Spring Today and Tomorrow

Rod Johnson

Founder, Spring
CEO, SpringSource

Agenda

- **Quick Review: Spring 2.5**
- Spring 3.0 Themes and Features
- Spring 3.0 Roadmap
- The Big Picture

Spring Framework 2.5

- Comprehensive support for ***annotation-based configuration***
  - @Autowired (+ @Qualifier or custom qualifiers)
  - @Transactional
  - @Component, @Service, @Repository, @Controller
- Common **Java EE 5** annotations supported too
  - @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 DAO with Lifecycle


```
@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"/>
```

Resolving Dependencies: @Autowired in Detail

- Injection at constructor/field/method level
- Supports multi argument methods
  - Concise
- Default behavior is Spring's traditional *autowire by type*
- Annotations make *autowiring* more useful

```
@Autowired
public void createTemplates(DataSource ds,
          ConnectionFactory cf) {
  this.jdbcTemplate = new JdbcTemplate(ds);
  this.jmsTemplate = new JmsTemplate(cf);
}
```

@Qualifier Annotation

- Autowiring by type may have too many candidates
- Provide hints using *qualifiers*
  - **`@Qualifier`** annotation
  - Can be used on fields / parameters or on custom annotations

Resolution of dependencies by name


```
public class JdbcOrderRepositoryImpl
    implements OrderRepository {

    @Autowired
    public void init(
        @Qualifier("myDS")
        DataSource orderDataSource,
        @Qualifier("otherDS")
        DataSource inventoryDataSource,
        MyHelper autowiredByType) {
        // ...
}
```

Resolution of dependencies by annotation


```
public class JdbcOrderRepositoryImpl
  implements OrderRepository {

  @Autowired
  public void setOrderServices(
   @Emea OrderService emea,
   @Apac OrderService apac) {
     // ...
}
```

Association of injection target with annotation: By annotation


```
@Emea
public class EmeaOrderService
    implements OrderService {

    ...
}
```

```
@Qualifier
@Component
public @interface Emea {
}
```

```
@Apac
public class ApacOrderService
    implements OrderService {
    ...
}
```

```
@Qualifier
@Component
public @interface Apac{
}
```

Association of injection target with annotation: XML

```
<bean class="example.EmeaOrderService">
   <qualifier type="example.Emea"/>
   <!-
      ...
        EmeaOrderService need not be annotated
     -->
</bean>

<bean class="example.ApacOrderService">
   <qualifier type="example.Apac"/>
   <!-- inject any dependencies required by this
  bean -->
</bean>
```

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";
    }
```

Test Context Framework

- Escape JUnit 3 concrete inheritance hell

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("test-config.xml")
public class RewardSystemIntegrationTests {

    @Autowired
    private RewardNetwork rewardNetwork;

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

Agenda

- Quick Review: Spring 2.5
- **Spring 3.0 Themes and Features**
- Spring 3.0 Roadmap
- Groovy and Grails

Spring 3.0 Themes

- **Java 5+** foundation
  - compatible with J2EE 1.4 and Java EE 5
- Spring **Expression Language**
  - Unified EL++
- Comprehensive **REST support**
  - and other Spring @MVC additions
- Support for **Portlet 2.0**
  - action/event/resource request mappings
- Declarative **model validation**
  - Hibernate Validator, JSR 303
- Early support for **Java EE 6**
  - JSF 2.0, JPA 2.0, etc

New Project Layout

- **Framework modules revised**
  - now managed in Maven style
  - one source tree per module jar
    - spring-beans.jar, spring-aop.jar, etc
  - **no spring.jar anymore!**
- Built with new Spring build system as known from Spring Web Flow 2.0
  - consistent deployment procedure
  - consistent dependency management
  - consistent generation of OSGi manifests

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 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"
    - global platform context
  - access to all Spring-defined beans by name
    - similar to managed beans in JSF expressions
  - extensible through Scope SPI
    - e.g. for step scope in Spring Batch 2.0

Web Context Attributes

- **Implicit web-specific attributes** exposed by default as well
  - "contextProperties": web.xml init-params
  - "contextAttributes": ServletContext attributes
  - "request": current Servlet/PortletRequest
  - "session": current Http/PortletSession
- Exposure of all **implicit JSF objects** when running within a JSF request context
  - "param", "initParam", "facesContext", etc
  - full compatibility with JSF managed bean facility

REST Support

- Spring MVC to provide 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**

@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) {
   ...
}
```

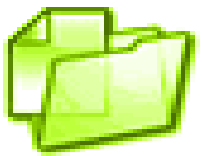
@MVC Extensibility

- Ability to register and handle custom annotations

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

Project Rearrangements

- Spring 3.0 will include a revised version of the **Object/XML Mapping (OXM)** module
  - known from Spring Web Services
  - also useful e.g. for SQL XML access
- Spring 3.0 will also feature **revised binding and type conversion** infrastructure
  - including the capabilities of Spring Web Flow's binding
  - stateless type converter objects with EL integration
- Spring 3.0 will include the core functionality of Spring **JavaConfig**
  - configuration classes defining managed beans



