_type($params[$key], $val);
    }

    $context = stream_context_create(array('http' => array(
        'method' => "POST",
        'header' => "Content-Type: text/xml",
        'content' => xmlrpc_encode_request($method, $params,
            array('encoding' => $encoding))
    )));
```

Webové služby (Web Services)

- Vzdáleně volané podprogramy umístěné na serveru
- Volání pomocí protokolu HTTP
  - Předání vstupních parametrů
  - Vrácení výsledku
- Předávají se XML data definovaná protokoly
  - WSDL – popis rozhraní služby
  - SOAP – volání služby

Popis rozhraní služby: WSDL

- Web Services Description Language
- Platformově nezávislý popis rozhraní
  - XML dokument
  - Využíva XML Namespaces a XML Schema
- Definuje
  - Názvy funkcí
  - Jejich parametry
  - Způsob volání (vstupní URL)

Příklad popisu služby

```
<message name="jePrvocisloRequest">
    <part name="cislo" type="xsd:long"/>
</message>
<message name="jePrvocisloResponse">
    <part name="return" type="xsd:boolean"/>
</message>
<portType name="Cisla">
    <operation name="jePrvocislo" parametrOrder="cislo">
        <input message="m:jePrvocisloRequest"
            name="jePrvocisloRequest"/>
         <output message="m:jePrvocisloResponse"
            name="jePrvocisloResponse"/>
```

Příklad popisu služby (II)

```
<binding name="cislaSoapBinding" type="m:Cisla">
 …
</binding>
<service name="CislaService">
    <port binding="m:cislaSoapBinding" name="cisla">
       <wsdlsoap:address location="http://nekde.cz/cisla"/>
     </port>
</service>
```

Použití WSDL

- Na základě popisu lze automaticky generovat rozhraní v cílovém jazyce
  - „Stub“ - zástupnou metodu, která implementuje volání skutečné metody přes HTTP
- Rovněž lze generovat WSDL popis z rozhraní v cílovém jazyce
- Současné vývojové nástroje umožňuji vytvoření WS z funkce „jedním kliknutím“
  - Např. v Eclipse

Volání služby: SOAP

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  env:encodingStyle="
            http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xs="http://www.w3.org/1999/XMLSchema"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
 <env:Header/>
 <env:Body>
  <m:jePrvocislo xmlns:m="urn:mojeURI">
    <cislo xsi:type="xs:long">1987</cislo>
  </m:jePrvocislo>
 </env:Body>
```

Odpověď služby

```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <env:Body>
  <ns1:jePrvocisloResponse
    xmlns:ns1="urn:mojeURI"
    env:encodingStyle="
            http://schemas.xmlsoap.org/soap/encoding/">
   <return xsi:type="xsd:boolean">true</return>
  </ns1:jePrvocisloResponse>
 </env:Body>
```

Komplexní příklady

- WSDL http://www.w3.org/TR/wsdl20-primer/#basics-greath-scenario
- SOAP http://www.w3.org/TR/soap12-part0/#L1165
- W3C Web Services Activity http://www.w3.org/2002/ws/#documents

Vyhledání webových služeb

- Idea centrálního registru
  - Protokol UDDI (Universal Description, Discovery and Integration)
  - Poskytovatelé ukládají WSDL popisy, klienti prohledávají
- Přehled služeb daného poskytovatele
  - WSIL (Web Services Inspection Language)
  - Seznam služeb v souboru `/inspection.wsil`

Implementace WS

- Jakarta EE
  - JAX-WS API součástí standardu
  - Vytvoření webových služeb pomocí anotací
  - Běží na Jakarta EE aplikačním serveru
- PHP
  - Rozšíření SOAP
  - Třídy SoapServer, SoapClient, ...

Jakarta EE

- Součástí Java EE je **Java API for XML Web Services (JAX-WS)**
- Server
  - Definice služeb pomocí anotací tříd a metod
  - Automaticky generuje WSDL popis
- Klient
  - Automatické generování proxy třídy z WSDL popisu

Java EE Implementace

```
package helloservice.endpoint;

import javax.jws.WebService;
import javax.jws.webMethod;

@WebService
public class Hello {

    public Hello() {    }

    @WebMethod
    public String sayHello(String name) {
```

PHP

- Rozšíření SOAP
- Server
  - Třída `SoapServer`
  - Registruje třídy a metody implementující službu
- Klient
  - Třída `SoapClient`
  - Zpracuje WSDL a zpřístupní vzdálené metody

