



**profinit.**  
PROFESSIONALS IN IT

## Spring portfolio

**Tomáš Krátký**

[tomas.kratky@profinit.eu](mailto:tomas.kratky@profinit.eu)

<http://www.profinit.eu>



# Obsah

- Když se řekne Spring ...
- Představení ukázkové aplikace
- Využití Spring portfolia
  - Spring Framework (core)
  - Spring Web Services
  - Spring Integration
  - Spring Batch
  - Spring Web Flow
  - Spring Modules
- Co se sem nevešlo (Spring DM, Security, ...)

# Když se řekne Spring ...



Spring



Spring Web Flow



Spring





# Co je Spring ?

- Spring je Lightweight *Application* Framework
- Spring je cílen na všechny aplikační vrstvy
- Spring poskytuje „plumbing code“





# Trocha historie

## Datum narození

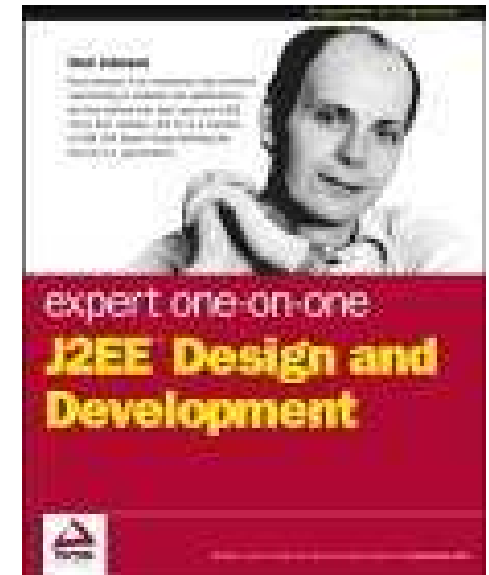
- 2002/2003

## Rodičové

- Rod Johnson a Juergen Holler

## Okolnosti narození

- kniha Expert One-on-One J2EE Design and Development
- V březnu 2004 uvolněn Spring 1.0
- V letech 2004 / 2005 se Spring stal přední JSE / JEE aplikační platformou



Spring



# Spring Mission Statement

**Autoři Springu věří, že:** (viz [springframework.org](http://springframework.org))

- J2EE should be easier to use.
- It is best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero.
- JavaBeans offer a great way of configuring applications.
- OO design is more important than any implementation technology, such as J2EE.
- Checked exceptions are overused in Java. A platform shouldn't force you to catch exceptions you're unlikely to be able to recover from.
- Testability is essential, and a platform such as Spring should help make your code easier to test.

# Spring Mission Statement

**Autoři Springu se zaměřují na:** (viz [springframework.org](http://springframework.org))

- Spring should be a pleasure to use
- Your application code should **not** depend on Spring APIs
- Spring should not compete with good existing solutions, but should foster integration.





# Existující projekty

- Spring Framework
- Spring Web Flow
- Spring Web Services
- Spring Batch
- Spring Integration
- Spring Modules
- Spring Security (Acegi Security)
- Spring Dynamic Modules
- SpringSource dm Server
- Spring LDAP, IDE, JavaConfig, Rich Client, ...

