The Spring framework – J2EE s lidskou tváří

Opravdu s lidskou tváří?

Trochu o mě

- Roman Pichlík
- HP Software
- Blog http://sweb.cz/pichlik
- Czech Java User Group
- CZ Podcast

Agenda

- J2EE
- Proč je tu Spring
- Spring IOC
- Spring AOP
- Spring DAO
- Dynamické jazyky
- Spring a vliv na J2EE

J2EE intro

- Nadstavba standardní Javy
  - Enterprise oblast
    - Misssion critical aplikace
      - Webové aplikace?
- Sada specifikací a API
  - Java Community Process
  - Middleware dodavatelé
- Klíčové technologie
  - EJB, JTA, Servlety a JSP, JSF, JNDI, JCA a další...

Jak bylo J2EE myšleno

- Robustnost
- Interoperabilita
- Bezpečnost
- Cluster-aware
- SOA
- Buzzwords...

Jak to dopadlo

- One size fits all
  - 80/20
  - Webové aplikace
- Velká komplexnost
  - Jednoduché věci složitě a složité nemožně
  - Příliš mnoho k učení
- Standardy vs Standardizace
  - Pomalá adopce
  - Rychlé zastarávání

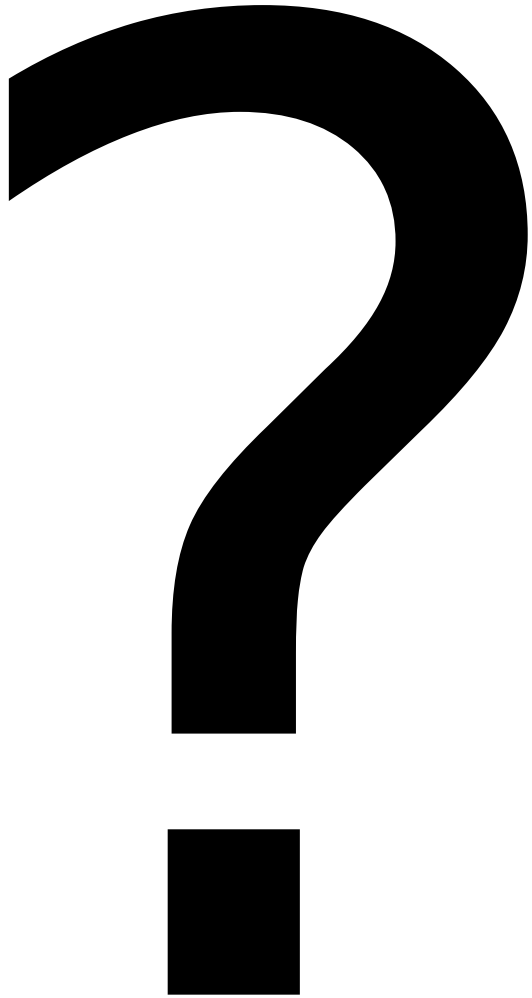
Co lidé opravdu chtějí

- Jednoduchost
  - Testovatelnost
  - Vývoj
  - Nasazení
- Agilnost
  - myšlenky
  - technologie
  - postupy