Spring Java Configuration

- Annotation-centric approach, but unique
  - Annotations are in dedicated configuration classes, *not* application classes
  - Preserves centralized configuration model of XML
    - Indeed, *stronger* centralization than with XML
- Allows objects to be created and wired in Java
- Research project since 2005
- Available in milestone form as a separate project since 2007
- Core functionality moves to Spring Framework in 3.0

@Configuration

- A configuration class is similar to a **<beans/>** document
- Specifies a configuration class that creates beans
- Defines defaults for the current context

```
@Configuration(
  defaultAutowire = Autowire.BY_TYPE,
  defaultLazy = Lazy.TRUE)
```

@Bean

- Analogous to **<bean>**
- Indicates a bean creation method
- Supports standard bean attributes from BeanDefinition internal metadata
  - lazy
  - scope
  - depends-on
  - …

@Bean

```
@Bean (scope = REQUEST)
public Page currentPage() { … }


@Bean (scope = SESSION,
   destroyMethodName = “shutdown”);
public Preferences prefs() { … }


@Bean (lazy = Lazy.FALSE);
public Admin admin() { … }
```

Java Configuration Class Example

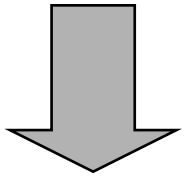
```
@Configuration
public abstract class JavaConfig {

    @Autowired
    private DataSource dataSource;

    @Bean
    public AccountDAO accountDAO() {
        // return new InMemoryAccountDAO();
        JdbcAccountDAO dao = new JdbcAccountDAO();
        dao.setDataSource(dataSource);
        dao.init();
        return dao;
    }
    @Bean
    public AccountService accountService() {
        DefaultAccountService service = new DefaultAccountService();
        service.setAccountDAO(accountDAO());
        return service;
    }
```

Method creates a bean

```
<bean name="accountDAO"
    class="...JdbcAccountDao">
    <property  name="dataSource"
               ref="dataSource" />
</bean>
```

Bean-to-Bean Dependencies handled elegantly, with correct lifecycle semantics

```
@Bean
public AccountDAO accountDAO() { ... }
...
service.setAccountDAO(accountDAO());
```

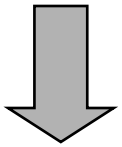


```
service.setAccountDAO(
        ctx.getBean("accountDAO"));
```

Referencing external beans

- Easy way to reference external beans using Spring 2.5 annotation-driven injection
- Strongly typed

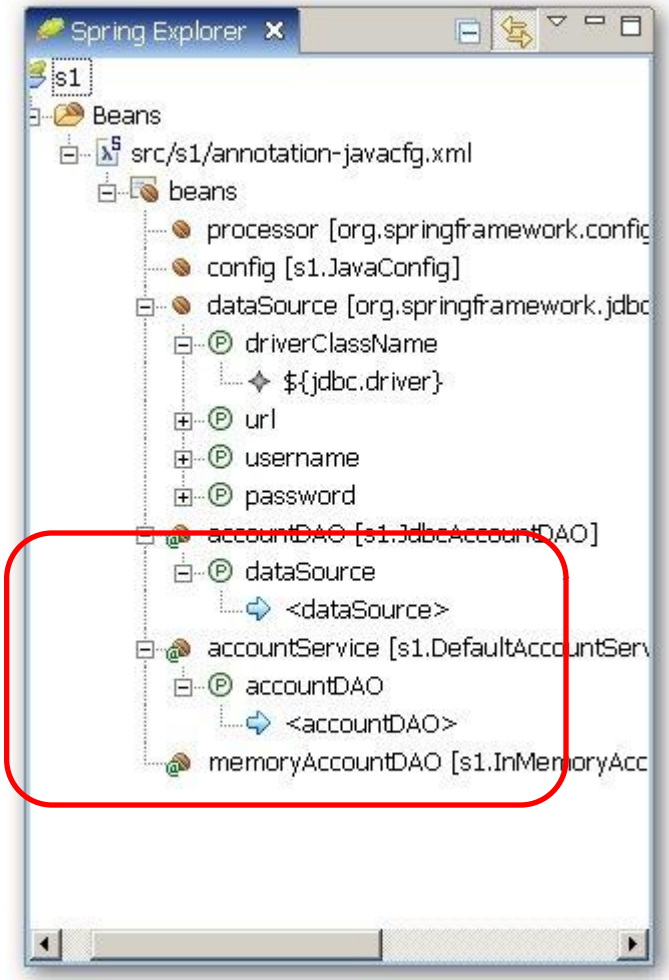
```
@Autowired
private DataSource dataSource;
```



```
public DataSource dataSource() {
     return (DataSource) ctx.getBean("dataSource");
}
```