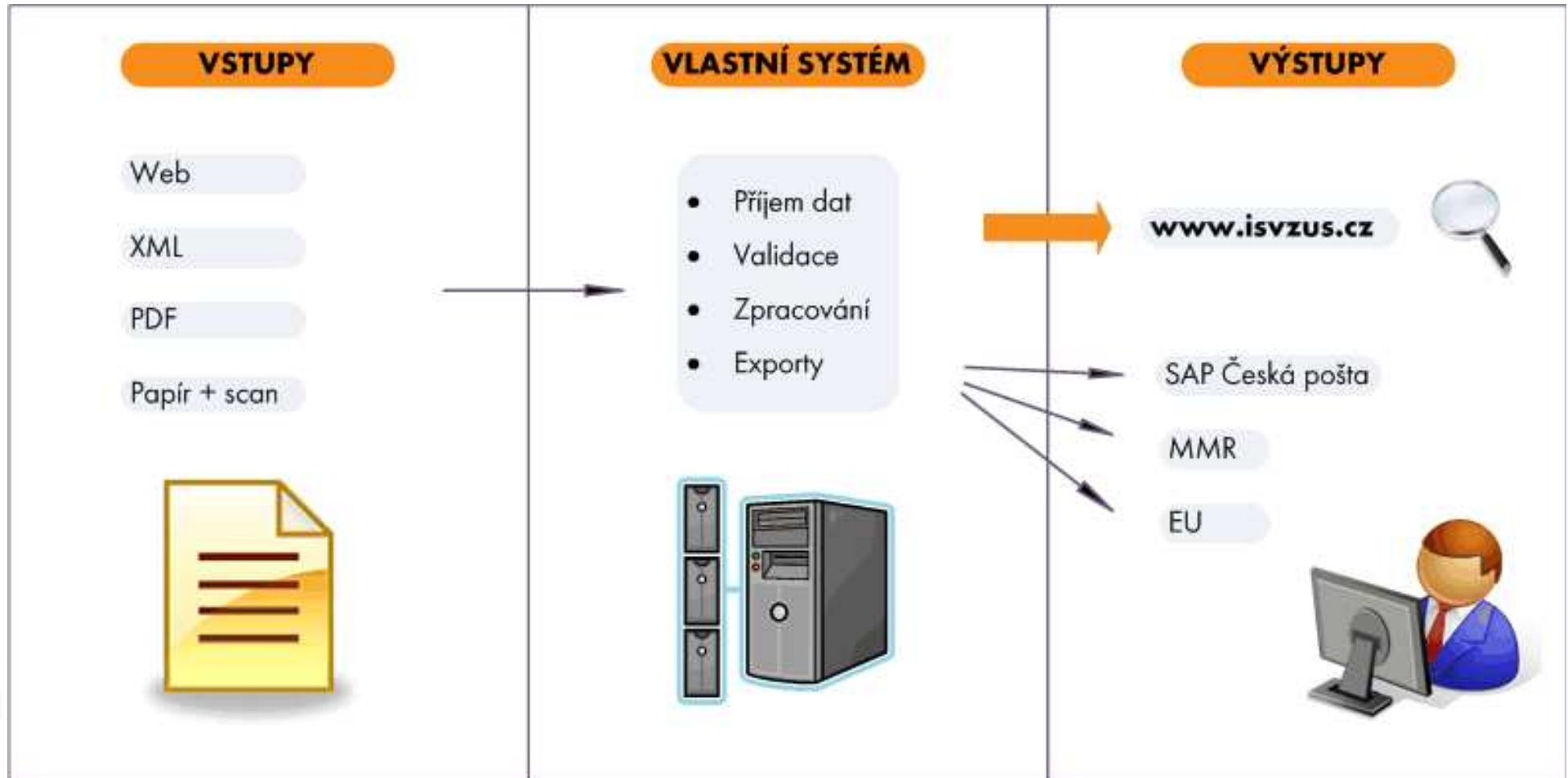


# Představení ukázkové aplikace





# Agenda





# Charakteristiky

- Rozsah cca 500 KSLOC
- Pracnost cca 2000 md
- Ve špičce kolem 10 lidí
- Denně 100 zveřejněných novinek
- 30 000 přístupů týdně
  - Z toho většina ve špičce 7 až 10 h





# Základem všeho je Spring framework



# Co je Spring framework ?

## Spring Features

Spring is a layered Java/J2EE application platform, based on code published in **Expert One-on-One J2EE Design and Development**

Spring includes:

