**spring框架入门到熟练**

官网:<http://spring.io/>

文档:<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/>

**spring的概况**

* spring是一个开源框架，为了解决企业应用开发的复杂性而创建的，现在已不止应用于企业运用。
* 是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

**优点：**

* 从大小与开销两方面而言都是轻量级的。
* 通过控制反转（IoC）的技术达到松耦合的目的。
* 提供了面向切面编程的丰富支持，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。
* 包含并管理应用对象的配置和生命周期，这个意义上是一种容器。

**为什么是spring？**

* 开发应用简单：对象创建，销毁，生命周期，生命过程管理，都可交给spring来做。
* 开发应用方便：把对象交给spring管理后，需要用实例时，通知spring， 即可返回所需实例。
* 开发应用快捷：……..

**spring作用：**

* 容器
* 提供了对多种技术的支持   
  —JMS （java message service）  
  —MQ支持   
  —UnitTest   
  —…….
* AOP(事物管理,日志)
* 提供了众多方便应用的辅助类(JDBC Template)
* 对主流应用框架(Hibernate 等)提供了良好的支持

**适用范围**

* 构建企业应用(SpringMVC+Spring+Hibernate/ibatis)
* 单独适用Bean容器(Bean管理)
* 单独适用AOP进行切面处理
* 其他的Spring功能，如：对消息的支持等
* 在互联网中的应用

**什么是框架？**

* 软件框架，通常指的是为了实现某个业界标准或完成特定基本任务的软件组件规范，也指为了实现某个软件组件规范时，提供规范所要的基本功能的软件产品
* 框架就是制定一套规范或者规则(思想)，大家(程序员)在该规范或者规则下工作，或者说就是使用别人搭好的舞台，你来表演
* 框架的特点：   
  — 半成品   
  — 封装了特定的处理流程和控制逻辑   
  — 成熟的，不断升级改进的软件
* 框架与类库的区别   
  —框架一般是封装了逻辑，高内聚的，类库则是松散的工具组合。   
  —框架专注于某一个领域，类库则是更通用的。

**接口**

* 用于沟通的中介的抽象化
* 实体把自己提供给外界的一种抽象化说明，用于由内部操作分离出外部沟通方法，使其能被修改内部而不受外界影响其他实体与其交互的方式。
* 对应Java接口即声明，声明了哪些方法是对外公开提供的。
* 在 Java8中， 接口可以拥有方法体。

**面向接口编程**

* 结构设计中，分清层次及调用关系，每层只向外（上层）提供一组功能接口，各层间仅依赖接口而非实现类
* 接口实现的变动不影响各层间的调用，这一点在公共服务中尤为重要
* 面向接口编程“中的”接口”是用于隐藏具体实现和实现多态的组件

**IOC**

* IOC：控制反转，控制权转移，应用程序本身不负责依赖对象的创建和维护，而是由外部容器负责创建和维护。(获得依赖对象的过程被反转了。控制反转之后，获得依赖对象的过程由自身管理变为了由IOC容器主动注入。控制反转更合适的名字叫“依赖注入”。所谓依赖注入，就是由IOC容器在运行期间，动态地将某种依赖关系注入到对象之中 )
* DI (依赖注入)是一种实现方式
* 目的：创建对象并组装对象之间的关系

**Spring的Bean配置**

整个版本编码的说明

<?xml version="1.0" encoding="UTF-8"?>

下面是命名空间 和schenma 的位置

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<!--激活注解管理base-package指定注解管理的文件地址-->

<context:component-scan base-package="dao"/>

<!--bean 标签代表java中一个具体的对象 new-->

<bean class="entity.User">

<!--property 配置对象属性值 name 属性名，value 属性值(基本数据类型) ref 属性值(对象类型)-->

<property name="id" value="1"/>

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

<property name="password" value="123"/>

</bean>

</beans>

测试代码：

public class Test {

public static void main(String[] args) {

/\*1.指定配置文件地址\*/

String path="spring.xml";

/\*2读取配置文件生成工具类对象\*/

ApplicationContext context = new ClassPathXmlApplicationContext(path);

/\* getBean(类描述),从spring管理的对象中找到一个符合该类型的对象\*/

User bean = context.getBean(User.class);

System.out.println(bean.getId());

System.out.println(bean.getName());

System.out.println(bean.getPassword());

System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

//注解打印

TestDao dao1 = context.getBean(TestDao.class);

dao1.showUser();

TestDao dao2 = context.getBean(TestDao.class);

//每个bean 在spring 工厂中只会创建一个

/\*用==判断两个地址是否一样\*/

System.out.println(dao1==dao2);

}

}

**Bean容器初始化**

* **基础包：**

—org.springframework.core （spring核心jar包 猜测：是一个重用率很高的工具集合）   
—org.springframework.beans （spring创建对象相关模块 ）   
—org.springframework.context （spring对象使用相关模块）   
—BeanFactory 提供配置结构和基本功能，加载并初始化Bean   
—ApplicationContext保存了Bean对象并在Spring中被广泛使用

* **方式，ApplicationContext**

—本地文件   
FileSystemXmlApplicationContext context= new FileSystemXmlApplicationContext(“盘符路径.xml”)；

—Classpath   
ApplicationContext context = new ClassPathXmlApplicationContext(….xml);

