Spring概况

<http://spring.io/>

<http://projects.spring.io/spring-framework/>

<http://maven.springframework.org/release/org/springframework/spring/>

Spring作为一个轻量级的J2EE的开源开发框架，为企业级的复杂应用的开发提供了一套简单轻量级的解决方案。但也不止应用于企业应用了，还有互联网应用，服务端应用等等。Spring为复杂的JavaEE开发带来了春天。

框架的特点

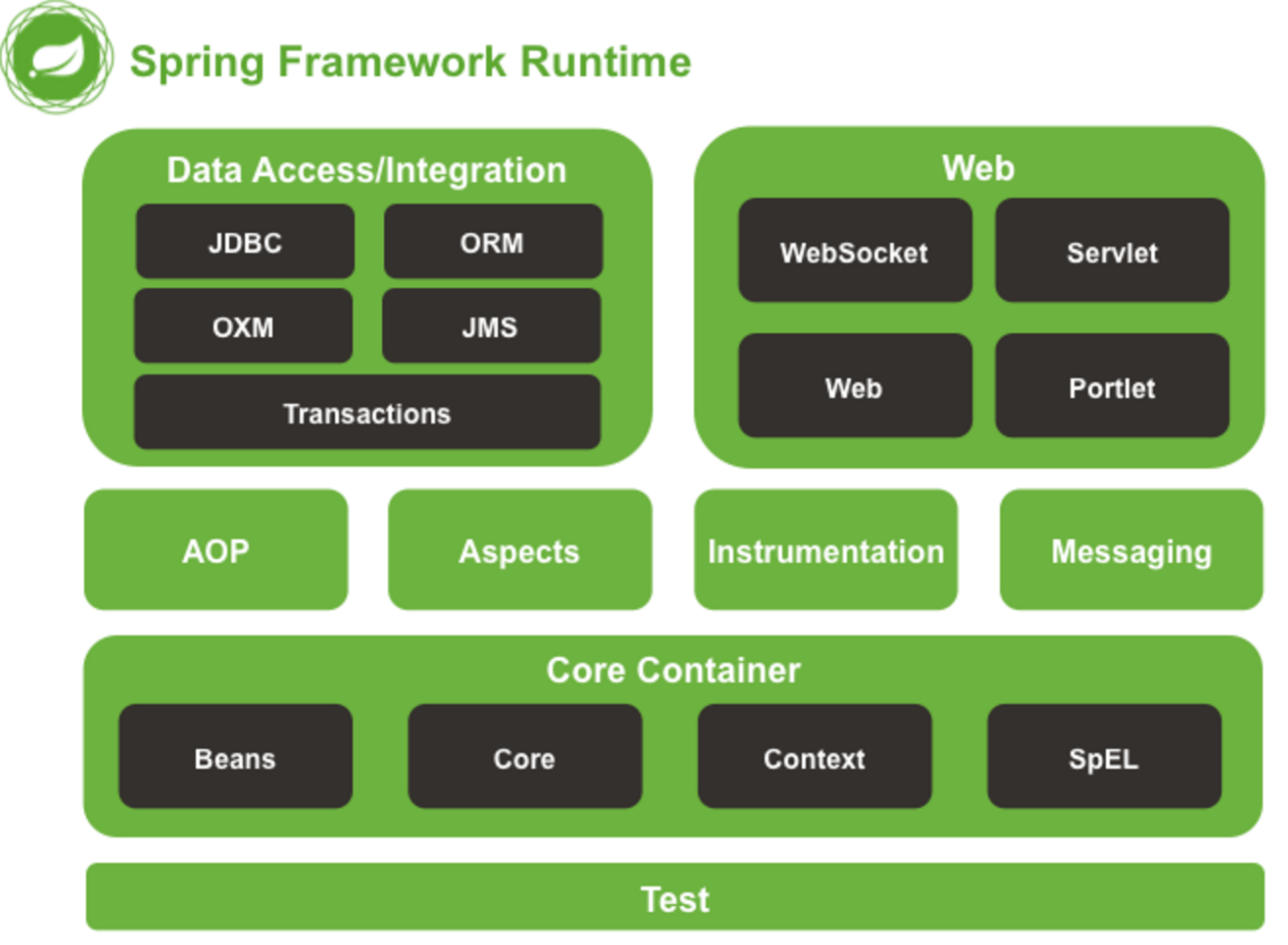
1. 半成品
2. 封装了特定的处理流程和控制逻辑
3. 成熟的、不断升级改进的软件

框架与类库的区别

1. 框架一般是封装了逻辑、高内聚的，类库这是松散的工具组合
2. 框架专注于某一领域，类库则是更通用的

Spring最初是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架。

1. 从大小与开销方面而言Spring都是轻量的。
2. 通过控制反转IoC达到松耦合的目的。
3. 提供了面向切面AOP编程的支持，允许通过分离应用的业务逻辑与系统级服务进行内聚性的开发。比如，商品管理中商品的入库、出库就是业务逻辑；系统服务指比如入库和出库都要进行统一的日志记录、事务管理与具体业务无关，无论什么业务都会涉及到。
4. 包含并管理对象的配置和生命周期，这个意义上是一种容器，比如Spring IoC容器管理Beans对象。
5. 将简单的组件配置、组合成为复杂的应用，这个意义上是框架。对主流应用框架(Hibernate, Struts等)提供了良好的支持。



适用范围：

1. 构建企业应用 (SpringMVC + Spring + Hibernate/MyBatis)
2. 单独使用Bean容器 (Bean管理)
3. 单独使用AOP进行切面处理
4. 其他的Spring功能，比如：对消息的支持等
5. 在互联网中的应用….

