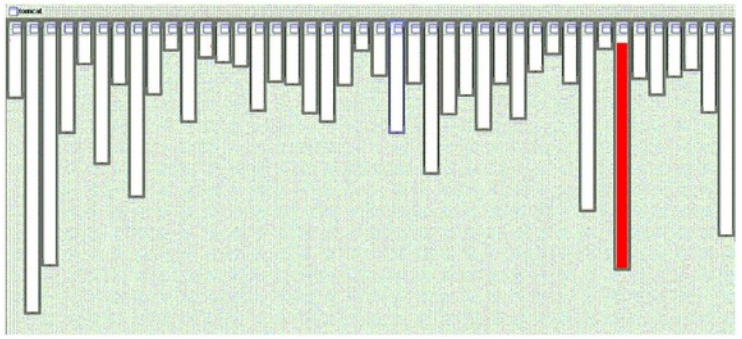
Aspekty (AOP)
AspectJ

AOP

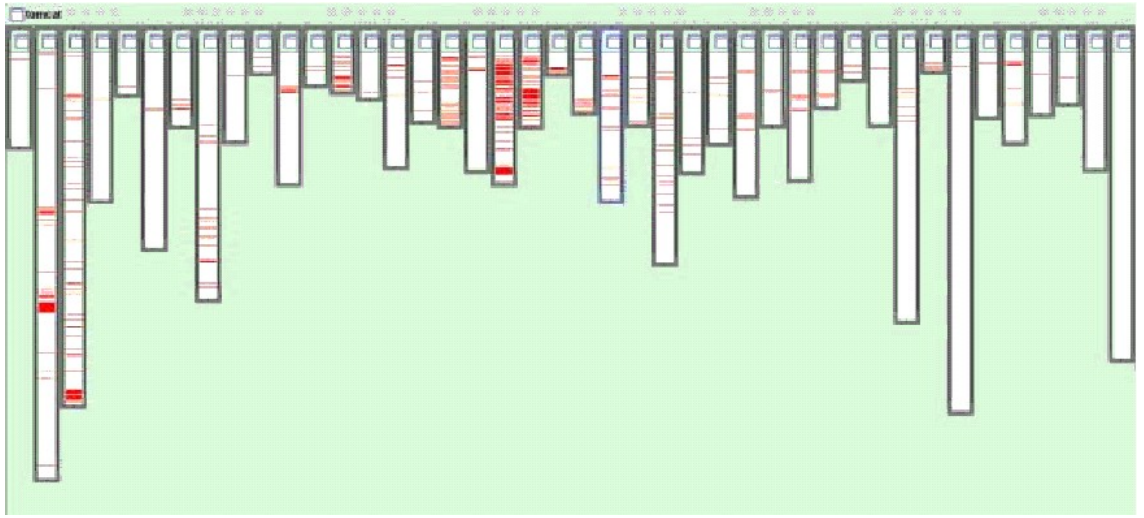
- Aspect-oriented programming
- „separation of concerns“
  - concern ~ část kódu programu související s nějakou funkčností

- většinou se chápe jako rozšíření OOP
- řeší problém, že ne vždy lze dat kód pro nějakou funkčnost do 1 nebo několika málo tříd
  - naopak je přes celou aplikaci

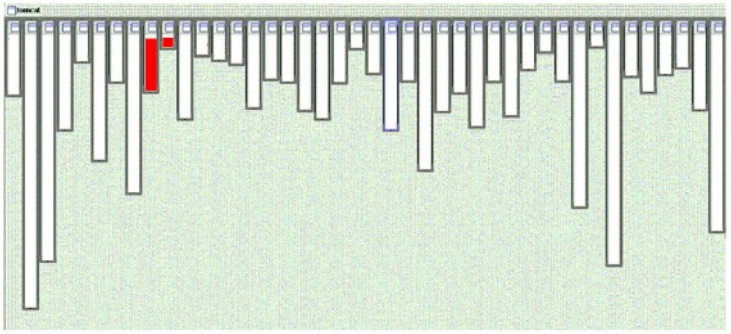
Modulárnost aplikací



parsování XML v Tomcatu



logování



práce s URL v Tomcatu

AspectJ

- http://www.eclipse.org/aspectj/
- rozšíření Javy
  - 1 koncept – **joinpoint**
    - místo v programu pro připojení kódu
  - několik konstrukcí
    - **pointcut**
      - definice joinpointu(ů)
    - **advice**
      - kód, který se má přidat
    - **inter-type declaration**
      - rozšíření deklarace třídy
    - **aspect**
      - „třída“ která může definovat výše zmíněné konstrukce

Pointcut

- call(void Point.setX(int))
- call(void Point.setX(int)) ||
  call(void Point.setY(int))
- call(void FigureElement.setXY(int,int)) ||
  call(void Point.setX(int)) || call(void Point.setY(int)) ||
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point))
- pointcut move():
  call(void FigureElement.setXY(int,int)) ||
  call(void Point.setX(int)) || call(void Point.setY(int)) ||
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));
- call(public * Figure.* (..))