hp invent

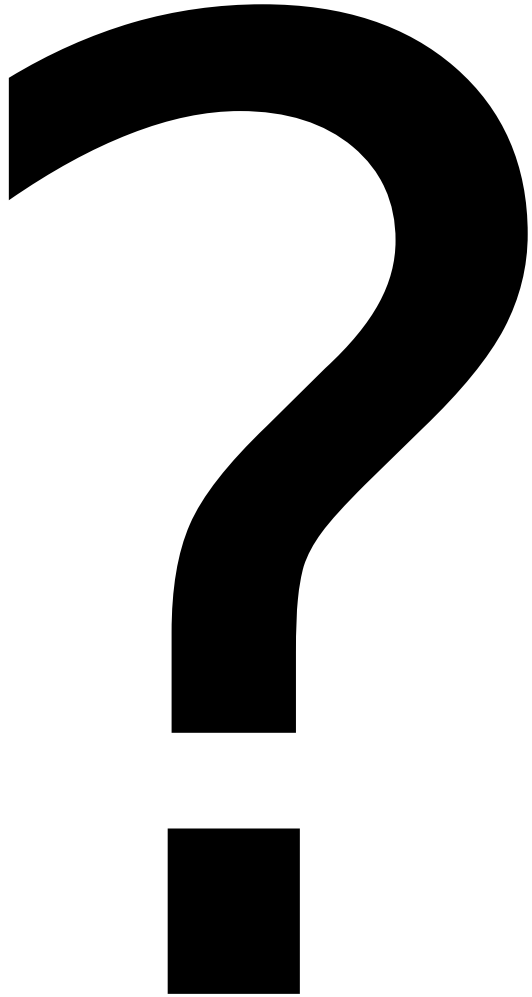
Nabídka se nestřetla s poptávkou

Přichází náš rytíř



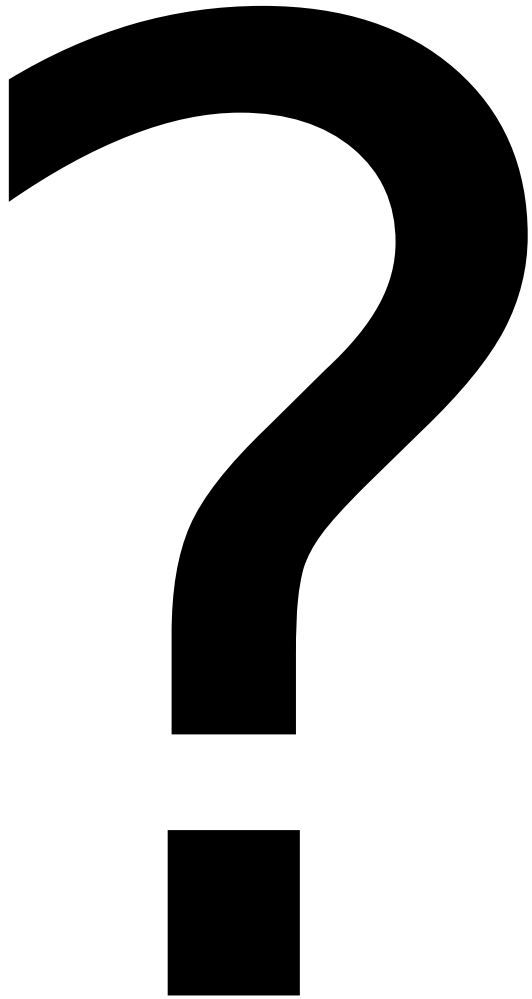
Přichází náš rytíř

Přichází náš rytíř

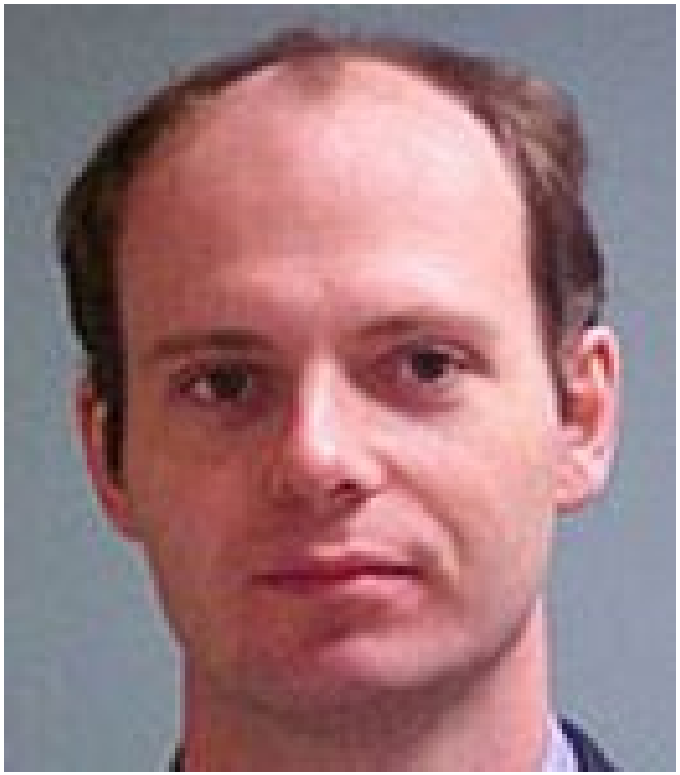


Přichází náš rytíř

Přichází náš rytíř

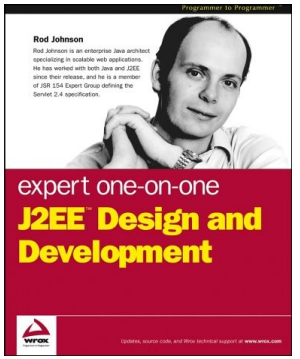


Přichází náš rytíř



Historie Spring frameworku

- 2002
  - Rod Johnson, J2EE Design and Development
    - Návod jak dobře a úspěšně používat J2EE
    - 30000 řádků kódu



- 2003 založení open source projektu
  - http://springframework.org/
  - http://www.springframework.org/documentation

Úhel pohledu

- Komplexní řešení
  - Web, Enterprise, Desktop
- Poskytovatel služeb
  - transakce, AOP, konfigurace, remoting...
- Rozhraní aplikace/prostředí
  - aplikační server, webový kontejner, testy
- Rozhraní aplikace/technologie
  - Hibernate, JMS, JSF, JPA, JDBC, EJB...

Co možná o Springu uslyšíte

- Náhrada J2EE a především EJB
- Sada modulů pro různé použití
- Továrna na továrny
- Webový framework
- Sada pomocných API pro Hibernate
- Přežitek v době EJB 3.0
- Výstřelek zhýralých vývojářů

Největší mýty

- Spring se nehodí pro Enterprise aplikace
  - 10 velkých bankovních institucí používá Spring
  - Weblogic, WebSphere AS certifikují Spring
- Spring nepoužívají velké firmy
  - eBay, Oracle, HP
- Spring nejde škálovat
  - Open Terracotta
  - Tangosol Coherence Data Grid

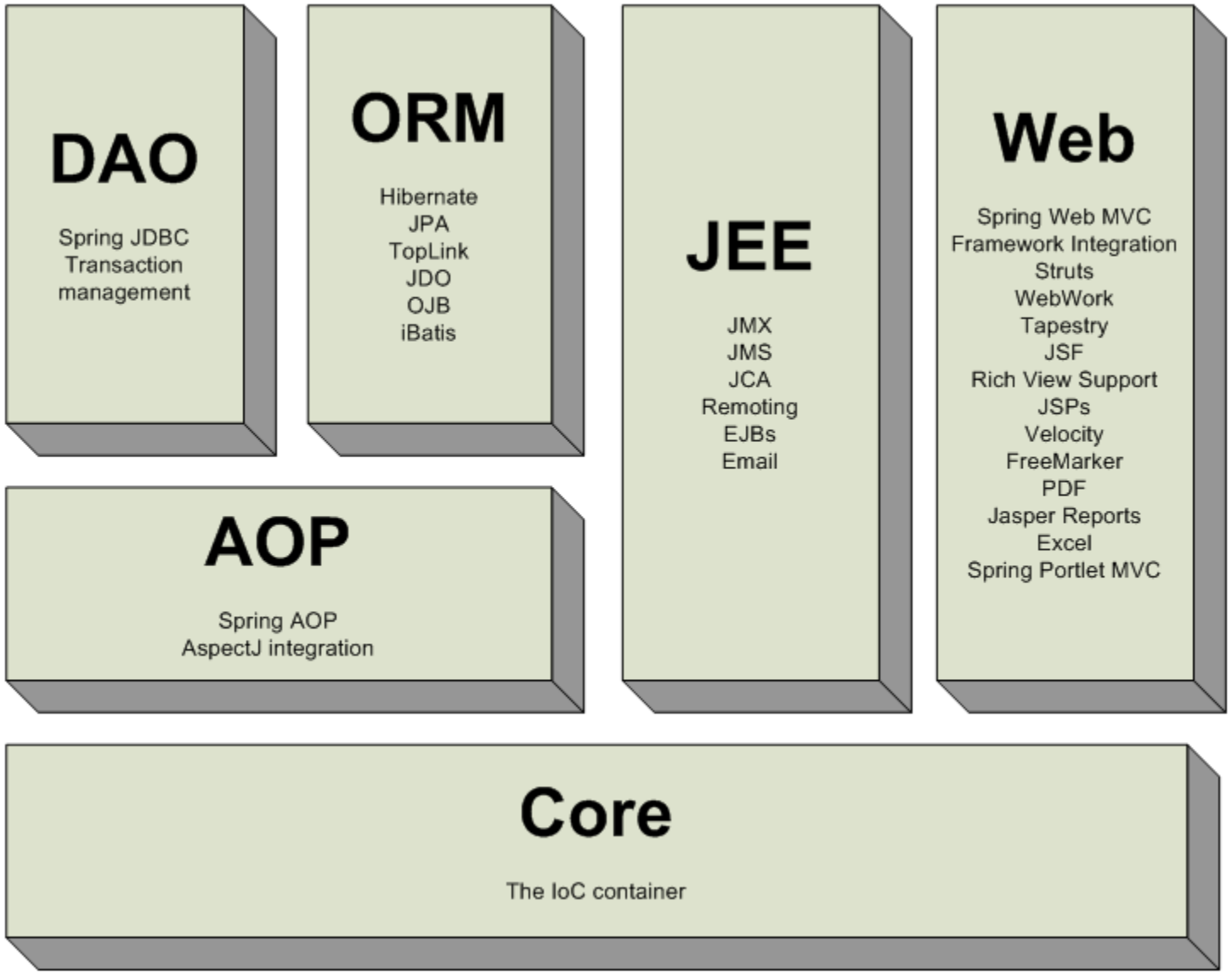
Základní myšlenky

- Zjednodušení komplexnosti
  - Nejenom J2EE
  - API Abstrakce
- Neinvazivnost
- Zaměření na architekturu aplikace
- Jednoduchá testovatelnost a použití
- Modulárnost