PHP Soap Server

```php
<?php

function sayHello($name) {
    return "Hello, " . $name;
}

$server = new SoapServer(null,
        array('uri' => "urn://helloservice/endpoint"));
$server->addFunction('sayHello');
$server->handle();
```

PHP SOAP Klient

```php
$client = new SoapClient(null,
    array('location' => "http://…/simple_server.php",
          'uri'      => "urn://my/namespace"));

$name = "Karel";
$result = $client->
            __soapCall("sayHello",array($name));

print $result;
```

PHP s WSDL

```
$soap = new SoapClient(
    'http://api.search.live.net/search.wsdl');

print_r($soap->__getFunctions());

$ret = $soap->Search(…);
```

Mikroslužby (Microservices)

Architektura orientovaná na služby

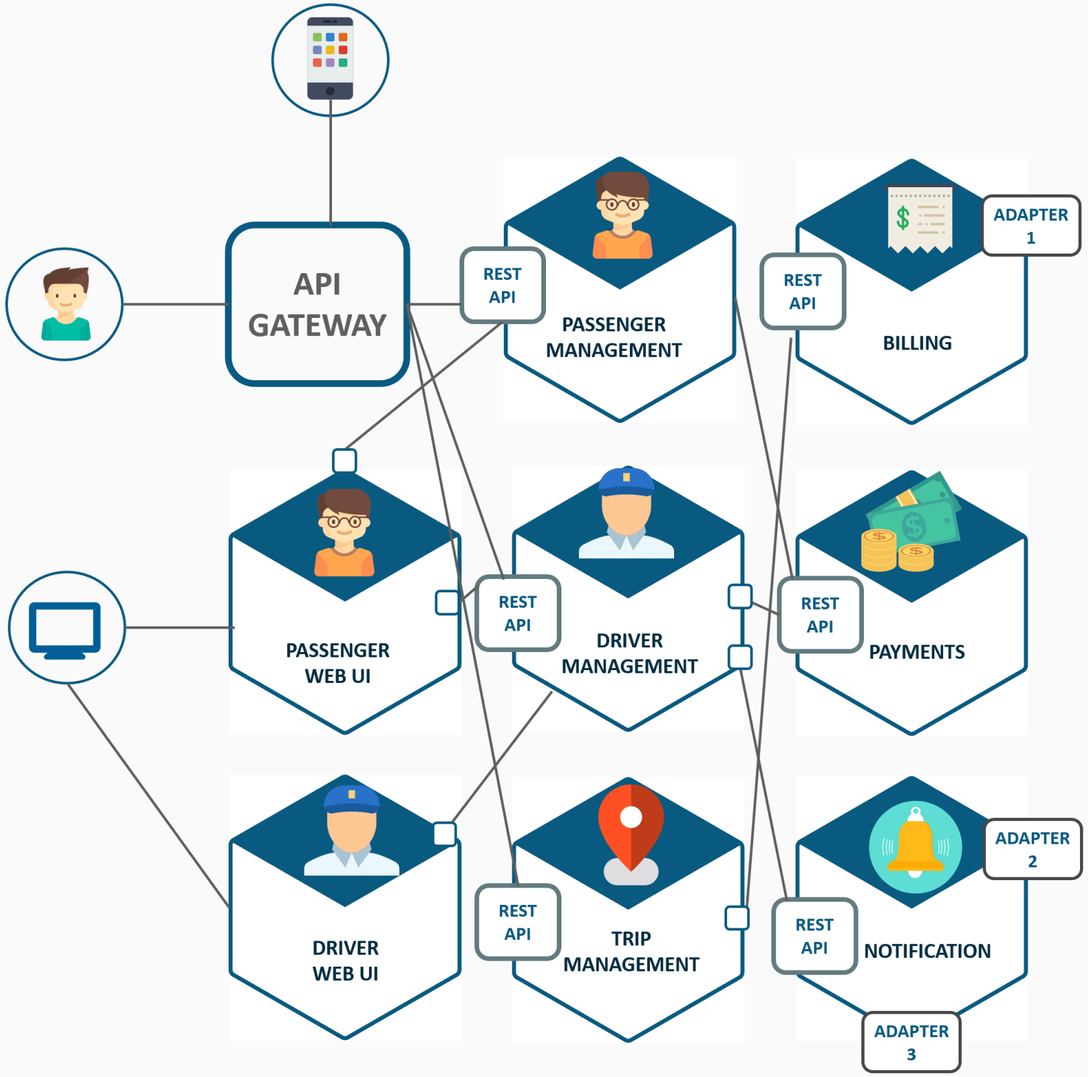
Monolitická architektura

- Jedna aplikace
  - Jedna databáze, webové (aplikační) rozhraní
  - Business moduly – např. objednávky, doprava, sklad, …
- Výhody
  - Jednotná technologie, sdílený popis dat
  - Testovatelnost
  - Rychlé nasazení – jeden balík
- Nevýhody
  - Rozměry aplikace mohou přerůst únosnou mez
  - Neumožňuje rychlé aktualizace částí, reakce na problémy
  - Pokud použité technologie zastarají, přepsání je téměř nemožné

Mikroslužby

- Aplikace je rozdělena na malé části
  - Vlastní databáze (nepřístupná vně)
  - Business logika
  - Aplikační rozhraní (REST)
- Typicky malý tým vývojářů na každou část (2 pizzas rule)
- Výhody
  - Technologická nezávislost
  - Snadné aktualizace, kontinuální vývoj
- Nevýhody
  - Testovatelnost – závislosti na dalších službách
  - Režie komunikace, riziko nekompatibility, řetězové selhání, ...

Mikroslužby (příklad: Uber)



Vlastnosti mikroslužby

- Vnější API
  - Dostatečně obecné – reprezentuje logiku, ne např. schéma databáze (která je skrytá)
- Externí konfigurace
- Logování
- Vzdálené sledování
  - Telemetrie – metriky (počty volání apod.), výjimky
  - Sledování živosti (Health check)
  - Logování, trasování

V čem implementovat mikroslužby?

- V čemkoliv – spojovacím bodem je pouze API
- Node.js (+ express + MongoDB)
  - Populární rychlé řešení
- Java
  - Spring Boot
  - Ultralehké frameworky Např. Spark - https://github.com/perwendel/spark
  - Microprofile
  - Quarkus.io, Helidon.io

Eclipse Microprofile

- https://microprofile.io/