Advice

- before(): move() {
  System.out.println("about to move");
  }

- after() returning: move() {
  System.out.println("just successfully moved");
  }

Inter-type declaration

- aspect PointObserving {
  private Vector Point.observers = new Vector();
  ...
  }

Aspect

```
aspect PointObserving {
    private Vector Point.observers = new Vector();
    public static void addObserver(Point p, Screen s) {
        p.observers.add(s);
    }
    public static void removeObserver(Point p, Screen s) {
        p.observers.remove(s);
    }
    pointcut changes(Point p): target(p) && call(void Point.set*(int));
    after(Point p): changes(p) {
        Iterator iter = p.observers.iterator();
        while ( iter.hasNext() ) {
            updateObserver(p, (Screen)iter.next());
        }
    }

    static void updateObserver(Point p, Screen s) {
        s.display(p);
    }
}
```

Aspect

- aspect SimpleTracing {
  pointcut tracedCall():
  call(void FigureElement.draw(GraphicsContext));

  before(): tracedCall() {
  System.out.println("Entering: " + thisJoinPoint);
  }
  }
- aspect SetsInRotateCounting {
  int rotateCount = 0;
  int setCount = 0;

  before(): call(void Line.rotate(double)) {
  rotateCount++;
  }

  before(): call(void Point.set*(int))
  && cflow(call(void Line.rotate(double))) {
  setCount++;
  }
  }

AspectJ

- aspekty lze definovat i přímo v Javě
  - pomocí anotací

```
@Aspect
public class Foo {

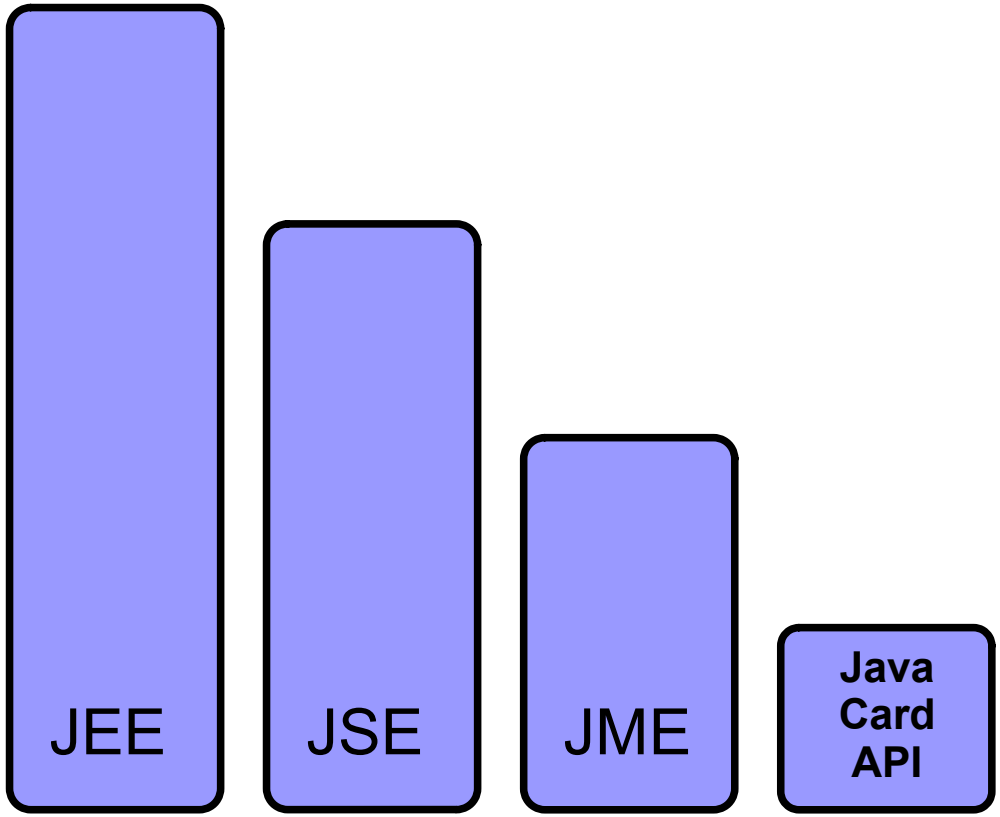
  @Pointcut("call(* *.*(..))")
  void anyCall() {}

  @Before("call(* org.aspectprogrammer..*(..))
                                        && this(Foo)")
  public void callFromFoo() {
  }

}
```

