Intro to Spring Framework

For Spring Framework 4.3.x
Non-Magical

Aside: What does “Spring” mean?

- In different contexts, it may refer to the Spring Framework or the wider Spring Project Ecosystem

- The Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications.

- The Spring Project Ecosystem is a collection of projects that complement and extend the Spring Framework

Aside: Spring Projects

- When we use “Spring” we’ll normally mean the entire project ecosystem. Sometimes we’ll just be referring to the Spring Framework though. While context should make it clear, it’s better to be specific.

- http://www.spring.io/projects
  - Note: check out Spring Framework at this link!

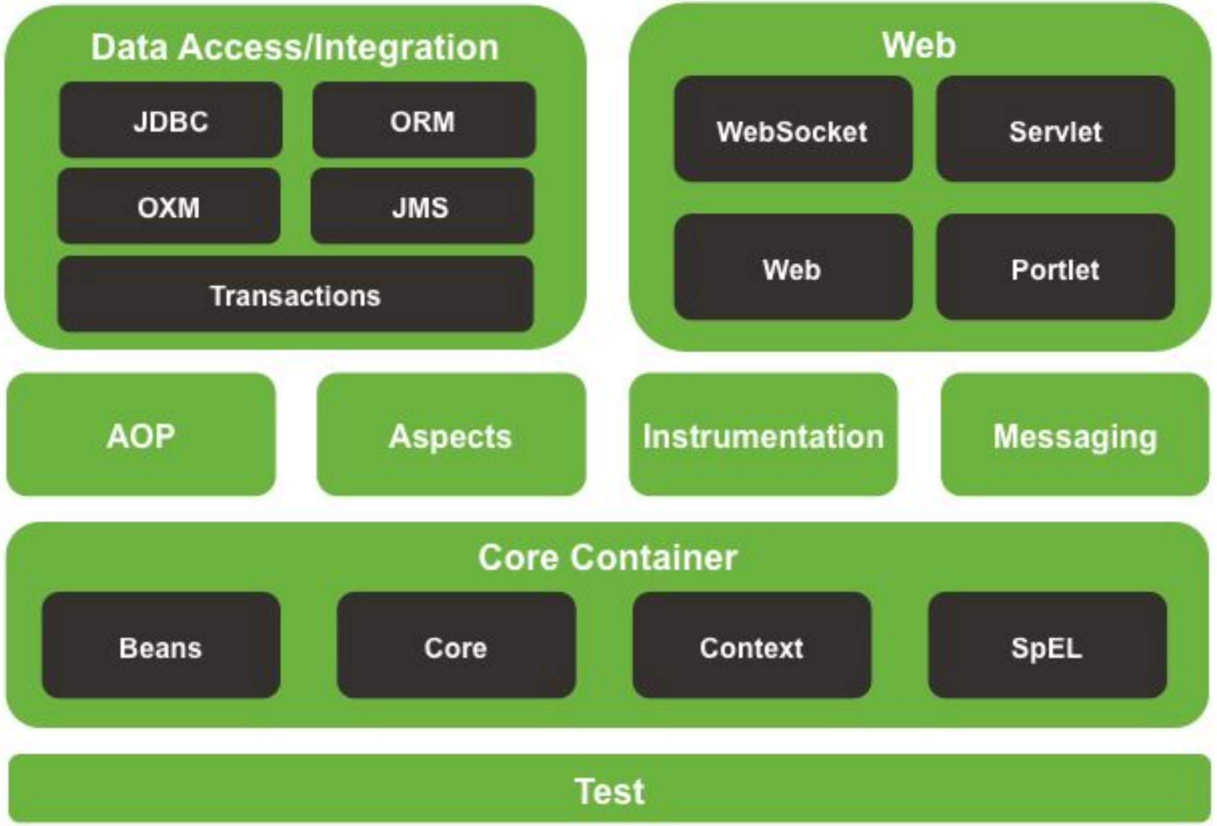
Spring Framework

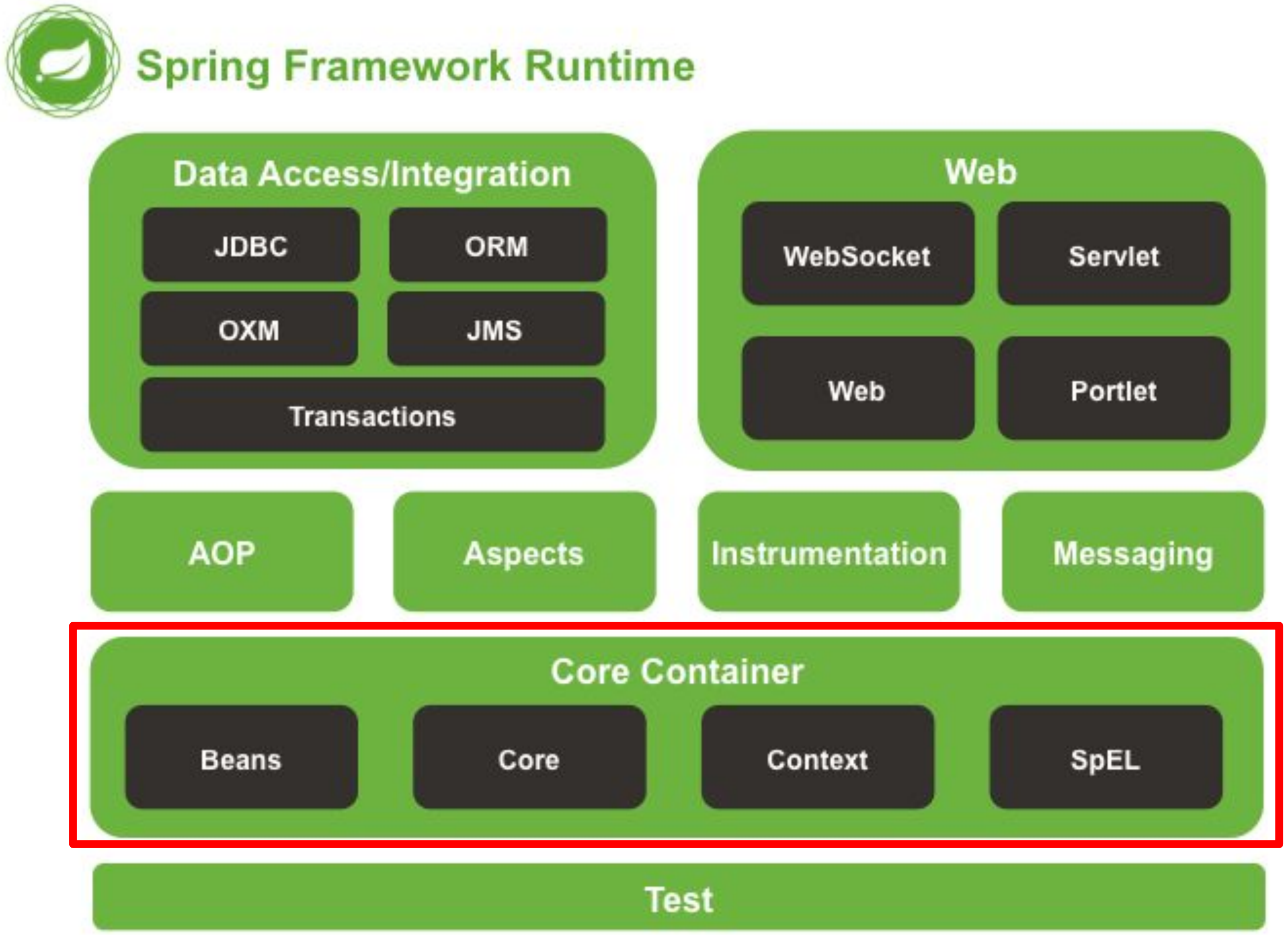
- The Spring Framework is a lightweight solution and a potential one-stop-shop for building your enterprise-ready applications. The Spring Framework provides infrastructure that supports your business logic.

- Spring Framework is modular, allowing you to use only those parts that you need, without having to bring in the rest. It contains about 20 modules.

- The core of the Spring framework includes an Inversion of Control (IoC) Container that manages your Java objects

Spring Framework Runtime




Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Spring Framework Core Container

4 modules in total, but we’ll only be looking at 3 of them:

- spring-core
- spring-beans
- spring-context

Core and Beans “provide the fundamental parts of the framework”: The IoC Container and beans.

Context might as well be fundamental for us -- we’ll never use core and beans without also using context.

Aside: What is a “Bean”?