Spring框架的主要优势之一就是其分层架构，允许开发者选择哪一个组件。同时，也为J2EE开发提供了一个集成的框架。

Spring框架使用基本的JavaBeans来代替EJB, 并且提供了更多企业级应用的功能。Spring致力于提供统一的方法来管理业务对象。Spring是全面的和模块化的，分层结构使得可以选择它孤立的任何一个部分。它的设计从底部帮助你编写易于测试的代码。

其他优点：

* 低侵入式设计，代码污染极低
* Write Once, Run Anywhere
* Dependency Injection有效的降低了耦合度
* AOP提供了通用任务的集中管理，从而更好的复用。任务比如：安全，事务管理，日志等
* ORM和DAO机制提供了与第三方持久层框架的良好整合，简化了对数据库的访问
* 高度开放性，并不强制。该特点贯穿于表现层，业务层，持久层。
* 可以有效的组织中间层对象。
* 使用统一的配置文件，消除使用各种不同格式的属性定制文件的需要，在整个工程应用中可以通过一致的方法来配置。
* 促进良好的编程习惯，减少编程代价。
* 易于简单的单元测试。提供了对JUnit的支持，通过注解来测试。
* 使EJB成为一种备选。
* 为数据存取提供了一致的框架。

主要特点：

* 方便解耦，简化开发。 通过Spring提供的IoC容器，可以将对象之间的依赖关系交由Spring来控制。避免硬编码，造成的程序过渡的耦合。用户可以不再为单实例模式类，属性文件解析等很底层的需求来编写代码。可以更加专注上层的应用。
* AOP编程的支持，用户可以实现面向切面的编程，许多不能用传统OOP实现的功能都可以用AOP编程来轻松面对。
* 声明式事务的支持
* 方便程序测试
* 方便集成各种优秀框架，比如Struts, Hibernate等。
* 降低JavaEE API的使用难度。比如：JDBC, Java Mail, 远程调用等。
* Spring的源码是经典学习范例

Spring框架搭建

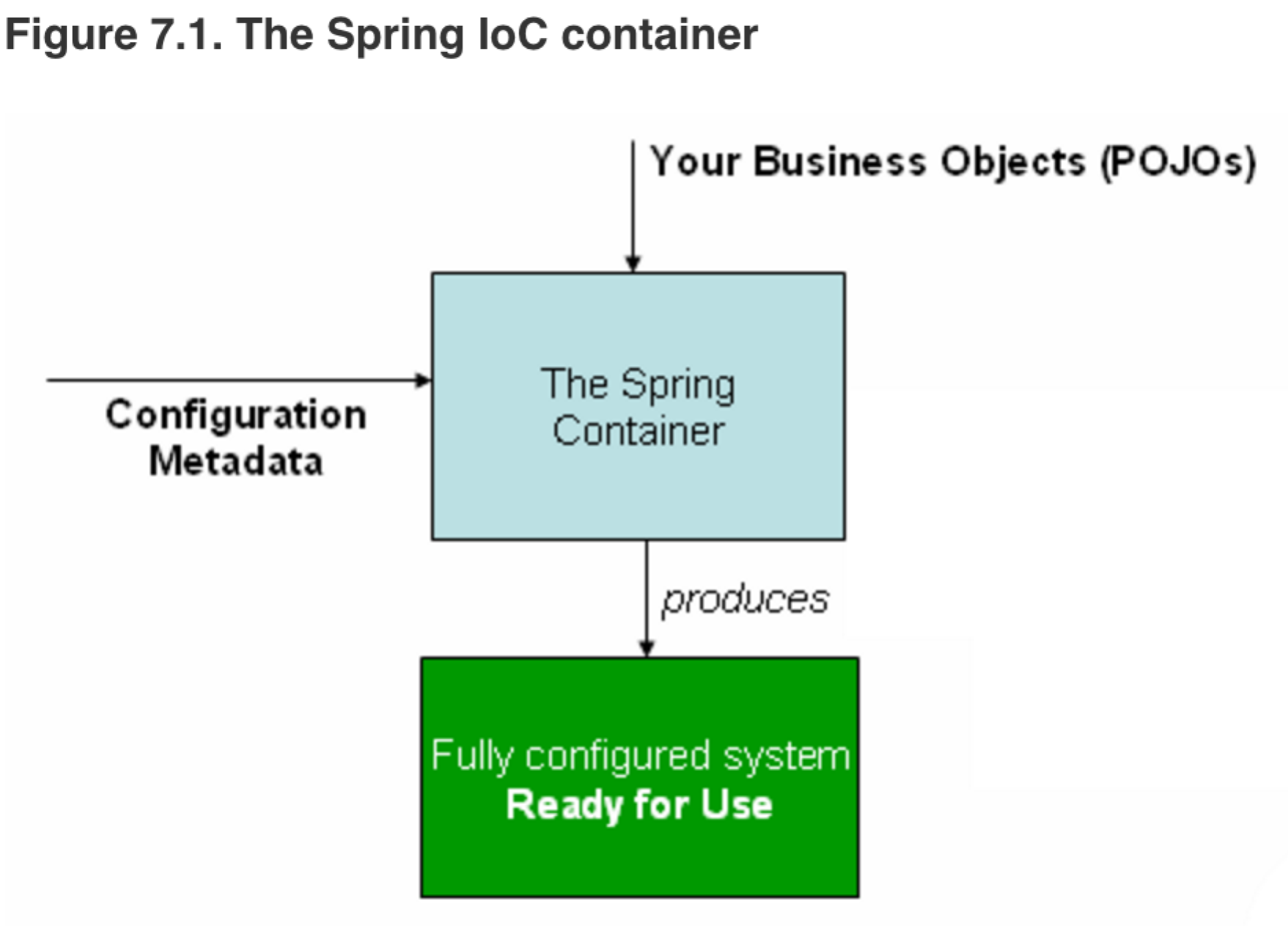
<http://www.tutorialspoint.com/spring/spring_environment_setup.htm>

