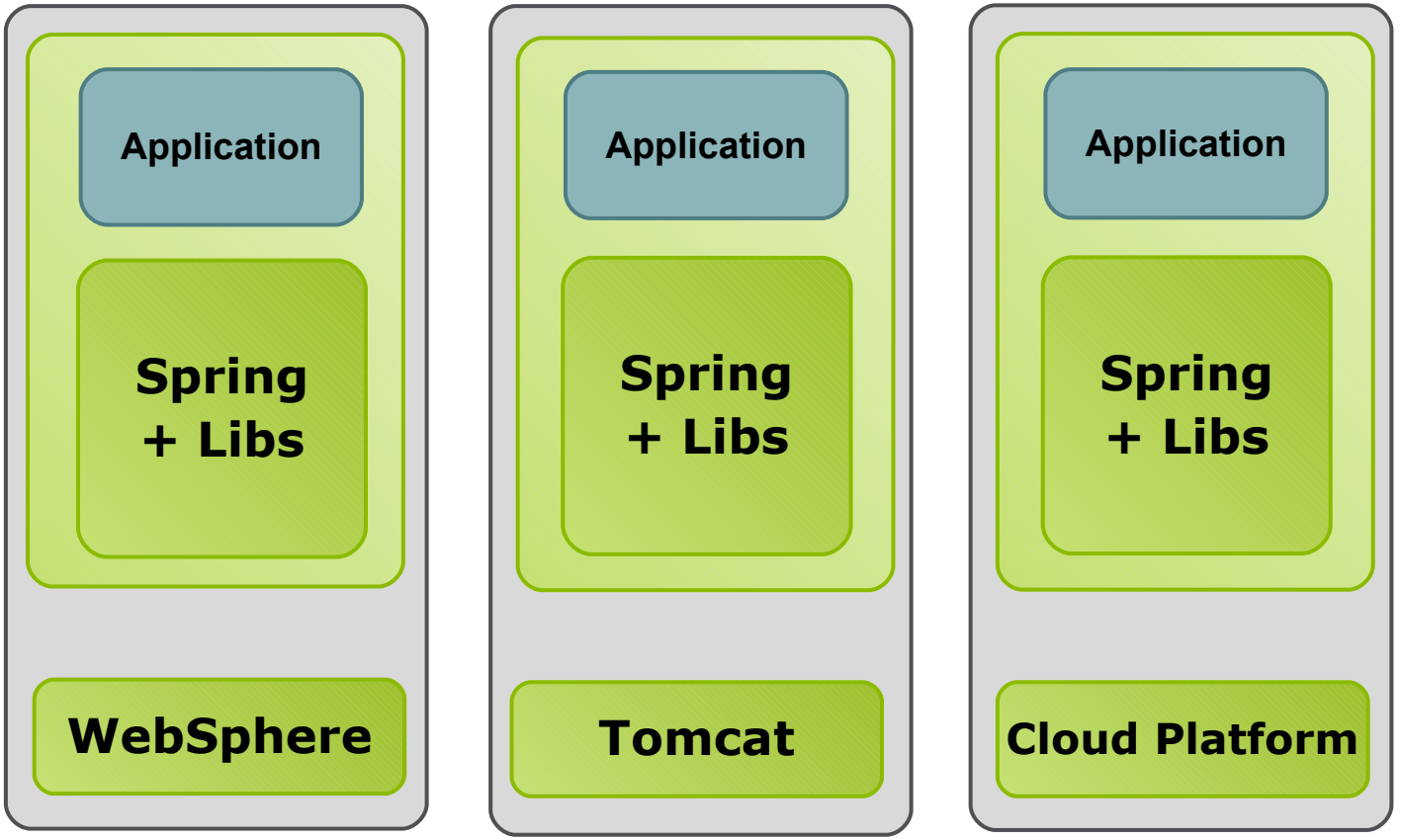
Spring 3.1 and Beyond – Themes and Trends

