Spring Framework 3.0
On The Way To 3.1

Jürgen Höller
Principal Engineer
SpringSource, a division of VMware

Agenda

- A Review
  - Spring 3.0 Themes & Trends
- A Preview
  - Spring 3.1 Themes & Trends

Spring 3.0 Themes

- Powerful **annotated component model**
  - stereotypes, factory methods, JSR-330 support
- Spring **Expression Language**
  - Unified EL++
- Comprehensive **REST support**
  - and other Spring @MVC additions
- Support for **Portlet 2.0**
  - action/event/resource request mappings
- Declarative **model validation**
  - integration with JSR-303 Bean Validation
- Support for **Java EE 6**
  - in particular for JPA 2.0

Enhanced Stereotype Model

- Powerful options for custom annotations
  - combining meta-annotations e.g. on stereotype
  - automatically detected (no configuration necessary!)

```
@Service
@Scope("request")
@Transactional(rollbackFor=Exception.class)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyService {}

@MyService
public class RewardsService {
    ...
}
```

Annotated Factory Methods

- Spring 3.0 includes the core functionality of the Spring **JavaConfig** project
  - **configuration classes** defining managed beans
  - common handling of **annotated factory methods**

```
@Bean @Primary @Lazy
public RewardsService rewardsService() {
    RewardsServiceImpl service = new RewardsServiceImpl();
    service.setDataSource(…);
    return service;
}
```

Standardized Annotations

```
@ManagedBean
public class RewardNetworkService
        implements RewardNetwork {

    @Inject
    public RewardNetworkService(AccountRepository ar) {
      …
    }

    @TransactionAttribute
    public RewardConfirmation rewardAccountFor(Dining d) {
      …
    }
}
```

JSR-330 and Co

- **@javax.inject.Inject** is part of JSR-330
  - "Dependency Injection for Java"
  - also defines @Qualifier semantics
  - and a Provider interface
- **@javax.annotation.ManagedBean** is part of JSR-250 v1.1
  - driven by the Java EE 6 specification
  - can be detected through classpath scanning
- **@javax.ejb.TransactionAttribute** is part of the EJB 3.0/3.1 specification
  - also supported for Spring beans

EL in Bean Definitions

```
<bean class="mycompany.RewardsTestDatabase">

    <property name="databaseName"
            value="“#{systemProperties.databaseName}”/>

    <property name="keyGenerator"
            value="“#{strategyBean.databaseKeyGenerator}”/>

</bean>
```

EL in Component Annotations

```
@Repository
public class RewardsTestDatabase {

    @Value("#{systemProperties.databaseName}")
    public void setDatabaseName(String dbName) { … }

    @Value("#{strategyBean.databaseKeyGenerator}")
    public void setKeyGenerator(KeyGenerator kg) { … }
}
```

REST Support

- Spring MVC to provide first-class support for **REST-style mappings**
  - extraction of URI template parameters
  - content negotiation in view resolver
- Goal: **native REST support** within Spring MVC, for UI as well as non-UI usage
  - in natural MVC style, preserving MVC strengths
- Consists of several independent features
  - also: client-side RestTemplate

REST in MVC - @PathVariable

http://rewarddining.com/rewards/12345

```
@RequestMapping(value = "/rewards/{id}", method = GET)
public Reward reward(@PathVariable("id") long id) {
    return this.rewardsAdminService.findReward(id);
}
```

Portlet 2.0 Support

- Portlet 2.0: major new capabilities
  - explicit action name concept for dispatching
  - **resource requests** for servlet-style serving
  - **events** for inter-portlet communication
- Spring's Portlet MVC 3.0 to support **explicit mapping annotations**
  - @ActionMapping, @RenderMapping, @ResourceMapping, @EventMapping
  - specializations of Spring's @RequestMapping

Spring Portlet MVC 3.0

```
@Controller
@RequestMapping("EDIT")
public class MyPortletController {
    …

    @ActionMapping("delete")
    public void removeBook(@RequestParam("book") String bookId) {
        this.myService.deleteBook(bookId);
    }

    @EventMapping("BookUpdate")
    public void updateBook(BookUpdateEvent bookUpdate) {
        // extract book entity data from event payload object
        this.myService.updateBook(…);
    }
}
```

Declarative Model Validation

```
public class Reward {
    @NotNull
    @Past
    private Date transactionDate;
}

@RequestMapping("/rewards/new")
public void newReward(@Valid Reward reward) { … }
```

- **JSR-303 "Bean Validation"** as the common ground
- Spring 3.0 fully supports JSR-303 for MVC data binding
- Same metadata can be used for persisting, rendering, etc

Scheduling Enhancements

- Spring 3.0 introduces a **major overhaul of the scheduling package**
  - TaskScheduler interface with Trigger abstraction
  - **XML scheduling namespace** with cron support
  - @Async annotation for asynchronous user methods
  - **@Scheduled annotation** for cron-triggered methods

```
@Scheduled(cron = "0 0 12 * * ?")
public void performTempFileCleanup() {
    ...
}
```