JEE
Java Enterprise Edition

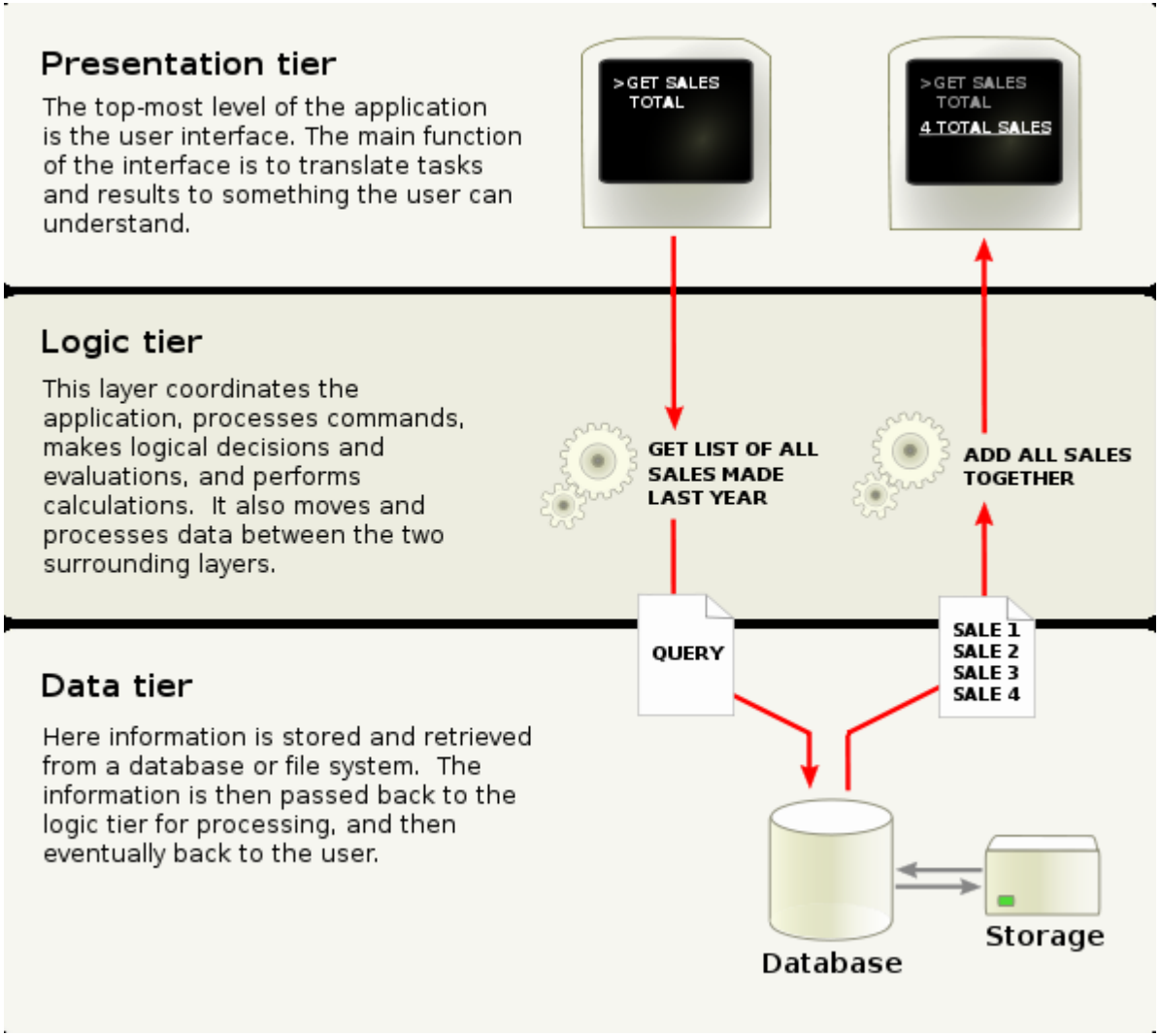
Přehled



„Enterprise“ aplikace

- „velké podnikové“ aplikace
- požadované vlastnosti
  - znovupoužitelnost
  - volné vazby
  - transakce
  - deklarativní rozhraní
  - persistence
  - bezpečnost
  - distribuované aplikace
  - ...

Třívrstvé architektura



zdroj obrázku: http://en.wikipedia.org/wiki/File:Overview_of_a_three-tier_application_vectorVersion.svg

JAVA

EJB
(nejdříve krátce EJB 2, tj. staré EJB)

Přehled

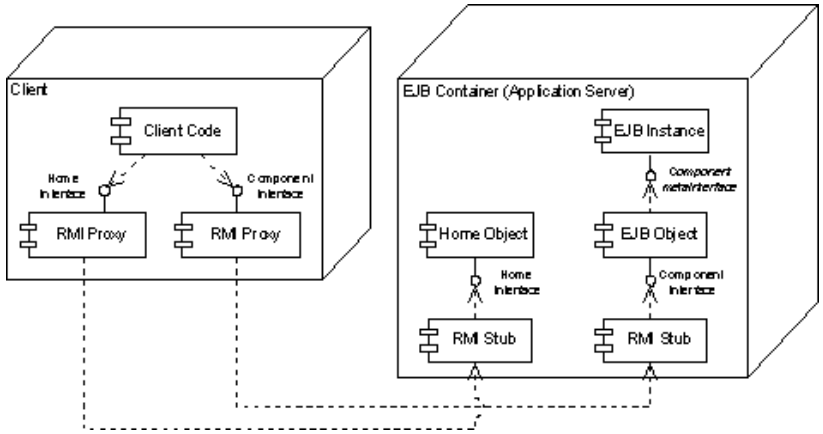
- Enterprise Java Beans
- komponenty
- běží na serveru
  - EJB kontejner
- lokální i vzdálený přístup
- kontejner poskytuje množství služeb
  - persistence
  - bezpečnost
  - transakce
  - „scalability“
  - „concurrency“

EJB

- typy bean
  - session beans – implementují business logiku (logic tier), nejsou persistentní
    - stateless – bezstavové
    - statefull – udržují stav
  - message-driven beans
    - implementují předepsaný interface
      - MessageListener – onMessage()
  - entity beans – přístup k persistentním datům
    - persistence
      - container managed
      - bean managed
- deployment descriptor
- EAR

EJB

- mnoho problémů
  - povinnost vytváření několika interfaců a tříd
    - třídy musely mít stejné metody ale neimplementovaly interfacy
    - EJB kontejner „sváže“ interface a implementaci
      - vygeneruje stuby a proxy
  - nutnost mnoha descriptorů
  - ...