Spring第一个Hello Word程序测试

<http://www.tutorialspoint.com/spring/spring_hello_world_example.htm>

Spring之IoC (控制反转Inversion of Control)

控制反转IoC: 获得依赖对象的过程被反转了, 一种控制权的转移。控制被反转后，应用程序本身不涉及对象的创建和维护，而是由外部容器负责创建和维护。也即 获得依赖对象的过程由自身管理变为了由Spring IoC容器主动注入。 Dependency Injection (DI)依赖注入是其一种实现方式， 就是由IoC容器在运行期间，动态的将某种依赖关系注入到对象之中，从而创建了对象并且组装了依赖对象之间的关系。

The container gets its instructions on what business objects to instantiate, configure, and assemble by reading configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The diagram is a high-level view of how Spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.

注意！！一个Spring IoC容器就是指的是一个上下文Context！！一个应用程序中，可以有多个Spring IoC容器即多个Context！！！

**Bean**

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 managed by a Spring IoC container. These **beans are created with the configuration metadata** that you supply to the container, for example, in the form of XML <bean/> definitions, which you have already seen in previous chapters. 注意：IoC容器中，所有的对象都称之为Bean. Beans需要通过配置文件来配置的！！

The bean definition contains the information called configuration metadata, which is needed for the container to know the followings:

* How to create a bean
* Bean's lifecycle details
* Bean's dependencies

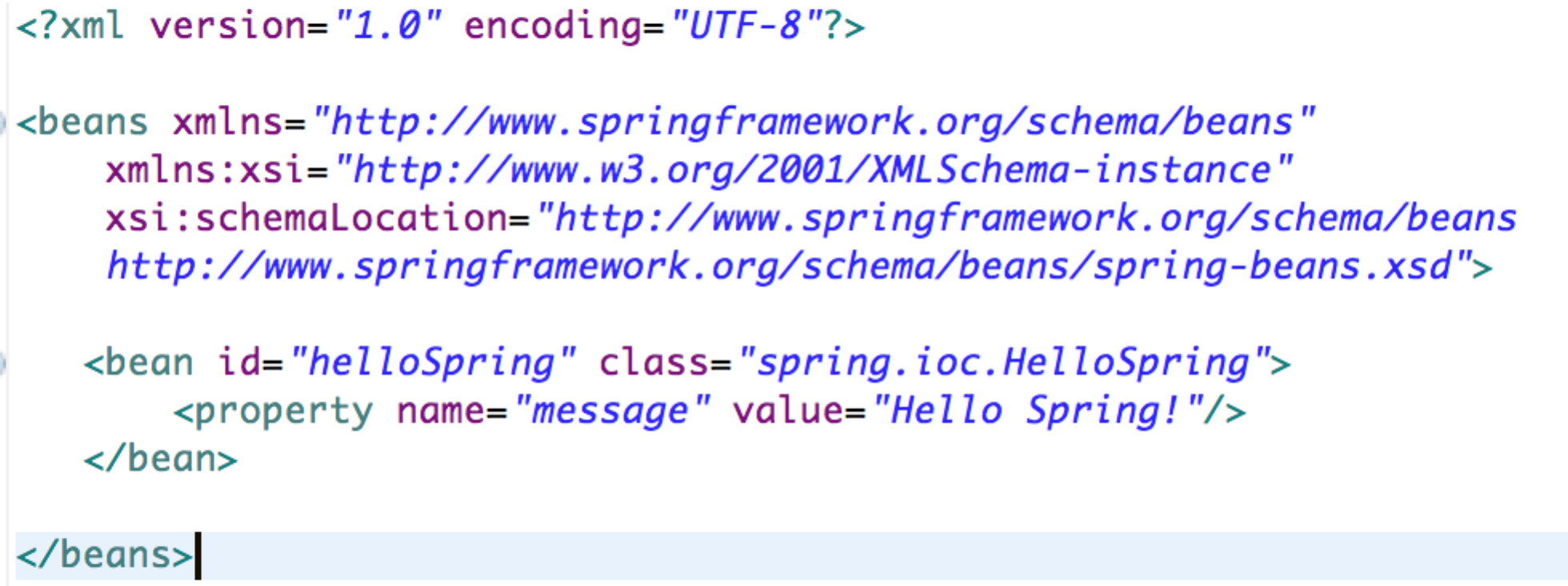
All the above configuration metadata translates into a set of the following properties that make up each bean definition.

注意！！一个Spring IoC容器就是指的是一个上下文Context！！一个应用程序中，可以有多个Spring IoC容器即多个Context！！！

**Bean Configuration File (Beans的配置项)**

Spring的Beans配置一般通过xml来实现。This file needs to be created under the **src** directory. 一般取名为Beans.xml

The Beans.xml is used to assign unique IDs to different beans and to control the creation of objects with different values without impacting any of the Spring source files.