Java Configuration Benefits

- Pure Java
  - Allows visibility control
  - Allows use of inheritance in configurations
- Powerful object creation
  - Ability to use arbitrary Java code
  - Good for configuring existing classes
- Refactoring friendly
- Strongly typed
- Preserves valuable application blueprint
- IDE support with Spring IDE



Annotation configuration vs Spring Java Configuration

- Different philosophies
  - Annotation driven injection *adds metadata to container identifying components and injection methods*
  - Java Configuration is *programmatic object creation*
    - Essentially a Java DSL for bean creation

Pruning & Deprecation in 3.0

- Some **pruning** planned
  - Commons Attributes support
  - traditional TopLink API support
    - in favor of JPA (EclipseLink)
  - subclass-style Struts 1.x support
- 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
  - several outdated helper classes

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 classes using 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!

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 Framework 3.0 M3** released at the end of March
  - With Java Config features
- **Spring Framework 3.0 RC1** scheduled for early May
  - after two further milestones
- **Spring Framework 3.0 final** expected in June
  - depending on RC feedback

Agenda

- Quick Review: Spring 2.5
- Spring 3.0 Themes and Features
- Spring 3.0 Roadmap
- **The Big Picture**

Productivity/Operational challenges

- Enterprise Java productivity has greatly improved from the bad old (pre-Spring) days
- …but there is still more to do
- Big vendors like IBM/Sun/Oracle have never really understood the problem and cannot solve it
  - At one level, big vendors *need* a certain amount of complexity to exclude competitors and justify the costs of acquiring and using their technologies
- Committees of vendors (JCP) have even less of a chance

Key Problem: Vendor/project fragmentation

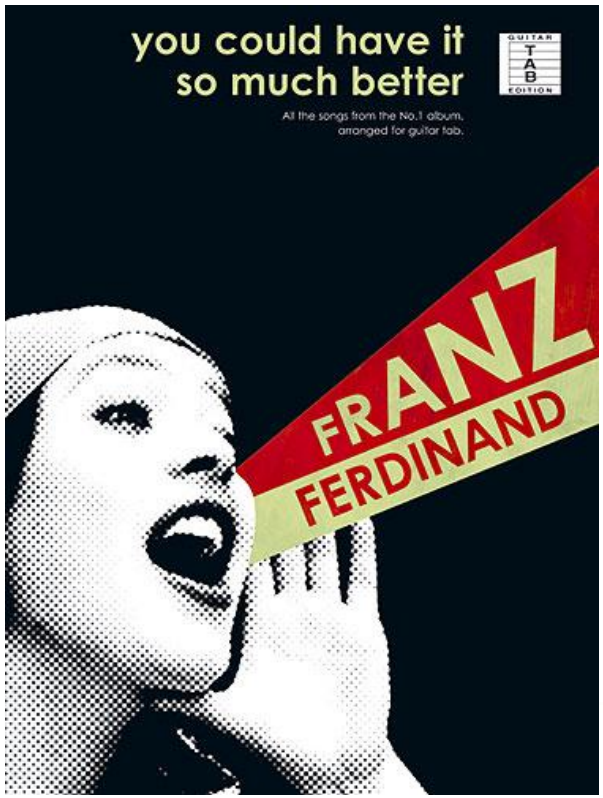
- Technical problem - Different sources for:
  - Tooling
  - Build solution
  - RAD solution
  - Frameworks and libraries
  - Servers

  - *… that developers actually want to use*
- Business problem:
  - Need one throat to choke

What happens if no one cares about the whole picture?

At SpringSource, we understand the problem, and we care

- In 2009 you will see us deliver a joined up strategy for
  - Software distribution
  - Tooling
  - RAD
  - Build
  - Servers
  - …
  - All kinds of workloads
  - Virtualization/Cloud



We also *know* the problem can be solved…

- We believe that enterprise Java has not yet reached its potential
- We are very bullish about the Java platform
- Conditions for success:
  - Developer empowerment
  - Open source projects and vendors
  - The decline of the influence of the JCP bureaucracy and design by committee
  - Lightweight infrastructure

