Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Spring Framework 3.0: New and Notable

Rod Johnson
SpringSource

Agenda

> Continuing the work of Spring 2.5
> Spring 3.0 new features
> Implications for best practice
> Spring beyond Spring Framework

BUILDING ON SPRING 2.5

Spring 2.5 theme: Simplification!

- Annotation based DI model
  - Ability to define Spring-managed objects without XML
  - Support for JSR-250: “Common Annotations for the Java Platform”
- Overhaul of Spring MVC
  - Annotation model replaces concrete inheritance to define MVC interaction

Spring Framework 2.5: Annotation based DI

- Comprehensive support for ***annotation-based configuration***
  - @Autowired (+ @Qualifier or custom qualifiers)
  - @Transactional
  - @Component, @Service, @Repository, @Controller
- Common **Java EE 5** annotations supported
  - @PostConstruct, @PreDestroy
  - @PersistenceContext, @PersistenceUnit
  - @Resource, @EJB, @WebServiceRef
  - @TransactionAttribute

Annotated Bean Component

```
@Service
public class RewardNetworkService
            implements RewardNetwork {

  @Autowired
  public RewardNetworkService(AccountRepository ar) {
    ...
  }

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

Annotated class with Lifecycle Methods

```
@Repository
public class JdbcAccountRepository
                        implements AccountRepository {

   @Autowired
   public JdbcAccountRepository(DataSource ds) { … }

   @PostConstruct
   public initCache() { … }

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

Minimal XML Bean Definitions

- **Spring no longer *requires* XML**
- Need to use XML only when you need to externalize something

```
<!-- Activating annotation-based configuration -->
<context:annotation-config/>

<!-- Just define beans – no constructor-arg/property -->
<bean class=”com.myapp.rewards.RewardNetworkImpl”/>

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


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

Minimal XML Bootstrapping

```
<!--
   // Scans for:
   //    @Component, @Service, @Repository, @Controller
   //    (and custom annotations) and deploys automatically
   // No user bean definitions at all!
   -->
<context:component-scan
    base-package="com.myapp.rewards"/>
```

Automatically picked up and injected

XML (optional) can supplement

```
@Service
public class RewardNetworkService
           implements RewardNetwork {

  @Autowired
  public RewardNetworkService(AccountRepository ar) {
    …
  }
```

Spring Servlet MVC 2.5

```
@Controller
public class BookController {

  private final BookService bookService;

  @Autowired
  public MyController(BookService bookService) {
    this.bookService = bookService;
  }
  // Responds to URL http://host/servlet/book/removeBook
  @RequestMapping
  public String removeBook(@RequestParam("book") String bookId) {
    this.bookService.deleteBook(bookId);
    return "redirect:myBooks";
  }
```

> Annotations replace Controller interface and framework superclasses

@RequestMapping Method signature conventions