<bean></bean>可以用于配置一些properties, 以上图为例：

class

This attribute is mandatory and specifies the bean class to be used to create the bean. 具体要实例化的哪一个类，是必须的。如果想得到一个Bean对象，可以通过类型来获取，即根据class的名字来得到。注意：class属性配置时要跟完整的类名，也即要包括package名。

id

This attribute specifies the bean identifier uniquely. In XML-based configuration metadata, you use the id and/or name attributes to specify the bean identifier(s). 在IoC容器中，bean的唯一标识。如果想得到一个Bean对象，可以通过配置id属性，然后根据id名字来得到。

properties

This is used to inject the dependencies. 用于实现依赖注入dependency injection, 具体参考下面笔记。设值注入, setter-based dependency injection

scope

This attribute specifies the scope of the objects created from a particular bean definition.该bean的作用域。

constructor-arg

This is used to inject the dependencies. 用于实现依赖注入dependency injection, 具体参考下面笔记。构造注入, constructor-based dependency injection

lazy-initialization mode

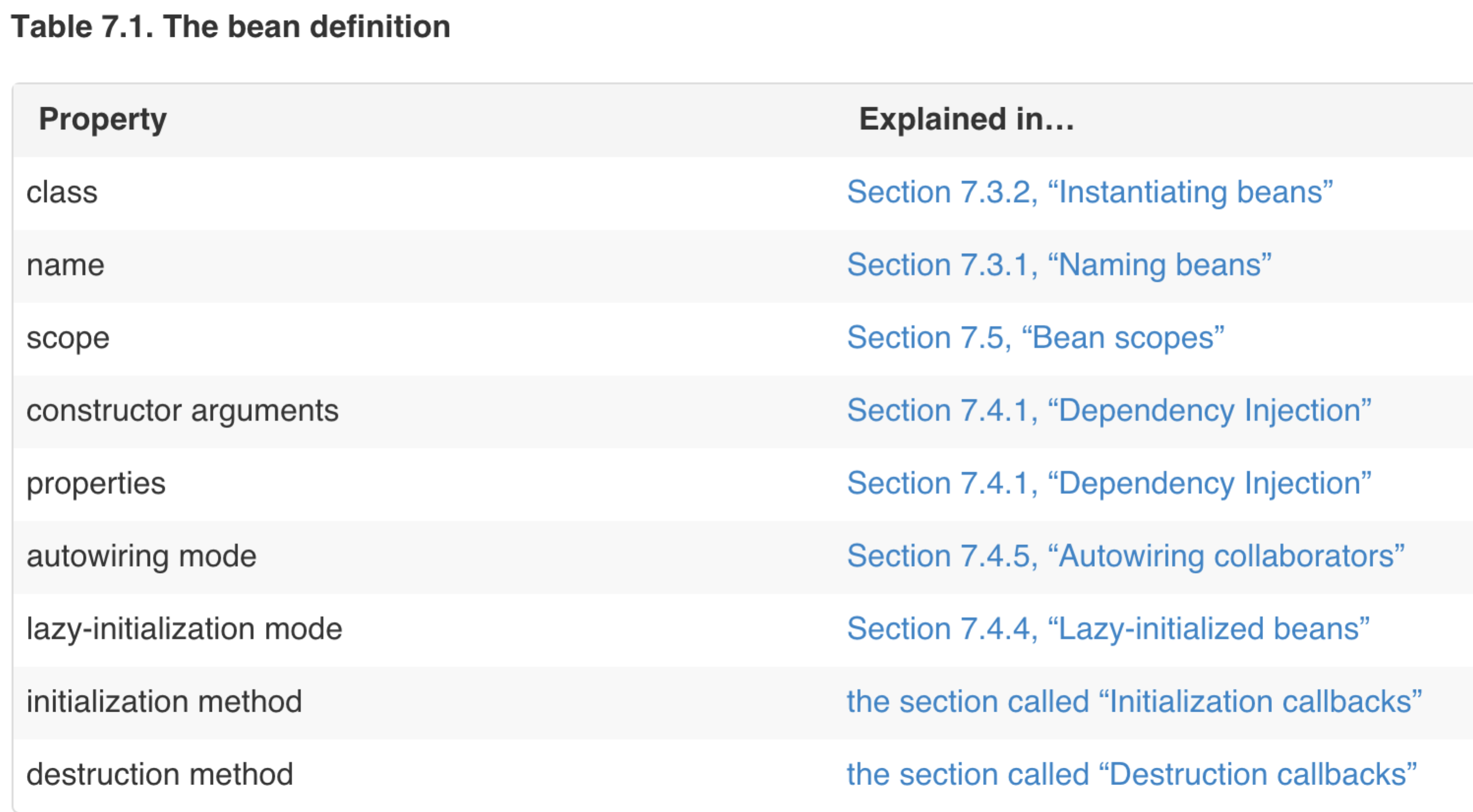
A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at startup.

initialization method

A callback to be called just after all necessary properties on the bean has been set by the container.

destruction method

A callback to be used when the container containing the bean is destroyed.



