Spring 2.5 on the Way to 3.0

Jürgen Höller
VP & Distinguished Engineer
SpringSource

Agenda

- Review: Spring 2.5
- Plans for Spring 3.0
- Spring 3.0 Roadmap

The Spring Framework

- The "classic" Spring project
  - established as early as February 2003
- Foundational, domain-independent framework
  - major generations: 1.2, 2.0, 2.5
  - currently moving on to 3.0
- Covering many areas of functionality
  - DI container, AOP framework
  - transaction abstraction, data access
  - messaging support, executor abstractions
  - web support, web MVC infrastructure

Spring Framework 2.5

- Extended **platform support**
  - Java SE 6, Java EE 5, and OSGi
- Enhanced **AspectJ support**
  - new pointcut and weaving options
- Comprehensive support for **annotation-based configuration**
  - bean lifecycle, autowiring
  - revised test context framework
  - major improvements to Spring MVC

Extended Platform Support

- **Java SE 6**
  - JDBC 4.0, JMX MXBeans
  - ServiceLoader API, HttpServer API
- **Java EE 5**
  - Servlet 2.5, JSP 2.1, JSF 1.2
  - JTA 1.1, JAX-WS 2.0/2.1
  - JSR-250 annotations support
- OSGi Bundles out of the box

Support for AspectJ LTW

- **AspectJ Load-Time Weaving**
  - transforming byte code of application classes
  - through Spring's LoadTimeWeaver abstraction
- Driven by AspectJ `META-INF/aop.xml` files
  - standard AspectJ deployment descriptor
  - aspects can be individually deployed as jars
- <context:load-time-weaver/>
  - <context:spring-configured/>
  - <tx:annotation-driven mode="aspectj"/>
- Requires platform support!

Annotated Bean Component


```
@Service
public class RewardNetworkService
        implements RewardNetwork {

  @Autowired
  public RewardNetworkService(AccountRepository ar) {
     …
  }

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

Annotated DAO

```
@Repository
public class HibernateAccountRepository
        implements AccountRepository {

  @Autowired
  public HibernateAccountRepository(SessionFactory sf) {
    ...
  }

  public Account loadAccount(String number) {
    // use Hibernate API here
  }
}
```

Annotated DAO with Lifecycle

```
@Repository
public class JdbcAccountRepository implements AccountRepository {

  @Autowired
  public JdbcAccountRepository(DataSource ds) { … }

  @PostConstruct
  public initCache() { … }

  @PreDestroy
  public cleanupCache() { … }
}
```

Using Java EE 5 Annotations

- Java EE 5 includes specific annotations
  - **@Resource**
    - injecting a JNDI reference into a managed bean
  - **@WebServiceRef / @EJB**
    - injecting a JAX-WS / EJB 3 service proxy
  - **@TransactionAttribute**
    - EJB 3 transaction demarcation
  - **@PersistenceContext / @PersistenceUnit**
    - JPA resource injection
- All consistently supported in Spring 2.5

Minimal XML Bootstrapping


```
<bean class="com.myapp.rewards.RewardNetworkImpl"/>

<bean class="com.myapp.rewards.JdbcAccountRepository"/>


<!- OR: even getting rid of explicit bean definitions completely! -->


<!-- Scans for @Components, @Services, etc to deploy -->
<context:component-scan base-package="com.myapp.rewards"/>


<!– Plus shared infrastructure configuration beans:
    PlatformTransactionManager, DataSource, etc -->
```

Test Context Framework


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

  private RewardNetwork rewardNetwork;

  @Autowired
  public void setRewardNetwork(RewardNetwork) { … }

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

Spring Servlet MVC 2.5


```
@Controller
public class MyController {

  private final MyService myService;

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

  @RequestMapping("/myBooks")
  public String showBooks(ModelMap model) {
    model.addAttribute("books", myService.findAllBooks());
    return "booksEdit";
  }

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

Conventional URL Mapping


```
@Controller
@RequestMapping("/rewards/**")
public class RewardsController {

    @RequestMapping
    public void index() {...}

    @RequestMapping
    public List<Reward> search(SearchCriteria criteria) {...}

    @RequestMapping
    public Reward show(@RequestParam Long id) {...}
}
```

Spring Portlet MVC 2.5

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

  private final MyService myService;

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

  @RequestMapping(params = "action=list")
  public String showBooks(ModelMap model) {
    model.addAttribute("books", myService.findAllBooks());
    return "booksEdit";
  }