```
@RequestMapping
  public String removeBook(
                    @RequestParam("book") String bookId,
                    HttpServletRequest req) {
      this.bookService.deleteBook(bookId);
      return "redirect:myBooks";
  }
```

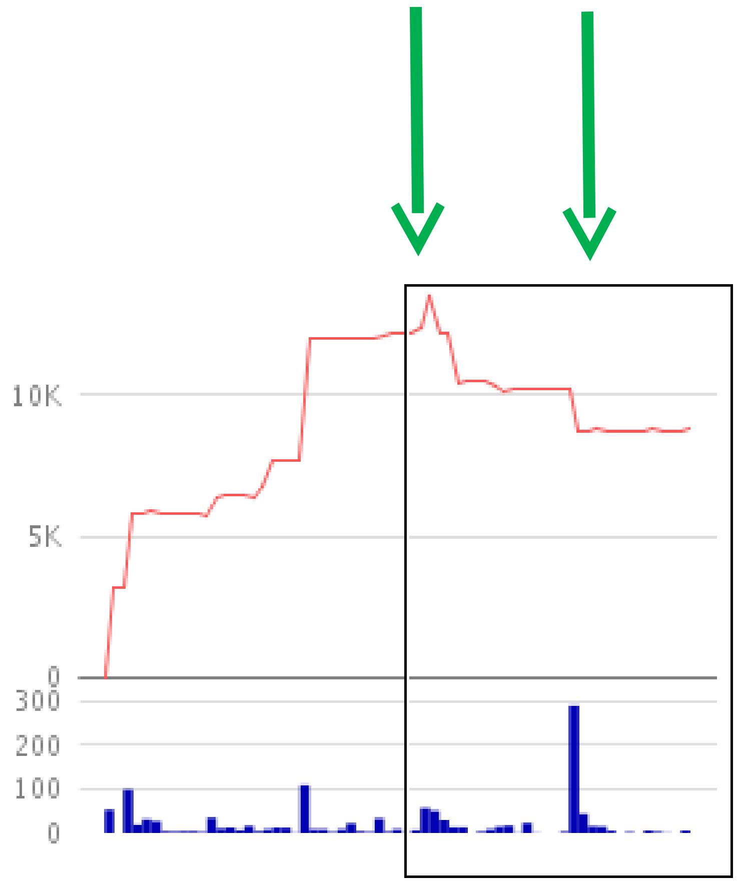
- Builds on Spring MVC fundamentals
- Return type rules:
  - String -> logical view name
  - ModelAndView: just like traditional Controller classes
  - Null -> the response will have been written to Servlet OutputStream
- Parameter processing rules:
  - Well known parameters such as Servlet request automatically processed
  - Binds parameters with @RequestParam

Spring MVC Best Practice Changes

- Do *not* use old Controller interface, SimpleFormController and friends
  - Annotation model is simply superior for MVC
  - Keeps everything good about Spring MVC
  - Old model will eventually be removed
- No need to use XML bean definitions for @Controllers
  - Rely on annotation scanning
  - If a controller is so complex that annotation injection isn’t enough, web tier has too much responsibility!

Ongoing Simplification…

- Each version of Spring has made Spring applications simpler
- Pet Clinic sample LOC stats, showing reduction due to Spring 2.0 and 2.5



Spring 3.0 goals

- Spring Framework becomes Java 5+ only
  - Enables us to use new language features *everywhere*
- Simplify/eliminate Spring configuration
  - Extend annotation support from 2.5
  - Add Expression Language (EL)
- Introduce comprehensive REST support
- Continue MVC improvements

Java 5+

- Users still on 1.4 or below should stay on 2.5.6
- Generification of internal APIs

```
Object getBean(String name)
```

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

```
Map<String, T> getBeansOfType<Class<T> type)
```

```
ApplicationListener<E>
```

```
TaskExecutor --> java.util.concurrent.Executor
```

KEY NEW FEATURES IN 3.0

Key New Spring 3.0 Features

- Spring EL
- Further Spring MVC improvements
- REST support
- Spring Java Configuration

Powerful Spring EL Parser

- **Custom expression parser implementation** shipped as part of Spring 3.0
  - package org.springframework.expression
  - next-generation expression engine inspired by Spring Web Flow 2.0's expression support
- Compatible with Unified EL but significantly more powerful
  - navigating bean properties, maps, etc
  - method invocations
  - construction of value objects

EL in XML 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.favoriteColor}")
   private String favoriteColor;

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

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

}
```

EL in Component Annotations (2)

```
@Repository
public class RewardsTestDatabase {
   @Value(“#{systemProperties.favoriteColor}”)
    private String favoriteColor;

    @Autowired
    public void init(@Value(“#{systemProperties.databaseName}”)
                                        String dbName,
                   @Value(“#{strategyBean.timeout}”
                                        int timeout) { … }
}
```

EL Context Attributes

- Example showed **access to EL attributes**
  - "systemProperties", "strategyBean"
- **Implicit attributes** exposed by default, depending on runtime context
  - e.g. "systemProperties", "systemEnvironment"
  - access to all Spring-defined beans by name
  - extensible through Scope SPI
    - e.g. for step scope in Spring Batch 2.0