Jürgen Höller, Principal Engineer, SpringSource

springsource
A division of vmware

CONFIDENTIAL

Deployment Platforms: Becoming More Diverse



Deployment Platforms in 2011: Latest Releases

- **Java EE moving on to Java EE 6**
  - GlassFish 3
  - JBoss 6
  - Other servers still on Java EE 5 (at best)
- **Tomcat moving on to Tomcat 7**
  - Servlet 3.0 based (Java EE 6 level)
- **Cloud platforms becoming a serious option for regular Java web application deployment**
  - Google App Engine: Jetty++
  - Amazon Elastic Beanstalk: Tomcat++

Wide Variety of Data and Datastores

- **Not all data resides in relational databases**
  - cloud environments often suggest alternatives for scalability reasons
  - BigTable, Redis, Mongo, etc
- **Distributed caches add challenges as well**
  - not least of it all in terms of application-level access patterns
  - GemFire, Coherence, etc
- **Hardly any standardization available**
  - just an abandoned caching JSR that never achieved a final release
  - caching – but only caching – possibly getting picked up in Java EE 7
  - alternative datastore space is too diverse

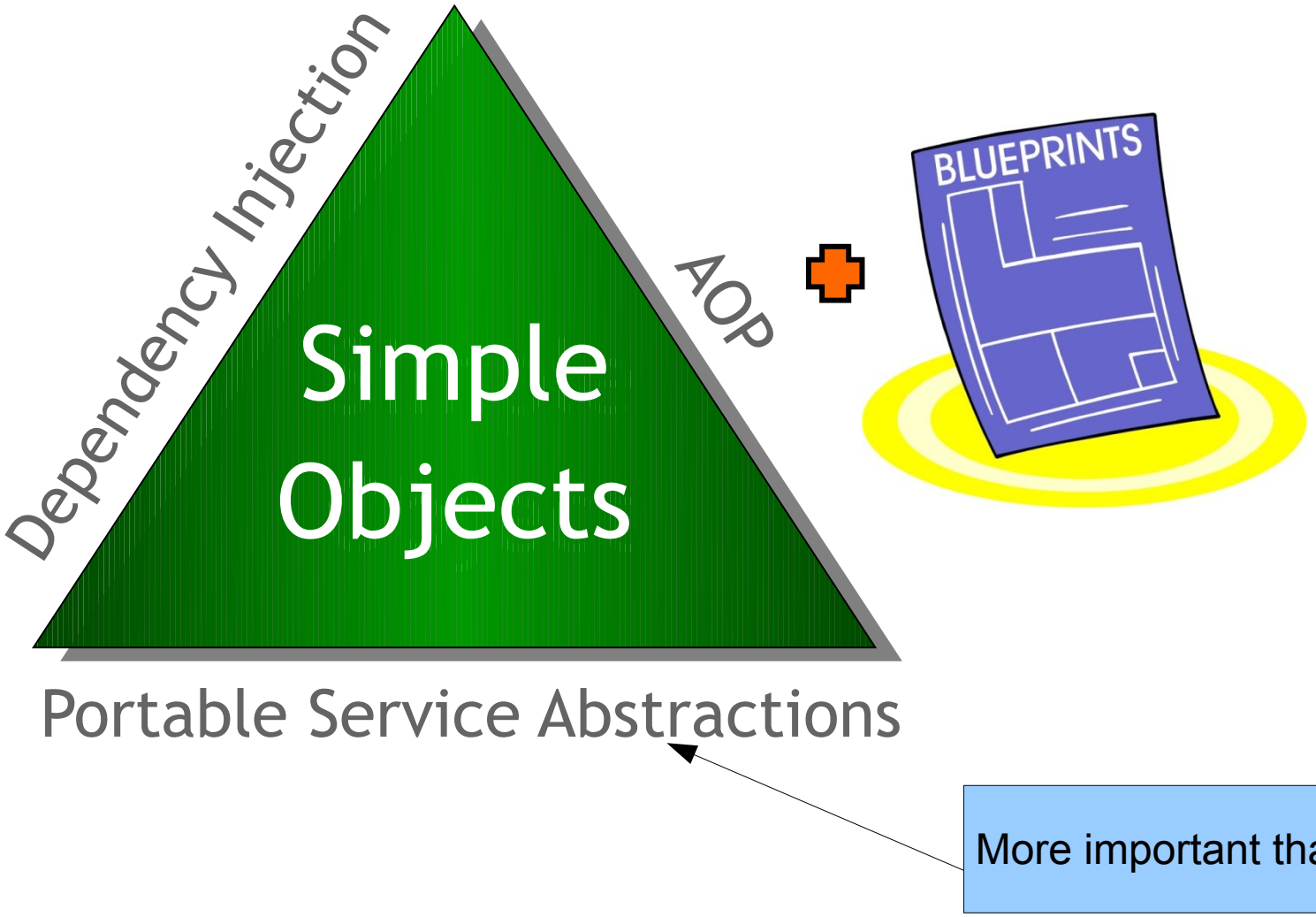
Wide Variety of Web Clients

- **More and more client-side web technologies**
  - HTML 5 as a next-generation browser standard
  - Adobe Flex as a rich client technology on the basis of Flash
- **Server-side state to be minimized or even removed completely**
  - in particular: no server-side user interface state
  - strictly controlled user session state
- **JSF's state-centric approach not too desirable anymore**
  - except for special kinds of applications (which it remains very useful for)
  - general web applications and web services based on JAX-RS / MVC style
  - nevertheless: JSF keeps evolving – JSF 2.2 coming up in Q4 2011