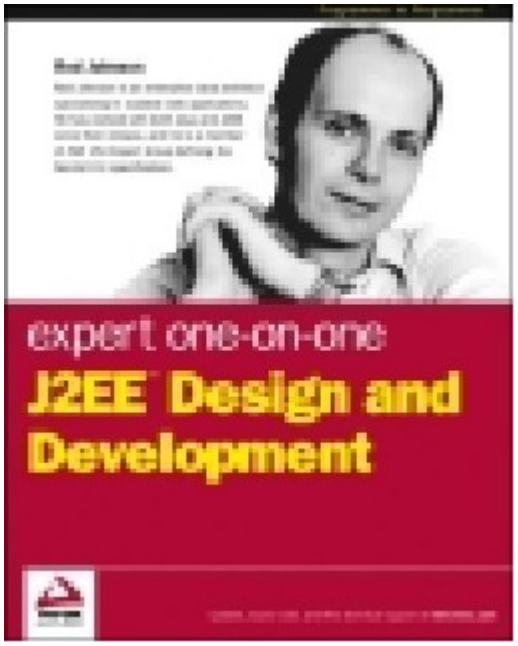
zdroj obrázku: B.Eckel: Thinking in Enterprise Java

JAVA

Spring

Přehled

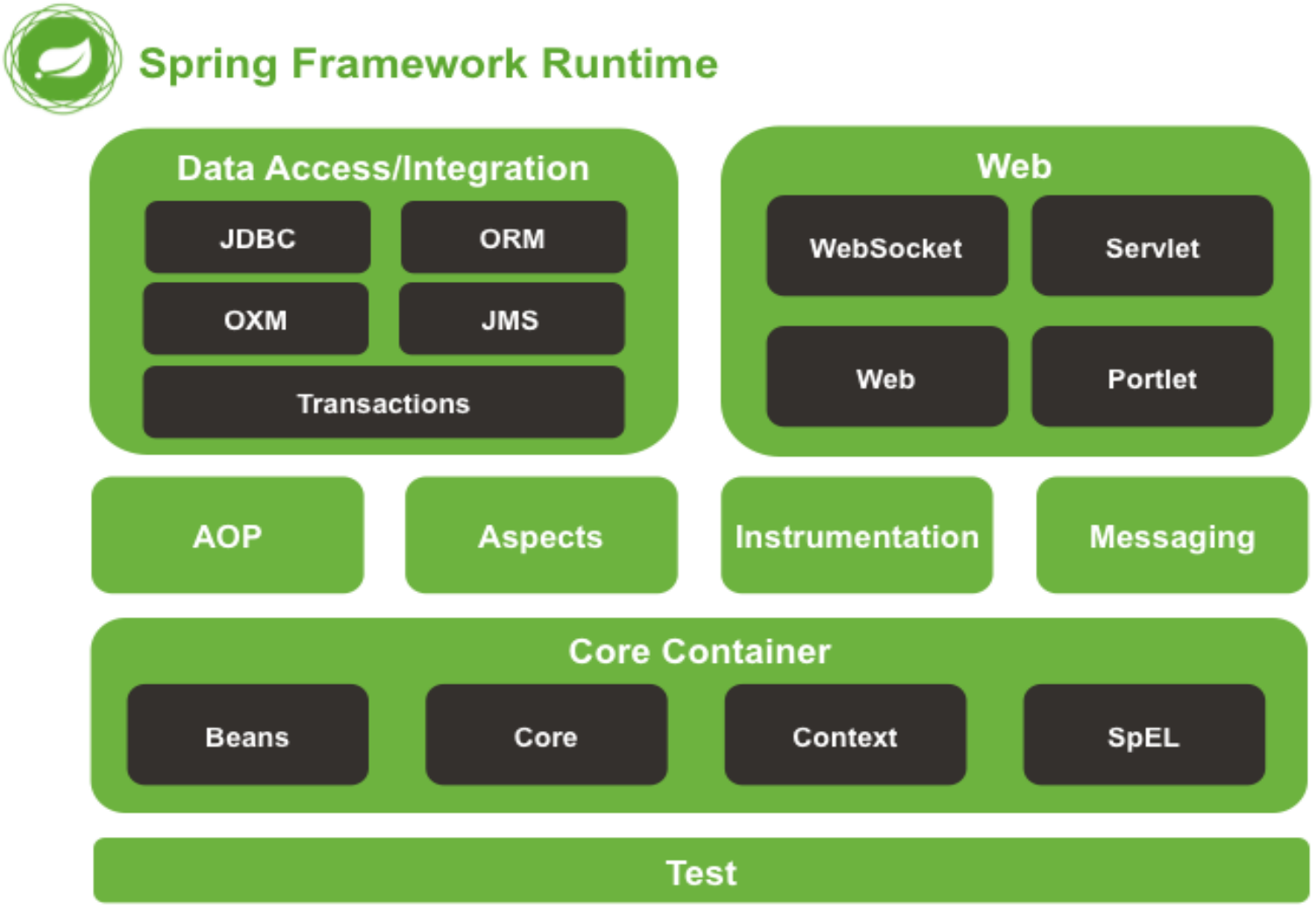
- 2002
- kritika EJB
  - příliš složité
  - těžko použitelné
  - těžko testovatelné
  - všude RemoteException
  - …
- Rod Johnson: Expert One-on-One J2EE Design and Development
  - kritika EJB +
    návrh lepší architektury
    - vyvinul se z ní Spring