Základní myšlenky řečí technologie

- Inversion Of Control
- Aspektově orientované programování (AOP)
- OOP přístup
  - programování rozhraním
- Open-Closed princip
  - otevřený pro rozšíření
  - uzavřený pro modifikace
- POJO (Plain Old Java Objects) přístup

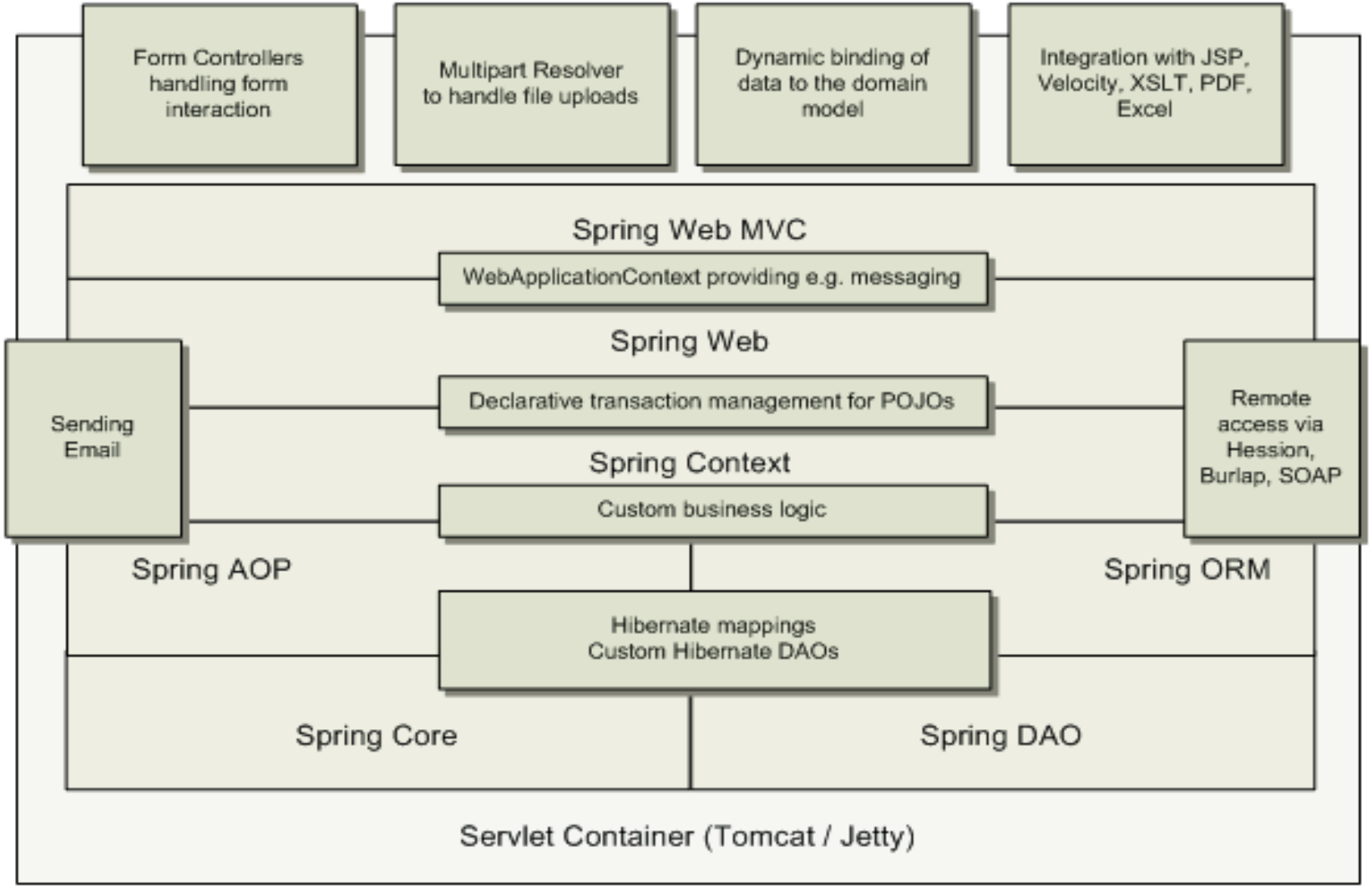
Stavební kameny



Modulárnost

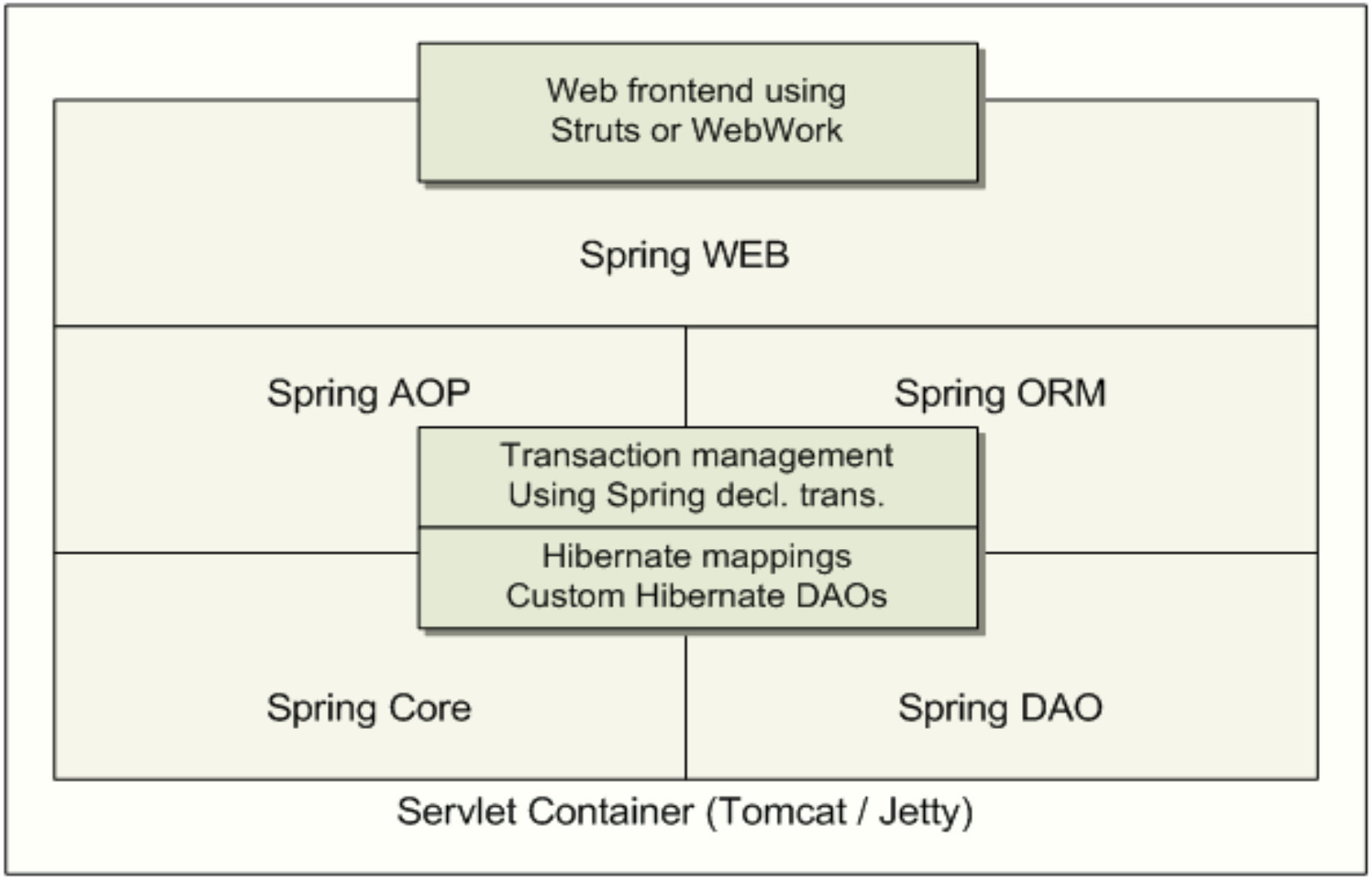
- Použij co potřebuješ
- JAR distribuce
  - kompletní (2,5 MB)
  - po modulech
- Malá závislost na knihovnách třetích stran
- Různé scénáře použití modulů