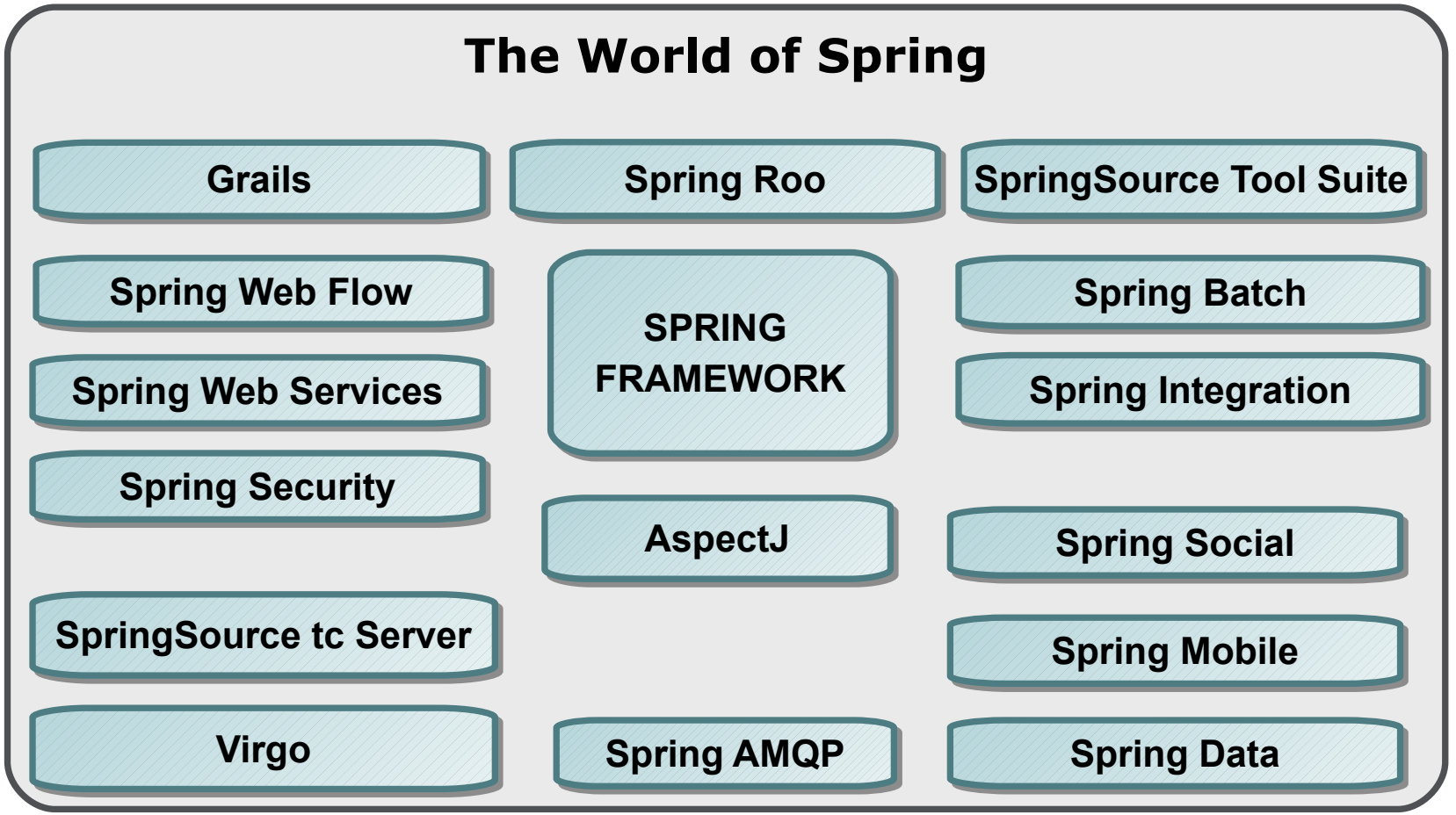
Java SE 7: Concurrent Programming

- **A challenge: concurrent programming in a multi-core world**
  - user-level APIs and recommended programming styles?
- **Servers with more cores than concurrent requests**
  - how to actually use your processor power in such a scenario?
- **Java SE 7: java.util.concurrent.ForkJoinPool**
  - specialized ForkJoinPools to be locally embedded within the application
  - different kind of pool, separate from regular Runnable-oriented Executors
- **Oracle JDK 7 scheduled for GA release in Q3 2011**

Key Elements of Spring: Ready for 2011 & Beyond



The World of Spring: Several New Projects



Spring Framework and The Spring Ecosystem

- **The Spring ecosystem is growing very rapidly**
  - lots of innovation happening in newly established projects
  - e.g. Spring Integration, Spring Data, Spring Mobile
- **Spring Framework remains at the core of all Spring efforts**
  - Spring Framework 3.0 as the basis of the modern Spring programming model
  - Spring Framework 3.1 supporting new infrastructural demands
- **This presentation has a focus on Spring Framework's evolution but don't forget to check out new Spring initiatives as well...**
  - recent candidates: Spring Social, Spring Mobile, Spring Data
  - development experience: Spring Roo, SpringSource Tool Suite

A Quick Review: Spring Framework 3.0

- **Powerful annotated component model**
  - stereotypes, factory methods, JSR-330 support
- **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**
  - integration with JSR-303 Bean Validation
- **Support for Java EE 6**
  - in particular for JPA 2.0

Spring Framework 3.1: Key Themes

- Environment profiles for bean definitions
- Java-based application configuration
- "c:" namespace for XML configuration
- Cache abstraction & declarative caching

- Customizable @MVC processing
- Conversation management
- Explicit support for Servlet 3.0
- Enhanced Groovy support

Environment Abstraction

- **Grouping bean definitions for activation in specific environments**
  - e.g. development, testing, production
  - possibly different deployment environments
- **Custom resolution of placeholders**
  - dependent on the actual environment
  - hierarchy of property sources

- Injectable environment abstraction API
  - org.springframework.core.env.Environment
- Property resolution SPI
  - org.springframework.core.env.PropertyResolver