还有一些可以参考如下：

<http://www.tutorialspoint.com/spring/spring_bean_definition.htm>

**Bean的初始化、配置加载**

Beans的初始化、配置加载过程涉及到两个基础的package, 一个是org.springframework.beans; 一个是org.springframework.context

在第一个XXX.beans包中，其中的BeanFactory类以及其子类提供配置结构和基本功能，加载并初始化Beans; 而在XXX.context包中的ApplicationContext类及其相关类保存了Beans对象并在Spring中被广泛使用。



初始化Beans时，常用的几种方式来得到ApplicationContext

1. 本地文件，指定具体的Beans配置文件的path, 来加载，一般放在src目录下。
2. 通过classpath相对工程的相对路径来指定。
3. Web应用中依赖Servlet或Listener

IoC容器在初始化、加载Beans配置时，会扫描beans.xml文件中的beans相关的配置，实例化这些beans以及完成依赖注入, 下面提到的Spring的依赖注入就是在随着这些beans的创建过程中，完成对成员变量的赋值。

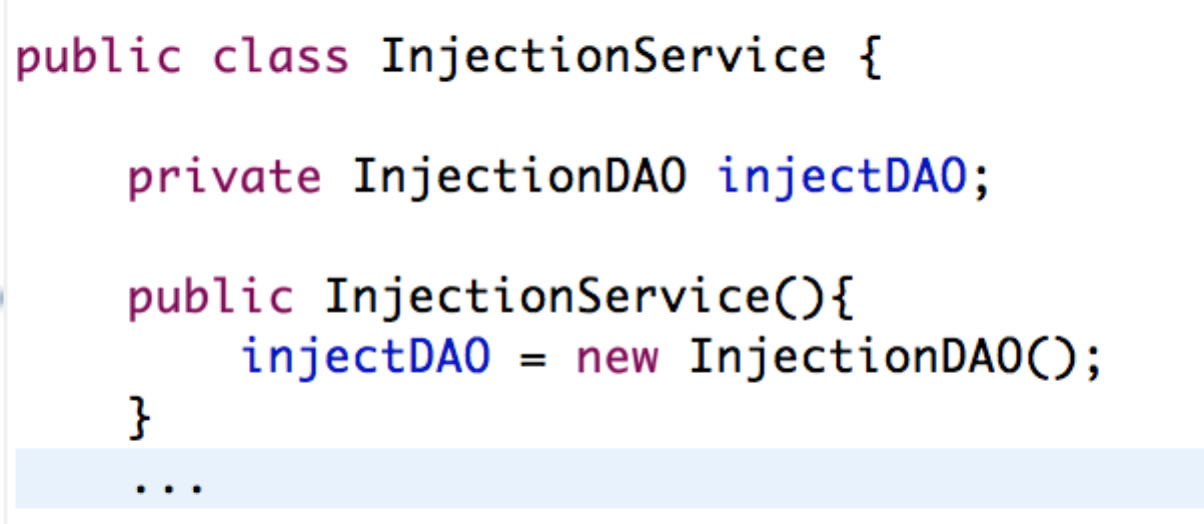
注意：IoC容器初始化、加载Beans时，默认就会实例化对象了。如果希望在第一次请求对象时才创建对象，需要配置<bean></bean>中的lazy-initialization mode这个属性。

注意！！一个Spring IoC容器(bean容器)，就是指的是一个上下文Context！！一个应用程序中，可以有多个Spring IoC容器即多个Context！！！

**Dependency Injection(wiring)依赖注入 (重要！！！！)**

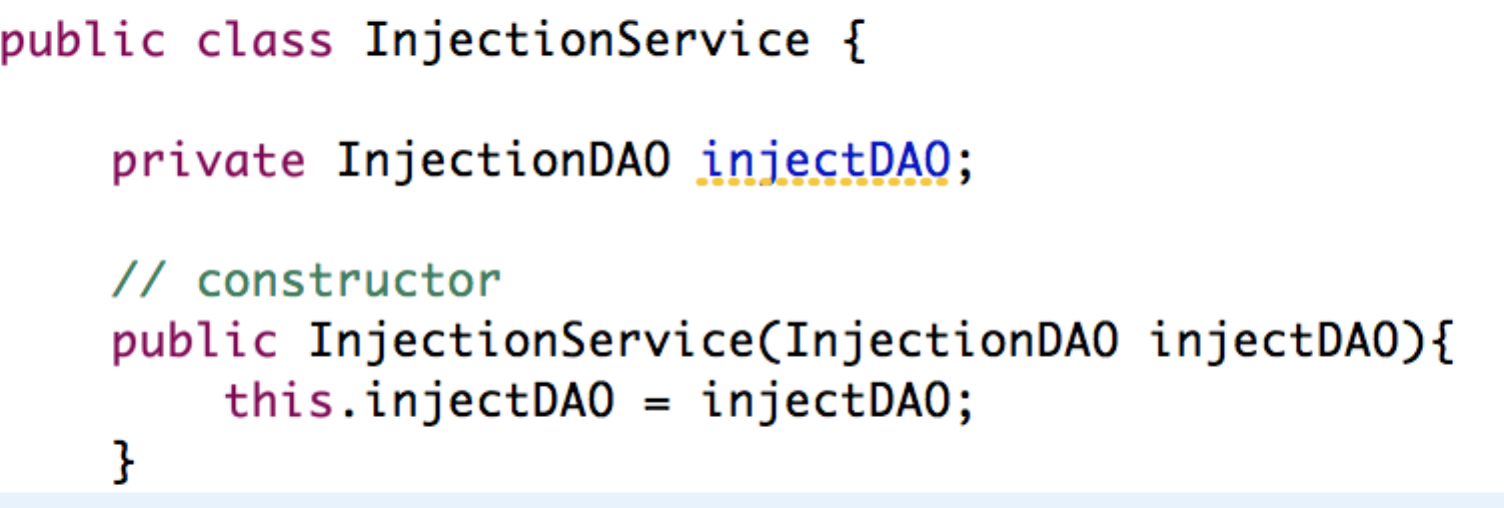
Spring注入是指在启动Spring容器后也即IoC容器初始化、加载beans配置的时候，完成对变量的赋值行为。常用的两种注入方式：设值注入和构造注入