Všechny moduly



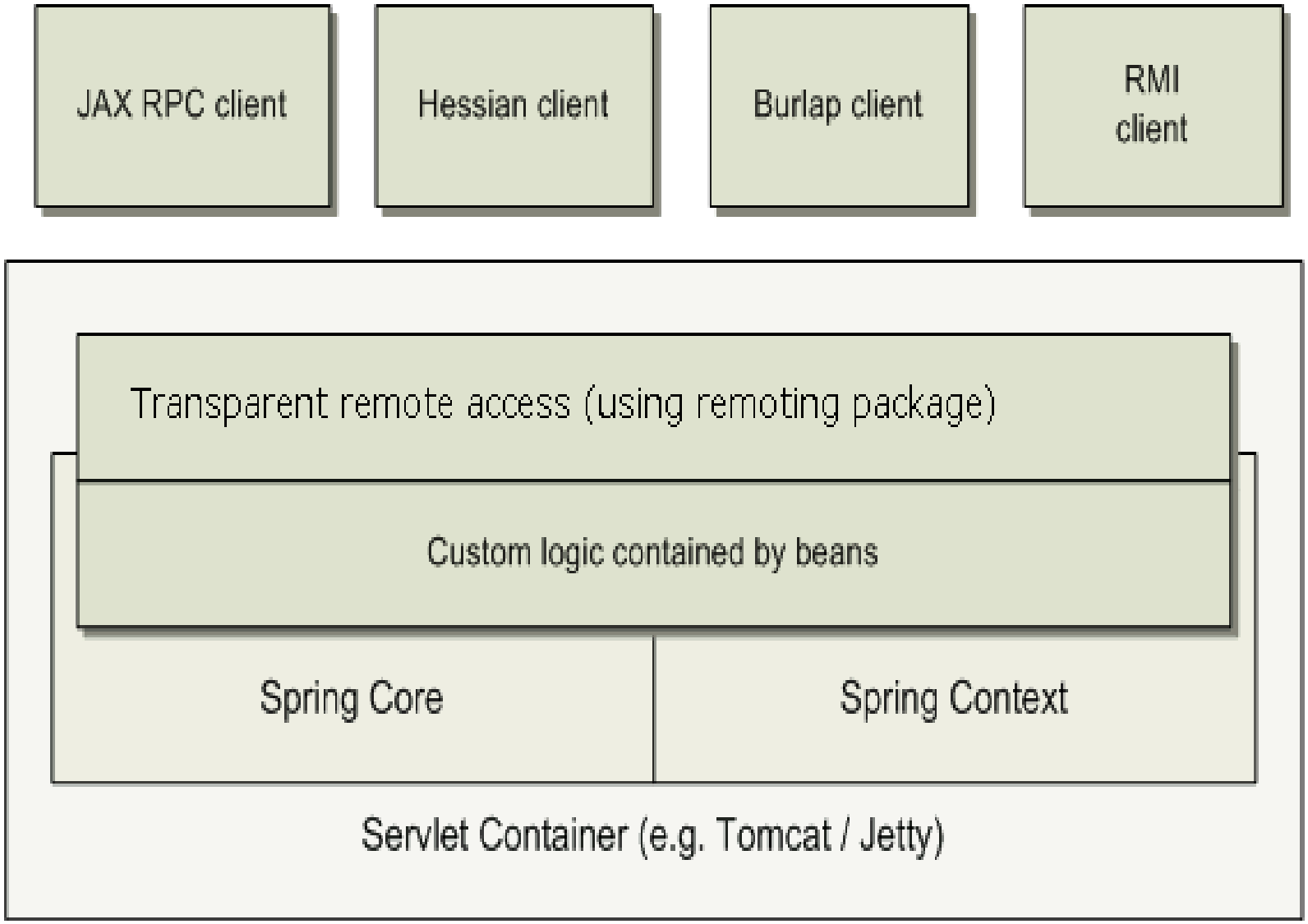
Typical full-fledged Spring web application

Integrace s web frameworky



Spring middle-tier using a third-party web framework

Remoting


JAX RPC client
Hessian client
Burlap client
RMI client
Transparent remote access (using remoting package)
Custom logic contained by beans
Spring Core
Spring Context
Servlet Container (e.g. Tomcat / Jetty)