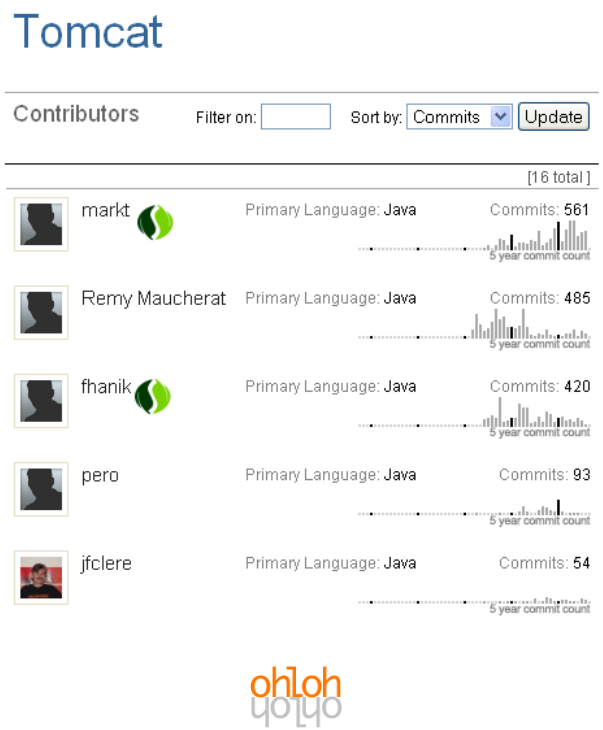
SOME OF OUR KEY INITIATIVES

SpringSource's wider role

- Some of you may still think of us as a framework vendor but today we are much more
- Spring has always spread into new areas and always demonstrated value
- SpringSource is broader still than Spring
- **Spring is central to the solution, but the next level of simplification can only be delivered through managing the whole stack**

SpringSource and Tomcat

- Today SpringSource employees are the leading contributors to Tomcat
- SpringSource is the leading provider of Tomcat support
  - Last 2 years
    - 83% of project commits
    - 96% of bug fixes



Tomcat: A Market Phenomenon

- By far the most popular application server today, in development and production
- Used by around 70% of organizations developing Java web applications

- Represents a developer-driven switch away from complexity

Why do people choose Tomcat?

- Fast
- Robust
- Better development experience than J2EE servers
- Spring's abstraction and portability allows them to choose the most appropriate server, removing the API barrier to Tomcat adoption