- “Bean” is a term we should be careful with. We’ll normally use “Bean” to refer to “Spring Bean”, but in other cases it might refer to “Enterprise Java Bean”, or EJB. It might also refer to a regular “JavaBean”. As always, it’s best to be specific.
- Spring Beans are used with the IoC Container in Spring
- EJBs are used with the EJB Container in J2EE
- JavaBeans are Java objects that satisfy:
    - No args constructor
    - Private properties with getters and setters
    - Serializable

Spring Beans

- In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans.
- A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container.
- Otherwise, a bean is simply one of many objects in your application.
  - Since there are no additional requirements for Spring Beans like there are for Java Beans, we refer to Spring Beans as POJOs, or Plain Old Java Objects.
- Not every Java object you use has to be a Spring Bean, even if you’re using Spring Framework

IoC Container

- The goal of an IoC Container is to decouple execution from configuration. By separating these concerns, an application's code base becomes more modular, loosely coupled, with less focus on how code will be implemented and more on its business logic.

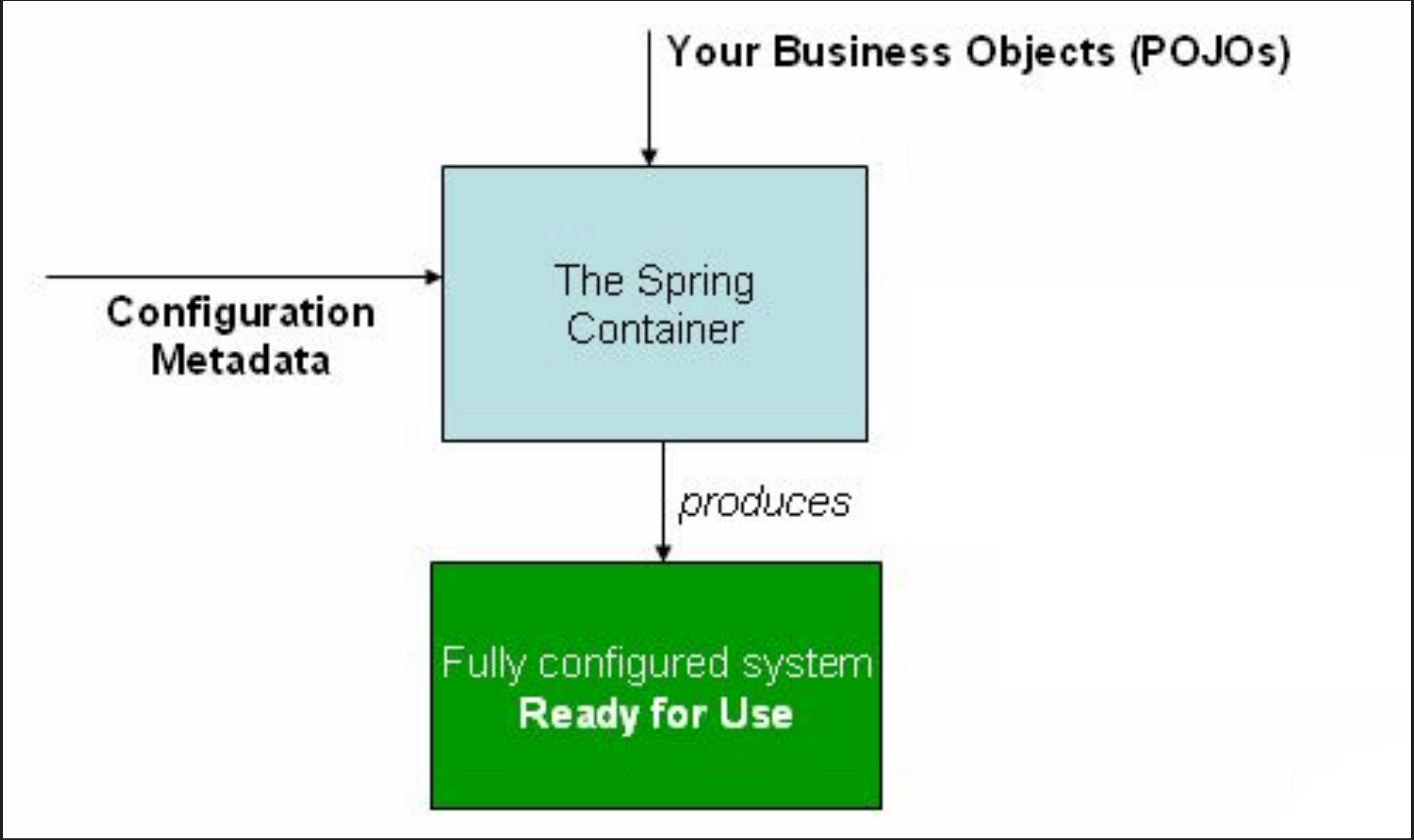
- The IoC container is responsible for instantiating, configuring, and assembling the beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.

