Spring: Paving The Way For Continuous Innovation

Jeremy Grelle

Agenda

- New And Cool Stuff in the Spring Ecosystem
  - Spring 3.0
  - Spring Integration
  - Spring BlazeDS
  - Spring Roo
- Putting It All Together
  - Demos

Spring Platform - Core Values

- *For Developers, by Developers*
- *Simplification*
- *Community*
- *Innovation*

Core Value: For Developers, By Developers

- We like code
  - We make it easier to work with code
- Avoiding the ivory tower
  - Spring team works with customers every day
- Practical solutions
  - It's only valuable if it works in the real world

Spring 3.0

It's almost here!!

(Finally!!!)

Spring 3 Themes

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

Spring 3 - Generics

```
public interface BeanFactory {

    <T> T getBean(String name, Class<T> requiredType);

    <T> Map<String, T> getBeansOfType(Class<T> type);
}
```


Spring 3 - Rearrangements

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

Spring 3 - Annotated Factory Methods

- Spring JavaConfig
  - configuration classes
  - annotated factory methods

Spring 3 - 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 {
    …
}
```


Spring 3 - SpEL in Bean Definitions

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

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

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

</bean>
```


Spring 3 - SpEL in Component Annotations

```
@Repository
public class RewardsTestDatabase {

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

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

Spring 3 - SpEL 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)

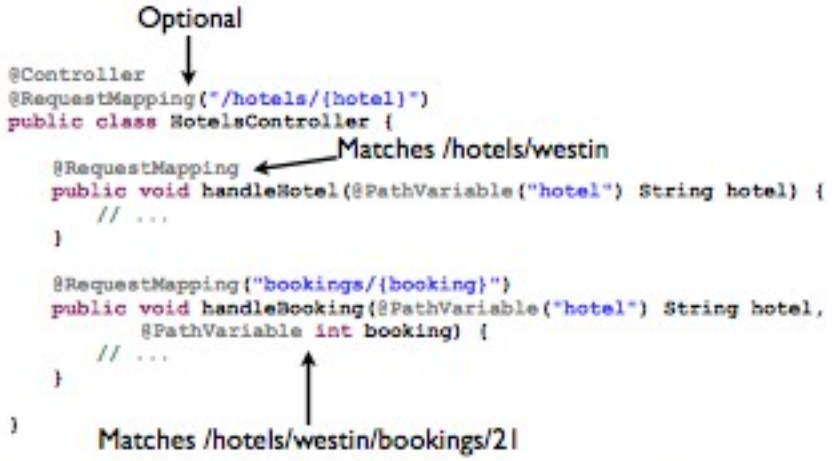
Web Context Attributes

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

Spring 3 MVC - 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

Spring 3 MVC @Controller - 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) {
        // ...
    }

}
```


Spring 3 - SpEL with Spring Security 3

```
@PreAuthorize("!isAnonymous() AND
               (#username == principal.username OR
               hasRole('ROLE_ADMIN'))")

@RequestMapping(value = "/{username}/preferences",
                method = RequestMethod.GET)

public void getPreferences(@PathVariable String username,
                           Model model) {

   List<Preference> preferences =
        travelService.findPreferences(username);
   model.addAttribute("model", new ModelHolder(preferences));

}
```


Spring 3 MVC Views - Resource Representation

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**

Spring 3 - RestTemplate

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

Spring 3 - 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);
```

Spring 3 - Declarative Validation

- Annotation-based
- Same metadata can be used for persisting, rendering, etc
- JSR-303 "Bean Validation" as the common ground

Spring 3 - Declarative Validation Sample

```
public class Reward {

    @NotNull

    @Past

    private Date transactionDate;

}
```

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

```
@RequestMapping(value = "/rewards", method = RequestMethod.POST)
public String addReward(@Valid Reward reward, BindingResult binding) {
```


Spring Integration

- An embedded Message Bus
  - Runs *within* any Spring ApplicationContext, *wherever Spring runs*
    - Allows integration without an ESB
  - All components are Spring-managed objects
    - Only integration framework that can take full advantage of Spring component model
- Also, an Application Integration Framework
  - Connects to other systems via adapters
  - File, JMS, HTTP, WS, Mail, UDB/TCP, Twitter...

Spring Blaze DS

- Bootstraps Adobe Blaze DS (Flex server) within a Spring environment
  - Allows Spring to manage Flex server-side components
- Direct remoting from Flex clients to Spring beans
- Easy integration with Spring Security, using regular Flex API
- Give Flex applications access to full power of Spring
  - Example: Asynchronous messaging with Spring Integration

Spring Roo

Mission Statement

"Roo's mission is to fundamentally and sustainably improve Java developer productivity without compromising engineering integrity or flexibility"

End User's Description

"Roo is a little genie who sits in the background and handles the things I don't want to worry about"

\W/

Easy To Learn and Use

- Usability you'll love
  - "hint" and "help" commands to guide you all the way
  - TAB completion for nearly everything
  - Command hiding and automatic contextual awareness
  - Even if you make a mistake, Roo will rollback changes!
- Scriptable UI that can even "replay" scripts
- You're in charge and Roo won't get in your way
  - Roo acts predictably and conservatively at all times

Familiar Libraries

- Roo builds on what you already know
  - Popular, proven, and mature APIs at your full disposal
- Java
- Spring Framework
- Java Persistence API (Hibernate)
- Java Server Pages
- Spring Security
- Spring Web Flow
- Log4J, Maven, AspectJ, Eclipse/STS...

Cloud-Class Scalability

- Zero performance overhead
  - Efficient, reflection-free code
  - No dynamic proxies, deferred compiling, LTW or similar
  - No Roo runtime means no runtime performance cost
- Zero memory overhead
  - No class creation means no Perm Gen memory issues
  - No Roo runtime means no runtime memory cost
- Zero deployment footprint
  - Roo apps leverage OSGi bundles out-of-the-box
  - No Roo runtime means no WAR footprint size

More Great Stuff

- No lock-in
  - Get rid of Roo from your project in under 10 minutes!
- Easy to write and deploy your own Roo add-ons
  - Roo can even write them for you (seriously!)
- Open source and SpringSource maintained
  - SpringSource-endorsed application architecture
- Consistent apps across your team and enterprise

Code Generation Approach

- Roo is a *hybrid* code generator
  - The best of the passive and active generation models
  - Achieved by use case-specific design decisions
- Passive generation
  - Use Roo shell to make Roo generate something
  - It's done and it's finished (usually .xml and .java files)
- Active generation (automatic round-tripping)
  - Builds detailed metadata model with help of @Roo*
  - *Incrementally* updates .aj, .jsp files - never .java files

Putting it together...

Demo

Resources

- http://www.springsource.org
- Spring Integration:
  - http://www.springsource.org/spring-integration
- Spring BlazeDS:
  - http://www.springsource.org/spring-flex
- Spring Roo:
  - http://www.springsource.org/roo