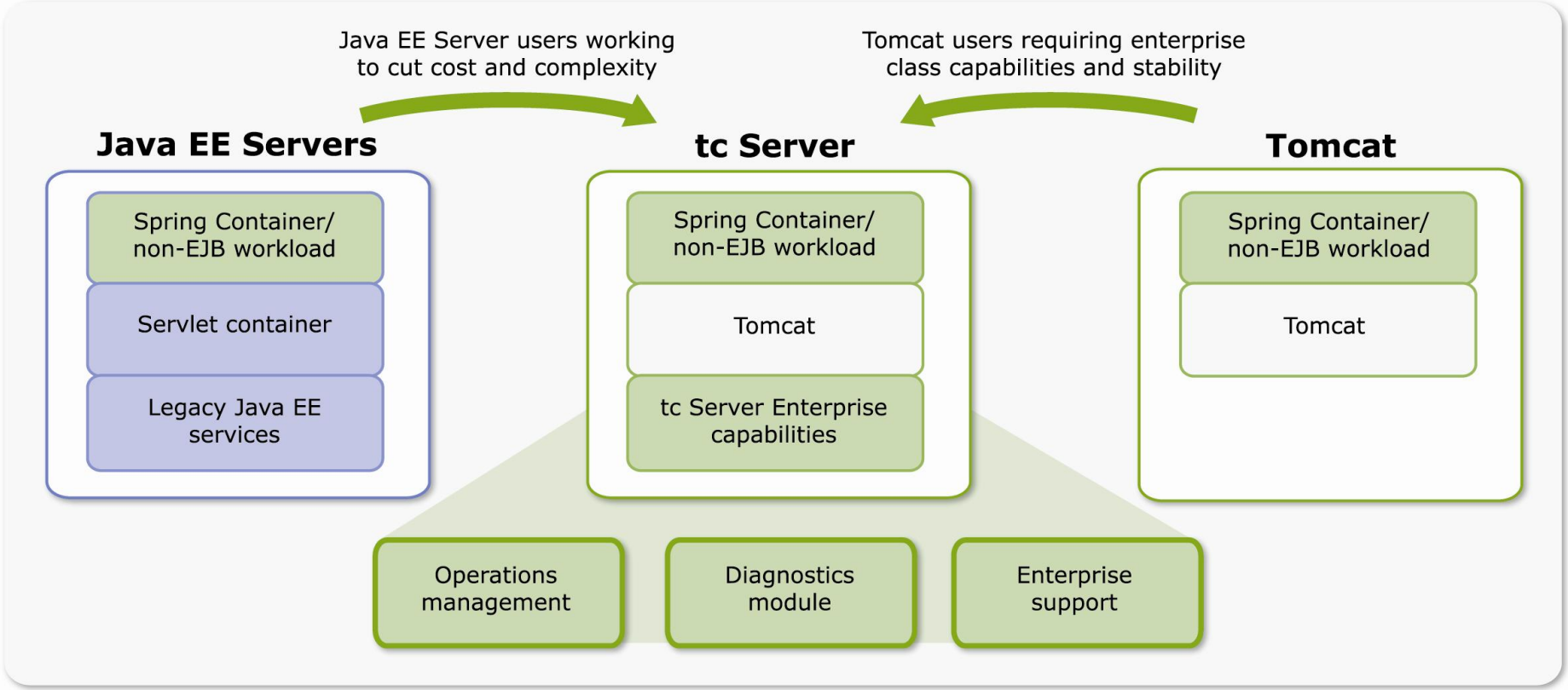
Tomcat limitations

- Tomcat is great for what it does, but not perfect for the data center
- Why do some people not adopt Tomcat or switch back?
  - Perceived lack of enterprise support
  - No management capabilities
  - Desire to use Java EE APIs such as EJB

SpringSource tc Server

- Addresses all but the third (and least important) Tomcat limitation
- Strong Tomcat/Spring solution reflects the market-leading choice
- **The Tomcat you know, the enterprise capabilities you need**
  - Enhanced operational management capabilities
  - Enterprise-level mission-critical support
  - Significantly lower cost than legacy app. servers
  - Powerful, yet lightweight solution

Who should use tc Server?

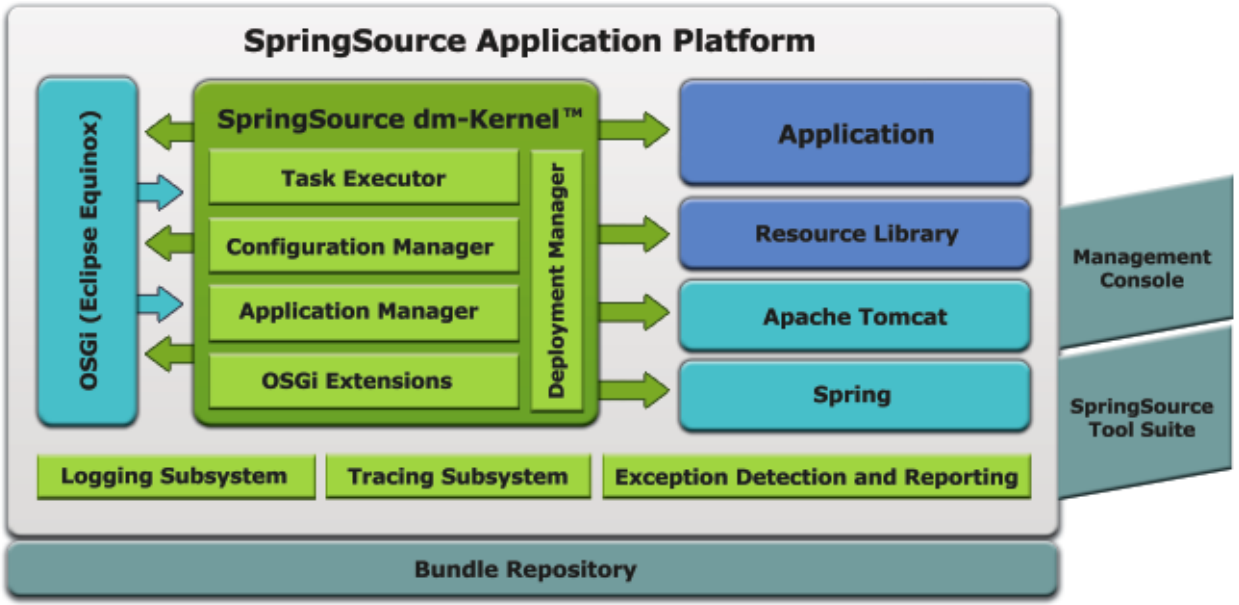


Our server strategy

- tc Server is a solution for WAR-based web applications which you'd already like to run on Tomcat
- dm Server is targeted at next-generation, modular applications
  - The best place to do OSGi on the server side
  - Not limited to web workloads
- Migration path from tc Server to dm Server will be easy

SpringSource dm Server

- Next generation, completely module-based application server
- Runs on the SpringSource Dynamic Module Kernel™
- Harnesses the power of Spring, Tomcat and OSGi



Benefits for developers

- Lightweight
  - Memory footprint < 10% of traditional monolithic application servers
- Strategic solution to shared library hell
  - No more version conflicts between servers and applications
  - Effective sharing of libraries between applications
- Modular server, with opportunity to modularize applications as well
  - Far superior choice for very large applications
- **Realizes vision of pluggable application server in *J2EE without EJB* (2004)**