- **The most complete lightweight container**, providing centralized, automated configuration and wiring of your application object system from a set of loosely-coupled components (POJOs) in a consistent and transparent fashion. The container brings agility and allows software components to be first developed and tested in isolation, then scaled up for deployment in any environment (J2EE or non-J2EE).
- **A common abstraction layer for transaction management**, allowing for pluggable transaction managers, and making it easy to use. Generic strategies for JTA and a single JDBC DataSource are included. In contrast to plain JTA or EJB CMT, Spring's transaction support is more flexible and easier to use.
- **A JDBC abstraction layer** that offers a meaningful exception hierarchy (no more pulling vendor codes out of SQLException), simplifying error handling. You'll never need to write another finally block to use JDBC again. The JDBC-oriented exceptions comply to Spring's standard exception hierarchy.
- **Integration with Toplink, Hibernate, JDO, and iBATIS SQL Maps**: in terms of resource holders, DAO implementation support, and IoC convenience features, addressing many typical Hibernate integration issues. All of these comply to Spring's generic transaction management.
- **AOP functionality**, fully integrated into Spring configuration management. You can AOP-enable any object managed by Spring, including beans. In Spring, you can have declarative transaction management without EJB... even without JTA, if you're using a single database in Tomcat.
- **A flexible MVC web application framework**, built on core Spring functionality. This framework is highly configurable via strategies like Velocity, Tiles, iText, and POI. Note that a Spring middle tier can easily be combined with a web tier based on any other web MVC framework.

You can use all of Spring's functionality in any J2EE server, and most of it also in non-managed environments. A central focus of Spring is to be tied to specific J2EE services. Such objects can be reused across J2EE environments (web or EJB), standalone applications, test environments, etc.

Spring's layered architecture gives you a lot of flexibility. All its functionality builds on lower levels. So you can e.g. use the JavaBeans support. But if you use the web MVC framework or AOP support, you'll find they build on the core Spring configuration, so you can apply

# Co je Spring framework ?


- Velmi zjednodušeně řečeno
  - IoC container
  - AOP framework
  - Service abstraction layer pro mnoho API třetích stran



- Spring ve své podstatě **pomáhá poskládat aplikaci dohromady** a usnadňuje integraci s dalšími technologiemi



# Jak může Spring pomoci

- Přináší konzistentní strukturu pro celou aplikaci
  - Poskytuje konzistentní přístup ke skládání jednotlivých částí aplikace dohromady
  - Poskytuje elegantní integrační body se standardními technologiemi jako JPA, Hibernate, EJB, ...
  - Díky tomu lze očekávat nárůst produktivity
- 



# IoC v akci

```
public class HlidaniZmenRunnerImpl implements HlidaniZmenRunner {
    private MessageSource msgSource;
    private EmailService emailService;
    private HlidaniZmenResultConvertor convertor;
    private HlidaniZmenRepository repository;

    public void setMessageSource( MessageSource s ) { this.msgSource = s; }
    public void setEmailService( EmailService s ) { this.emailService = s; }
    public void setResultConvertor( HlidaniZmenResultConvertor c ) { this.convertor = c; }
    public void setHlidaniZmenRepository( HlidaniZmenRepository r ) { this.repository = r; }

    public void runHlidaniZmen( HlidaniZmen h, Timestamp aktualniSpusteni ) {
        ...
        for ( HlidanaZakazka z : h.getHlidaneZakazky() ) {
            HlidanaZakazka aktualni = repository.findLastChangeForZakazka( ... );
            ...
        }
        ...
    }
}
```



# IoC v akci

```
<bean id="hlidaniZmenRunner"  
  class="cz.profinit.kos3.model.hlidanizmen.HlidaniZmenRunnerImpl">  
  <property name="hlidaniZmenRepository" ref="hlidaniZmenRepository" />  
  <property name="resultConvertor" ref="resultConvertor" />  
  <property name="messageSource" ref="messageSource" />  
  <property name="emailService" ref="emailService" />  
</bean>
```





# AOP v akci

```
@Transactional (  
    readOnly = true,  
    propagation = Propagation.REQUIRED,  
    isolation = Isolation.DEFAULT,  
    rollbackFor = Throwable.class )  
public Aktualita findDetailAktualita( long idAktuality ) {  
    return repository.findDetailAktualita(idAktuality);  
}
```

```
<bean id="txManager"  
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
    <property name="dataSource" ref="dataSource"/>  
</bean>
```

```
<tx:annotation-driven transaction-manager="txManager" proxy-target-class="false"/>
```