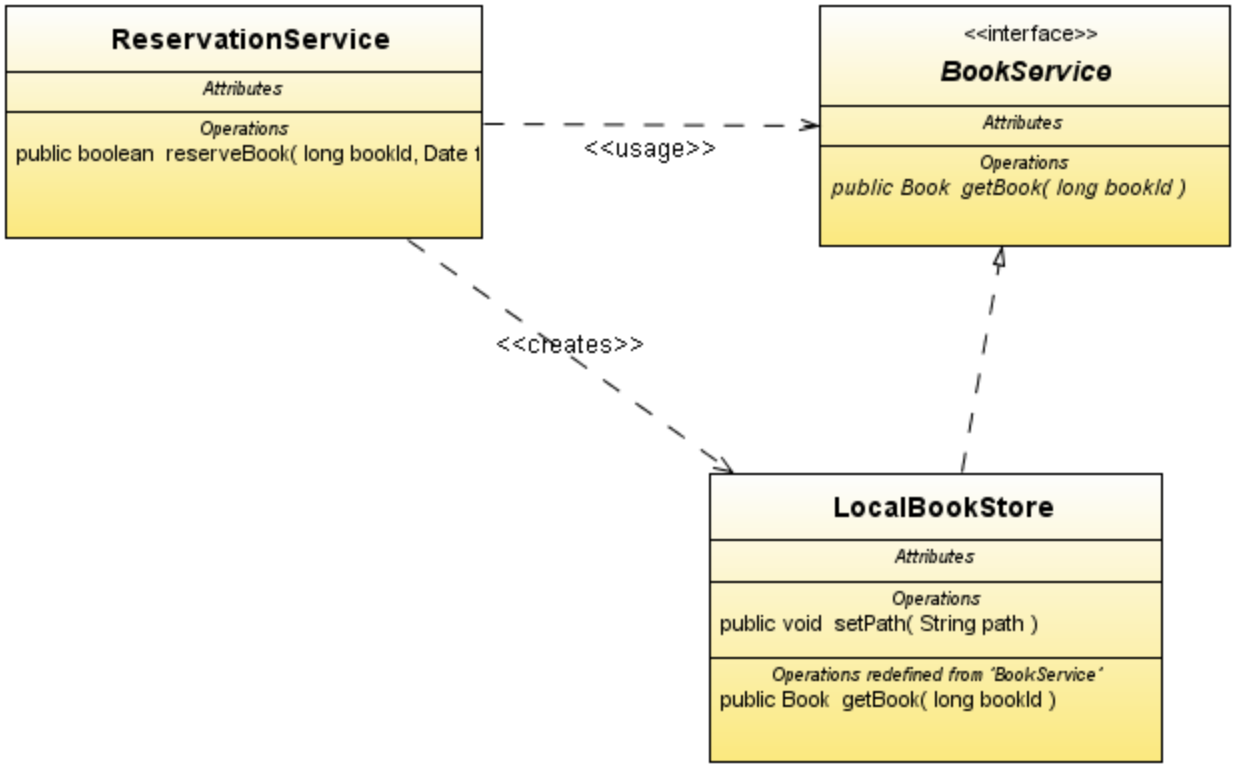
Subprojekty kolem Springu

- Spring Web Flow
- Spring Security
- Spring Dynamic Modules
- Spring Web Services
- Spring Batch
- Spring Integration
- Spring IDE
- Spring SpringSource dm Server

Inversion Of Control

Příklad

- Rezervace knih



Typický kód

```
public class ReservationService{
    private BookService bookService;

    public ReservationService() {
        init();
    }
    private void init () {
        //create concrete instance of BookService implementation
        LocalBookStore localBookStore = new LocalBookStore();
        localBookStore.setPath("/usr/home/data.xml"); //setup
        bookService = localBookStore;
    }
    public boolean reserveBook(long bookId, Date from,Date to,User u) {
        Book book = bookService.getBook(bookId);
        //reservation code omitted
        return true;
    }
}
```

Typický kód - problémy

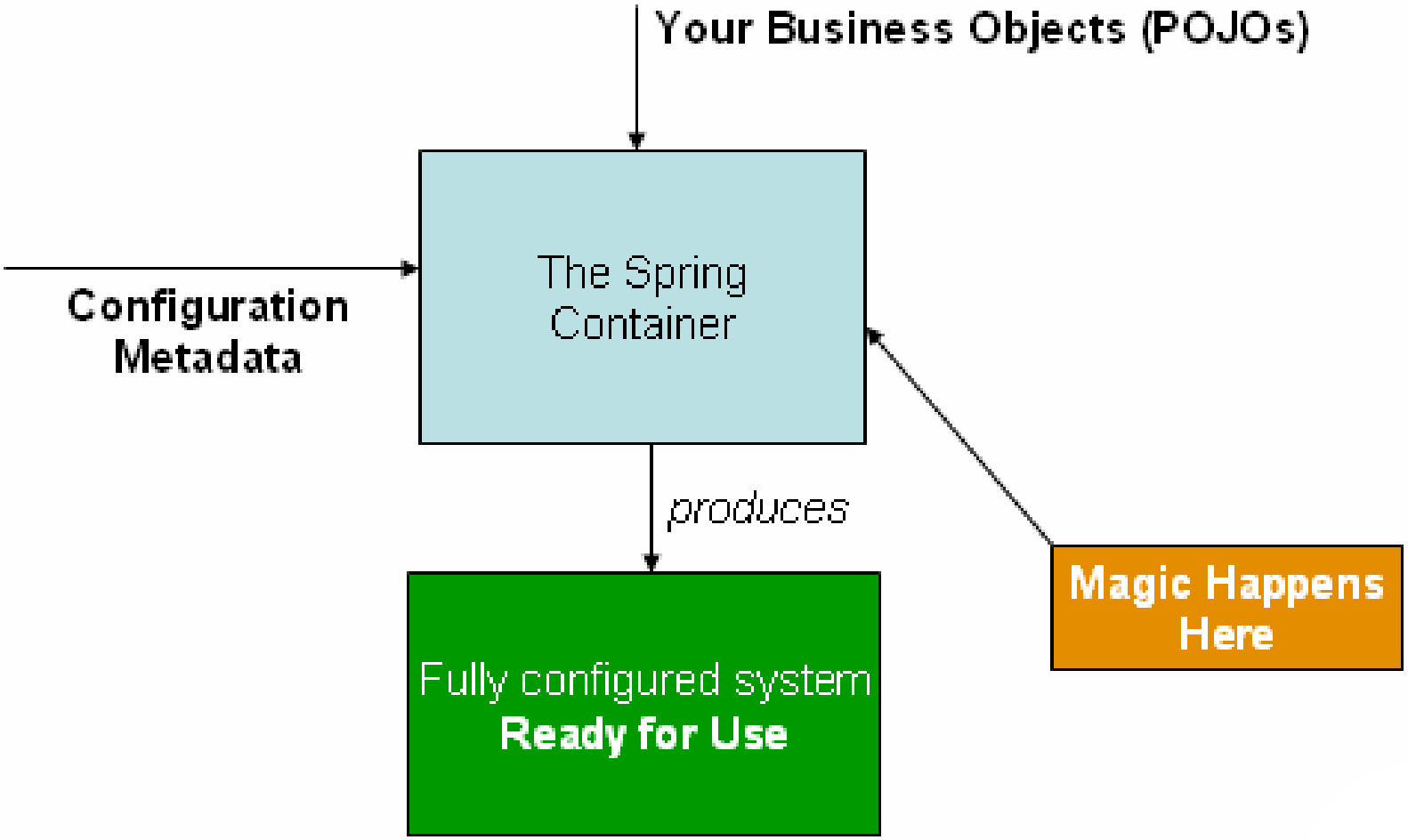
- Těsné vazby mezi objekty
  - Volba konkrétní implementace
  - Odpovědnost za inicializaci

```
private void init () {
        //create concrete instance of BookService implementation
        LocalBookStore localBookStore = new LocalBookStore();
        localBookStore.setPath("/home/dagi/data.xml"); //setup
        bookService = localBookStore;
}
```

Inversion Of Control

- M. Fowler (2004) návrhový vzor
- Proč
  - Těsné vazby mezi objekty
  - Vytváření a skládání objektů podle jejich vazeb
- Jak
  - Obrácená kontrola
  - Odpovědnost na někom třetím
    - IoC kontejner
- Chce to mentální kotrmelec ;-)

IoC kontejner



The Spring IoC container

Objekty (POJOs)

```
public class ReservationService {
    private BookService bookService;

    public boolean reserveBook(long bookId, Date from, Date to, User user) {
        Book book = bookService.getBook(bookId);
        //reservation code omitted
        return true;
    }
}
```

```
public class LocalBookStore implements BookService {
    private String path;

    public Book getBook(long bookId) {
        //accessing code omitted
    }

}
```