Environment Example

```
<beans profile="production">
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
    <property name="driverClass" value="${database.driver}"/>
    <property name="jdbcUrl" value="${database.url}"/>
    <property name="username" value="${database.username}"/>
    <property name="password" value="${database.password}"/>
  </bean>
</beans>

<beans profile="embedded">
  <jdbc:embedded-database id="dataSource" type="H2">
    <jdbc:script location="/WEB-INF/database/schema-member.sql"/>
    <jdbc:script location="/WEB-INF/database/schema-activity.sql"/>
    <jdbc:script location="/WEB-INF/database/schema-event.sql"/>
    <jdbc:script location="/WEB-INF/database/data.sql"/>
  </jdbc:embedded-database>
</beans>
```

Environment Profiles

- **Environment association of specific bean definitions**
  - **XML 'profile' attribute** on <beans> element
  - **@Profile annotation** on configuration classes
  - **@Profile annotation** on individual component classes
- **Activating specific profiles by name**
  - e.g. through a system property
    - -Dspring.profiles.active=development
  - or other means outside of the deployment unit
    - according to environment conventions
- **Ideally: no need to touch deployment unit across different stages/environments**

Java-Based Application Configuration

- **Application-specific container configuration**
  - aligned with the @Configuration style
  - focus on customizing the annotation-based processing parts of Spring
- **Equivalent to XML namespace functionality**
  - but not a one-on-one mapping
  - 'natural' container configuration from an annotation-oriented perspective
- **Typical infrastructure setup**
  - transactions
  - scheduling
  - MVC customization

Application Configuration Example

```
@FeatureConfiguration
@Import(DataConfig.class)
public class TxFeatures {
        @Feature
        public TxAnnotationDriven tx(DataConfig dataConfig) {
                return new TxAnnotationDriven(dataConfig.txManager()).proxyTargetClass(true);
        }
}

@Configuration
public class DataConfig {
        @Bean
        public PlatformTransactionManager txManager() {
                return new DataSourceTransactionManager(dataSource());
        }

        @Bean
        public DataSource dataSource() {
                // ... configure and return JDBC DataSource ...
        }
}
```

```
<tx:annotation-driven transaction-manager="txManager"
                      proxy-target-class="true"/>
```

"c:" Namespace for XML Configuration

- **New XML namespace for use with bean configuration**
  - shortcut for <constructor-arg>
    - inline argument values
    - analogous to existing "p:" namespace
  - use of constructor argument names
    - recommended for readability
    - debug symbols have to be available in the application's class files

```
<bean class="…" c:age="10" c:name="myName"/>

<bean class="…" c:name-ref="nameBean"
      c:spouse-ref="spouseBean"/>
```

Cache Abstraction

- **CacheManager and Cache abstraction**
  - in org.springframework.cache
    - which up until 3.0 just contained EhCache support
  - particularly important with the rise of distributed caching
    - not least of it all: in cloud environments

- **Backend adapters for EhCache, GemFire, Coherence, etc**
  - EhCache adapter to be shipped with Spring core
  - plugging in custom adapters if necessary

- **Specific cache setup per environment profile?**
  - potentially even adapting to a runtime-provided service

Declarative Caching

```
@Cacheable
public Owner loadOwner(int id);

@Cacheable(condition="name.length < 10")
public Owner loadOwner(String name);

@CacheEvict
public void deleteOwner(int id);
```

Cache Configuration

- **Cache namespace**
  - <cache:annotation-driven>
  - convenient setup for annotation-driven caching
  - pointing to a "cacheManager" bean by default

- **CacheManager SPI**
  - EhCacheCacheManager
    - backed by an EhCacheFactoryBean
  - GemFireCacheManager
    - part of the Spring GemFire project

Customizable @MVC Processing

- **@MVC turned out to be one of the most successful recent developments in Spring core**
  - superseding the form controller template classes
    - SimpleFormController, AbstractWizardFormController etc deprecated since 3.0
  - mapping flexibility of handler methods as a key success factor
    - @RequestMapping in combination with request params, path variables, etc
    - very natural to follow REST conventions