# AOP v akcii

@Aspect

```
public class LogAccessChecking {
```

```
    private static final Logger LOG = ...
```

```
    @AfterReturning(
```

```
        pointcut="target(AccessChecker) and execution(public * checkAccess(..))",  
        returning="accessType" )
```

```
    public void doLogAccessCheckerResult( JoinPoint jPoint, Object accessType ) {
```

```
        AccessType t = (AccessType) accessType;
```

```
        if ( t != null && t == AccessType.NONE && LOG.isDebugEnabled() ) {
```

```
            Object checkTarget = jPoint.getTarget();
```

```
            Object[] checkArgs = jPoint.getArgs();
```

```
            LOG.debug( ... );
```

```
        }
```

```
    }
```

```
}
```



# Abstraction v akci

```
class AktualityRepositoryImpl extends JdbcDaoSupport implements AktualityRepository {  
  
    private AktualitaRowMapper aktualitaMapper;  
  
    public Aktualita findDetailAktualita( long idAktuality ) {  
        return (Aktualita) getJdbcTemplate().queryForObject(  
            "{call sp_ctiAktuality ?, ?}",  
            new Object[] { idAktuality, null },  
            new int[] { Types.DECIMAL, Types.TIMESTAMP },  
            aktualitaMapper );  
    }  
}
```

- Podobné zjednodušení při práci s emailem, EJB, ...



# Spring Web Services





# Klasický přístup

## Code first (Contract Last)

- Vytvořit POJO service v Javě
- Vygenerovat WSDL pomocí XFire, CXF, ...
- Klienti používají vygenerované WSDL
  
- Závislost na způsobu generování WSDL
- `java.util.Map` nelze v XSD
- Může narušit zapouzdření



# Bolestivý přístup

## Contract first

- Vytvořit WSDL
- Vygenerovat či napsat kostru a implementaci v Javě
- Příliš složité
- Příliš mnoho zbytečného kódu

```
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:wadl="http://example.com/stockquote.wadl"
  xmlns:soap="http://schemas.xmlsoap.org/soap/encl/"
  xmlns="http://schemas.xmlsoap.org/wdl/">

  <complexType base="http://www.w3.org/2001/XMLSchema" >
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="TradePrice">
      <complexType>
        <all>
          <element name="price" type="float"/>
        </all>
      </complexType>
    </element>
  </definitions>

  <message name="GetLastTradePriceInput">
    <part name="body" element="wdl:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="wdl:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>My first service</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>
</definitions>
```



# Přístup Spring Web Services

- Contract first ...
- ... ale ne WSDL
- Většinou je podstatný formát dat
- ... který je definován pomocí XSD
  
- Spring Web Services mi umožní soustředit se na XSD pro data, zbylý SOAP balast se generuje



# XSD v akci – doménové typy

```
<schema ...>  
  <complexType name="FormInfo">  
    <sequence>  
      <element name="podaciCislo" type="string" />  
      <element name="znackaFormulare" type="string" />  
      ...  
      <element name="datumZverejneni" type="date" />  
    </sequence>  
  </complexType>
```



# XSD v akci – Request, Response

```
<element name="formInfoRequest">
  <complexType>
    <sequence>
      <element name="podaciCislo" type="string" />
    </sequence>
  </complexType>
</element>

<element name="formInfoResponse">
  <complexType>
    <sequence>
      <element name="result" type="tsn:FormInfo" />
    </sequence>
  </complexType>
</element>

</schema>
```



# Obsluha messages

- Object / XML mapper (OXM)
  - JAXB, XStream, ...
- Request a Response Java objekty
- Jednoduché
- Omezující je deserializace z XML

## @Endpoint

```
public class FormInfoEndpoint {  
    @PayloadRoot (  
        localPart = "FormInfoRequest",  
        namespace = "http://www.cpost.cz/forminfo")  
    public FormInfoResponse handleFormInfo(FormInfoRequest req) {  
        ...  
    }  
}
```