- Standard založený na Jakarta EE
  - Podmnožina rozhraní JEE (např. CDI, JAX-RS, JSON-B)
  - Specifická rozhraní pro mikroslužby
    - Config, Health Check, Metrics, JWT Auth, REST Client, ...
- Příklad služby
  - https://github.com/DIFS-Teaching/java-micro-service

Microprofile – další API

- Config
  - Externí konfigurace služby – zdroje, priority, …
- Fault Tolerance
  - Řešení výpadků kvůli závislosti služeb
  - Timeout, Retry, …
- Health Check
  - Vzdálené zjištění živosti mikroslužby

Microprofile – další API (II)

- JWT Authentication
- Metrics
  - Statistiky o využití služby – vzdálené měření výkonu
- OpenAPI
  - Generování formalizované dokumentace API služby
- REST Client
- Příklady https://github.com/payara/Payara-Examples/tree/master/microprofile

Health Check

- Liveness
  - Jestli služba žije, nebo jestli je třeba ji restartovat
  - `/health/live`
- Readiness
  - Žije a může správně pracovat – např. zdroje k dispozici
  - `/health/ready`
- HTTP Status code 200 (UP), 503 (DOWN), 500 (nelze zjistit)
- https://download.eclipse.org/microprofile/microprofile-health-2.1/microprofile-health-spec.html

Metrics

- Sběr metrik různých typů v aplikaci
  - Gauge – spojitá hodnota (měřidlo), např. délka fronty
  - Counter – počítadlo, např. počet reg. uživatelů
  - Timer – časové údaje, např. průměrný čas operace, atd.
- Počty volání a čas strávený v metodách
  - `@Metered`, `@Timed`
- Centrální API pro sběr metrik
  - Data sbírá a zpřístupňuje server
  - /metrics, /metrics/application
- Předpokládá využití řešení pro sběr metrik (např. Prometheus) a případně vizualizaci (např. Grafana)

Distribuované logování

- Chyby mohou nastat v jednotlivých mikroslužbách i v komunikaci mezi nimi
- V případě výpadku nelze procházet jednotlivé logy odděleně
- Distribuované logování
  - Centralizace logů, programátor rozhoduje, co se loguje
- Distribuované sledování (tracing)
  - Sledování celého průběhu operací a jejich výsledků

Distribuované logování

- Nástroje pro centralizovaný sběr logů
  - Např. ELK stack:
    - Elastic Search – ukládání, prohledávání a analýza dat (JSON)
    - Logstash – sběr logů z různých zdrojů
    - Kibana – vizualizace a procházení
- Podpora v aplikacích
  - Nutný výstup ve vhodném formátu (např. pro logstash)
  - V Javě např. log4j, logback.

