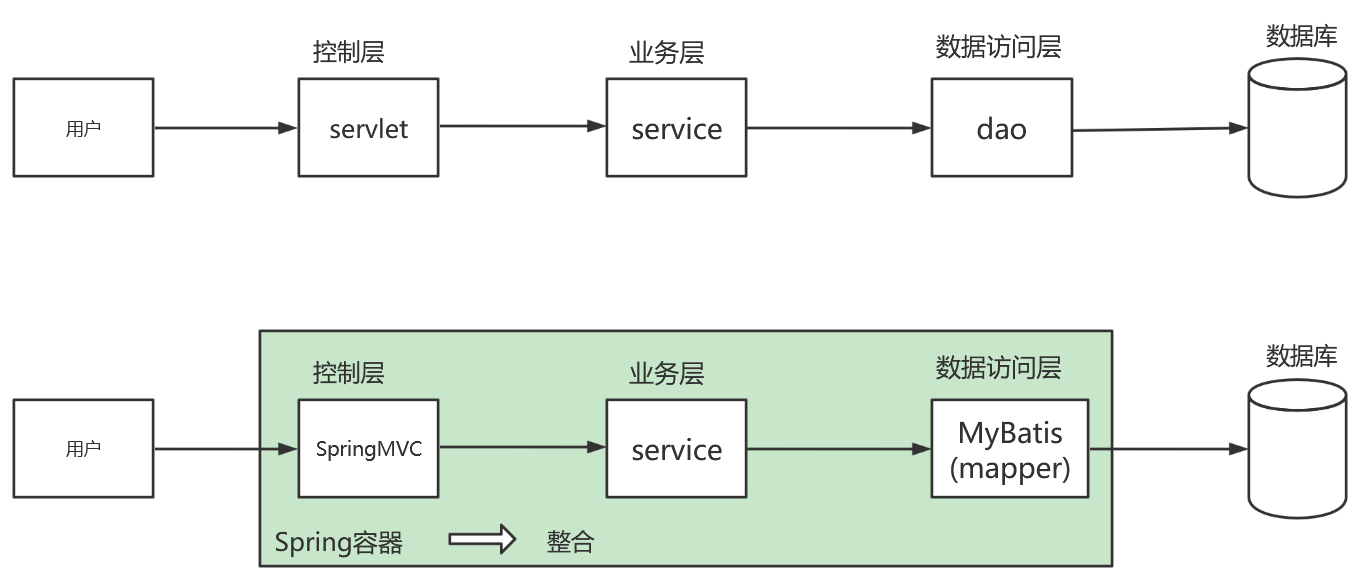
## Spring Framework

Spring框架是一个Java平台，为开发Java应用程序提供全面的基础架构支持。可以将单层框架以最佳的组合揉和在一起建立一个连贯的体系。Spring处理基础结构，让我们的精力专注于应用程序开发。

Spring使我们能够从POJO[普通的Java对象]构建应用程序，并将企业服务非侵入性地应用于POJO。此功能适用于Java SE编程模型以及全部和部分Java EE。



◆目的：使JAVAEE开发更加容易

◆功能：使用基本的JavaBean(就是一个具体和公共的，具有无参数构造器的Java类)开发和部署多层结构的Java应用系统的跨平台的构件体系结构，并提供了更多的企业应用功能

◆范围：任何Java应用

Spring是一个轻量级控制反转(IOC)和面向切面(AOP)的容器框架。

IOC技术介绍:

**控制反转**（Inversion of Control，缩写为**IoC**），是[面向对象编程](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1%E7%BC%96%E7%A8%8B" \t "https://baike.baidu.com/item/%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC/_blank)中的一种设计原则，可以用来减低计算机代码之间的[耦合度](https://baike.baidu.com/item/%E8%80%A6%E5%90%88%E5%BA%A6" \t "https://baike.baidu.com/item/%E6%8E%A7%E5%88%B6%E5%8F%8D%E8%BD%AC/_blank)。最常见的方式是**依赖注入**（Dependency Injection，简称**DI**）

|  |
| --- |
| 本来由资源使用者直接创建对象使用,改为由第三方(IOC容器)控制被使用对象的实例创建和生命周期,使用者直接调用即可,即控制反转  什么是反转?  正转:由资源使用者直接创建对象使用,  反转:就是由第三方new好对象,通过依赖注入(@AtutoWard)的手段让使用者直接用 |
|  |

