Building Next-Gen Web Applications with the Spring 3.0 Web Stack

Jeremy Grelle  (@jeremyg484)
Senior Member Technical Staff  (a.k.a, Open Source Web Dude)
SpringSource a division of VMware
Battling the rising complexity of web application development by building on a lightweight Spring foundation.
Next-Gen? - The Modern Web Dilemma

- Web application requirement complexity continues to rise
  - How do we manage this complexity?
  - How do we expose our services to the largest possible audience?
  - How do we give users the best possible experience in the shortest amount of time?
The Spring Web Stack - Simplicity and Power

- The Spring Web Stack gives you:
  - unified programming model
  - multiple client types served with the same foundation
  - adaptability - the right approach (i.e., stateless, stateful) for the given use case
The Spring Web Stack

- Spring Faces
- Spring BlazeDS Integration
- Spring Web Flow
- Spring JavaScript
- Spring Security
- Spring Framework and Spring MVC
Web Stack Components

- **Spring Framework and Spring MVC**
  - The foundation for all other components

- **Spring JavaScript**
  - Ajax integration

- **Spring Web Flow**
  - Framework for stateful web conversation
Web Stack Components (cont.)

- **Spring Security**
  - Security framework

- **Spring Faces**
  - Integration with JavaServer Faces via Web Flow

- **Spring BlazeDS Integration**
  - Integration with Adobe Flex clients
Next-Gen? - Productivity: More Important than Ever Before

- *Must* resolve first obstacle to Java in the cloud
  - Expectations are higher than ever before
  - Success of Ruby on Rails et al has raised the bar for building simpler applications
  - Developer choice crucial in determining cloud
    - Cloud apps often new
    - Often start simple
  - Java/JVM technologies perceived as complex
Increasing Productivity: The Opinionation Pyramid

- Ideal is to build on top of powerful, extensible layers
- No need to move to introduce new runtime
- Never hit a wall

Choice, Power

Grails/Roo

Spring

Servlet/other specifications

JVM

Opinionated, Productive
Grails

- The most popular rapid development framework for the JVM
- Solutions built on solid foundations
Spring Roo

- Higher Java productivity
- Familiar Java
  - Roo uses the Java APIs and standards you already know and trust.
- Usable and learnable
  - Roo features an extremely high level of usability and an advanced shell
- Trusted Spring Stack
  - Roo has no runtime – It’s just Spring
- Easy Roo removal
  - Roo can be easily removed from a user project in under five minutes.
Getting Started

- Create a new Spring MVC project from a template
  - Most use Roo to do this, either from an IDE like STS or the command-line

- Typical setup:

- One DispatcherServlet registered in `web.xml`
  - FrontController that dispatches web requests to your application logic
  - Generally the “default servlet” mapped to “/”

- Two Spring Containers (or ApplicationContexts) instantiated
  - 1 “root” context to host “shared resources” required by Servlets / Filters
  - 1 “web” context to host local application components delegated to by the DispatcherServlet
    - Your application components are typically discovered via classpath scanning
Demo
The Spring Web Stack

- Spring Faces
- Spring BlazeDS Integration
- Spring Web Flow
- Spring JavaScript
- Spring Security

Spring Framework and Spring MVC
Introduction to the MVC programming model

- DispatcherServlet requests are mapped to @Controller methods
  - @RequestMapping annotation used to define mapping rules
  - Method parameters used to obtain request input
  - Method return values used to generate responses

- Simplest possible @Controller:

```java
@Controller
public class SimpleController {

    @RequestMapping("/simple")
    public @ResponseBody String simple() {
        return "Hello world!";
    }
}
```
Mapping Requests

- **By path**
  - `@RequestMapping("path")`

- **By HTTP method**
  - `@RequestMapping("path", method=RequestMethod.GET)`
  - POST, PUT, DELETE, OPTIONS, and TRACE are also supported

- **By presence of query parameter**
  - `@RequestMapping("path", method=RequestMethod.GET, params="foo")`
  - Negation also supported: `params={ "foo", "!bar" }`

- **By presence of request header**
  - `@RequestMapping("path", header="content-type=text/*")`
  - Negation also supported
Mapping Requests (2)

