

What's new in Spring 3.0?

Arjen Poutsma
SpringSource

About Me

- Fifteen years of experience in Enterprise Software Development
- Development lead of Spring Web Services
- Developer on Spring 3
- Contributor to various Open Source frameworks: (XFire, Axis2, NEO, ...)

Themes

- Java 5+
- Spring Expression Language
- REST support
- Portlet 2.0
- Declarative model validation
- Early support for Java EE 6

Java 5+

Generics

```
public interface BeanFactory {  
    <T> T getBean(String name, Class<T> requiredType);  
  
    <T> Map<String, T> getBeansOfType(Class<T> type);  
}
```

Rearrangements

- Object/XML Mapping (OMX) module
 - REST, SQL XML access
- Conversion infrastructure
 - stateless Java 5+ type converters and formatters
 - superseding PropertyEditors

Annotated Factory Methods

- Spring JavaConfig
 - configuration classes
 - annotated factory methods

```
@Configuration
public class AppConfig{

    @Bean @Lazy
    public RewardsService rewardsService() {

        RewardsServiceImpl service =

            new RewardsServiceImpl();

        service.setDataSource(...);

        return service;

    }

}
```

Meta-Annotations

```
@Service
```

```
@Scope("request")
```

```
@Transactional(rollbackFor=Exception.class)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface MyService {}
```

```
@MyService
```

```
public class RewardsService {
```

```
    ...
```

```
}
```

Expression Language

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) { ... }
```

```
}
```

Default Context Attributes

- Exposed by default
 - "systemProperties", "systemEnvironment"
 - access to all Spring-defined beans by name
 - extensible through SPI
- Determined at runtime (not init-time)

Web Context Attributes

- Web-specific attributes:
 - "contextParameters"
 - "contextAttributes"
 - "request"
 - "session"
- JSF objects in a JSF request context

Web

URI Templates

- URI-like string, containing one or more variable names
- Variables can be substituted for values to become a URI
- Helps to create nice, “RESTful” URLs

Examples

URI Template	Request	Variables
<code>/hotels/{hotelId}</code>	<code>/hotels/ westindiplomat</code>	<code>hotelId= westindiplomat</code>
<code>/hotels/{hotelId}/ bookings</code>	<code>/hotels/ westindiplomat/ bookings</code>	<code>hotelId= westindiplomat</code>
<code>/hotels/{hotelId}/ bookings/ {bookingId}</code>	<code>/hotels/ westindiplomat/ bookings/21</code>	<code>hotelId= westindiplomat bookingId=21</code>

@PathVariable

- Spring 3.0 M1 introduced the @PathVariable annotation
- Allows you to use URI Templates in @MVC

Example

```
@Controller
@RequestMapping("/hotels/{hotel}")
public class HotelsController {

    @RequestMapping
    public void handleHotel(@PathVariable("hotel") String hotel) {
        // ...
    }

    @RequestMapping("bookings/{booking}")
    public void handleBooking(@PathVariable("hotel") String hotel,
        @PathVariable int booking) {
        // ...
    }
}
```

Example

Optional



```
@Controller
@RequestMapping("/hotels/{hotel}")
public class HotelsController {

    @RequestMapping
    public void handleHotel(@PathVariable("hotel") String hotel) {
        // ...
    }

    @RequestMapping("bookings/{booking}")
    public void handleBooking(@PathVariable("hotel") String hotel,
        @PathVariable int booking) {
        // ...
    }
}
```

Example

Optional



```
@Controller
@RequestMapping("/hotels/{hotel}")
public class HotelsController {

    @RequestMapping ← Matches /hotels/42
    public void handleHotel(@PathVariable("hotel") String hotel) {
        // ...
    }

    @RequestMapping("bookings/{booking}")
    public void handleBooking(@PathVariable("hotel") String hotel,
        @PathVariable int booking) {
        // ...
    }
}
```

Example

Optional



```
@Controller
@RequestMapping("/hotels/{hotel}")
public class HotelsController {

    @RequestMapping ← Matches /hotels/42
    public void handleHotel(@PathVariable("hotel") String hotel) {
        // ...
    }

    @RequestMapping("bookings/{booking}")
    public void handleBooking(@PathVariable("hotel") String hotel,
                              @PathVariable int booking) {
        // ...
    }
}
Matches /hotels/42/bookings/21
```

Views

Mime Type	View
application/xml	MarshallingView
application/atom+xml	AbstractAtomFeedView
application/rss+xml	AbstractRssFeedView
application/json	MappingJacksonJsonView

Other Goodies

- `HiddenHttpMethodFilter`
- `ShallowEtagHeaderFilter`
- `@RequestHeader`, `@RequestBody`,
`@ResponseBody`, `@CookieValue`

<http://blog.springsource.com/2009/03/08/rest-in-spring-3-mvc/>

RestTemplate

- RestTemplate as core client-side component
- Similar to other templates in Spring
 - JdbcTemplate
 - JmsTemplate
 - WebServiceTemplate

RestTemplate

```
String uri = "http://example.com/hotels/{id}";  
template = new RestTemplate();  
HotelList result = template.getForObject(uri,  
    HotelList.class, "1");
```

```
Booking booking = // create booking object  
uri = "http://example.com/hotels/{id}/bookings";  
Map<String, String> vars = Collections.singletonMap("id", "1");  
URI location = template.postForLocation(uri, booking, vars);
```

```
template.delete(location.toString());
```

```
template.execute(uri, HttpMethod.GET,  
    myRequestCallback,  
    myResponseCallback);
```

<http://blog.springsource.com/2009/03/27/rest-in-spring-3-resttemplate/>

Portlet 2.0 Support

```
@Controller
@RequestMapping("EDIT")
public class MyPortletController {
    @RequestMapping("delete")
    public void removeBook(@RequestParam("book") String bookId) {
        this.myService.deleteBook(bookId);
    }
    @RequestMapping("BookUpdate")
    public void updateBook(BookUpdateEvent bookUpdate) {
        // extract book entity data from event payload object
        this.myService.updateBook(...);
    }
}
```

Declarative Validation

- Annotation-based
- Same metadata can be used for persisting, rendering, etc
- JSR-303 "Bean Validation" as the common ground
- Spring 3.x will fully support JSR-303 as it becomes final

Declarative Validation Sample

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

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

Various

Spring 3.0 and Java EE 6

- Early Java EE 6 support in Spring 3.0
 - JSF 2.0
 - JPA 2.0
 - JSR-303 Bean Validation
- Spring 3.1: full support for Java EE 6

Pruning & Deprecation

Pruned:

- Commons Attributes support
- traditional TopLink API support

Deprecated:

- MVC controller hierarchy
- JUnit 3.8 support
- Struts 1.x support
- several outdated helper classes

Other

- Scoped Bean Serializability
 - reobtain references on deserialization
- Scheduling Enhancements
 - **@Async**, cron-style triggers
- Backwards compatible with Spring 2.5
 - 100% programming model
 - 95% compatibility of extension points

Summary

- Java 5+
- Full-scale REST support
- Broad Unified EL++ support in the core
- Annotation-based model validation
- Remains backwards compatible

Roadmap

- 3.0 RC1: September 2009
 - feature-complete and fully documented
- 3.0 RC2: November 2009
- 3.0 GA: December 2009
- 3.1: Q2 2010 already
 - compatibility with Java EE 6
 - support for conversations