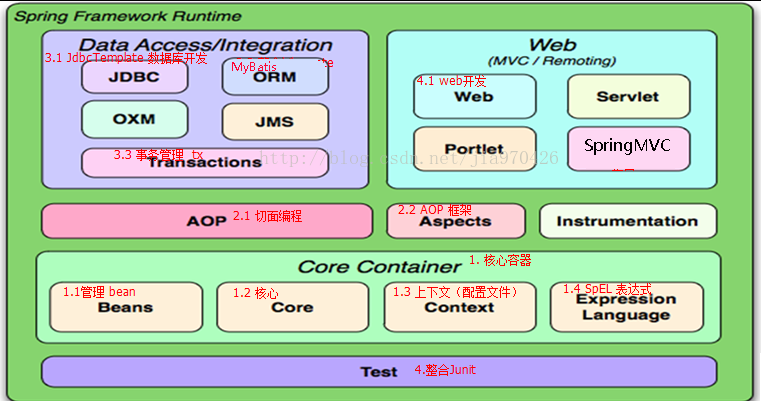
Sping中文官方api文档(用google浏览器可以在线翻译为中文)

<https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html>

在线中文 Spring5.0 API文档  
https://www.cnblogs.com/yiwangzhibujian/p/5893219.html

## **模块**

Spring框架包含组织为约20个模块的功能。这些模块分为核心容器，数据访问/集成，Web，AOP（面向方面​​的编程），检测，消息传递和测试，如下图所示



### 需要的jar包的介绍:

**spring-core-4.1.6.RELEASE.jar**

核心模块提供了框架的基本组成部分，包含IOC和依赖注入功能。

**spring-beans-4.1.6.RELEASE.jar**

Bean模块提供BeanFactory，它是一个工厂模型的复杂实现

**spring-context-4.1.6.RELEASE.jar**

上下文模块建立在有核心和Bean模块提供的坚实的基础上，他是访问定义和配置的任何对象的媒介，AppliactionContext接口是上下文接口的重点