WAR-based deployment options: Incremental adoption path

- Standard WAR
  - Get started immediately deploying your existing web apps
- *Shared Library* WAR
  - Share libraries by explicitly importing libraries
- *Shared Services* WAR
  - Share libraries and services between applications
- ...OSGI bundles, without WAR boilerplate, or if not web-specific

“Shared Library” WAR

META-INF/MANIFEST.MF (from WAR)

```
Manifest-Version: 1.0
Import-Library:
  org.springframework.spring;version="2.5.4",
  org.hibernate.ejb;version="[3.3.2.GA,3.3.2.GA]"
Import-Package: javax.annotation
```

- No more library bloat in WEB-INF/lib
- Just import libraries/versions
- Just deploy your business logic and resources, not half your server
  - Much smaller WAR files
  - Faster deployment

Simplifying how you *obtain* software

- Currently unnecessarily complex
- Waste of time locating and integrating software
  - We may have gotten used to it, but it's still time better spent on developing software
- Many sources of software
- No guarantee things will work together

Before/After

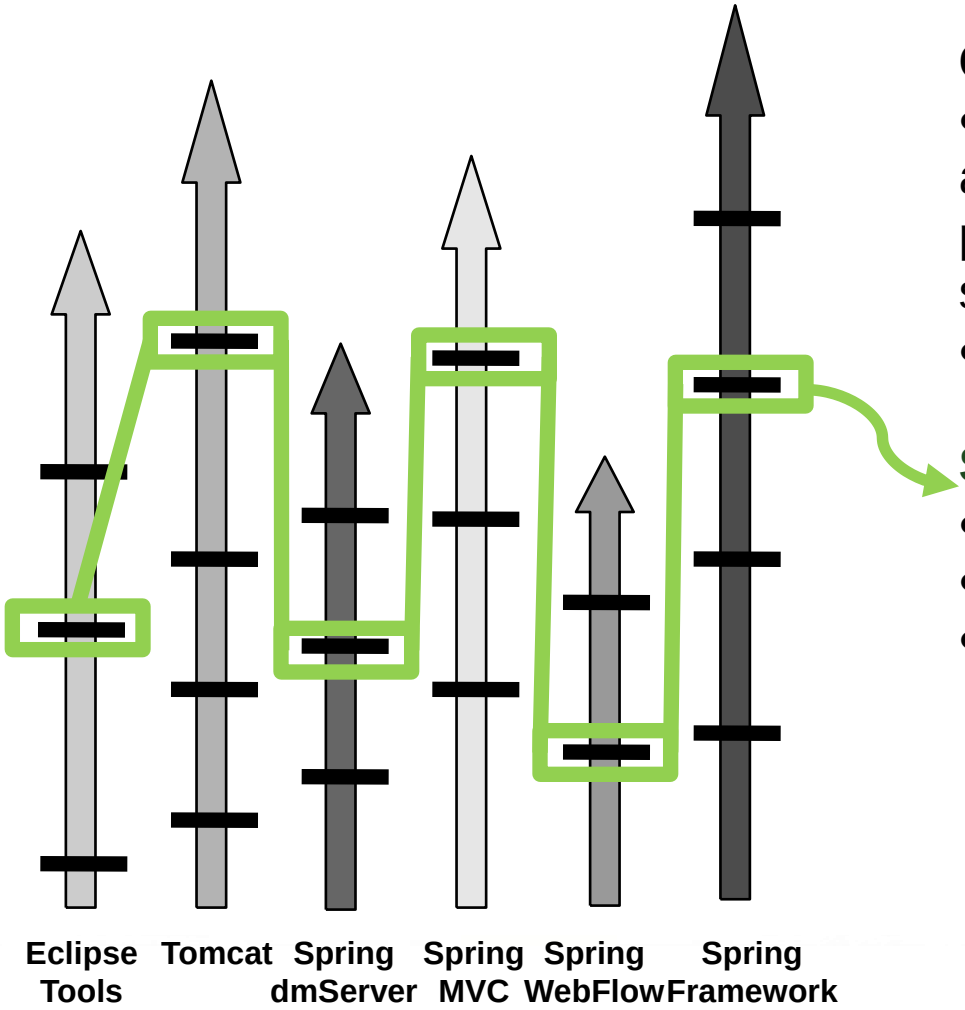
Old

- Need to download multiple open source projects to get anything done
  - Even multiple *Spring* projects
- No authoritative source for what works with what

New

- Bundle repository authoritative source of JARs
- Single SpringSource Web package
- Tools to customize your own distribution

Constant in-house integration effort



Challenge:

- Integrate, manage, and support changes across a wide range of open source projects; each with their own release schedules, versions, & dependencies.
- Time intensive/ Expensive