# Obsluha messages

## @Endpoint

```
public class FormInfoEndpoint {  
    @PayloadRoot (  
        localPart = "FormInfoRequest",  
        namespace = "http://www.cpost.cz/forminfo")  
    public FormInfoResponse handleFormInfo(  
        @XPathParam("/tns:formInfoRequest/tns:formInfo/tns:podaciCislo/text()")  
        String podCis) {  
        ...  
    }  
}
```





# Spring Integration





# Klasická situace

- Vstupem rozumíme Webové služby, soubory, ...
- Výstupem rozumíme dávky, JMS, jiné služby, ...
- Framework poskytuje standardní prvky, např.
  - Splitter
  - Router
  - Transformer
  - Filter
  - ...

# Spring Web Services v akci

<beans>

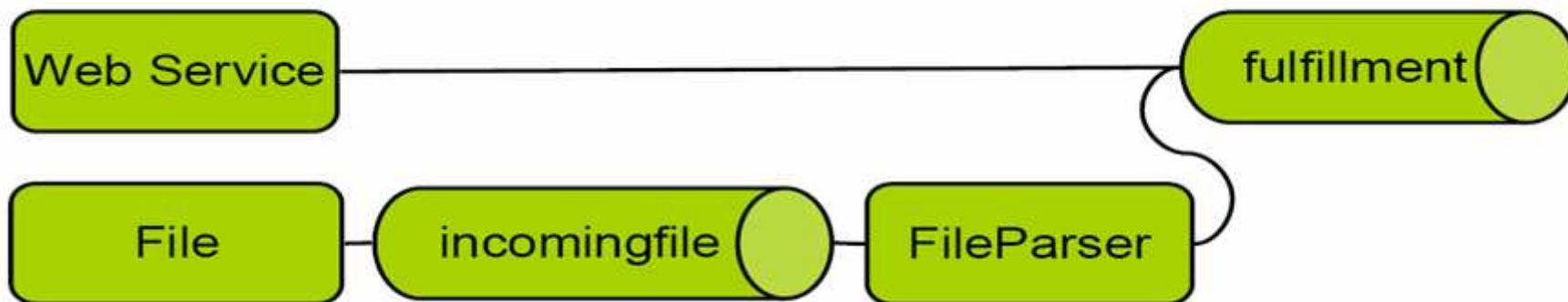
<integration:annotation-driven />

<integration:message-bus auto-create-channels="false" />

<integration:channel id="fulfillment" />

<integration:channel id="incomingfile" />

<integration:file-source directory="/tmp/files,, channel="incomingfile" poll-period="1000" />