Distribuované sledování

- Nástroje pro sledování aplikací
  - Např. Jaeger, Zipkin
- Podpora v aplikacích
  - Např. Microprofile Open Tracing, Spring Cloud Sleuth

Microprofile Open Tracing

```
@RequestScoped
@Path("/items")
@Produces(MediaType.APPLICATION_JSON)
public class CatalogService {
  @Inject
  Tracer tracer;

  @Timeout(value = 2, unit = ChronoUnit.SECONDS)
  @Retry(maxRetries = 2, maxDuration = 2000)
  @Fallback(fallbackMethod = "fallbackInventory")
  @GET
  @Traced(value = true, operationName = "getCatalog.list")
```

Viz Monitor and Debug Java Microservices with MicroProfile OpenTracing nebo Open Liberty guides.

Config Injection

- Hodnoty parametrů dodaných z vnějšku (`@Inject`, `@ConfigProperty`)
- Různé datové typy (zabudované a vlastní konvertory)
- Zdroje konfiguračních hodnot
  - `META-INF/microprofile-config.properties`
  - Environment variables
  - System properties
    - Např. Open Liberty: `server/jvm.options`
  - Možné vlastní zdroje konfigurace

REST Client

- Vytvoření REST klienta z definice služby
- Anotované rozhraní služby, podobně jako JAX-RS
- Vytvoření REST klienta
  - `RestClientBuilder`
- Injektování hotového klienta
  - `@RestClient`
- https://github.com/payara/Payara-Examples/tree/master/microprofile/rest-client
- https://github.com/eclipse/microprofile-rest-client

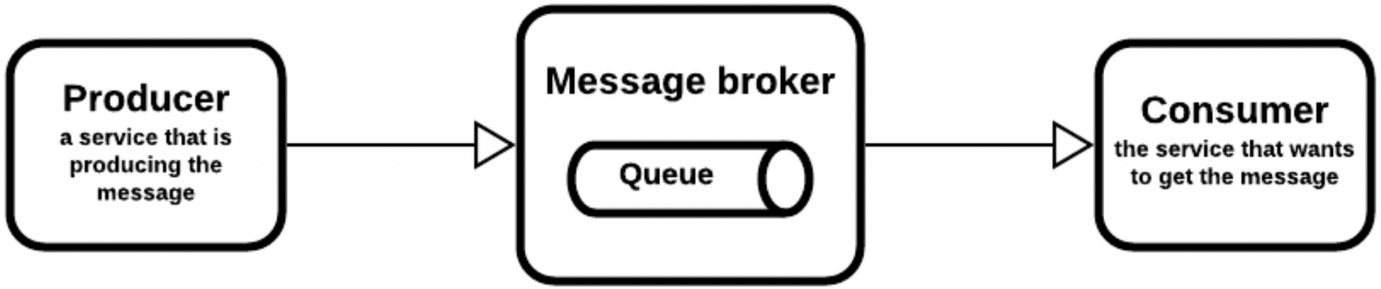
Fault tolerance

- Automatické opakování metody, pokud dojde k výjimce
  - `@Retry`
- Timeout volání metody
  - `@Timeout`
- Fallback metody
  - `@Fallback`
  - `@CircuitBreaker`
- https://github.com/eclipse/microprofile-fault-tolerance
- https://github.com/payara/Payara-Examples/tree/master/microprofile/fault-tolerance