- Simplest possible @Controller revisited

```java
@Controller
public class SimpleControllerRevisited {

    @RequestMapping(value="/simple/revisited",
                    method=RequestMethod.GET,
                    headers="Accept=text/plain")
    public @ResponseBody String simple() {
        return "Hello world revisited!";
    }
}
```
Request Mapping at the Class Level

- **@RequestMapping can be used at the class level**
  - Concise way to map all requests *within* a path to a @Controller

```java
@Controller
@RequestMapping("/accounts/*")
public class AccountsController {

    @RequestMapping("active")
    public @ResponseBody List<Account> active() {
        ...
    }

    @RequestMapping("inactive")
    public @ResponseBody List<Account> inactive() {
        ...
    }
}
```
Obtaining Request Data

- **Obtain request data by declaring method arguments**
  - A query parameter value
    - `@RequestParam("name")`
  - A group of query parameter values
    - A custom JavaBean with a `getName()/setName()` pair for each parameter
  - A path element value
    - `@PathVariable("var")`
  - A request header value
    - `@RequestHeader("name")`
  - A cookie value
    - `@CookieValue("name")`
  - The request body
    - `@RequestBody`
  - The request body and any request header
    - `HttpEntity<T>`
Injecting Standard Objects

- A number of “standard arguments” can also be injected
  - Simply declare the argument you need as a method param

- HttpServletRequest (or its more portable WebRequest wrapper)
- Principal
- Locale
- InputStream
- Reader
- HttpServletResponse
- OutputStream
- Writer
- HttpSession
Validation

- **Trigger validation by marking a JavaBean parameter as @Valid**
  - The JavaBean will be passed to a Validator for validation
  - JSR-303 auto-configured if a provider is present on the classpath

- **Binding and validation errors can be trapped and introspected by declaring a BindingResult parameter**
  - Must follow the JavaBean parameter in the method signature
  - Errors automatically exported in the model when rendering views
  - Not supported with other request parameter types (@RequestBody, etc)
Generating Responses

- Return a POJO annotated with @ResponseBody
  - POJO marshaled as the body of the response
  - Converted to a representation based on Accept header
  - Default converters auto-configured for JSON, XML, Atom, etc

- Return a logical view name
  - ViewResolver -> View layer kicks in
  - View abstractions for numerous templating/rendering technologies (i.e., JSP, Tiles, Freemarker, Excel, PDF, Atom, JSON, XML)
  - ContentNegotiatingViewResolver can chain ViewResolvers and select based on Accept header

- Return a new ResponseEntity<T> object
  - More powerful, low-level; allows for setting custom response headers and status
@RequestMapping
("/hotels")
public @ResponseBody List<Hotel> getHotels(@Valid SearchCriteria criteria) {
    return travelService.findHotels(criteria);
}

@RequestMapping
("/hotels/{id}")
public @ResponseBody Hotel getHotel(@PathVariable Long id) {
    return travelService.findHotelById(id);
}

@RequestMapping(value = "/hotels", method = RequestMethod.POST)
public String addHotel(@Valid Hotel hotel) {
    hotel = travelService.addHotel(hotel);
    return "redirect:/hotels/" + hotel.getId();
}
Pluggability

- Spring MVC allows you to customize just about any part of the request chain
  - HttpMessageConverters
  - HandlerInterceptors
  - ExceptionHandlers
  - ViewResolvers
  - HandlerMappings
  - HandlerAdapters
  - WebArgumentResolvers
  - TypeConverters
  - Validators
  - ...and more
Demo
The Spring Web Stack

- Spring Faces
- Spring BlazeDS Integration
- Spring Web Flow
- Spring JavaScript
- Spring Security
- Spring Framework and Spring MVC
Goals of Spring JavaScript

- Encapsulate use of Dojo for common enterprise use cases
  - Ajax (fragment rendering support)
  - Client-side validation

- Promotes progressive enhancement
  - Robust in the face of JavaScript failure
  - Maximizes potential audience
  - Accessibility
Working with the Spring JavaScript API

