Overview of Spring Framework

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

## Dependency Injection and Inversion of Control

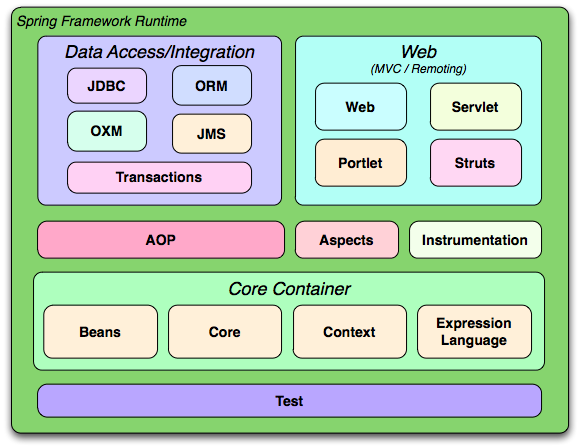
Java applications -- a loose term that runs the gamut from constrained applets to n-tier server-side enterprise applications -- typically consist of objects that collaborate to form the application proper. Thus the objects in an application have dependencies on each other.

Although the Java platform provides a wealth of application development functionality, it lacks the means to organize the basic building blocks into a coherent whole, leaving that task to architects and developers. True, you can use design patterns such as Factory, Abstract Factory, Builder, Decorator, and Service Locator to compose the various classes and object instances that make up an application. However, these patterns are simply that: best practices given a name, with a description of what the pattern does, where to apply it, the problems it addresses, and so forth. Patterns are formalized best practices that you must implement yourself in your application.

The Spring Framework Inversion of Control (IoC) component addresses this concern by providing a formalized means of composing disparate components into a fully working application ready for use. The Spring Framework codifies formalized design patterns as first-class objects that you can integrate into your own application(s).

## Modules

The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test, as shown in the following diagram-



### Core Container

The [Core Container](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/beans.html" \l "beans-introduction) consists of the Core, Beans, Context, and Expression Language modules.

1. The [Core and Beans](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/beans.html" \l "beans-introduction) modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The BeanFactory is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.
2. The [Context](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/beans.html" \l "context-introduction) module builds on the solid base provided by the [Core and Beans](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/beans.html" \l "beans-introduction) modules: it is a means to access objects in a framework-style manner that is similar to a JNDI registry. The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, resource-loading, and the transparent creation of contexts by, for example, a servlet container. The Context module also supports Java EE features such as EJB, JMX ,and basic remoting. The ApplicationContext interface is the focal point of the Context module.
3. The [Expression Language](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/expressions.html) module provides a powerful expression language for querying and manipulating an object graph at runtime. It is an extension of the unified expression language (unified EL) as specified in the JSP 2.1 specification. The language supports setting and getting property values, property assignment, method invocation, accessing the context of arrays, collections and indexers, logical and arithmetic operators, named variables, and retrieval of objects by name from Spring's IoC container. It also supports list projection and selection as well as common list aggregations.

### Data Access/Integration

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

1. The [JDBC](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/jdbc.html" \l "jdbc-introduction) module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.
2. The [ORM](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/orm.html" \l "orm-introduction) module provides integration layers for popular object-relational mapping APIs, including [JPA](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/orm.html" \l "orm-jpa), [JDO](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/orm.html" \l "orm-jdo), [Hibernate](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/orm.html" \l "orm-hibernate), and [iBatis](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/orm.html" \l "orm-ibatis). Using the ORM package you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.
3. The [OXM](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/oxm.html) module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and Xstream.
4. The Java Messaging Service ([JMS](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/jms.html)) module contains features for producing and consuming messages.
5. The [Transaction](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/transaction.html) module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs (plain old Java objects).