Synchronní a asynchronní komunikace

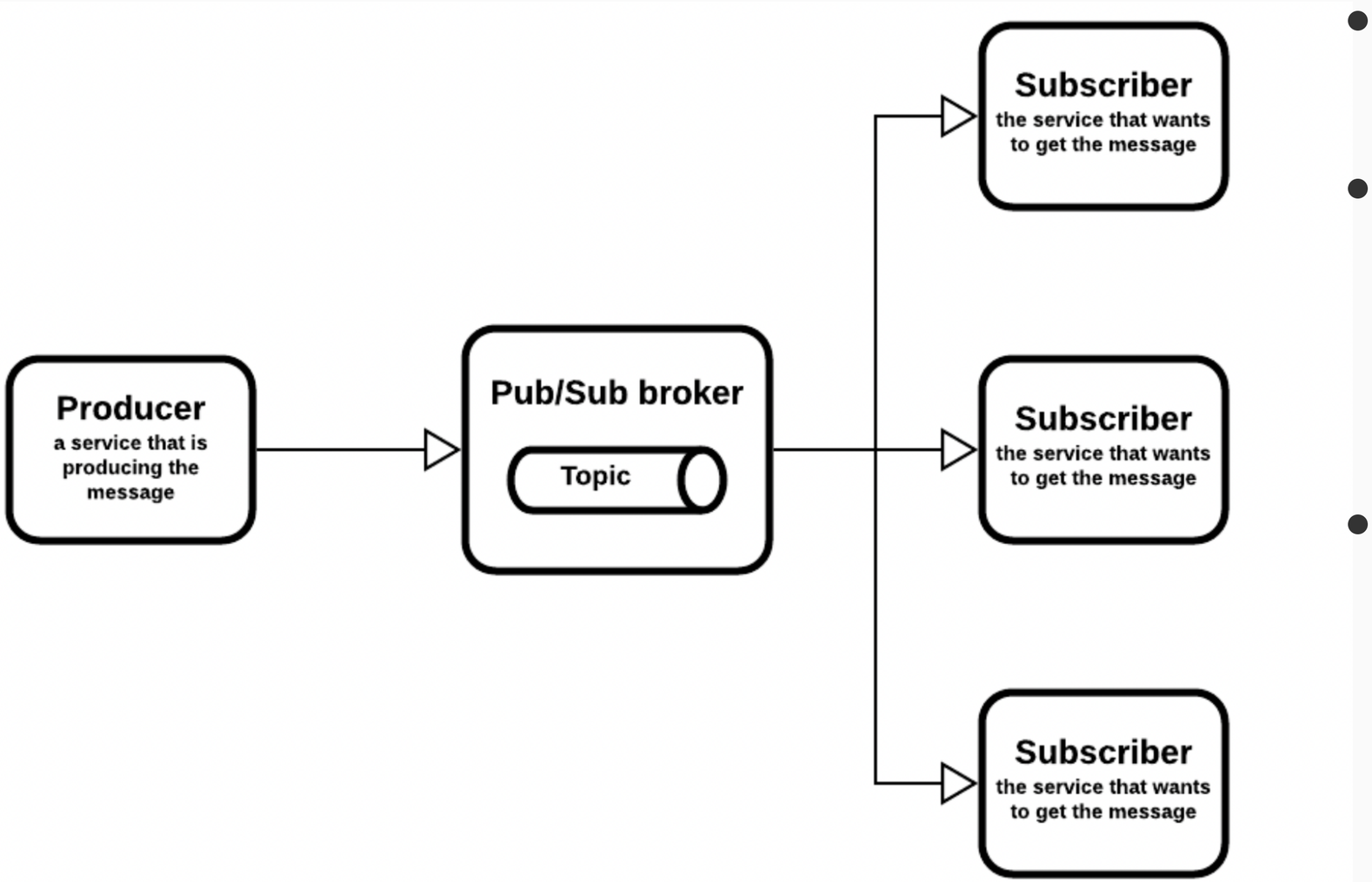
- Synchronní komunikace
  - Nejčastěji REST
  - Klientská služba čeká na odpověď od volané služby (blokování)
  - Pokud volaná služba není dostupná, klientská služba čeká nebo selže
- Asynchronní komunikace
  - Služba zasílá zprávy, nečeká na odpověď
  - Výměnu zpráv zajišťuje centrální *message broker*
  - *Message queue* vs. *Publish/subscribe*

Message queue



- Tvoří se fronty zpráv od konkrétního producenta pro konkrétního konzumenta
- Např. RabbitMQ, Apache ActiveMQ nebo Amazon SQS

Publish / subscribe



- Zprávy jsou přiřazovány k tématům (*topics*)
- Producent publikuje zprávy k tématu, konzument se přihlašuje k doběru tématu
- Apache Kafka, Pulsar, Amazon SNS

Microprofile API

- Standard Microprofile Reactive Messaging
- Podpora různých brokerů
  - Apache Kafka, Amazon Kinesis, RabbitMQ, Apache ActiveMQ, ...
- Anotace metod produkujících zprávy `@Outgoing("my-channel")` a přijímajících zprávy `@Incoming("my-incoming-channel")`
- Demo pro OpenLiberty
  - Creating reactive Java microservices

A to je vše!

Dotazy?