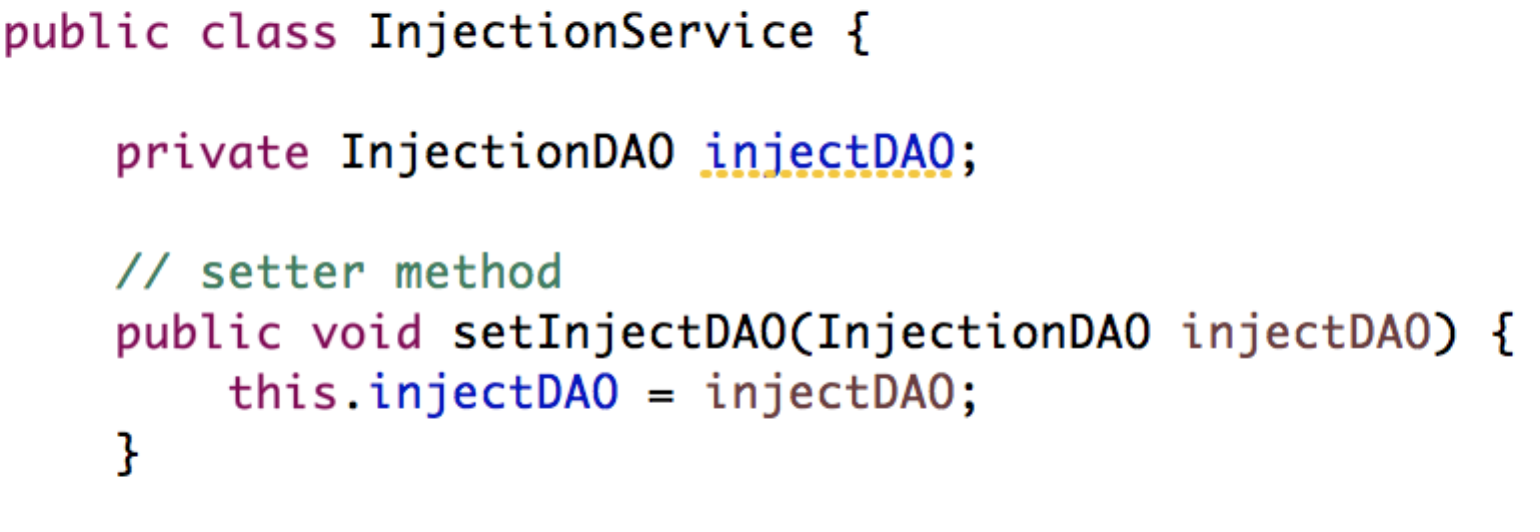
<http://www.tutorialspoint.com/spring/spring_dependency_injection.htm>



假如我们有一个application (另外一个类), 它有一个业务是实现InjectionService, 其中涉及到Injection DAO层，那么我们可能有如上的代码。因此，我们说在InjectionService和InjectionDAO之间是存在dependency的。