Přehled

- Spring
  - http://www.spring.io/
  - založeno na POJO
    - plain old Java objects
    - ale lze integrovat s EJB
  - „lehké“ řešení
    - co nejmenší závislosti aplikačního kód na Springu
    - není nutný žádný server
      - použitelné pro jakýkoliv typ aplikací
  - snaha o integraci s dalšími frameworky
    - „neobjevovat kolo“
    - používat úspěšná existující řešení

Architektura



Spring core

- balíček org.springframework.beans
- „inversion of control“ kontejner
  - Dependency Injection
  - Hollywood Principle: "Don't call me, I'll call you."

- objekty se nepropojují v kódu, ale v konfiguračním souboru
- objekt není zodpovědný za hledání svých závislostí
- závislosti nadeklarovány
  - kontejner je ”dodá” – nastaví konkrétní objekty pomocí setterů
    - obvyklá jmenná konvence setXxx()
    - nebo přes parametry konstruktoru
- žádné speciální požadavky na objekty

Spring core

- vytváření objektu pomocí „factory“
  - interface org.springframework.beans.factory.BeanFactory
  - mnoho implementací

Spring core – příklad

```
public class nameBean {
    String name;

    public void setName(String a) {
        name = a;
    }

    public String getName() {
        return name;
    }
}
```

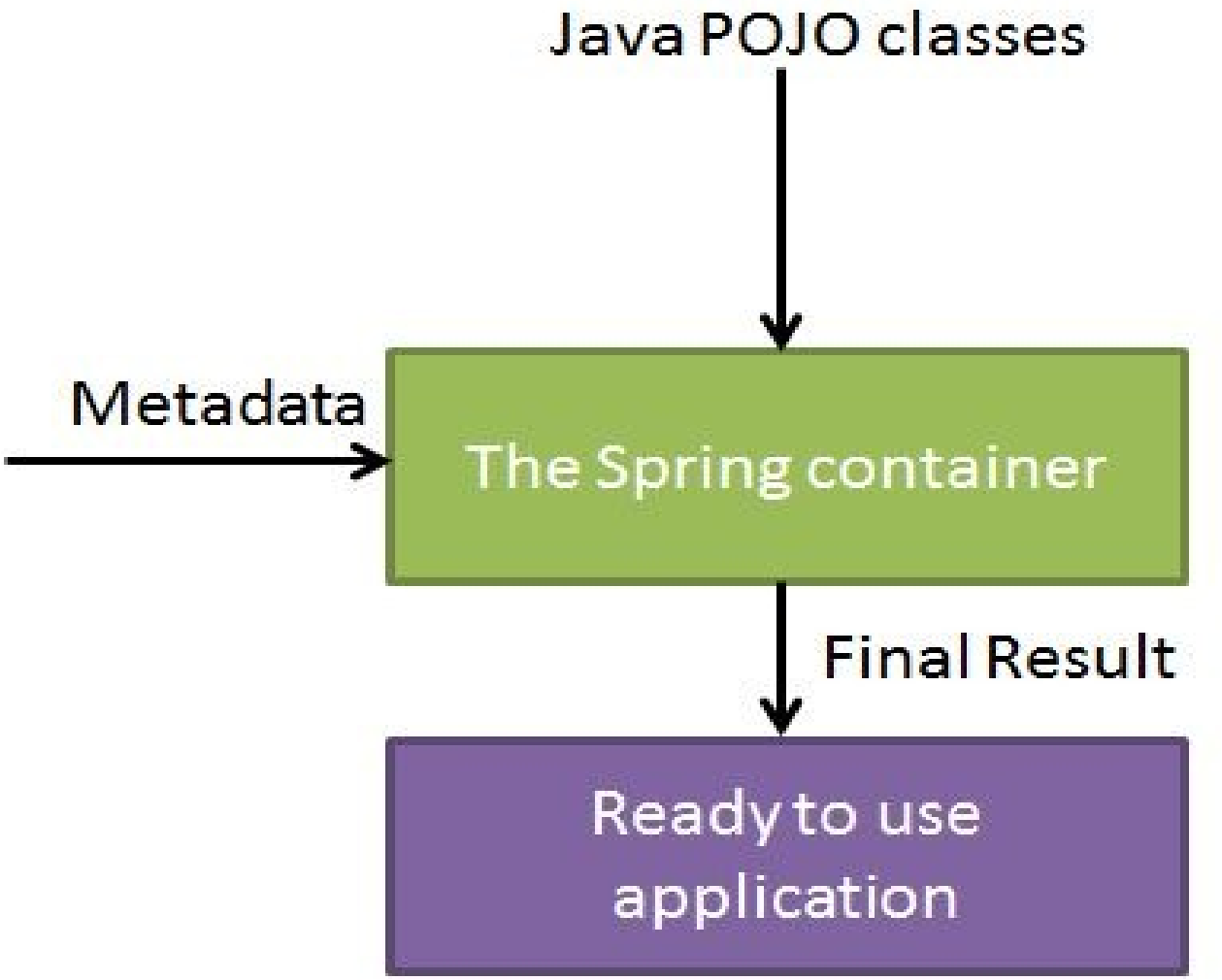
```
<bean id="bean1" class="nameBean">
  <property name="name" >
     <value>Tom</value>
  </property>
</bean>
```