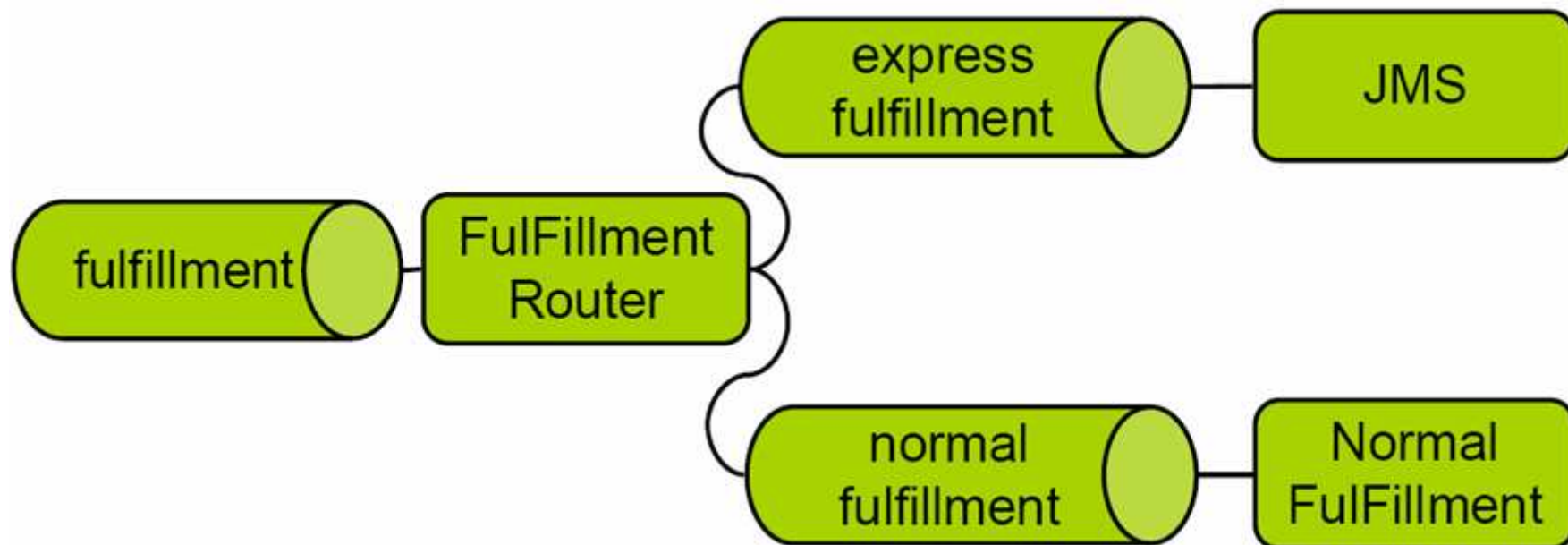
# Spring Web Services v akci

@Component

```
public class FileParser {  
    @Resource  
    private MessageChannel fulfillment;  
  
    @Subscriber(channel = "incomingfile")  
    public void handleFile(String content){  
        Order order = ...;  
        fulfillment.send(orderMessage);  
    }  
}
```



# Spring Web Services v akci





# Spring Web Services v akci

**@MessageEndpoint(input="fulfillment")**

```
public class FulFillmentRouter {  
  
    @Router  
    public String routeOrder(Order order) {  
        if (order.isExpress()) {  
            return "expressfulfillment";  
        } else {  
            return "normalfulfillment";  
        }  
    }  
}
```



# Spring Batch





# Dávky

- Typicky složené z kroků
- Obvykle čtou a zapisují data
- Problémy jsou restarty, optimalizace, velké množství dat, ...
- V aktuálním příkladu
  - Přečíst formulář
  - Zpracovat formulář
  - Zapsat je zpět



# Jak na čtení dat

- Pouze primární klíče
  - Méně dat
  - Postupné dočítávání
- Alternativně lze načítat data pomocí kurzoru
- Další alternativa je načítat shluky dat
- Prakticky nemožné je načíst všechna data současně



# Java v akcii

```
public class FormBatchVerification extends AbstractItemWriter {  
    private FormDao formDao;  
  
    @Autowired  
    public void setFormDao(FormDao formDao) {  
        this.formDao = formDao;  
    }  
  
    public void write(Object podaciCislo) throws Exception {  
        Form form = (Form) formDao.findById((Integer) podaciCislo);  
        form.setProcessed(true);  
        // do the processing  
        formDao.update(form);  
    }  
}
```



# Konfigurace v akci

```
<beans ...>
  <bean id="" parent="simpleJob">
    <property name="steps">
      <bean id="step1" parent="simpleStep">
        <property name="itemReader">
          <bean class="....DrivingQueryItemReader">
            <property name="keyCollector">
              <bean class="....SingleColumnJdbcKeyCollector">
                <property name="sql"
                  value="SELECT ID FROM XXX WHERE ... " />
                <property name="jdbcTemplate" ref="jdbcTemplate" />
              </bean>
            </property>
          </bean>
        </property>
      </bean>
    </property>
  </bean>
</property>
</beans>
```




# Konfigurace v akci

```
<property name="itemWriter">
  <bean class="...FormBatchVerification" />
</property>
<property name="commitInterval" value="15" />
</bean>
</property>
</bean>
</beans>
```

- Po každých 15 formulářích commit



# Spring Batch složitěji

- Závislé kroky, validace, ...
  - Různé datové zdroje (soubory, ...)
  - Může být kontrolován přes JMX
  - Perzistence stavu
  - Snadný restart
- 