但是在Spring框架中，我们可以通过IoC容器，按照下面的2种方式来实现。



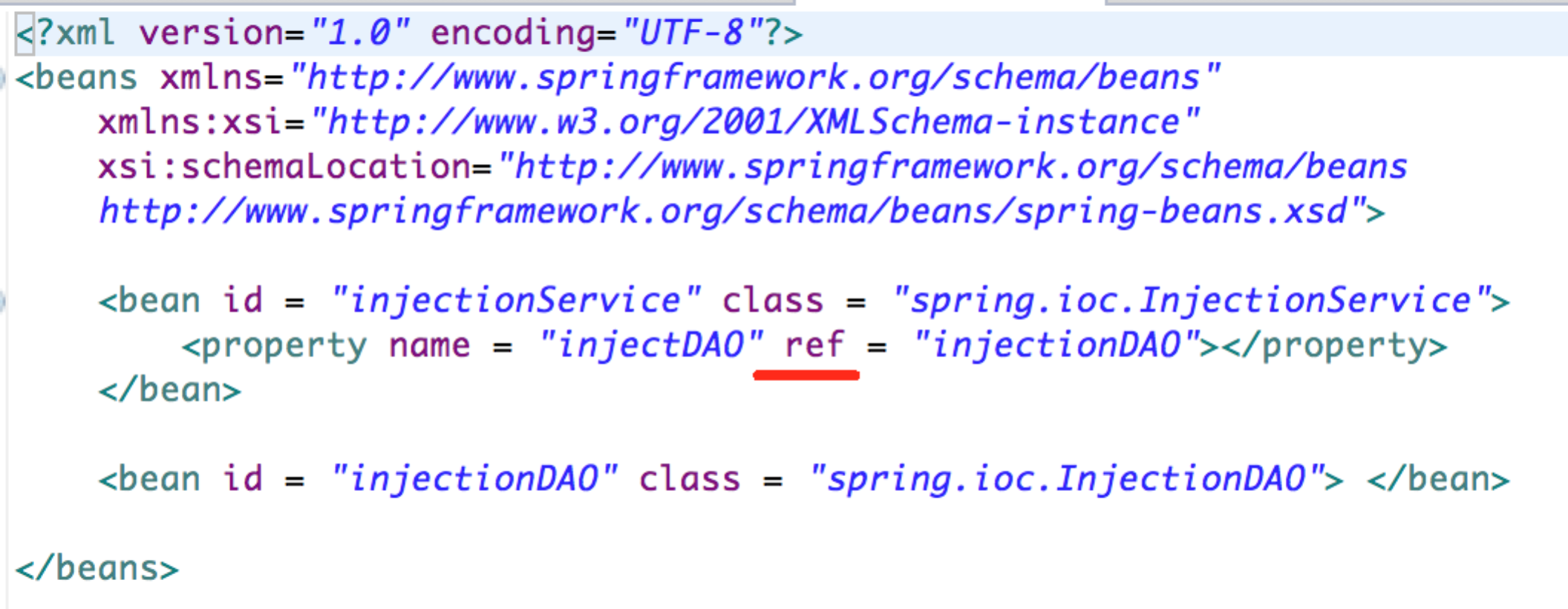


We have removed the total control from **InjectionService** and kept it somewhere else (ie. XML configuration file) and the dependency (ie. class **InjectionDAO**) is being injected into the class **InjectionService** through a Class Constructor. Thus flow of control has been "inverted" by Dependency Injection (DI) because you have effectively delegated dependences to some external system.

Second method of injecting dependency is through Setter Methods of **InjectionService** class where we will create **InjectionDAO** instance and this instance will be used to call setter methods to initialize **InjectionService** ’s properties.

**setter-based dependency injection**

设值注入，就是通过一个属性或成员变量的setter方法来进行依赖注入。Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

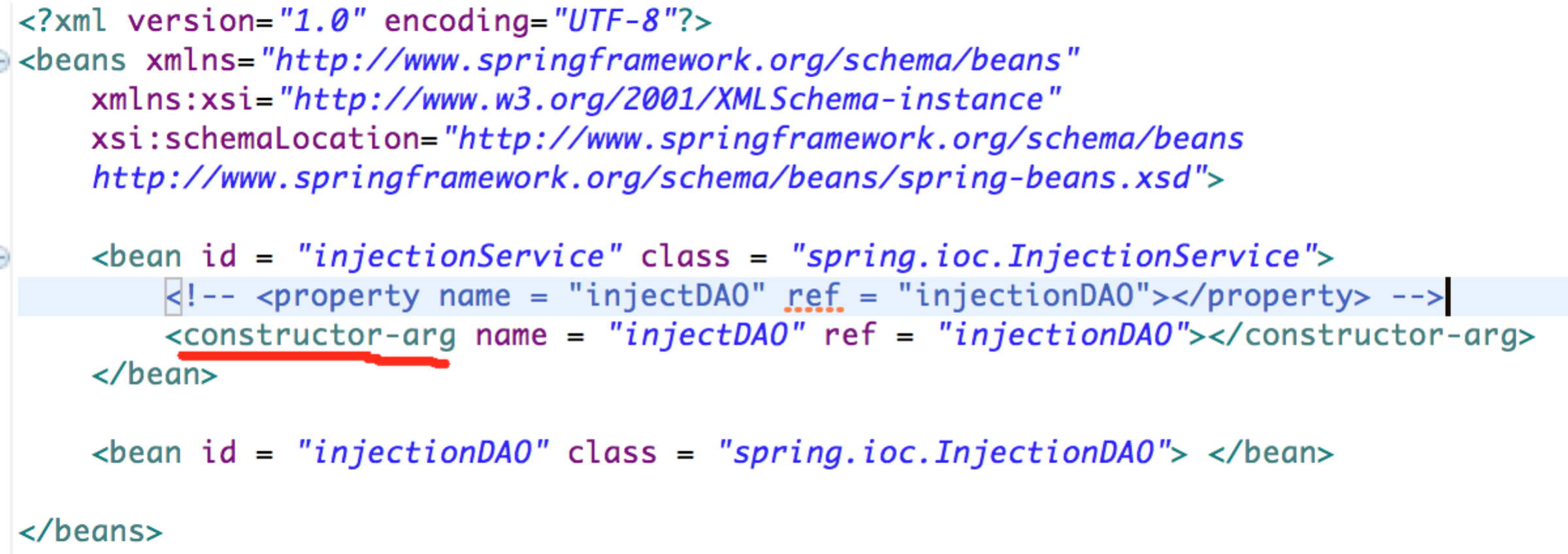


注意：第一个bean配置中的<property>标签里的name的值就是一个成员变量，该成员变量的setter method需要另外传入一个对象引用作为参数也即InjectionDAO, 因此通过ref作为一个引用，然后ref的值就是另外一个Bean的id.

设值注入会自动调用类中相关成员变量的setter method.

注意：如果成员变量是String类型的话，即setter method的参数也是String, 那么在依赖注入时，在beans.xml配置时可以使用name = “XXXX” value = “YYYY”的方法，完成DI.

**constructor-based dependency injection**



Constructor-based DI (构造注入), is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.

