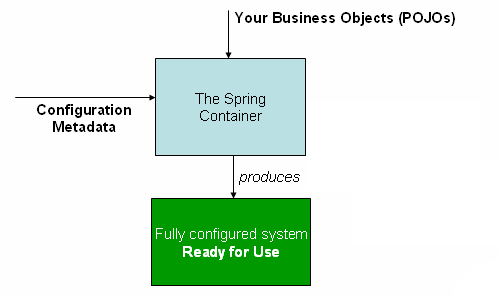
**New features of Spring 2 versus Spring 3**



.

* Spring Expression Language
* IoC enhancements/Java based bean metadata
* General-purpose type conversion system and field formatting system
* Object to XML mapping functionality (OXM) moved from Spring Web Services project
* Comprehensive REST support
* @MVC additions
* Declarative model validation
* Early support for Java EE 6
* Embedded database support

**@MVC additions**

* A mvc namespace has been introduced that greatly simplifies Spring MVC configuration.
* Additional annotations such as @CookieValue and @RequestHeaders have been added.
* As with most annotation-based alternatives, keep in mind that the annotation metadata is bound to the class definition itself, while the use of XML allows for multiple beans of the same type to provide variations in their qualifier metadata, because that metadata is provided per-instance rather than per-class.
* XML-based metadata is not the only allowed form of configuration metadata. The Spring IoC container itself is totally decoupled from the format in which this configuration metadata is actually written
* While XML has been the traditional format for defining configuration metadata you can instruct the container to use Java annotations or code as the metadata format by providing a small amount of XML configuration to declaratively enable support for these additional metadata formats.

Spring 2.5 introduced support for annotation-based configuration metadata

**Java5 :**  The entire framework code has been revised to take advantage of Java 5 features like generics, varargs and other language improvements.

All callback interfaces such as TransactionCallback and HibernateCallback declare a generic result value now.

the Spring core codebase is now freshly revised and optimized for Java 5.

The following is an example of how the Expression Language can be used to configure some properties of a database setup

<bean class="mycompany.RewardsTestDatabase">

<property name="databaseName"

value="#{systemProperties.databaseName}"/>

<property name="keyGenerator"

value="#{strategyBean.databaseKeyGenerator}"/>

</bean>

This functionality is also available if you prefer to configure your components using annotations:

@Repository

**public** **class** RewardsTestDatabase {

@Value("#{systemProperties.databaseName}")

**public** **void** setDatabaseName(String dbName) { … }

@Value("#{strategyBean.databaseKeyGenerator}")

**public** **void** setKeyGenerator(KeyGenerator kg) { … }

}

The framework modules have been revised and are now managed separately with one source-tree per module jar:

* org.springframework.aop
* org.springframework.beans
* org.springframework.context
* org.springframework.context.support
* org.springframework.expression
* org.springframework.instrument
* org.springframework.jdbc
* org.springframework.jms
* org.springframework.orm
* org.springframework.oxm
* org.springframework.test
* org.springframework.transaction
* org.springframework.web
* org.springframework.web.portlet
* org.springframework.web.servlet
* org.springframework.web.struts

Here is an example of a Java class providing basic configuration using the new JavaConfig features:

package org.example.config;

@Configuration

public class AppConfig {

private @Value("#{jdbcProperties.url}") String jdbcUrl;

private @Value("#{jdbcProperties.username}") String username;

private @Value("#{jdbcProperties.password}") String password;

@Bean

public FooService fooService() {

return new FooServiceImpl(fooRepository());

}

@Bean

public FooRepository fooRepository() {

return new HibernateFooRepository(sessionFactory());

}

@Bean

public SessionFactory sessionFactory() {

// wire up a session factory

AnnotationSessionFactoryBean asFactoryBean =

new AnnotationSessionFactoryBean();

asFactoryBean.setDataSource(dataSource());

// additional config

return asFactoryBean.getObject();

}

@Bean

public DataSource dataSource() {

return new DriverManagerDataSource(jdbcUrl, username, password);

}

}

To get this to work you need to add the following component scanning entry in your minimal application context XML file.

<context:component-scan base-package="org.example.config"/>

<util:properties id="jdbcProperties" location="classpath:org/example/config/jdbc.properties"/>

Or you can bootstrap a @Configuration class directly using AnnotationConfigApplicationContext:

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);

FooService fooService = ctx.getBean(FooService.class);

fooService.doStuff();

}