- **However, there are several things on the wish list for 3.1…**
  - arbitrary mappings to handler methods across multiple controllers
  - request interception on a per-handler-method basis
  - better customization of handler method arguments

Conversation Management

- **Abstraction for conversational sessions**
  - basically HttpSession++
  - more flexible lifecycle
  - more flexible storage options
- **Management of a current conversation**
  - e.g. associated with browser window/tab
  - or manually demarcated
- **For use with MVC and JSF**
  - 'conversation' scope for scoped beans
  - programmatic access at any time

Window-Specific Sessions

- **Common problem:** isolation between browser windows/tabs
  - windows sharing the same HttpSession
  - HttpSession identified by shared cookie
- **Window id managed by the framework**
  - associating the current window session
  - e.g. for MVC session form attributes
  - special SessionAttributeStore variant
- **Simpler problem, simpler solution**
  - as opposed to full conversation management

Support for Servlet 3.0

- **Explicit support for Servlet 3.0 containers**
  - such as Tomcat 7 and GlassFish 3
  - while at the same time preserving compatibility with Servlet 2.4+
- **Support for XML-free web application setup (no web.xml)**
  - Servlet 3.0's ServletContainerInitializer in combination with Spring 3.1's AnnotationConfigWebApplicationContext plus the environment abstraction
- **Exposure of native Servlet 3.0 functionality in Spring MVC**
  - support for asynchronous request processing
  - standard Servlet 3.0 file upload support behind Spring's MultipartResolver abstraction

Enhanced Groovy Support

- **Enhanced <lang:groovy> support**
  - base script classes
  - custom bindings
  - implicit access to Spring beans by name
    - analogous to SpEL's context attributes
- **Inclusion of Grails BeanBuilder in Spring core**
  - Groovy-based bean configuration
- **Groovy-based template files?**
  - as alternative to Velocity and FreeMarker
  - e.g. for email templates

Related Portfolio Projects

- **Spring Framework 3.1 serves as a foundation for several new projects**
  - Spring Data
  - Spring Mobile
  - Spring Social
  - Spring Roo
  - Greenhouse sample application

- **Refinements to Spring MVC and to RestTemplate driven by the needs of several of those projects**
  - e.g. <mvc:annotation-driven> enhancements
  - e.g. new ClientHttpRequestInterceptor facility

Spring 3.1 Summary

- Selected improvements to the Spring 3.0 programming model

- M1 – out now!
  - **Environment profiles for bean definitions**
  - **Java-based application configuration**
  - **"c:" namespace for XML configuration**
  - **Cache abstraction & declarative caching**

- M2 – following soon…
  - **Customizable MVC processing**
  - **Conversation management**
  - **Explicit support for Servlet 3.0**
  - **Enhanced Groovy support**

Spring 3.1 Release Plan

- **3.1 M1 in February 2011**
- 3.1 M2 in April 2011
- 3.1 RC1 in May 2011
- 3.1 GA in June 2011

A Quick Preview: Spring Framework 3.2

- **Spring 3.2 plan already being prepared**
  - direct follow-up to Spring 3.1
  - scheduled for early 2012
- **Key driver: Java SE 7 support**
  - making best use of JRE 7 at runtime
  - support for JDBC 4.1
  - support for fork-join framework
- **Further inspiration: EE specification updates**
  - e.g. JSF 2.2, JPA 2.1
  - focus on individual specifications with standalone releases

Spring 3.2 Strategy

- **Early support for the latest JDK generation**
  - Java 7 as the central theme
  - with Java 8's language enhancements in mind already
- **Early support for EE specification updates**
  - preparing for a comprehensive Java EE 7 update in a later release
  - Spring keeps track of relevant EE specifications
- **Preserving compatibility with Java 5+**
  - Java SE 5+ as well as Java EE 5+
  - for the entire Spring 3.x branch