Solution

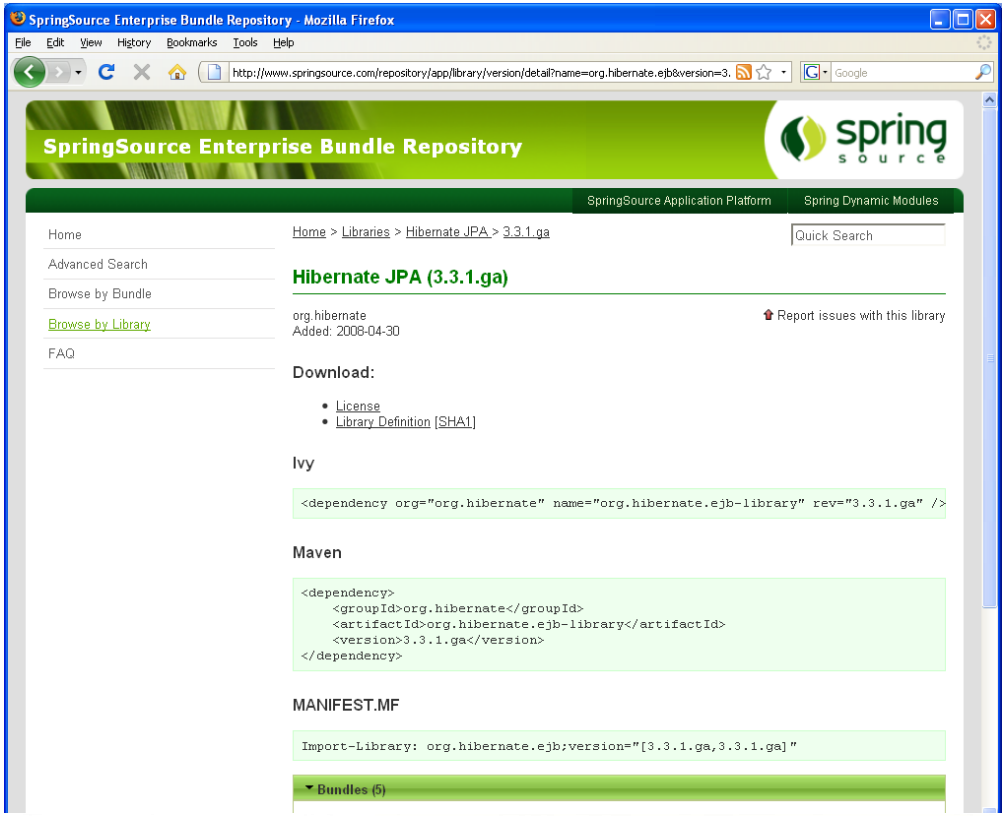
- SpringSource Subscriptions
- SpringSource Bundle Repository +
- ...Further technologies, coming soon

Vision

- One authoritative place to obtain JAR files and dependency information
- One convenient interface for obtaining complete distributions
  - For open source community
  - For SpringSource customers