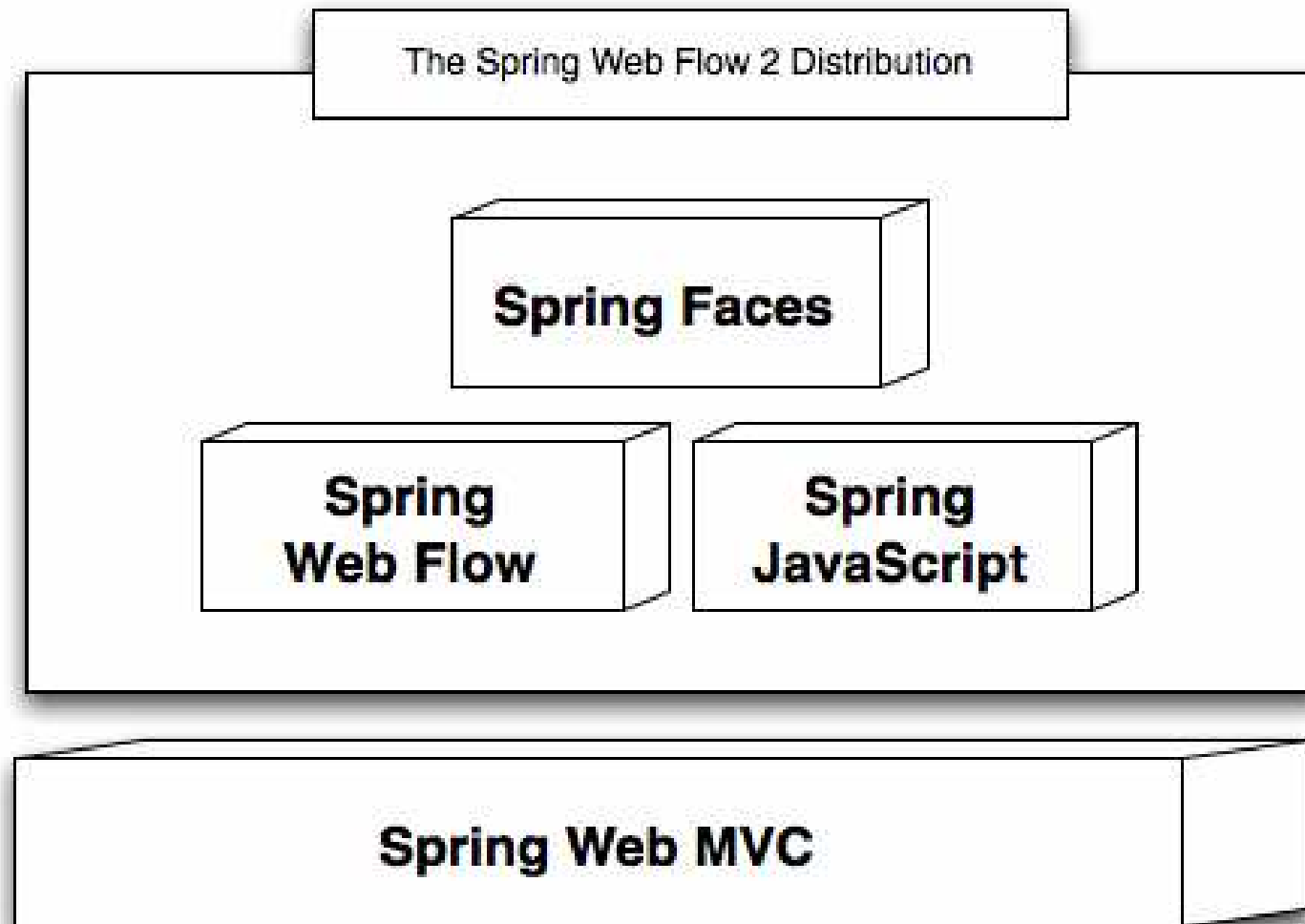


# Spring Web Flow





# Komponenty WebFlow





# Spring Web Flow

- Definice znovupoužitelných flow
- **Pokročilý controller engine**
- Vynikající podpora pro AJAX (problémy s knihovnou Trinidad, ...)
  - Partial refresh
  - Render popup
  - ...
- Vynikající podpora pro využívání JSF



