Spring Framework 3.0
The Next Generation

Jürgen Höller
VP & Distinguished Engineer
SpringSource

Quick Review: Spring 2.5

- **Annotation-based component style**
  - dependency injection: *@Autowired*
    - with optional *@Qualifier* or custom qualifier
  - middleware services: *@Transactional*
  - stereotypes: *@Component, @Repository, @Controller*
- Common **Java EE 5** annotations supported too
  - *@PostConstruct, @PreDestroy, @Resource*, etc
- **Component scanning** in the classpath
  - as alternative to (minimal) XML bean definitions
- **Annotated web controllers** (a.k.a. @MVC)

Annotated Bean Component

```
@Service
public class RewardNetworkService
        implements RewardNetwork {

    @Autowired
    public RewardNetworkService(AccountRepository ar) {
      …
    }

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

Test Context Framework


```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class RewardSystemIntegrationTests {

    @Autowired
    private RewardNetwork rewardNetwork;

    @Test
    @Transactional
    public void testRewardAccountForDining() {
        // test in transaction with auto-rollback
    }
}
```

@MVC Controller Style

```
@Controller
public class MyController {

    private final MyService myService;

    @Autowired
    public MyController(MyService myService) {
        this.myService = myService;
    }

    @RequestMapping("/removeBook")
    public String removeBook(@RequestParam("book") String bookId) {
        this.myService.deleteBook(bookId);
        return "redirect:myBooks";
    }
}
```

Spring 3.0 Themes

- **Java 5+** foundation
  - even stronger support for annotated components
- 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**
  - Hibernate Validator, JSR-303
- Early support for **Java EE 6**
  - JSF 2.0, JPA 2.0, etc

Use of Meta-Annotations

- More 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;
}
```

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) { … }
}
```

EL Context Attributes

- Example showed **access to EL attributes**
  - "systemProperties", "strategyBean"
  - implicit references in expressions
- **Implicit attributes** to be exposed by default, depending on runtime context
  - e.g. "systemProperties", "systemEnvironment"
    - global platform context
  - access to all Spring-defined beans by name
    - similar to managed beans in JSF expressions
  - extensible through Scope SPI
    - e.g. for step scope in Spring Batch

Web Context Attributes

- **Implicit web-specific attributes** to be exposed by default as well
  - "contextParameters": web.xml init-params
  - "contextAttributes": ServletContext attributes
  - "request": current Servlet/PortletRequest
  - "session": current Http/PortletSession
- Exposure of all **implicit JSF objects** when running within a JSF request context
  - "param", "initParam", "facesContext", etc
  - full compatibility with JSF managed bean facility

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
- Alternative: **using JAX-RS** through integrated JAX-RS provider (e.g. Jersey)
  - using the JAX-RS component model to build programmatic resource endpoints

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);
}
```

Common @MVC Refinements

- More options for handler method parameters
  - in addition to @RequestParam and @PathVariable
  - **@RequestHeader:** access to request headers
  - **@CookieValue:** HTTP cookie access
  - supported for Servlet MVC and Portlet MVC

```
@RequestMapping("/show")
public Reward show(@RequestHeader("region") long regionId,
      @CookieValue("language") String langId) {
   ...
}
```

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
  - portlet filters analogous to servlet filters
- Spring's Portlet MVC 3.0 to support **explicit mapping annotations**
  - @ActionMapping, @RenderMapping, @ResourceMapping, @EventMapping
  - specializations of Spring's @RequestMapping
    - supporting action names, window states, etc

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(…);
    }
}
```

Model Validation

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

In view:

```
<form:input path="transactionDate">
```

- Same metadata can be used for persisting, rendering, etc
- Spring 3.0 RC1: to be supported for MVC data binding
- **JSR-303 "Bean Validation"** as the common ground

Spring 3.0 and Java EE 6

- **Early Java EE 6 API support** in Spring 3.0
  - integration with **JSF 2.0**
    - full compatibility as managed bean facility
  - integration with **JPA 2.0**
    - support for lock modes, query timeouts, etc
  - support for **JSR-303 Bean Validation** annotations
    - through Hibernate Validator 4.0 integration
  - all embeddable on Tomcat 5.5+ / J2EE 1.4+
- Spring 3.x: **support for Java EE 6 platforms**
  - Servlet 3.0 (waiting for GlassFish 3 and Tomcat 7)
  - JSR-236 "Concurrency Utilities for Java EE"

Spring 3.0 Summary

- Spring 3.0 **embraces REST and EL**
  - full-scale REST support
  - broad Unified EL++ support in the core
- Spring 3.0 significantly extends and **refines annotated web controllers**
  - RESTful URI mappings
  - annotation-based model validation
- Spring 3.0 remains **backwards compatible with Spring 2.5** on Java 5+
  - enabling a smooth migration path