Spring 3.1 Features Worth Knowing About
(Available For Download Today)

Rossen Stoyanchev, Software Engineer

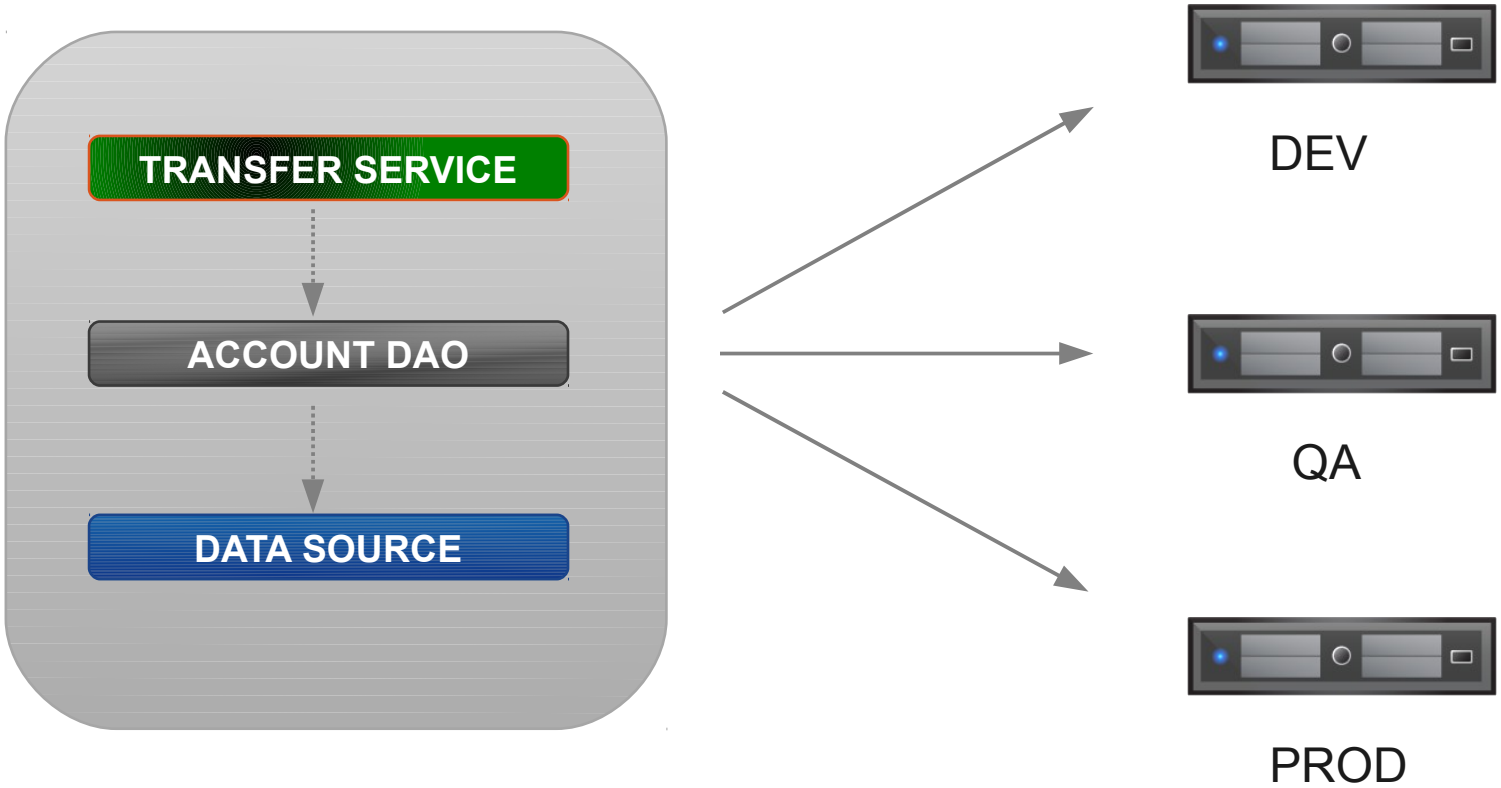
springsource
A division of vmware

CONFIDENTIAL

Agenda

- *Environment*
- Java Configuration DSL
- Cache Abstraction

Moving an Application Across Environments



Environment-Specific Properties In XML

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
        p:driverClassName="org.hsqldb.jdbcDriver"
        p:url="${database.url}"
        p:username="${database.user}"
        p:password="${database.password" />
```

Property Source:
dev/database.properties

```
<context:property-placeholder location="${ENV}/database.properties"/>
```

Property Source: Java system property

```
java -DENV=prod ...
```

dev/database.properties

qa/database.properties

prod/database.properties

Environment-Specific Properties In Java

Property Source:
Java system property?