Spring 3.0 and Java EE 6

- **Java EE 6 API support** in Spring 3.0
  - integration with **JPA 2.0**
    - support for query builder, shared cache setup, etc
  - integration with **JSF 2.0**
    - full compatibility as managed bean facility
  - **JSR-303 Bean Validation** integration
    - through Hibernate Validator 4.1
  - **JSR-330**: common dependency injection annotations
    - natively supported by Spring itself

Spring 3.0 Summary

- Spring 3.0 **embraces REST and EL**
  - full-scale REST support in Spring MVC
  - broad Unified EL++ support in the core
- Spring 3.0 significantly extends its **annotated component model**
  - JSR-330 dependency injection annotations
  - validation, scheduling, formatting
- Spring 3.0 **integrates with Java EE 6 APIs** such as JSF 2.0, JPA 2.0, JSR-303

On The Way To Spring 3.1

- We're about to release Spring Framework 3.0.5 in October
  - last planned release in the 3.0.x branch
- Moving on to **Spring 3.1 milestones**
  - first milestone scheduled for November
- Spring 3.1 is a straightforward next step after 3.0
  - **building on the Spring 3.0 foundation**

Spring 3.1 Themes

- **Environment profiles** for beans
- Java-based **application configuration**
- **Cache abstraction**
- **Conversation management**
- Servlet 3.0 & JSF 2.0

Environment Profiles

- Grouping bean definitions for activation in specific environments only
  - **development, testing, production**
  - possibly different deployment environments
- **Environment abstraction**
  - special injectable API type
- Custom **resolution of placeholders**
  - dependent on the actual environment
- Ideally: no need to touch deployment unit across different stages/environments

Environment Example

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
    <property name="driverClass" value="${database.driver}"/>
    <property name="jdbcUrl" value="${database.url}"/>
    <property name="username" value="${database.username}"/>
    <property name="password" value="${database.password}"/>
</bean>

<beans profile="embedded">
    <jdbc:embedded-database id="dataSource" type="H2">
        <jdbc:script location="/WEB-INF/database/schema-member.sql"/>
        <jdbc:script location="/WEB-INF/database/schema-activity.sql"/>
        <jdbc:script location="/WEB-INF/database/schema-event.sql"/>
        <jdbc:script location="/WEB-INF/database/data.sql"/>
    </jdbc:embedded-database>
</beans>
```

Java-Based App Config

- Application-specific container configuration in @Configuration classes
  - **equivalent to XML namespace functionality**
  - focus on customizing the annotation-based processing parts of Spring
- Typical infrastructure setup
  - AOP configuration
  - **transactions**
  - **scheduling**

App Config Example

```
@Configuration
public void AppConfig {

    @Autowired
    private DataSource dataSource;

    @Bean
    public RewardsService rewardsService() {
        return new RewardsServiceImpl(dataSource);
    }

    @Bean
    public PlatformTransactionManager txManager() {
        return new DataSourceTransactionManager(dataSource);
    }

    public TransactionConfiguration txConfig() {
        return annotationDrivenTx().withTransactionManager(txManager());
    }
}
```

Cache Abstraction

- CacheManager and Cache abstraction
  - in **org.springframework.cache**, which up until 3.0 just contained EhCache support
  - particularly important with the rise of **distributed caching** in cloud environments
- Backend adapters for EhCache, **GemFire**, Coherence, etc
  - several to be shipped with Spring core
  - plugging in custom adapters if necessary

Caching Annotations

```
@Cacheable
public Owner loadOwner(int id);

@Cacheable(condition="name.length < 10")
public Owner loadOwner(String name);

@CacheInvalidate
public void deleteOwner(int id);
```

Conversation Management

- Abstraction for conversational sessions
  - basically **HttpSession++**
  - more flexible lifecycle
  - more flexible storage options
- Management of **current conversation**
  - e.g. associated with browser window/tab
  - or manually demarcated
- Foundation for Web Flow 3
  - but also **for use with MVC and JSF**
  - programmatic access at any time

Window-Specific Sessions

- Common problem: **isolation between browser windows/tabs**
  - windows sharing the same HttpSession
  - HttpSession identified by shared cookie
- Window id managed by the framework
  - associating the **current window session**
  - e.g. for MVC session form attributes
- Simpler problem, simpler solution
  - as opposed to full conversation management

Servlet 3.0 & JSF 2.0

- Explicit support for **Servlet 3.0 containers**
  - such as Tomcat 7 and GlassFish 3
  - options for **automatic deployment of framework listeners**
  - standard file upload support behind MultipartResolver abstraction
- Richer support for **JSF 2.0**
  - e.g. with respect to conversations

Spring 3.1 Summary

- **Environment profiles** for beans
  - flexible placeholder resolution
- Java-based **application configuration**
  - infrastructure setup in configuration classes
- **Cache abstraction**
  - plus caching annotations
- **Conversation management**
  - window sessions & conversations for MVC
- Servlet 3.0 & JSF 2.0

Spring 3.1 Release Plan

- 3.1 M1 in November 2010
- 3.1 M2 in December 2010
- …
- 3.1 GA in late Q1 2011?