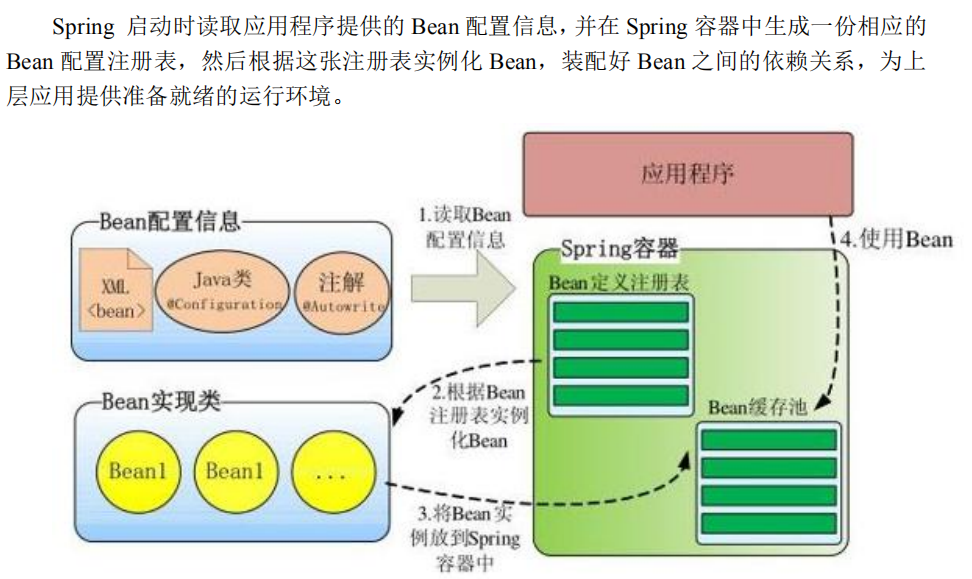
**spring-expression-4.1.6.RELEASE.jar**

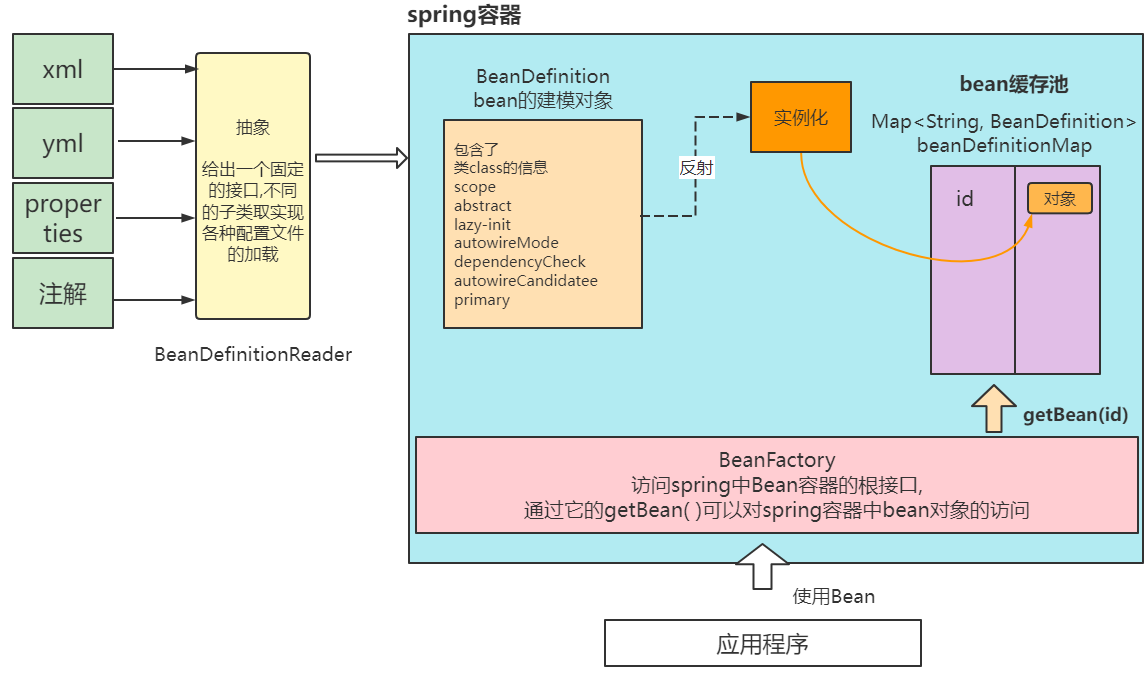
模块提供了一种功能强大的[表达式语言，](D:/2004%E7%8F%AD%E8%B5%84%E6%96%99/%E8%AF%BE%E4%BB%B6%E6%96%87%E6%A1%A3/05-SSM%E6%A1%86%E6%9E%B6/2-Spring/%E8%B5%84%E6%96%99/spring-framework-4.1.6.RELEASE-dist/spring-framework-4.1.6.RELEASE/docs/spring-framework-reference/html/expressions.html" \o "8.春季表达语言（SpEL）)用于在运行时查询和操作对象图。它是对JSP 2.1规范中指定的统一表达语言（统一EL）的扩展。该语言支持设置和获取属性值，属性分配，方法调用，访问数组，集合和索引器，逻辑和算术运算符，命名变量以及按名称从Spring的IoC容器中检索对象的内容。它还支持列表投影和选择以及常见的列表聚合。