- propojení objektů

```
<bean id="bean" class="beanImpl">
  <property name="conn">
     <ref bean="bean2"/>
  </property>
</bean>

<bean id="bean2" class="bean2impl"/>
```

Spring core


Java POJO classes
Metadata
The Spring container
Final Result
Ready to use application

Spring a datová vrstva

- lze používat cokoliv
  - JDBC
  - ORM
    - Hibernate
    - …
- lze používat samostatně
  - zjednodušuje používání DB
  - jednotné výjimky
  - ...

Spring a datová vrstva

- JdbcTemplate template = new JdbcTemplate(dataSource);
  List names = template.query("SELECT USER.NAME FROM USER",
  new RowMapper() {
  public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
  return rs.getString(1);
  }
  });

- int youngUserCount = template.queryForInt("SELECT COUNT(0) FROM USER WHERE USER.AGE < ?", new Object[] { new Integer(25) });

- class UserQuery extends MappingSqlQuery {
  public UserQuery(DataSource datasource) {
  super(datasource, "SELECT * FROM PUB_USER_ADDRESS
  WHERE USER_ID = ?");
  declareParameter(new SqlParameter(Types.NUMERIC));
  compile();
  }
  protected Object mapRow(ResultSet rs, int rownum) throws SQLException{
  User user = new User();
  user.setId(rs.getLong("USER_ID")); user.setForename(rs.getString("FORENAME"));
  return user; }
  public User findUser(long id) { return (User) findObject(id); }
  }
  User user = userQuery.findUser(25);

Spring AOP

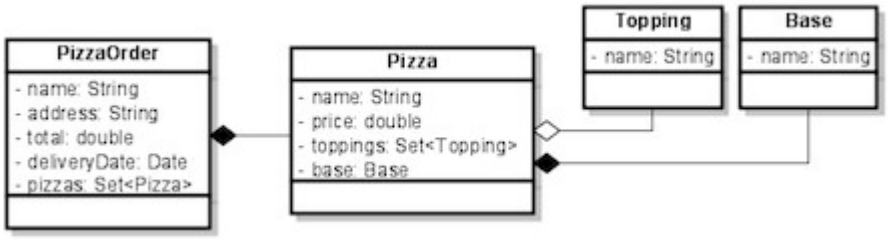
- implementováno v čisté Javě
  - lze integrovat s AspectJ
- určeno pro tu funkčnost, na kterou se aspekty hodí
  - původně pro přidaní JEE služeb do Springu
  - transakce
  - logování
  - ...

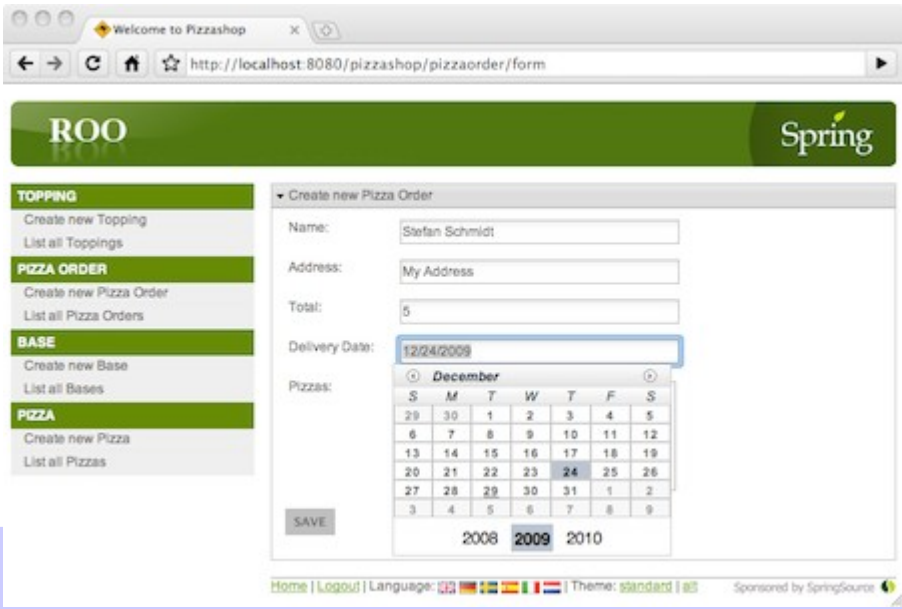
Další Spring součásti

- Spring MVC
  - web MVC framework
  - inspirováno frameworkem Struts
  - nepředepisuje, co použít pro generování stránek
    - JSP
    - šablonovací systémy (Velocity,...)
    - …
- EJB
  - místo POJO lze používat EJB
- ...