Configuration

- Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

- Spring allows us to configure our container and beans in three different ways:
  - Classic XML configuration
  - Annotation-driven configuration with @Component, @Controller, etc.
  - Java configuration with @Configuration and @Bean

- Many modules provide their own preconfigured beans


Your Business Objects (POJOs)
Configuration Metadata
The Spring Container
produces
Fully configured system
Ready for Use

IoC Specifics : ApplicationContext

- The IoC container IS the ApplicationContext in Java.
- ApplicationContext is provided by the spring-context module
- ApplicationContext is an interface
    - It’s also a sub-interface of BeanFactory
- Earlier versions of the Spring Framework used the BeanFactory as the IoC container.

ApplicationContext vs BeanFactory

- The Bean Factory was the original interface for Spring, and has been superceded by the more capable Application Context.

- Bean Factory was a sophisticated implementation of the factory design pattern which lazily and programmatically instantiate beans as singletons.

- Application Context is an extension of the Bean Factory, eagerly instantiating beans and capable of defining several different scopes besides singleton.

Creating The ApplicationContext

- Several factories exist that can build the ApplicationContext, according to the manner of configuration:
  - ClassPathXmlApplicationContext or FileSystemXmlApplicationContext for XML files
  - XmlWebApplicationContext for XML while using Spring-Web
  - AnnotationConfigApplicationContext for java configuration.

What the ApplicationContext does

Once your ApplicationContext is built from configuration metadata, it manages your beans and their dependency relationships. Remember ApplicationContext loads beans eagerly, then manages them. Managing a bean involves three tasks:

- Providing Dependency Injection (Bean Wiring)
- Managing The (Spring) Bean Lifecycle
- Managing Bean Scope

Review: Dependency Injection

“A process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean.”

Objects ONLY define their dependencies via arguments during construction or post-construction property setting. The container then handles instantiating/finding/injecting those dependencies, rather than the object.

Aside: IoC and Dependency Injection

- The Spring docs cite a 2004 blog post by one Martin Fowler as the source of the term “Dependency Injection” as a better/more specific descriptor than “Inversion of Control”, though both terms refer to the same thing.
- https://martinfowler.com/articles/injection.html
- You’ll want to use the term “IoC Container” to talk about the container at the core of the Spring Framework

Bean Wiring

- The process of actually connecting beans and their configured dependencies is called “Wiring.” We set this up in our configuration:
  - property element attributes in an XML configuration file
  - Annotations
- Dependencies can be registered and referenced either by name or by type (Java Class).
- The easiest method is “Autowiring”, where we use the @Autowired annotation
  - Default Autowiring finds beans by their type.

The Bean Lifecycle

- In general, the setup phase involves instantiation of the object, handling of its dependencies, initialization of properties and default values, and any custom initialization methods before the bean is ready for use within the program.

- The teardown phase dereferences the bean when it passes out of scope (or the container is itself shutdown), but also calls any custom destroy methods along the way.

The Bean Lifecycle, Specifically:

- Instantiate Bean
- Populate Bean Properties
- BeanNameAware.setBeanName()
- BeanFactoryAware.setBeanFactory()
- BeanPostProcessor Pre-Initialization
- InitializingBean.afterPropertiesSet()
- (Optional) Custom init method
- BeanPostProcessor Post-Initialization

Bean Ready for use!

- DisposableBean.destroy()
- (Optional) Custom destroy method

Interacting with the Bean Lifecycle

- As a rule, we do not need to interfere with the lifecycle.
- If we want to interact with it, we can implement the InitializingBean/DisposableBean interfaces on our beans and override their methods.
- We can also define custom init and destroy methods in the XML

Bean Scope

The scope tells us how many objects in Java correspond to a given Bean Definition. Remember that BeanFactory only had one scope: singleton. The normal ApplicationContext has two scopes:

- Singleton: Only a single bean object exists in the container.
- Prototype: Any number of bean objects can exist in the container.

To see more scopes, we’ll have to wait until we use a WebApplicationContext.