```
@Configuration
public class DatabaseBeans {

    @Value("${ENV}") String environment;

    @Bean
    public DataSource dataSource() throws IOException {
        return new DriverManagerDataSource(
                databaseProps().getProperty("database.url"),
                databaseProps().getProperty("database.username"),
                databaseProps().getProperty("database.password"));
    }

    @Bean
    public Properties databaseProps() throws IOException {
        String location = this.environment + "/database.properties";
        return PropertiesLoaderUtils.loadAllProperties(location);
    }

}
```

Spring EL Expressions in Spring 3.0

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
        p:driverClassName="org.hsqldb.jdbcDriver"
        p:url="#{databaseProps['database.url']}"
        p:username="#{databaseProps['database.user']}"
        p:password="#{databaseProps['database.password']}" />

<util:properties id="databaseProps"
    location="#{systemProperties['ENV']}/database.properties"/>
```

- **Easier to see where properties originate from**

What About “Structural” Differences ?

Environment-Specific Configuration in XML

```
<jee:jndi-lookup id="dataSource"
    jndi-name="java:comp/env/jdbc/datasource" />
```

prod/database-context.xml

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
        p:driverClassName="org.hsqldb.jdbcDriver"
        p:url="#{databaseProps['database.url']}"
        p:username="#{databaseProps['database.user']}"
        p:password="#{databaseProps['database.password']}" />
```

dev/database-context.xml

```
<import resource="${ENV}/db-context.xml" />
```

app-context.xml

Choose from multiple files to accommodate differences

Environment-Specific Configuration in Java

```java
@Configuration
public class DatabaseBeans {

    @Value("${ENV}") String environment;

    @Bean
    public DataSource dataSource() throws IOException, NamingException {
        if ("prod".equals(environment)) {
            return containerDataSource();
        } else {
            return standaloneDataSource();
        }
    }

    @Bean @Lazy
    public DataSource containerDataSource() throws NamingException {
        return (DataSource) new JndiTemplate().lookup(
                "java:comp/env/jdbc/datasource");
    }

    @Bean @Lazy
    public DataSource standaloneDataSource() throws IOException {
        return new DriverManagerDataSource(
                databaseProps().getProperty("database.url"),
                databaseProps().getProperty("database.username"),
                databaseProps().getProperty("database.password"));
    }
```

Accommodate differences in code

Spring 3.1 "Environment" Abstraction

- **A concrete representation with two key aspects**
  - Property Sources
  - Bean Profiles

Property Source:

A variety of sources: property files, system properties, servlet context, JNDI, etc.

Bean Profile:

A logical group of bean definitions. Registered only if the profile is ***active***.

Managing Property Sources

- **In standalone code**

```
ConfigurableApplicationContext ctx = new GenericApplicationContext();
MutablePropertySources sources = ctx.getEnvironment().getPropertySources();
sources.addFirst(new PropertiesPropertySource("myProps", props));
```

- **In a Web application**
  - Implement `ApplicationContextInitializer`
  - Register via `contextInitializerClasses` context parameter in web.xml

- **Default property sources**
  - JVM properties
  - System envrionment variables

Accessing Environment Properties

- **In standalone code**

```
ApplicationContext ctx = new GenericApplicationContext();
Environment env = ctx.getEnvironment();
boolean containsFoo = env.containsProperty("foo");
```

- **Via** `@Autowired` **or** `EnvironmentAware` **interface**

- **In configuration (via placeholders)**

```
<import resource="${ENV}/db-context.xml" />
```

- `<context:property-placeholder/>` **also now falls back on Environment property sources**

Beans Profiles With XML

- **Assign all beans to profile**

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="..."
    profile="dev">

</beans>
```

Multiple profiles can be listed

- **Assign enclosed beans to a profile**

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="...">

    <beans profile="dev"> <!-- ... --> </beans>

    <beans profile="prod"> <!-- ... --> </beans>

</beans>
```

Bean Profiles In Java

```
@Configuration
@Profile("dev")
public class DatabaseBeans {

    @Value("${ENV}") String environment;