EL and Best Practice

- Makes all forms of configuration more concise
- Makes annotation model much more powerful
  - XML needed in fewer cases
- Avoids need to recompile Java code when a simple configuration value changes
- Will become default EL for Spring Web Flow
- Will simplify use of other projects such as Spring Integration

REST Support

- Spring MVC now provides 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/show/12345

```
@RequestMapping(value = "/show/{id}", method = GET)
public Reward show(@PathVariable("id") long id) {
    return this.rewardsAdminService.findReward(id);
}
```

Similar to *@RequestParam*, but from URL path

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**

Other @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) {
    ...
}
```

@MVC Extensibility

> Ability to register and handle custom annotations

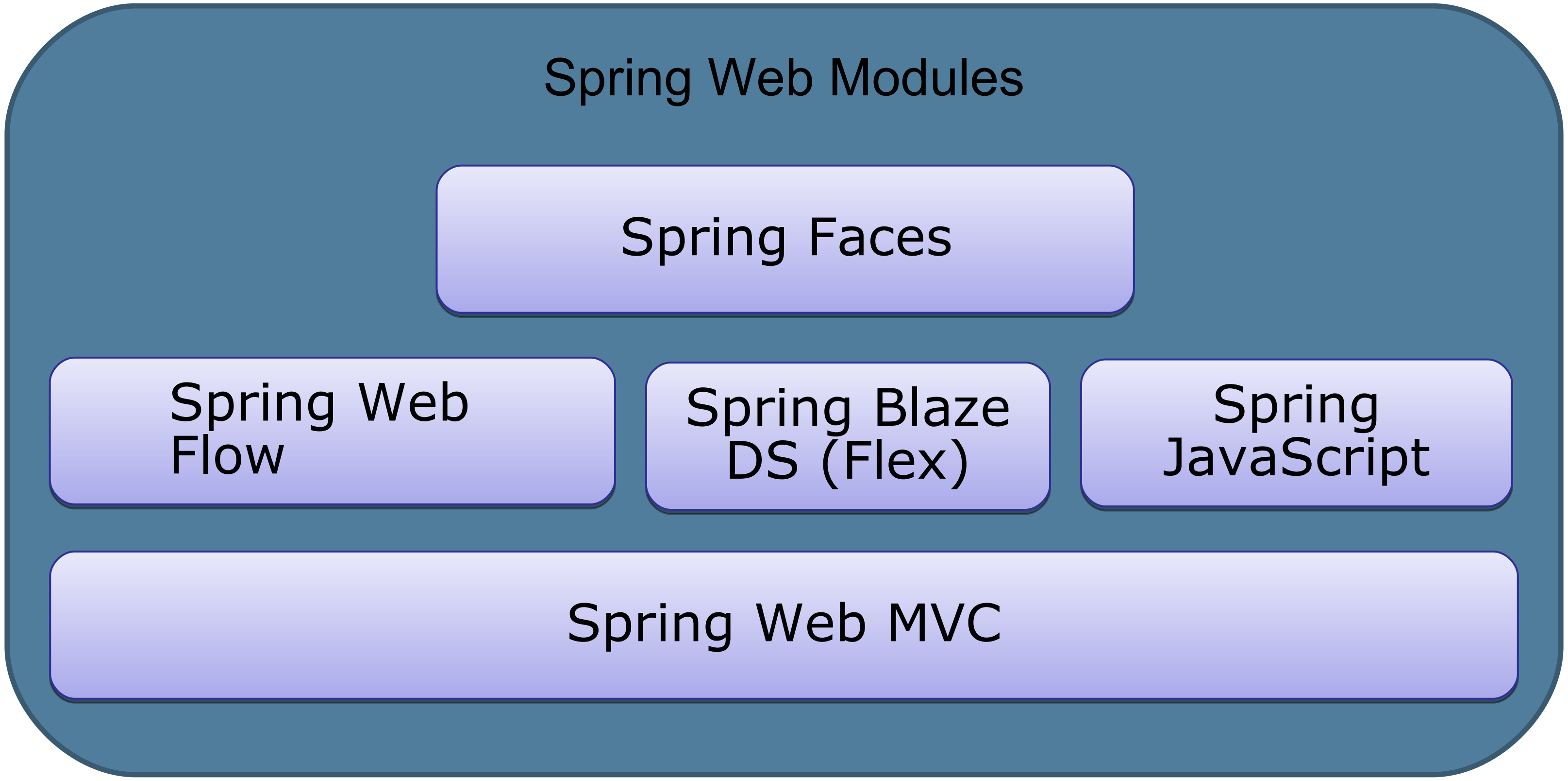
```
@RequestMapping("/show")
public Reward show(@RequestHeader("region") long regionId,
      @CookieValue("language") String langId,
      @MyMagicContextValue Magical m) {
    ...
}
```

Can teach Spring to
handle new annotations

Understanding the Spring Web Stack

- Spring Web MVC
  - foundation for all other web modules
- Spring JavaScript
  - AJAX support library
- Spring Web Flow
  - framework for stateful conversations
- Spring Faces
  - JavaServer Faces support library

Layered, Integrated Web Modules



Our Philosophy, and an Admission

- We didn’t offer a single, easy web technology in the past
- We have made enormous progress in this area
- We know there are many choices in the web tier
  - Traditional template oriented MVC
  - JSF
  - RIA…
- Fundamental Architectural Philosophy: **Build on a strong service layer**
- Today, Spring has a compelling story whatever your preferences in the web tier

Spring Java Configuration

- Annotation-centric approach, but unique
  - Annotations are in dedicated configuration classes, *not* application classes
  - Preserves centralized configuration model of XML
- Allows objects to be created and wired in Java
- **Essentially a Java DSL for configuration**
- Research project since 2005
- Now moves to Spring Framework core

@Configuration

- A configuration class is similar to a **`<beans/>`** document
- Specifies a configuration class that creates beans
- Defines defaults for the current context
- A Spring-managed object itself
  - Can be injected
  - Can be picked up by component scanning without needing an XML bean definition

`@Bean`

- Analogous to **`<bean>`**
- Indicates a bean creation method

Example configuration class