Metadata

```
@Component
public class ReservationService {
    @Autowired
    private BookService bookService;
    ...
}
```

```
@Component
public class LocalBookStore  implements BookService {
    @Autowired
    @Qualifier("file.path")
    private String path;
    ....
}
```

```
<beans>
    <context:annotation-config/>
    <context:component-scan base-package="cz.sweb.pichlik"/>
    <bean id="file.path" class="java.lang.String">
        <constructor-arg value="/home/dagi.data.xml"></constructor-arg>
    </bean>
</beans>
```

Kontejner

```
ApplicationContext container =
          new ClassPathXmlApplicationContext("applicationContext.xml");
ReservationService reservationService =
       (ReservationService) container.getBean(ResrvationService.class);
```

Test

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"applicationContext.xml"})
public class ReservationServiceTest {
    @Autowired
    private ReservationService reservationService;

    @Test
    public void testReserveBook() {
        reservationService.reserveBook
        ...
    }
}
```

Spring IoC kontejner

- Základní a nejdůležitější komponenta
- Metadata
  - XML, Anotace, Konfigurační kód atd.
- Snadná inicializace
  - Testy, Web, EJB, programově
- Atributy
  - Hierarchie, Události, Dependency injection
  - Scopes, Placeholders, vysoká customizovatelnost
  - ...

Spring kontejner vs. EJB kontejner

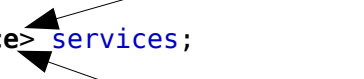
	Spring IoC	**EJB 3.0 IoC**
definice závislosti	XML descriptor, anotace, properties, konfigurační java kód	anotace, XML descriptor
aplikovatelnost	bez omezení	Servlet, listener classes, web services end-point, JAX-RPC handlers, DataSource, JMS, Mail, EJB, Environment entries, EntityManager, UserTransaction, EJB Beans, interceptors, web services end-point, DataSource, JMS, Mail, Environment entries, EntityManager, EJB Context,
nepřímé závislosti	ano	ne
způsob nastavení závislosti	setter, anotace proměnné, konstruktor	setter, anotace proměnné
extension pointy	AOP, BeanPostProcessor, Factory, lifecycle rozhrani, typová konverze	interceptory, lifecycle rozhrani
autowiring	typem, jménem	ne

Autowiring – beans collecting

- Loosely coupled extensibility
- Sběr objektů určitého typu

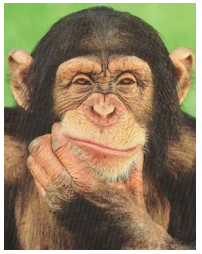
```
@Component
public class ServiceRegistry {
    @Autowired
    private List<BusinessService> services;

}
```

```
@Component("serviceA")
public class BusinessServiceA implements BusinessService {


}

@Service("serviceB")
public class BusinessServiceB implements BusinessService {


}
```



Aspect Oriented Programming (AOP)

AOP



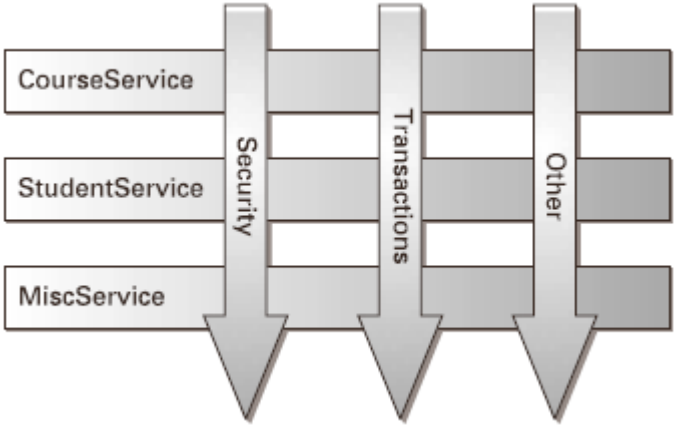
- Myšlenka

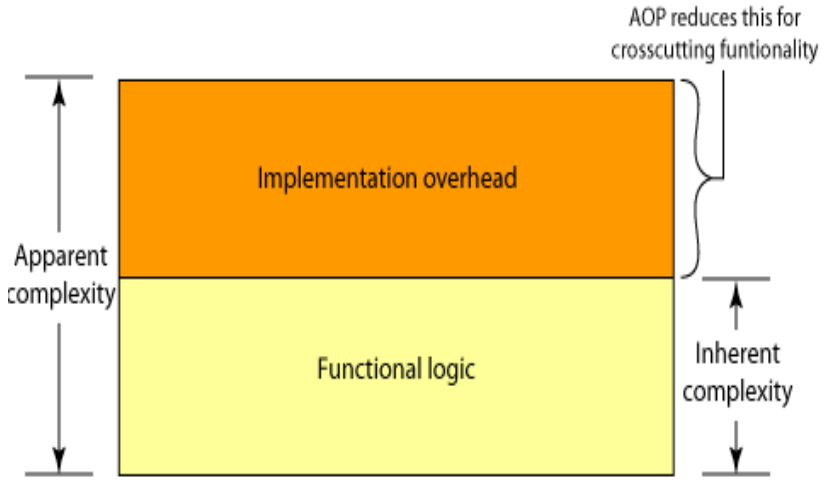
  „V každé aplikaci máme kousky kódu, které se nám prolínají všemi vrstvami naší aplikace, ale do žádné nepatří konkrétně. Těmto kouskům kódu můžeme říkat aspekty.

  AOP nabízí možnost tyto aspekty prolínat stávajícím kódem aniž bychom tento kód museli modifikovat.“

AOP

- Nový přístup pro řešení tradičních oblastí
  - implementační overhead
- Aspekt
  - Ucelený kus kódu
  - Java třída
  - Typy
    - Produkční
    - Vývojové



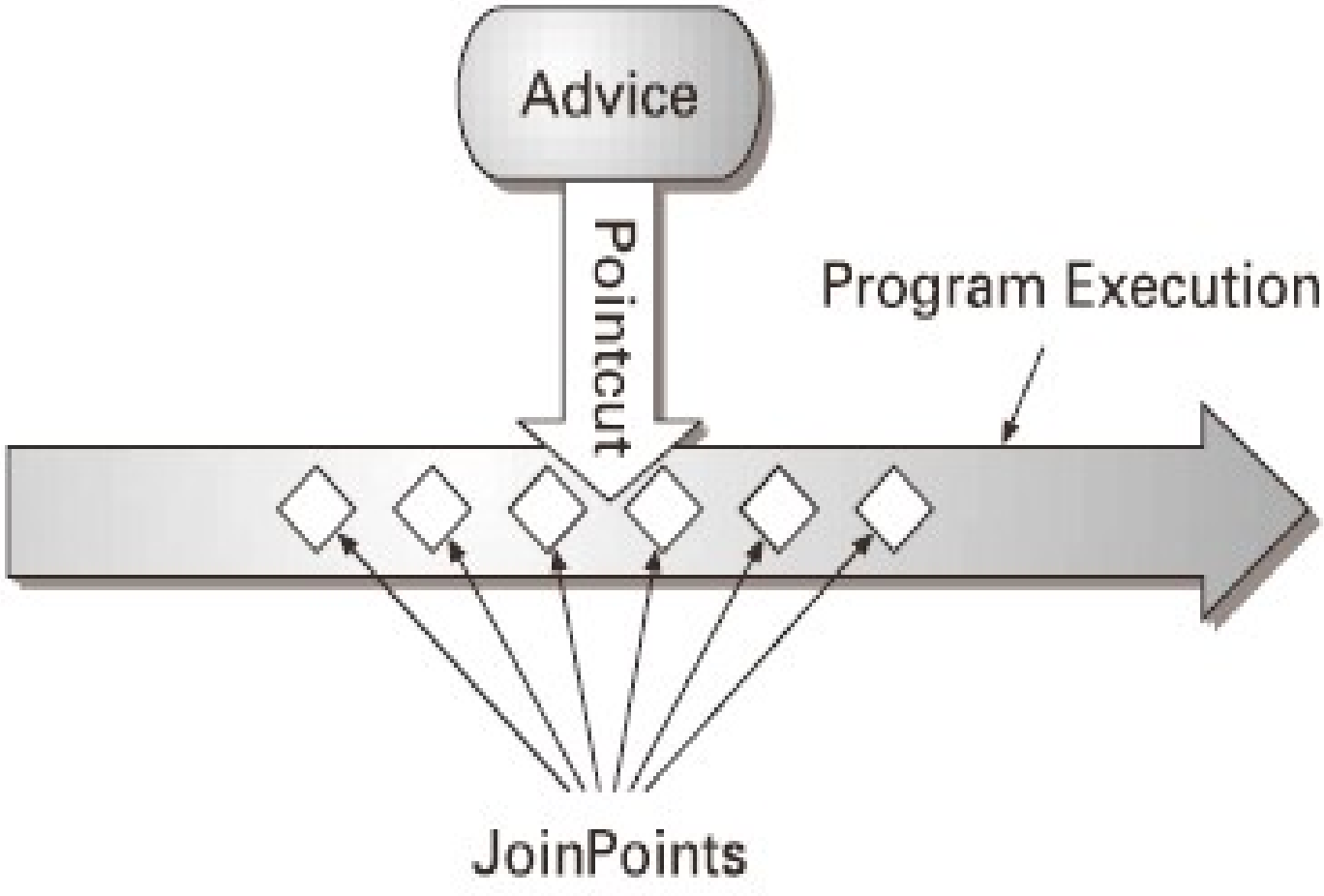


Základní pojmy

- Join Point
  - Definuje stavy, ve kterých se může nacházet vykonávání kódu např. volání metody, volání konstruktoru, statická inicializace atd.
- Pointcut
  - Definuje jazyk, pomocí kterého se lze na jednotlivé Join Ponty zavěsit
- Advice
  - Umožňuje nadefinovat vlastní kód, který se vykoná v závislosti na zavěšení pointcatu. V podstatě když nastane tohle, udělám toto.
- Weaving
  - začlenění aspectu do kódu aplikace
- Target object
  - cílový objekt, který je obohacen o aspekt

Aspect

- Aspect = Advice + Pointcut



Jednoduchý aspect (AspectJ)

anotace označující třídu jako aspect