  @RequestMapping(params = "action=delete")
  public void removeBook(@RequestParam("book") String bookId,
      ActionResponse response) {
    myService.deleteBook(bookId);
    response.setRenderParameter("action", "list");
  }
}
```

Plans for Spring 3.0

- **Java 5+** foundation
  - compatible with J2EE 1.4 and Java EE 5
- Spring **Expression Language**
  - Unified EL++
- Notable Spring **MVC additions**
  - comprehensive REST support
  - first-class Ajax support
  - declarative model validation
- Support for **Portlet 2.0**

Bean Definition Example

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

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

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

</bean>
```

Unified EL + Factory Example

```
@Factory
public class RewardsTestDatabaseFactory {

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

  @FactoryMethod
  public DataSource createTestDatabase() { … }
}
```

REST in MVC - @PathParam

http://rewarddining.com/show/12345

```
@RequestMapping(method = RequestMethod.GET)
public Reward show(@PathParam Long id) {
    return this.rewardsAdminService.findReward(id);
}
```

REST Routing Conventions


```
@Controller
public class AccountsController
        implements RestController<Account, Long> {
    public List<Account> index() {}
    public void create(Account account) {}
    public Account show(Long id) {}
    public void delete(Long id) {}
    public void update(Account account) {}
}
```

GET http://rewarddining.com/accounts

POST http://rewarddining.com/accounts

GET http://rewarddining.com/accounts/1

DELETE http://rewarddining.com/accounts/1

PUT http://rewarddining.com/accounts/1

Different Representations

- JSON

  GET http://rewarddining.com/accounts/1 accepts **application/json**
  GET http://rewarddining.com/accounts/1**.json**

- XML

  GET http://rewarddining.com/accounts/1 accepts **application/xml**
  GET http://rewarddining.com/accounts/1**.xml**

- ATOM

  GET http://rewarddining.com/accounts/1 accepts **application/atom+xml**
  GET http://rewarddining.com/accounts/1**.atom**

Model Validation

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

In view:

```
<form:input value="#{reward.transactionDate}">
```

- Enforced client-side and server-side
- Automatic Ajax refresh of validation errors
- We're considering JSR-303 and Hibernate Validator…

Further Servlet Web Features

- No definitive list yet
  - potentially covered by other portfolio projects
- Conversation management
  - isolation of concurrent windows in same browser
  - conversation scope with shorter lifetime than session
- Stateful controller objects
  - rather than stateless controllers with some form of session attribute management
  - particularly worth exploring in combination with conversation scope

Portlet 2.0 Support

- Portlet 2.0 introduces major new features
  - explicit action name concept for action dispatching
  - resource requests for servlet-style serving
  - events for inter-portlet communication
  - portlet filters analogous to servlet filters
- Nice fit with Spring Portlet MVC
  - in particular with annotation-based dispatching
- Portlet MVC 3.0 to support explicit annotations
  - @ActionMapping, @RenderMapping
  - @ResourceMapping, @EventMapping

Spring 2.5 Mission Continued

- Spring 3 **continues Spring 2.5's mission**
  - fully embracing Java 5 in the core Spring programming and configuration model
  - now with even the core framework requiring Java 5
    - all framework API to use Java 5 language syntax
- **Backwards compatibility** with Spring 2.5
  - 100% compatibility of programming model
  - 95% compatibility of extension points
  - all previously deprecated API to be removed
    - Make sure you're not using outdated Spring 1.2 / 2.0 API anymore!

Pruning & Deprecation in 3.0

- Some **pruning** planned
  - Commons Attributes support
  - traditional TopLink API support
    - in favor of JPA (EclipseLink)
- Some **deprecation** planned
  - traditional MVC controller class hierarchy
    - superseded by annotated controller style
  - traditional JUnit 3.8 test class hierarchy
    - superseded by test context framework

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

Spring 3.0 Roadmap

- **Spring 3.0 M1** to be released in **September 2008**
  - first cut of REST and EL support
- **Spring 3.0 RC1** scheduled for **December 2008**
  - you guessed it: SpringOne America ☺
- **Spring 3.0 final** expected in **January 2009**
  - depending on RC feedback

Spring 3.x Roadmap

- **Spring 3.1** expected in July 2009
  - full support for Java EE 6 environments
  - Servlet 3.0, JSF 2.0, JPA 2.0, JAX-RS
  - support for Web Beans annotations?
  - waiting on specs to be finalized…
- **Spring 3.2** expected in December 2009
  - possibly introducing explicit Java 7 support
  - still compatible with Java 5+

Your Feedback Opportunity!

- Let us know about your most serious pain points in Spring 2.5…
  - REST, EL and conversation support have been the most common requests up to now
- Spring 3.0 feature scope is largely determined already
  - However, it's still early enough to change priorities…
  - Review the enhancement requests in JIRA!