### 构建第一个项目：

### 默认装配方式：Spring执行第一个程序

**Spring的运行原理:**





### Bean的作用域(单例模式singleton\原型模式prototype)

scope="singleton"

singleton类型的bean定义从容器启动到第一次被请求而实例化开始，只要容器不销毁或退出，Spring IoC 容器中只会存在一个共享的 Bean 实例，无论有多少个Bean 引用它，始终指向同一对象。该模式在多线程下是不安全的。

scope="prototype"

每次通过 Spring 容器获取 prototype 定义的 bean 时，容器都将创建

一个新的 Bean 实例

如果要创建两个不同的bean对象

方式1：

|  |
| --- |
|  |

方式2：

|  |
| --- |
|  |

### 基于XML的注入(di)

所谓注入，可理解为对象的属性赋值

1 简单数据类型和引用数据类型注入

|  |
| --- |
| *<!--下面等同于User user = new User()-->* <**bean id="user" class="com.spring.pojo.User"**>  *<!--给对象user的属性注入值:相当于user.setUname("静静")-->* <**property name="uname" value="静静"**></**property**>  *<!--相当于user.setPwd("123")-->* <**property name="pwd" value="123"**></**property**> </**bean**> |
| *//控制反转* @Test **public void** run2(){  *//加载spring.xml配置文件* ApplicationContext ac = **new** ClassPathXmlApplicationContext(**"spring.xml"**);*//spring.xml文件必须放在src目录下* User user = (User)ac.getBean(**"user"**);  System.***out***.println(user); } |
| *<!--下面等同于User user = new User()-->* <**bean id="user" class="com.spring.pojo.User"**>  *<!--给对象user的属性注入值:相当于user.setUname("静静")-->* <**property name="uname" value="静静"**></**property**>  *<!--相当于user.setPwd("123")-->* <**property name="pwd" value="123"**></**property**>   *<!--引用数据类型的注入值:使用ref-->* <**property name="account" ref="account"**></**property**> </**bean**>  <**bean id="account" class="com.spring.pojo.Account"**>  <**property name="aid" value="101"**></**property**>  <**property name="money" value="4500.0"**></**property**> </**bean**> |

通过构造器注入

|  |
| --- |
| *<!--下面等同于User user = new User()-->* <**bean id="user" class="com.spring.pojo.User"**>  <**constructor-arg name="uname" value="小莉"**></**constructor-arg**>  <**constructor-arg name="pwd" value="321"**></**constructor-arg**>  <**constructor-arg name="account" ref="account"**></**constructor-arg**> </**bean**>  <**bean id="account" class="com.spring.pojo.Account"**>  <**constructor-arg name="aid" value="102"**></**constructor-arg**>  <**constructor-arg name="money" value="12000.53"**></**constructor-arg**> </**bean**> |

2 域属性自动注入（byName\byType；局部和全局配置）

autowire=*"byName"*

default-autowire=*"byType"*

**局部自动注入:**

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd"**>   <**beans**>   *<!--下面局部的自动注入放在bean标签中;autowire="byType"是根据类型自动注入;autowire="byName": 根据属性名自动注入-->* <**bean id="user" class="com.spring.pojo.User"**  **autowire="byName"**>  <**property name="uname" value="静静"**></**property**>  <**property name="pwd" value="44444"**></**property**>  </**bean**>   <**bean id="account" class="com.spring.pojo.Account"**>  <**constructor-arg name="aid" value="102"**></**constructor-arg**>  <**constructor-arg name="money" value="12000.53"**></**constructor-arg**>  </**bean**>  </**beans**> </**beans**> |
|  |

**全局配置配置自动注入**

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd"  default-autowire="byType"** >   <**beans**>   *<!--下面局部的自动注入放在bean标签中;autowire="byType"是根据类型自动注入;autowire="byName": 根据属性名自动注入-->* <**bean id="user" class="com.spring.pojo.User"**>  <**property name="uname" value="静静"**></**property**>  <**property name="pwd" value="44444"**></**property**>  </**bean**>   <**bean id="account" class="com.spring.pojo.Account"**>  <**constructor-arg name="aid" value="102"**></**constructor-arg**>  <**constructor-arg name="money" value="12000.53"**></**constructor-arg**>  </**bean**>  </**beans**> </**beans**> |