```
@Configuration
public class AppConfig {

    @Value("#{jdbcProperties.batchSize}") int batchSize;

    @Autowired private DataSource dataSource;

    @Bean
    public FooService fooService() {
        return new FooServiceImpl(fooRepository());
    }

    @Bean
    public FooRepository fooRepository() {
        return new JdbcFooDao(
                dataSource, batchSize);
    }

}
```

Identifies Configuration class – Could be automatically picked up

Configuration class is injected itself

Method defines a bean

Bean to bean dependency

Another Way of Thinking About It

```
@Configuration
public class AppConfig {

    @Value("#{jdbcProperties.batchSize}") int
    batchSize;

    @Autowired private DataSource dataSource;

    @Bean
    public FooService fooService() {
            return new FooServiceImp(
                                fooRepository());
    }
    @Bean
    public FooRepository fooRepository() {
            return new JdbcFooDao(
                        dataSource, batchSize);
    }

}
```

```
<bean id = “fooService” class=“…FooServiceImpl”>
    <constructor-arg ref=“fooRepository” />
</bean>
```

```
<bean id=“fooRepository”>
    <constructor arg ref=“dataSource” />
    <constructor arg value=“${jdbcProperties.batchSize)” />
</bean>
```

Bean to bean References

```
@Configuration
public class AppConfig {
    ...