```
@Aspect
public class ProfilingAspect {

    @Around("execution(* cz.sweb.pichlik.springaop.dao.UserStorageDao.*(..))")
    public Object tracingMethod(ProceedingJoinPoint call) throws Throwable{
        long start = System.nanoTime();
        try{
            return call.proceed();
        }finally{
            long time = System.nanoTime() - start;
            System.out.println("Invocation time:"+ time + "ns");
        }
    }
}
```

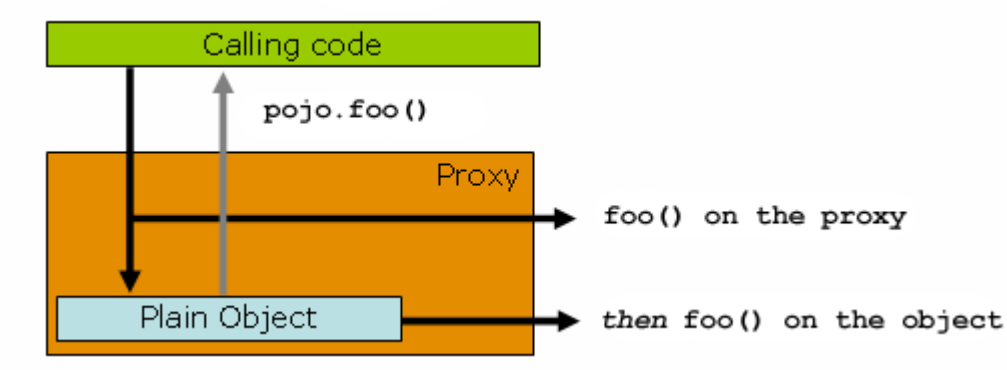
pointcut definice

volání metody target objektu

anotace označující advice

Spring AOP

- Integrace AspectJ
  - ne všechny vlastnosti (ořezaný pointcut language)
- Anotace, XML
- Runtime weaving (proxy)



AOP netradiční případy užití

- Synchronizace
  - Kontrola zámků
- Překlad výjimek
- Cacheování výsledků volání metod
- Introduction

Přístup do databáze

Problém jménem JDBC

- Využití JDBC se skládá z těchto činností
  - Získání databázového připojení
  - Vytvoření statementu
  - Nastavení parametru a exekuce SQL
  - Procházení resultsetu a zpracování dat
  - Uzavření resultsetu, statementu a datab. připojení
  - Ošetření `java.sql.SQLException`
- Typický kód pak vypadá následovně

JDBC klasika

```
Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;
try{
  con = getConnection();//ziskej databazove pripojeni
  String sql = "select count(*) from user";
  ps = con.prepareStatement(sql); //vytvor statement
  rs = ps.executeQuery(); //vykonej
  while(rs.next()){
    //zpracuj hodnoty
  }
}finally{ //a prosimte nezapomen vse uvolnit...
  rs.close();
  ps.close();
  con.close();
}
```

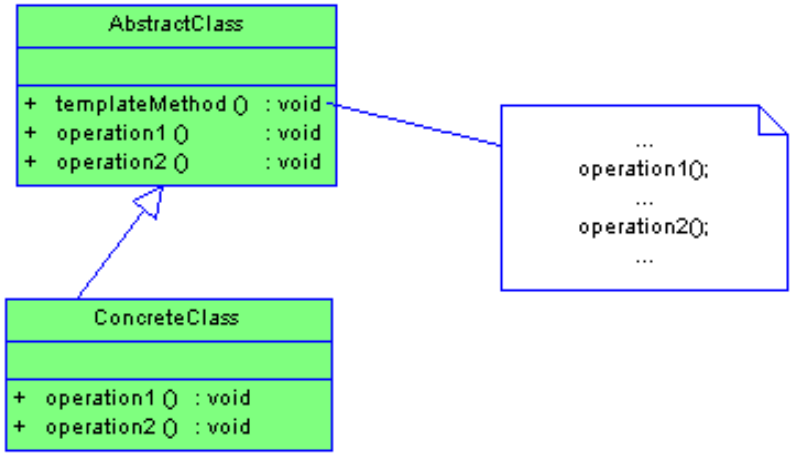
JDBC klasika