### Web

The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

1. Spring's Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web-related parts of Spring's remoting support.
2. The Web-Servlet module contains Spring's model-view-controller ([MVC](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/mvc.html" \l "mvc-introduction)) implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms, and integrates with all the other features of the Spring Framework.
3. The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application. Note that this support is now deprecated as of Spring 3.0. Consider migrating your application to Struts 2.0 and its Spring integration or to a Spring MVC solution.
4. The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

### AOP and Instrumentation

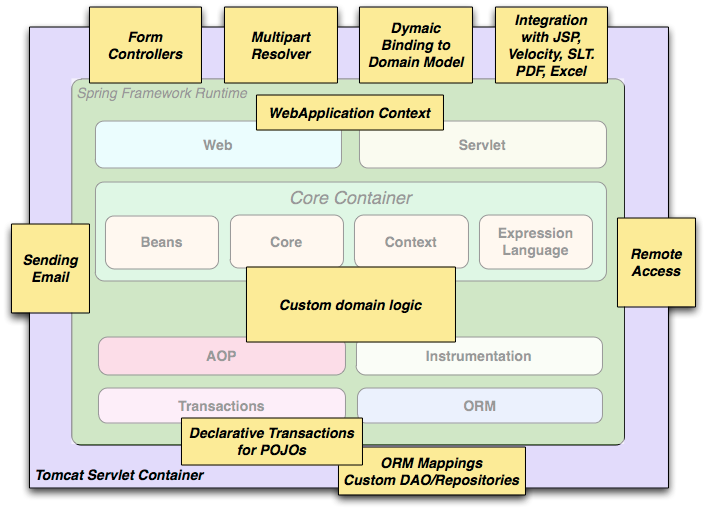
1. Spring's [AOP](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/aop.html" \l "aop-introduction) module provides an AOP Alliance-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated. Using source-level metadata functionality, you can also incorporate behavioral information into your code, in a manner similar to that of .NET attributes.
2. The separate Aspects module provides integration with AspectJ.
3. The Instrumentation module provides class instrumentation support and classloader implementations to be used in certain application servers.

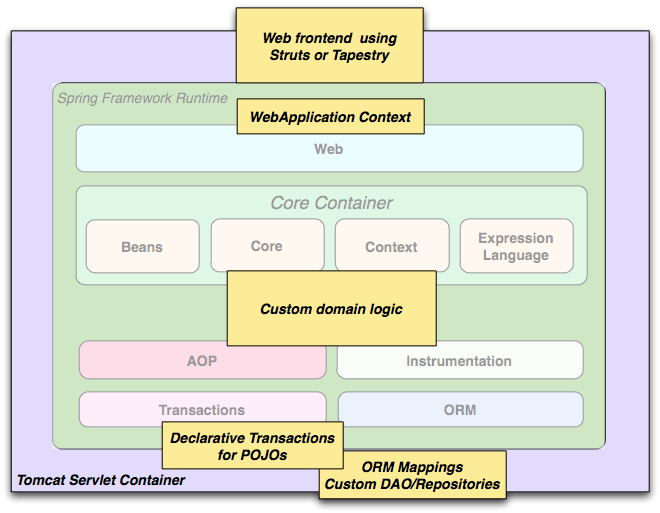
### Test

The Test module supports the testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.

## Usage scenarios

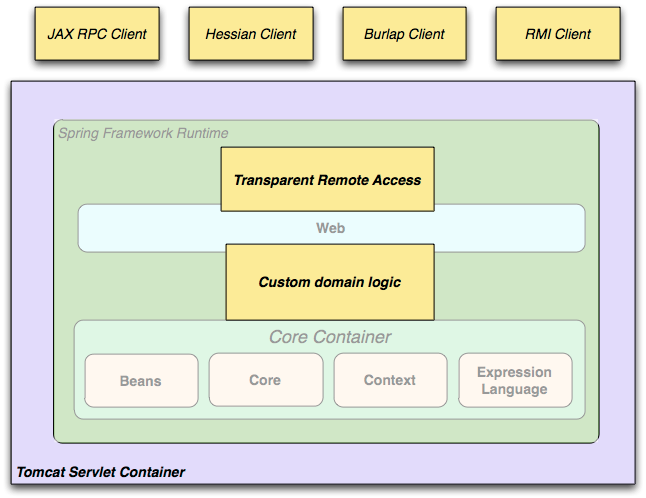
The building blocks described previously make Spring a logical choice in many scenarios, from applets to full-fledged enterprise applications that use Spring's transaction management functionality and web framework integration.