3 空字符串或null的注入

使用<value/>相当于:””

<null/>：null

<property name="name" ><value/></property>

<property name=*"name"* ><null/></property>

4空值注入

|  |
| --- |
| <**bean id="user" class="com.spring.pojo.User" autowire="byName"**>  *<!-- <property name="uname" ><value/></property> -->  <!-- <property name="uname" value=""></property>-->  <!-- <property name="uname" value="null"></property>-->* <**property name="uname"** ><**null**/></**property**>  <**property name="pwd" value="44444"**></**property**> </**bean**> |

5 集合属性注入(array、set、list、map、properties)(**了解**)

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  https://www.springframework.org/schema/beans/spring-beans.xsd"  default-autowire="byName"** >   <**beans**>  <**bean id="data" class="com.spring.pojo.Data"**>  <**property name="arr"**>  <**array**>  <**value**>长沙</**value**>  <**value**>株洲</**value**>  <**value**>湘潭</**value**>  </**array**>  </**property**>   <**property name="list"**>  <**list**>  <**value**>11</**value**>  <**value**>22</**value**>  <**value**>33</**value**>  </**list**>  </**property**>  <**property name="set"**>  <**set**>  <**value**>湖南</**value**>  <**value**>湖南</**value**>  <**value**>湖北</**value**>  <**value**>广东</**value**>  <**value**>广西</**value**>  </**set**>  </**property**>  <**property name="map"**>  <**map**>  <**entry key="长沙" value="0731"**></**entry**>  <**entry key="北京" value="010"**></**entry**>  <**entry key="广州" value="020"**></**entry**>  </**map**>  </**property**>  <**property name="pro"**>  <**props**>  <**prop key="provice"**>广东</**prop**>  <**prop key="phone"**>020</**prop**>  <**prop key="语言"**>粤语</**prop**>  </**props**>  </**property**>  </**bean**>   </**beans**> </**beans**> |

## 基于注解的DI注入(使用更常见)

1环境搭建：导入aop包（spring-aop-4.1.6.RELEASE.jar）, 添加context约束头信息（组件扫描器）

2常用注解：

@Component

@Value:给属性的基本数据类型赋值,可以放在属性上,页可以放在属性的set方法上

@Scope:指定类是单态模式还是原型模式

给属性的应用数据类型注值

@Autowired : 设置对象类型的属性的值(byType)。但它是按照对象的类型来注入，跟注解的名字无关(来自spring)。

@Resource : 设置对象类型的属性的值(byType)。但它是按照对象的类型来注入，跟注解的名字无关(来自JDK的注解)

@Qulifer(“bean的name”): 与@Autowired搭配,指定根据bean的name自动注入,也可以单独放在方法中,指定name来注入的bean对象

|  |
| --- |
| https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#xsd-schemas-context |
|  |

## Spring的IOC(注解)

@Component:

表示当前修饰的类交给Spring容器管理

修饰一个类，将这个类交给Spring管理。

与@Component相同功能的还有三个衍生注解，都是用来修饰类：

@Repository:添加在mapper的接口类上(数据访问层)

@Service：添加在Service实现类上(业务层)

@Controller:添加在Controller类上(控制层)

@Configration:添加在用于配置信息的类上(springboot中自动配置)

### Bean的其他的注解

Bean作用范围的注解

@Scope("singleton ")：默认是单态(单例)模式（常用）

@Scope("prototype")：原型模式(常用)

### IOC的XML和注解开发比较



使用场景：

XML:可以使用任何场景

开发中,用来注入框架实例

基于xml操作的是.class文件

注解：有些地方用不了，这个类不是自己写的(注解必须写在源代码上)

开发中,用来注入自己写的java类

基于注解操作的是源代码