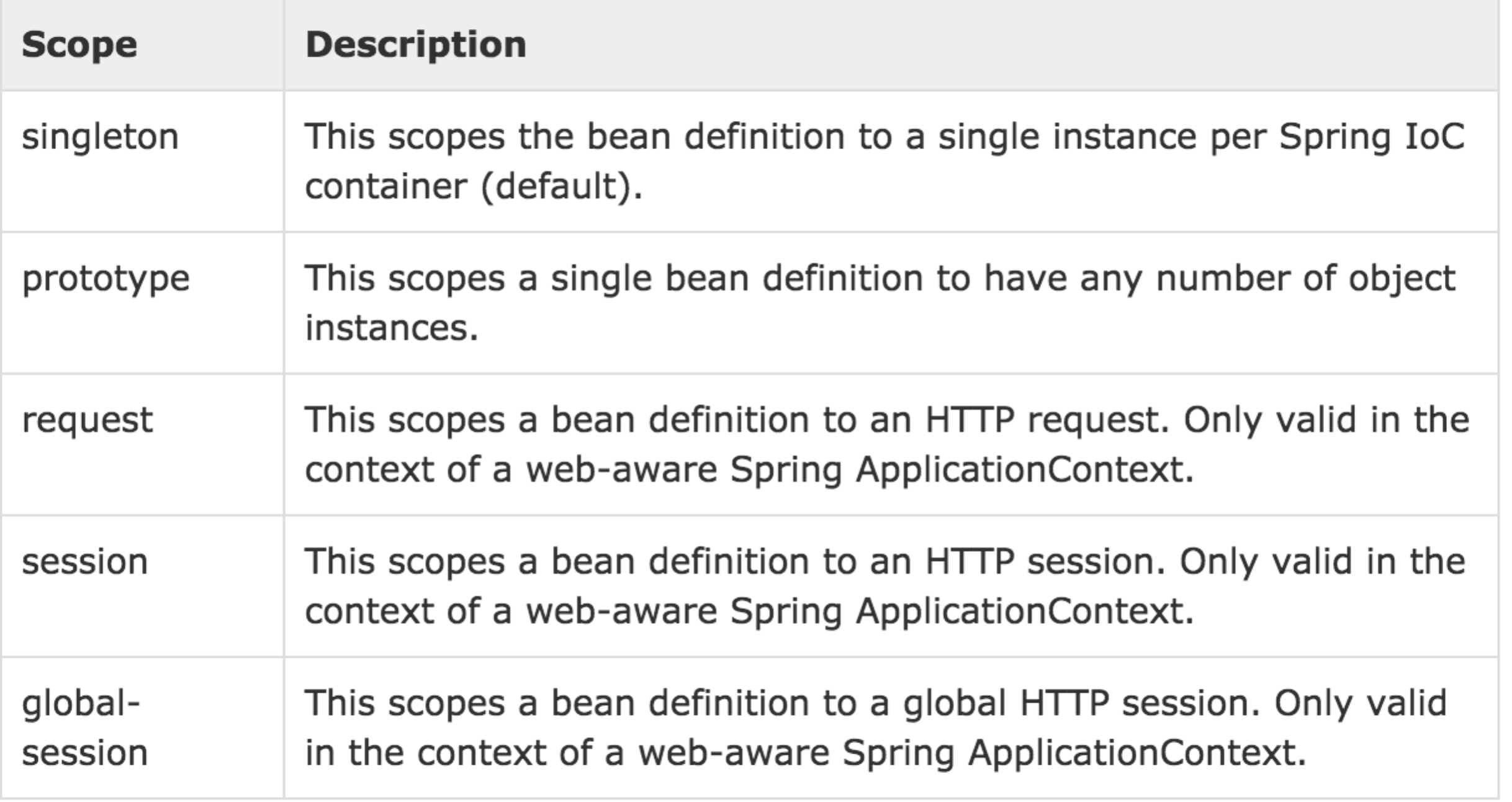
You can mix both Constructor-based and Setter-based DI, but it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies.

Code is cleaner with the DI principle and decoupling is more effective when objects are provided with their dependencies. The object does not look up its dependencies, and does not know the location or class of the dependencies rather everything is taken care by the Spring Framework.

注意！！一个Spring IoC容器(bean容器)，就是指的是一个上下文Context！！一个应用程序中，可以有多个Spring IoC容器即多个Context！！！

**Bean的作用域 (Bean Scope)**

The Spring Framework supports following five bean scopes, three of which are available only if you use a web-aware ApplicationContext.



注意：默认情况的bean scope是单例singleton, 指一个Spring IoC容器中只存在一份。

注意！！一个Spring IoC容器(bean容器)，就是指的是一个上下文Context！！一个应用程序中，可以有多个Spring IoC容器即多个Context！！！

注意：如果bean scope是prototype相当于是每次请求bean(每次使用)是都会new新的实例，但destroy方法不生效！！

另外三种bean scope都是与Web应用有关的。Bean scope为request的话，每次http请求创建一个实例且仅在当前request内有效；同理，bean scope为session的话，每次http请求创建一个实例仅在当前session内有效；还有一个全局的global session. 注意生存周期！

注意：Spring IoC容器(bean容器)也即一个上下文context, 再通过getBean()方法得到Bean对象时返回的还是一个Object类型，需要通过转型得到具体的类型。

**Bean的生命周期 (Bean life cycle)**

The life cycle of a Spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.

//----------------------------------------------------------------------------------------------------------------------//

Definition(定义), bean的定义在src目录下的beans.xml文件中配置的。

Initialization(初始化), 当Spring IoC容器初始化、加载beans.xml配置文件时，完成了bean的初始化，生成bean的实例。

Usage(使用), 在单元测试或者实际开发中，从Spring IoC容器即一个context容器中，取出一个bean来使用，比如通过id或者class来得到一个bean.

Destruction(销毁), 在Spring IoC容器即一个context容器中，当容器停止时要去销毁destroy由这个容器所创建的那些bean的实例。

//----------------------------------------------------------------------------------------------------------------------//

接下来介绍两个在bean life cycle中，非常重要的callback函数。分别在初始化initialization和销毁destruction的时候执行。

We simply declare the <bean> with **init-method** and/or **destroy-method** parameters. The **init-method** attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, **destroy-method** specifies a method that is called just before a bean is removed from the container.

Spring之AOP (面向切面)

Spring框架中的核心模块之一，也是系统中其他组建模块和应用开发的基础。