    @Bean
    public FooService fooService() {
        return new FooServiceImpl(
                fooRepository()
        );
    }

    @Bean
    public FooRepository fooRepository() {
        ...
}
```

- Has same effect as:

```
@Bean
public FooService fooService() {
    return new FooServiceImpl(
        appCtx.getBean("fooRepository")
    );
}
```

- Handles scope (prototype/singleton/custom) the same way, via container

If you're Creating Objects in Java, Why use Spring?

- Spring Java Configuration is Java object creation on steroids
  - Spring configures configuration objects themselves
  - Spring manages all created objects
  - Still get all Spring enterprise features
    - Declarative enterprise services like transaction management
    - AOP capabilities
    - Benefit from all Spring extension points
  - Benefit from Spring's ability to externalize configuration

Advantages of Spring Java Configuration

- Type safe
- Allows inheritance of configurations
  - Including abstract configuration methods
- Allows creation of objects using arbitrary method calls
- Robust bean-to-bean dependencies
  - No dependence on String names

Java Configuration versus Annotation driven injection

- Complementary, not mutually exclusive
- Java configuration classes benefit from Annotation injection
  - @Autowired
  - Lifecycle annotations

- Spring Java Configuration is a DSL for Configuration – *Annotations go in configuration*
- Spring Annotation DI, like EJB3, Guice and other annotation approaches, *annotates components being configured*

Best Practice: Annotations vs XML

> Only need to use XML when you need to externalize configuration from Java code

> Best practice
- Use XML for generic classes
  - Classes that will be used multiple times, configured differently
  - Often, classes that you didn’t write and can’t annotate
  - DataSources etc.
- Use component scanning and annotations (no XML) for most application objects

SPRING BEYOND SPRING FRAMEWORK

Other Spring Projects

- Spring Web Flow
- Spring Security
- Spring Batch
- Spring Integration
- Spring Web Services
- Spring Blaze DS
- Spring LDAP
- Spring Dynamic Modules
- Grails
- …

- Spring Roo
  - New!

Spring Roo Mission Statement

- What?
  - Roo's mission is to dramatically improve Java developer productivity without compromising power or flexibility
- How?
  - Round tripping code generator that enables rapid delivery of robust high performance enterprise Java applications

Create and Work on Spring Projects in a Fraction of the Time

> Java focus
  - For developers who want to work directly with the Spring programming model, using Java

> Promotes best practice

> Eliminates the busywork of creating projects

> Continues to add value throughout the project lifecycle
  - Sophisticated round tripping
  - Allows modification of code outside Roo

JavaOne

Example Shell Usage

- Tab completion saves most keystrokes from being typed
- "hint" and command visibility hiding assists new users
- In 76 keystrokes build a full application with MVC and passing tests