# HTML v akcii

```
<?xml version="1.0" encoding="windows-1250"?>
<ui:composition ...>
<ui:param name="contentTitle" value="#{msg.debSoModify_cnf_title}"/>
<ui:param name="formId" value="form_debSoModify_cnf"/>

<ui:define name="innerContent">
  <ib:label_value_in_row fieldId="clientaccount" label="#{msg.clientAccountLabel}"
    value="#{debSoModifyBean.cblRequest.activeTx.clientAccountNumber}">
    <ib:accountConverter showBankCode="false" />
  </ib:label_value_in_row>

  <ib:label_value_in_row
    fieldId="referencenumber" label="#{msg.referenceNumberLabel}"
    trClass="delimiter"
    value="#{debSoModifyBean.cblRequest.activeTx.referenceNumber}">
  </ib:label_value_in_row>

  ...
</ui:composition>
```



# Java v akcii

```
public class DebSoGetBean extends DebSoBeanBase {
    private List<StandingOrderWrapper> standingOrders;

    public void onFlowStart( ... ) {
        // nacist prikazy
        List<StandingOrderWrapper> orders = ...
        ...
    }
    public StandingOrderWrapper onOrderSelection( String refNum, String verNum ) {
        StandingOrderWrapper chosen = ...
        return chosen;
    }
    public String onOrderDetailPageSelection( StandingOrderWrapper chosen ...) {
        if ( chosen.isCurrentAccountType() ) {
            return "DEB_SOCA";
        } else ....
    }
}
```



# Konfigurace v akci

```
<?xml version="1.0" encoding="UTF-8"?>
<flow ...>
  <on-start>
    <evaluate expression="debSoGetBean.onFlowStart(infrastructure, ...)" />
  </on-start>

  <view-state id="ordersView" view="lst_deb_so_get.xhtml">
    <transition on="doShowDetail" to="ordersDetailPrepare">
      <evaluate
expression="debSoGetBean.onOrderSelection(requestParameters.id,requestParameters.ver)"
      result="flowScope.chosenOrder" />
    </transition>
  </view-state>
```



# Konfigurace v akci

```
<action-state id="ordersDetailPrepare">
  <evaluate expression="debSoGetBean.onOrderDetailPageSelection(chosenOrder ...)" />
  <transition on="DEB_SOCA" to="..."> ... </transition>
  <transition on="DEB_SO" to="orderDetailView" />
  ...
</action-state>
<view-state id="orderDetailView" view="ib_trn_deb_so_get.xhtml">
  <on-entry>
    <evaluate expression="debSoGetBean.onStandingOrderDetailEntry(chosenOrder)" />
  </on-entry>
  <transition on="doChange" to="modifyStandingOrder" />
  ...
</view>
<subflow-state id="modifyStandingOrder" subflow="flow_deb_so_modify">
  <input name="orderToModifyParam" value="debSoGetBean.chosenOrder" />
  <transition on="modifyConfirmed" to="modifyOrderConfirmed" />
  <transition on="modifyCancelled" to="orderDetailView" />
</subflow-state>
```



# Na co nezbyl čas

- Spring Modules
- Spring DM
- Spring dm server
- ...





# Slovo závěrem

Spring je mnohem víc  
než jen Spring Framework





# Reference

- [1] [Spring framework](#) homepage
- [2] [Spring source](#) homepage
- [3] [Spring reference manual](#) (verze 2.5)
- [4] [Spring projects](#)
- [5] Expert One-on-One J2EE Design and Development, Rod Johnson, Wrox, 2002
- [6] Expert One-on-One J2EE Development without EJB, Rod Johnson, Juergen Hoeller, 2004
- [7] Spring in Action, Craig Walls and Ryan Breidenbach, Manning, 2007



# Diskuse

- Komentáře
- Otázky
- Připomínky
- Upřesnění
- Poznámky
- ...