- Use API to apply decorations to HTML elements

- Different types of decorations
  - WidgetDecoration
  - AjaxEventDecoration
  - ValidateAllDecoration
Ajax with Partial Rendering

```html
<a id="moreResultsLink" href="search?q={$criteria.q}&page={$criteria.page+1}"
   onclick="Spring.addDecoration(new Spring.AjaxEventDecoration({
     elementId: "moreResultsLink",
     event: "onclick",
     params: {
       fragments: "searchResults"
     }
   }));">
  More Results
</a>
<script type="text/javascript">
  Spring.addDecoration(new Spring.AjaxEventDecoration({
    elementId: "moreResultsLink",
    event: "onclick",
    params: {
      fragments: "searchResults"
    }
  }));
</script>
```

Name of tile to re-render on server

No callback function necessary to link in response
<form:input path="creditCard"/>
<script type="text/javascript">
    Spring.addDecoration(new Spring.ElementDecoration({
        elementId: "creditCard",
        widgetType: "dijit.form.ValidationTextBox",
        widgetAttrs: {
            required: true,
            invalidMessage: "A 16-digit number is required.",
            regExp: "[0-9]{16}"
        }
    }));
</script>
Future of Spring JavaScript

- Resource Handling and Partial Rendering ideas will be incorporated directly into Spring 3.1

- Client-side bits will be incorporated into Spring Roo
  - Much better platform for realizing some of the original ideas (i.e., generation of client-side validation based on Java model)
    - Can support multiple libraries through Spring Roo Addons

- Will continue to support the API in Spring Web Flow releases
The Spring Web Stack

- Spring Faces
- Spring BlazeDS Integration
- Spring Web Flow
- Spring JavaScript
- Spring Security
- Spring Framework and Spring MVC
Spring Web Flow

- For implementing stateful flows
  - Reusable multi-step user dialogs

- Plugs into Spring MVC

- Spring Web Flow 2 available now
  - Incorporates lessons learned from 1.0
  - Offers many new features
Web Flow Sweet Spot
New Web Flow 2 Features

• Ajax support
  • Partial page re-rendering in flow DSL

• Spring security integration

• Flow-managed persistence

• Convention-over-configuration
  • View rendering
  • Data binding and validation
Spring Web Flow 3 - @Flow

- Extends @Controller model
- Define stateful UI flow control using plain Java
- Builds on Spring Web Flow infrastructure
@Flow Example

@Flow
public class BookHotel {

    private Booking booking;

    @Autowired private transient BookingService booking;

    public State start(@Input Long hotelId, Principal user) {
        booking = bookingService.createBooking(hotelId, user);
        return new EnterBookingDetails();
    }

    private class EnterBookingDetails extends ViewState {
        @Transition
        State next() { return new ReviewBooking(); }
    }

    private class ReviewBooking extends ViewState {}
}
The Spring Web Stack

- Spring Faces
- Spring BlazeDS Integration
- Spring Web Flow
- Spring JavaScript
- Spring Security
- Spring Framework and Spring MVC
Spring BlazeDS Integration

- Spring’s Adobe Flex integration project
  - Connects Flex clients to Spring-managed services
    - Using BlazeDS MessageBroker boot-strapped by Spring
  - Makes Flex natural fit in a Spring environment
    - Uses Spring XML namespace for simplified setup
    - Reduces the need for BlazeDS-specific config
Spring BlazeDS Integration

• Expose Spring beans as Remoting destinations (for RPC style interaction)
  • `<flex:remoting-destination>` or `@RemotingDestination`

• Message-style communication through native BlazeDS messaging, JMS, and Spring Integration channels
  • enables server-side push

• Integrate Spring Security to secure Flex apps
  • `<flex:secured>`
Demo
The Future

- Other ideas being considered for Spring 3.1
  - Incorporation of Servlet 3.0 features (auto-config, async servlets, etc.)
  - HTML5 support in the JSP tag library
  - Offline caching support
  - Cometd and WebSocket support
  - Server-sent events

- SpringOne 2010 in Chicago (www.springone2gx.com)
  - See lots more content on Spring Roo, Spring MVC, and deep-dives into mobile and social integration
Resources

- http://www.springsource.org
- http://www.springsource.org/roo
- http://www.springsource.org/webflow
- http://www.springsource.org/spring-flex
- http://blog.springsource.com/2010/07/22/spring-mvc-3-showcase
Thank you!