    @Bean
    public DataSource standaloneDataSource() throws IOException {
        return new DriverManagerDataSource(
                databaseProps().getProperty("database.url"),
                databaseProps().getProperty("database.username"),
                databaseProps().getProperty("database.password"));
    }

    @Bean
    public Properties databaseProps() throws IOException {
        String location = this.environment + "/database.properties";
        return PropertiesLoaderUtils.loadAllProperties(location);
    }
}
```

Bean Profile Activation

- In code

```
GenericXmlApplicationContext ctx = new GenericXmlApplicationContext();
ctx.getEnvironment().setActiveProfiles("dev");
ctx.load("classpath:${ENV}/database-context.xml");
ctx.refresh();
```

- In Web applications

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework...DispatcherServlet</servlet-class>
    <init-param>
        <param-name>spring.profiles.active</param-name>
        <param-value>production</param-value>
    </init-param>
</servlet>
```

Agenda

- Environment
- *Java Configuration DSL*
- Cache Abstraction

Custom XML Namespaces

- **Custom XML namespaces have become very popular**
  - In Spring and in other frameworks

- **Succinct and highly expressive**

```
<tx:annotation-driven transaction-manager="txManager"/>
```

- **Each namespace element can result in the registration of any number of bean definitions**

- **Question:**

  How do get the same benefits in Java configuration?

Feature Specifications

- Java configuration equivalent to an XML namespace

```
new TxAnnotationDriven(txManager);
```

- Declared in @FeatureConfiguration classes

```
@FeatureConfiguration
class TxFeature {

    @Feature
    public TxAnnotationDriven txAnnotationDriven(PlatformTransactionManager txm) {
            return new TxAnnotationDriven(txm);
    }

}
```

Beans declared elsewhere can be autowired in.

- Results in same bean declarations as `<tx:annotation-driven />`

FeatureSpecifications Are Designed For Ease of Use

```
@FeatureConfiguration
class Features {

    @Feature
    public TxAnnotationDriven txAnnotationDriven(PlatformTransactionManager txManager) {
        return new TxAnnotationDriven(txManager);
    }

    @Feature
    public ComponentScanSpec componentScan() {
        return new ComponentScanSpec("com.bank.service")
            .useDefaultFilters(false)
            .
```

```
beanNameGenerator(BeanNameGenerator beanNameGenerator) : ComponentScanSpec
excludeFilters(TypeFilter... excludeFilters) : ComponentScanSpec - ComponentScanSp
includeAnnotationConfig(Boolean includeAnnotationConfig) : ComponentScanSpec - C
includeFilters(TypeFilter... includeFilters) : ComponentScanSpec - ComponentScanSpe
resourcePattern(String resourcePattern) : ComponentScanSpec - ComponentScanSpec
scopedProxyMode(ScopedProxyMode scopedProxyMode) : ComponentScanSpec - Com
scopeMetadataResolver(ScopeMetadataResolver scopeMetadataResolver) : Component
useDefaultFilters(Boolean useDefaultFilters) : ComponentScanSpec - ComponentScanS
Press '^Space' to show Template Propos
```

```
    }

}
```

Agenda

- Environment
- Java Configuration DSL
- *Cache Abstraction*

Spring 3.1 Cache Abstraction

- **`CacheManager` and `Cache` abstractions**
  - `org.springframework.cache`
- **Backend adapters for EhCache, GemFire, Coherence, etc.**
- **Cache namespace**
  - `<cache:annotation-driven />`
- **CacheManager SPI**
  - `EhCacheManager`
  - `GemFireCacheManager`

Annotation-Based Caching

Cache name

```
@Cacheable("books")
public Book findBook(ISBN isbn) {
}
```

Custom key

```
@Cacheable(value="books", key="#isbn.rawNumber")
public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed) {
}
```

Cache condition

```
@Cacheable(value="books",  condition="name.length < 32")
public Book findBook(String name) {
}
```

Spring 3.1 M1 Blog Series:

---

http://blog.springsource.com/2008/03/26/spring-java-configuration-whats-new-in-m3/

```
https://github.com/cbeams/spring-3.1-profiles-xml
https://github.com/cbeams/spring-3.1-profiles-java
https://github.com/cbeams/spring-3.1-featurespec
https://github.com/rstoyanchev/spring-3.1-mvc-java-config
```

Review updated Spring Reference Documentation

Provide feedback via JIRA and Forums

Thank You