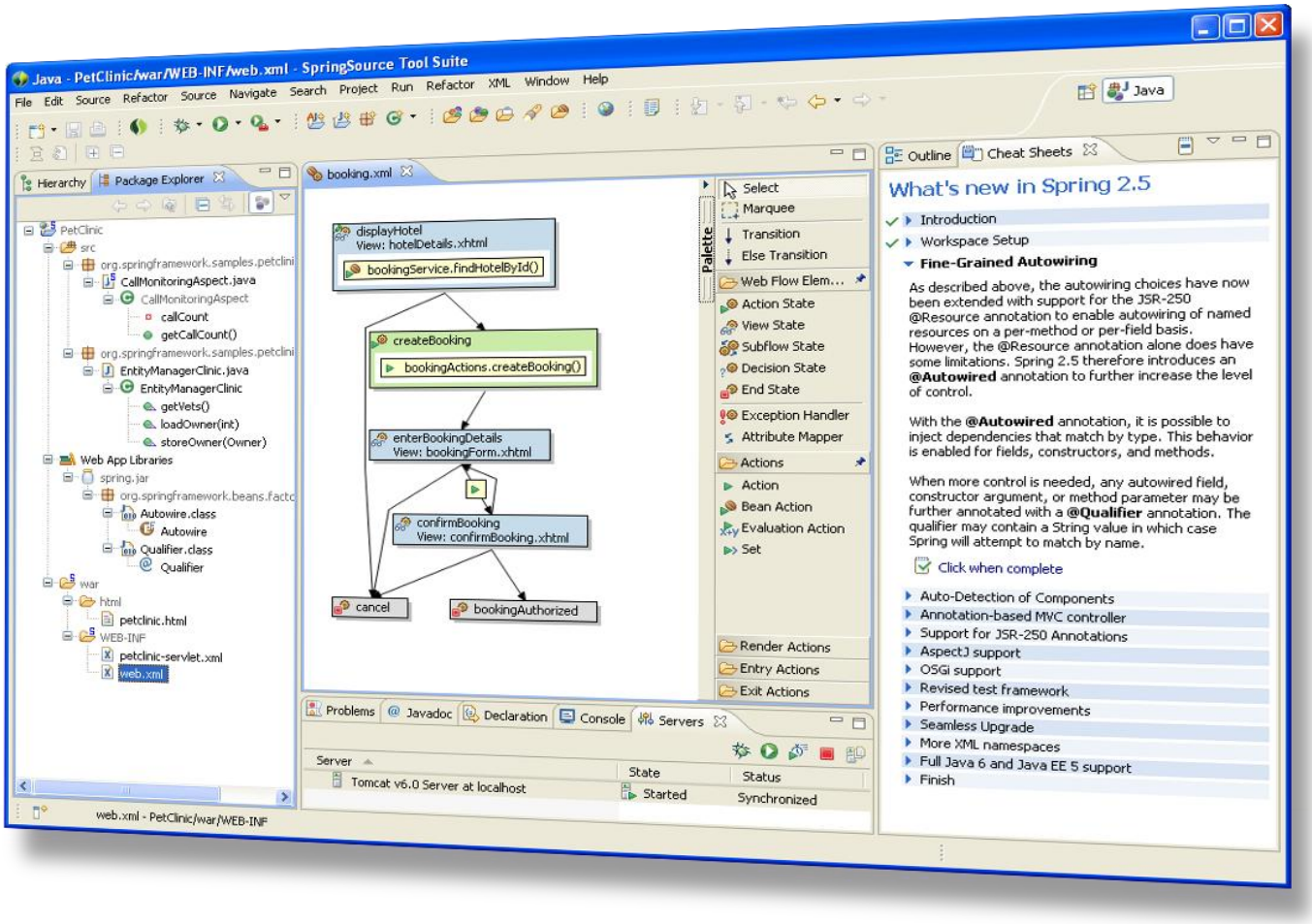
SpringSource Enterprise Bundle Repository

- Complete dependency resolution
- Version compatibility information
- Maven/Ant support
- Development tools
- Open source bundles (OSGi) and libraries



Helping to simplify how you *develop* software

- SpringSource Tool Suite



Simplifying how you author projects: RAD initiatives

- Rails introduced some great ideas but its limitations are more and more apparent
  - Struggles with enterprise scale/complexity
    - Twitter
  - New, alien stack too disruptive to introduce
- Grails offers the same key benefits, based on the power of enterprise Java and Spring
- Stay tuned for additional productivity technologies from SpringSource

Grails

A Web platform

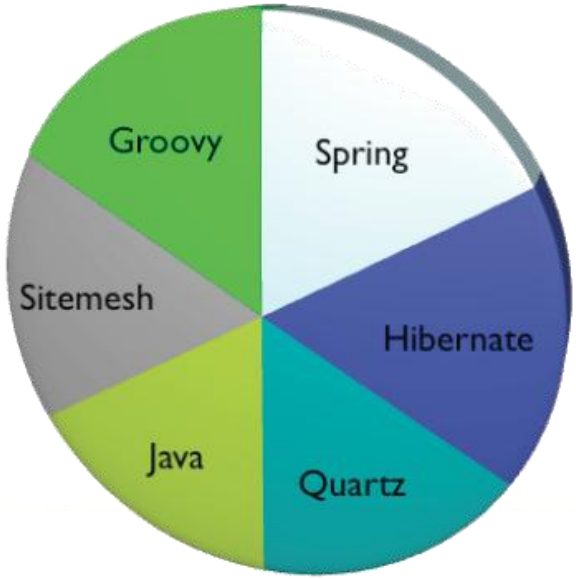
- implements the full stack from build system down to ORM layer
- simple commands to auto-generate application

Leverages existing technologies

- Spring, Hibernate, Quartz etc.
- no re-inventing the wheel

Extensible plug-in system

- Spring-based



Closing Thoughts

- Spring 3.0 is a major release with numerous important new features
- We're taking a broader view of enterprise Java
  - The "Spring Way" of simplification can be applied throughout the software stack and throughout the development/ops experience

Closing Thoughts

- It’s going to be a great year for us
  - Customers are more receptive than ever to our message of simplification
  - We have bigger plans than ever, and are bringing together many of our efforts to make you more productive
- SpringSource will be your partner to work more effectively and get better results in these challenging times

Coming Soon: SpringOne Europe

- The world's biggest conference on Spring technologies
- April 27-29
- Amsterdam
- Most key Spring committers will present
- 3 tracks
  - Rich Web Application Development
  - Enterprise Production Systems
  - Essential Spring

SpringAlong

- Tonight at 19:00
- Old Star pub
  - 66 Broadway
- Also a chance to meet Grails lead, Graeme Rocher