```
Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;
try{
  con = getConnection();
  String sql = "select count(*)from user";
  ps = con.prepareStatement(sql);
  rs = ps.executeQuery();
  while(rs.next()){
    //zpracuj hodnoty
  }
}finally{
  rs.close();
  ps.close();
  con.close();
}
```

- Co je špatně
  - Často se opakující kód
    - DP „Najdi a vlep“
  - Starat se o uzavírání zdrojů
  - Ošetření SQLException
  - Míchání odpovědností
    - Vykonání
    - Zpracování výsledků

Řešení

- společný kód je nadefinován v předkovi
  - potomek přepisuje pouze to co potřebuje
    - obvykle zpracování řádku ResultSetu



- Spring definuje pro JDBC účely třídu
  - `org.springframework.jdbc.core.JdbcTemplate`

JdbcTemplate – příklady užití

```
int rowCount = jdbcTemplate.queryForInt("select count(*) from user");

Actor actor = (Actor) jdbcTemplate.queryForObject(
    "select first_name, surname from t_actor where id = ?",
    new Object[]{new Long(1212)},
    new RowMapper() {
       public Object mapRow(ResultSet rs, int rn) throws SQLException {
            Actor actor = new Actor();
            actor.setFirstName(rs.getString("first_name"));
            actor.setSurname(rs.getString("surname"));
            return actor;
      }
    });
```

Spring DAO

- Abstrakce
  - Usnadňuje použití
    - JDBC, Hibernate, JPA, iBatis
  - Překlad SQLException na vlastní hierarchie výjimek
- Napojení na datasource
- Transakce
  - Programově
  - Deklarativně (XML, anotace)

Integrace dynamické jazyky

Proč dynamické jazyky

- Protože jsou dynamické oproti Jave
  - Deployment
  - MOP
  - Pestřejší syntaxe (DSL)
- čitelnější API
  - use-case orientované
  - XML, filesystem
- Vhodné pro určité typy úkolů
  - Testy, Frontend, Customizace, Prototypy...

Spring integrace

- Groovy, JRuby, BeanShell

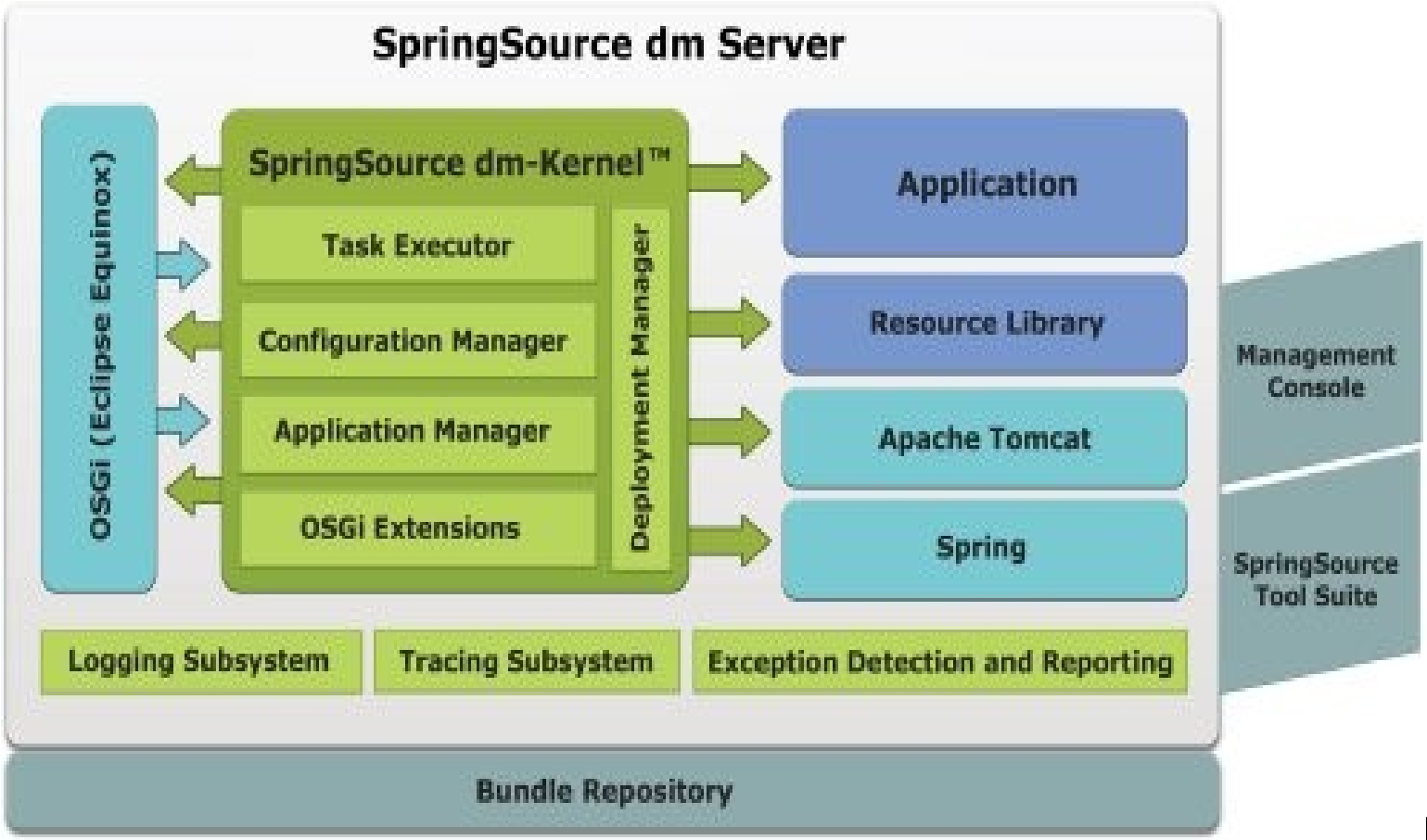
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-2.5.xsd">
    <!-- this is the bean definition for the Groovy-backed Messenger implementation -->
    <lang:groovy id="messenger" script-source="classpath:Messenger.groovy">
        <lang:property name="message" value="I Can Do The Frug" />
    </lang:groovy>
    <!-- an otherwise normal bean that will be injected by the Groovy-backed Messenger -->
    <bean id="bookingService" class="x.y.DefaultBookingService">
        <property name="messenger" ref="messenger" />
    </bean>
</beans>
```

Spring a vliv na J2EE

- J2EE 5
- EJB 3.1
- J2EE 6
  - Profily
  - SPI
- Budoucnost
  - OSGi

SpringSource dm Server

- Spring + Apache + OSGi



Díky za pozornost

Q&A