```
~/petclinic> roo
roo> create project org.springframework.petclinic
roo> install jpa HIBERNATE
roo> create entity class ~.domain.Pet
roo> create entity field string -fieldName name -notNull -lengthMax 40
roo> create controller class ~.web.ClinicController
roo> quit
~/petclinic> mvn test install
```

- If you prefer a GUI, our free STS IDE offers full ROO support!

Sun microsystems

.java File Structure

> Previous slide produced several small .java files (full code shown):

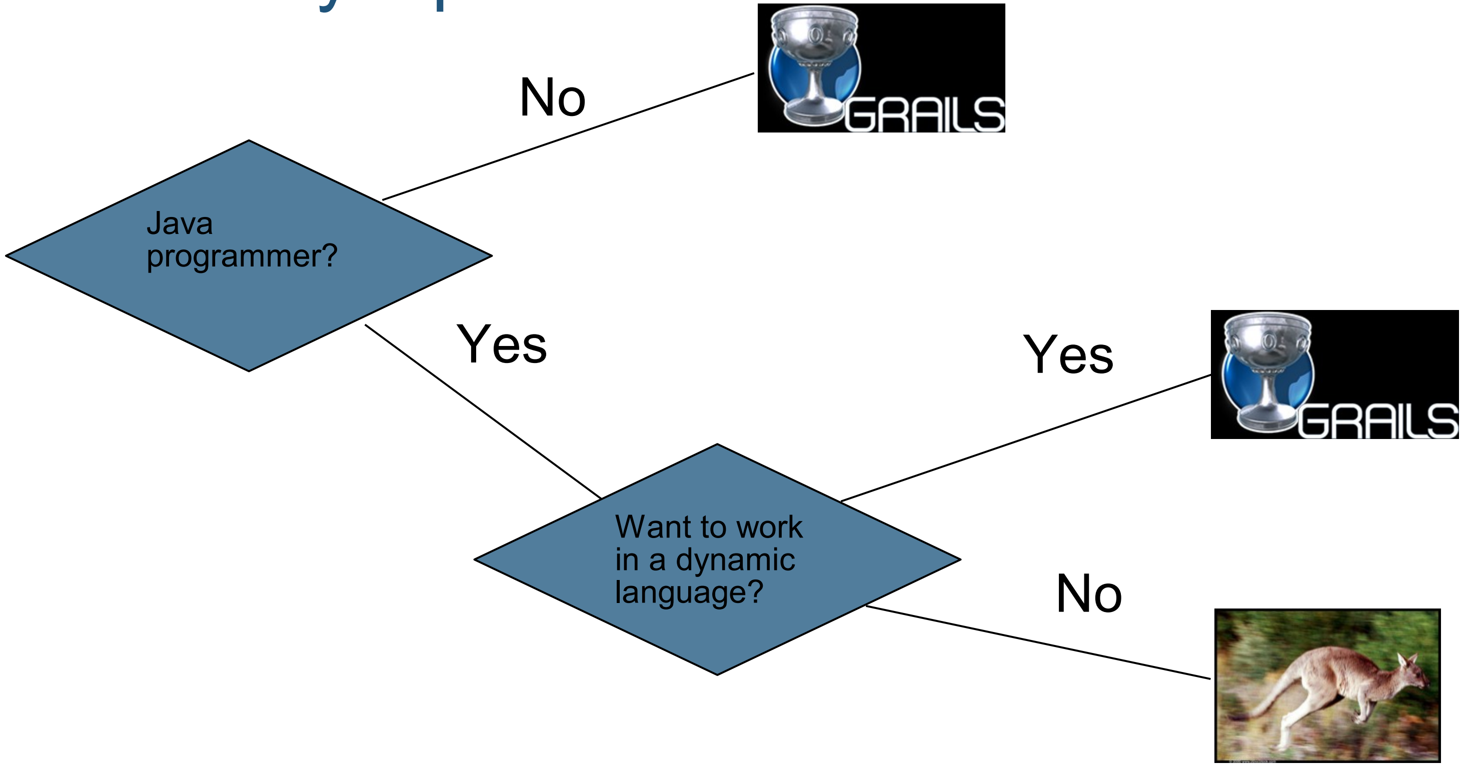
```
@RooEntity
@RooJavaBean
public class Pet {
  @NotNull
  @Length(min=0, max=40)
  private String name;
}
```

```
@RooIntegrationTest
public class PetIntegrationTest {
  @Test
  public void testMarkerMethod() {}
}
```

```
@RooDataOnDemand
public class PetDataOnDemand {}
```

```
@RooWebScaffold(automaticallyMaintainView = true, entity = Pet.class)
@Controller
@RequestMapping("/clinic/*")
public class ClinicController {}
```

Grails and Roo: Choosing Between Two Great Productivity Options



Whatever you want to do, the days of creating projects by hand are over – Bye Bye Boilerplate

Conclusion

- Spring 3.0 continues work of Spring 2.5 toward simplifying Spring configuration
  - Extensive use of Java 5+ language features
  - New EL
  - REST support
  - MVC enhancements
  - Java Configuration

Conclusion

- Many Spring projects beyond Spring Framework
  - Provided an integrated, consistent solution to enterprise Java problems
- Shared focus on enhanced productivity
- New Spring Roo project takes Java productivity to a new level

To Learn More

- Spring Framework
  - www.springframework.org
- Spring Projects
  - www.springsource.org
- Spring Roo
  - www.springsource.org/roo

JavaOne℠

Thank You

Rod Johnson
rod@springsource.com

SpringSource