Spring Roo

- framework pro snadné „generování“ enterprise aplikací
  - zjednodušeně
    vytvoření aplikace pomocí „průvodce“ v několika krocích





JAVA

EJB 3

Přehled

- inspirace Springem
- místo implementování interfaců jsou anotace
- používání „dependency injection“
- odstranění nutnosti deskriptorů
- …
- entity beans nahrazeny Java Persistence API
  - „mapování“ tříd na tabulky v relační databázi
  - JPQL dotazovací jazyk
    - „SQL nad objekty“

Session bean – příklad

```
@Remote
public interface Converter {
  public BigDecimal dollarToYen(BigDecimal dollars);
}

@Stateless
public class ConverterBean implements converter.ejb.Converter {
  private BigDecimal euroRate = new BigDecimal("0.0070");

  public BigDecimal dollarToYen(BigDecimal dollars) {
    BigDecimal result = dollars.multiply(yenRate);
    return result.setScale(2, BigDecimal.ROUND_UP);
  }
}
```

Message-driven bean – příklad

```
@MessageDriven(mappedName="MDBQueue")
public class MDB implements MessageListener {
  public void onMessage(Message msg) {
    System.out.println("Got message!");
  }
}
```

Entity – příklad

```
@Entity
@Table(name = "phonebook")
public class PhoneBook implements Serializable {
  @Column(name="number") private String number;
  @Column(name="name") private String name;

  public PhoneBook() {}

  public PhoneBook(String name, String number) {
    this.name = name;
    this.number = number;
  }

  @Id public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public String getNumber() { return number; }
  public void setNumber(String number) { this.number = number; }
}
```

JPQL

- inspirováno HQL
  - podmnožina HQL

- SELECT ... FROM ...
  [WHERE ...]
  [GROUP BY ... [HAVING ...]]
  [ORDER BY …]
- DELETE FROM ... [WHERE ...]
- UPDATE ... SET ... [WHERE …]

- SELECT a FROM Author a ORDER BY a.firstName, a.lastName
- SELECT DISTINCT a FROM Author a INNER JOIN a.books b WHERE b.publisher.name = 'MatfyzPress'

JAVA

Hibernate

Architektura

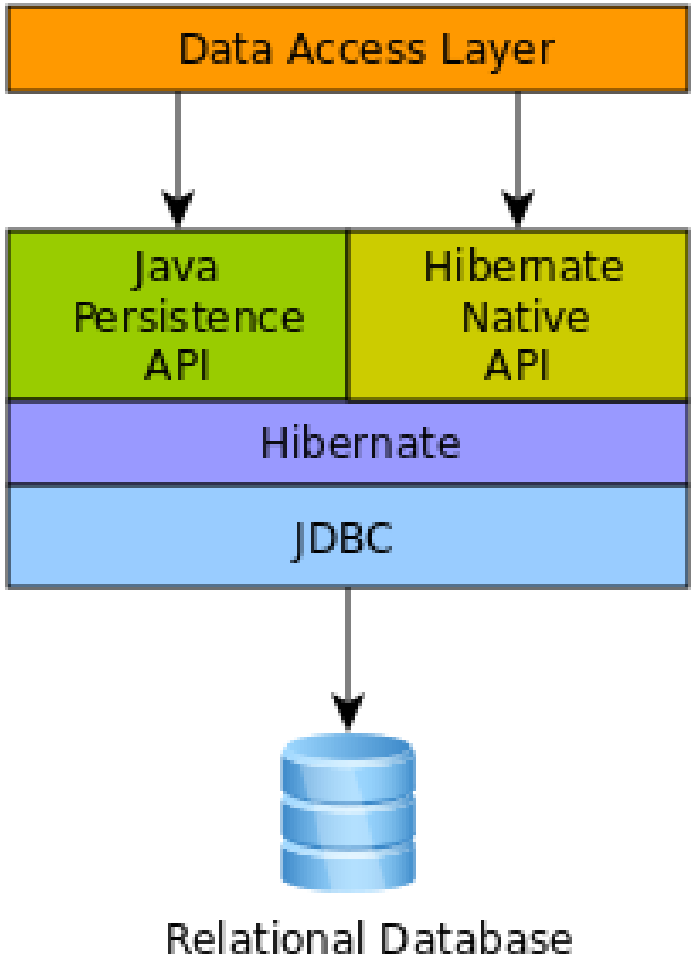


image source: http://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html

Základní API

- Session
  - propojení mezi DB a aplikací
  - „schovává“ v sobě spojení do DB
    - JDBC connection
  - spravuje objekty
    - obsahuje cache objektů
- SessionFactory
  - „tvůrce“ session
  - obsahuje mapovaní mezi objekty a DB
  - může obsahovat cache objektů
- persistentní objekty
  - obyčejné Java objekty
    - POJO/JavaBeans
  - měly by dodržovat pravidla pro JavaBeans
    - ale není to nutné