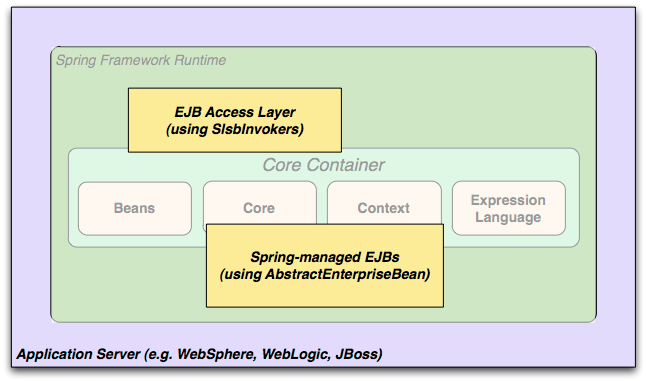
Typical full-fledged Spring web application

Spring's [declarative transaction management features](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/transaction.html" \l "transaction-declarative) make the web application fully transactional, just as it would be if you used EJB container-managed transactions. All your custom business logic can be implemented with simple POJOs and managed by Spring's IoC container. Additional services include support for sending email and validation that is independent of the web layer, which lets you choose where to execute validation rules. Spring's ORM support is integrated with JPA, Hibernate, JDO and iBatis; for example, when using Hibernate, you can continue to use your existing mapping files and standard Hibernate SessionFactory configuration. Form controllers seamlessly integrate the web-layer with the domain model, removing the need for ActionForms or other classes that transform HTTP parameters to values for your domain model.

Spring middle-tier using a third-party web framework

Sometimes circumstances do not allow you to completely switch to a different framework. The Spring Framework does not force you to use everything within it; it is not an all-or-nothing solution. Existing front-ends built with WebWork, Struts, Tapestry, or other UI frameworks can be integrated with a Spring-based middle-tier, which allows you to use Spring transaction features. You simply need to wire up your business logic using an ApplicationContext and use a WebApplicationContext to integrate your web layer.

Remoting usage scenario

When you need to access existing code through web services, you can use Spring's Hessian-, Burlap-, Rmi- or JaxRpcProxyFactory classes. Enabling remote access to existing applications is not difficult.

EJBs - Wrapping existing POJOs

The Spring Framework also provides an [access and abstraction layer](https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/ejb.html) for Enterprise JavaBeans, enabling you to reuse your existing POJOs and wrap them in stateless session beans for use in scalable, fail-safe web applications that might need declarative security.

### Dependency Management and Naming Conventions

The process of dependency management involves locating those resources, storing them and adding them to classpaths. Dependencies can be direct (e.g. my application depends on Spring at runtime), or indirect (e.g. my application depends on commons-dbcp which depends on commons-pool). The indirect dependencies are also known as "transitive" and it is those dependencies that are hardest to identify and manage.

If you are going to use Spring you need to get a copy of the jar libraries that comprise the pieces of Spring that you need. To make this easier Spring is packaged as a set of modules that separate the dependencies as much as possible, so for example if you don't want to write a web application you don't need the spring-web modules.

#### Maven Dependency Management

If you are using Maven for dependency management you don't even need to supply the logging dependency explicitly. For example, to create an application context and use dependency injection to configure an application, your Maven dependencies will look like this:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>3.0.0.RELEASE</version>

<scope>runtime</scope>

</dependency>

</dependencies>

Note the scope can be declared as runtime if you don't need to compile against Spring APIs, which is typically the case for basic dependency injection use cases.