Použití

- zjednodušeně
  - vytvořit konfiguraci
    - XML
  - vytvořit třídy
    - Java
  - vytvořit mapovaní
    - XML nebo
    - Java anotace

Konfigurace

- XML soubor
- definuje
  - připojení do DB
  - typ (dialekt) DB
  - kde je mapovaní
  - …

```
<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">org.h2.Driver</property>
        <property name="connection.url">jdbc:h2:mem:db1;DB_CLOSE_DELAY=-1;MVCC=TRUE</property>
        <property name="connection.username">sa</property>
        <property name="connection.password"/>

        <property name="connection.pool_size">1</property>

        <property name="dialect">org.hibernate.dialect.H2Dialect</property>

        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

        <property name="show_sql">true</property>

        <property name="hbm2ddl.auto">create</property>

        <mapping resource="org/hibernate/tutorial/hbm/Event.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Třídy pro persistentní data

- POJO
- měly by dodržovat pravidla pro JavaBeans
  - není nutné
- je potřeba konstruktor bez parametrů
  - jeho viditelnost je libovolná

```
public class Event {
  private Long id;
  private String title;
  private Date date;

  public Event() {}

  public Event(String title, Date date) {
    this.title = title;
    this.date = date;
  }

  public Long getId() { return id; }
  private void setId(Long id) {  this.id = id;  }

  public Date getDate() {  return date;  }
  public void setDate(Date date) {  this.date = date;  }

  public String getTitle() {  return title;  }
  public void setTitle(String title) {  this.title = title; }
}
```

Mapování

- XML soubor
- mapování atributů třídy na sloupce
- definuje se
  - jméno
  - typ
    - není nutný, pokud je „zřejmý“
    - Hibernate typy
      - nejsou to ani Java ani SQL typy
      - jsou to „převodníky“ mezi Java a SQL typy
  - sloupec
    - není nutný pokud je stejný jako jméno

```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="increment"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>
</hibernate-mapping>
```

Mapování

```
@Entity
@Table( name = "EVENTS" )
public class Event {
  private Long id;
  private String title;
  private Date date;

  public Event() { }

  public Event(String title, Date date) {
    this.title = title;
    this.date = date;
  }

  @Id
  @GeneratedValue(generator="increment")
  @GenericGenerator(name="increment", strategy = "increment")
  public Long getId() {  return id;  }

  private void setId(Long id) {  this.id = id; }

  @Temporal(TemporalType.TIMESTAMP)
  @Column(name = "EVENT_DATE")
  public Date getDate() { return date; }

  public void setDate(Date date) {  this.date = date; }

  public String getTitle() { return title; }
  public void setTitle(String title) { this.title = title; }
}
```

- mapování lze i pomocí anotací
- v konfiguraci je pak v mapování přímo odkaz na třídu

Používání

- ```
  SessionFactory sessionFactory =
          new Configuration().configure().buildSessionFactory();
  ```

- ```
  Session session = sessionFactory.openSession();
  session.beginTransaction();
  session.save(new Event("Our very first event!", new Date()));
  session.save(new Event("A follow up event", new Date()));
  session.getTransaction().commit();
  session.close();
  ```

- ```
  List result = session.createQuery( "from Event" ).list();
  ```
  ```

Stavy objektů

- Transient
  - vytvořený objekt (new)
  - ale ještě neasociovaný s Hibernate session
- Persistent
  - objekt asociovaný se session
    - vytvořený a pak uloženy nebo načtený
- Detached
  - perzistentní objekt jehož session byla skončena
  - lze asociovat s novou session

Používání objektů

- načítání
  - sess.load(Event.class, new Long(id));
    - při neexistenci vyhazuje výjimku
    - nemusí nutně sahat ihned do DB
  - sess.get(Event.class, new Long(id));
    - při neexistenci vrací null
- dotazování
  - sess.createQuery(...).list()
- modifikování objektů
  - Event e = sess.load(Event.class, new Long(69));
    e.set...
    sess.flush();

Používání objektů

- modifikace „odpojených“ objektů
  - Event e = sess.load(Event.class, new Long(69));
    e.set...
    ...
    secondSess.update(e);
- mazání objektů
  - sess.delete(e);

Dotazování

- HQL – Hibernate query language
  - obdoba SQL

```
select foo
from Foo foo, Bar bar
where foo.startDate = bar.date
```

- lze používat i nativní SQL

```
sess.createSQLQuery("SELECT * FROM CATS").list();
```

Hibernate...

- další součásti
  - vytváření tříd podle tabulek
  - podpora pro full-text vyhledávání
  - verzování objektů
  - validace objektů
  - podpora JPA (Java Persistence API)
  - ...