—Web应用中依赖Servlet或Listener

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>context</servlet-name>

<servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>

</servlet>

**Spring注入**

* **\* Spring 注入是指在启动Spring容器加载bean配置的时候，完成对变量的赋值行为\***
* **常用的两种注入方法**

--设值注入

class 就是创建对象，相当于让spring做了new对象实例

property相当于调一个set方法给一个属性赋值必须命名规范，不要乱起名，会自动调用 ，Spring也不会违背封装属性，name 是一个成员变量， value 属性值(基本数据类型) ， ref 属性值(对象类型)引用

<bean class="entity.User">

<property name="id" value="1"/>

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

<property name="password" value="123"/>

<property name="userInfo" ref="userInfo"/>

<property name="bills">

<!-- list 子标签配置集合-->

<list>

<!-- ref 子标签调用bean-->

<ref bean="bill"/>

<ref bean="bill1"/>

<!-- <value>fdafdafds</value>-->

</list>

</property>

</bean>

--构造注入

<bean class="com.imooc.ioc.injection.service.InjectionService">

<constructor-org name="id" value="1"/>

</bean>

**Bean配置项**

* id：在整个IOC容器中这个bean的唯一标识
* Class：具体要实例化的哪一个类
* Scope ：范围（作用域）
* Constructor arguments：构造器的参数
* Properties：属性
* Autowiring mode：自动装配的模式
* lazy-initalization ：懒加载模式
* Initialization/destructio：初始化和销毁的方法

以上都是一些比较常用的

**Bean的作用域**

* singleton：单例，指一个Bean容器中只存在一份
* prototype：每次请求(每次使用)创建新的实例，destroy方式不生效
* Request：每次http请求创建一个实例且仅在Request内有效
* session：同上，每次http请求创建，当前session内有效
* global session: 基于portlet的wed中有效（portlet定义了global session），如果是在web中，同session

**Aware**

* Spring中提供了一些以Aware结尾的接口，实现了Aware接口的bean在被初始化之后，就可以获得相应资源
* 通过Aware接口，可以对Spring相应资源进行操作（一定要慎重）
* 为对Spring进行简单的扩展提供了方便的入口

**Bean的自动装配(Autowiring)**

* **No:**不做任何操作\*\*
* **byname:**根据属性名自动装配。此选项将检查容器并根据名字查找也属性完全一致的bean，并将其与属性自动装配
* **byType:**如果容器中存在一个与指定属性类型相同的bean，那么将与该属性自动装配;如果存在多个该类型bean，那么抛出异常，并指出不能使用byType方式进行自动装配;如果没有找出到匹配的bean，则什么事都不发生
* **Constructor:**与byType方式类似，不同之处在于它应用于构造参数。如果容器中没有找到与构造参数类型一致的bean，那么抛出异常

**Resources**

* 针对于资源文件的统一接口
* Resources
  + URLResource:URl对应的资源，根据一个URL地址即可构建
  + ClassPathResource:获取类路径下的资源文件
  + FileSystemResource:获取文件系统里面的资源
  + ServletContextResource:ServletContext封装的资源，用于访问ServletContext环境下的资源
  + InputStreamResource:针对于输入流封装的资源
  + ByteArrayResource:针对于字节数组封装的资源

**Bean管理的注解实现**

**Classpath扫描与组件管理**

* 从Spring3.0开始，Spring JavaConfig项目提供了很多特性，包括使用java而不是XML定义bean，比如@Configuration，@Bean，@import，@DependsOn
* @Component是一个通用注解，可用于任何bean
* @Repository，@Service，@Controller 是更有针对性的注解
  + @Repository通常用于注解DAO类，即持层
  + @Service通常用于注解Service类，即业务层
  + @Controller通常用于Controller类，即控制层（MVC）

**元注解(Meta-annotations)**

* 许多Spring提供的注解可以作为自己的代码，即“元数据注解”，元注解是一个简单的注解，可以应用到另一个注解

@Target((ElementType.TYPE))

@Retention(RetentionPolicy.RUNTIME)

@Deprecated

@Component

public @interface Service(){

//.....

}

* 除了value(),元注解还可以有其他的属性，允许定制

@Target((ElementType.TYPE))

@Retention(RetentionPolicy.RUNTIME)

@Deprecated

@Scope("session")

public @interface SessionScope{

ScopedProxyMode proxyMode() default ScopedProxyMode.DEFAULT

}

**类的自动检测及Bean的注册**

* Spring可以自动检测类并注册Bean到ApplicationContext中

@Service

public class SimpleMoviester{

private MocieFinder movieFinder;

@Autovired

public SimpleMovieLister(MocieFinder movieFinder){

this.movieFinder=movieFinder

}

}

@Repository

public class JpaMovieFinder implements MovieFinder{

// implementation elided for clarity

}

* 通过在基于XML的Spring配置如下标签(请注意包含上下文命名空间)
* 仅会查找在同一个applicationContext中的bean注解

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!--开启对象注解-->

<context:annotation-config/>

</beans>

* 为了能够检测这些类并注册相应的Bean ，需要下面内容

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<!--开启对象注解-->

<context:component-scan base-package="controller"/>

</beans>

* 包含,通常在使用前者后，不能再使用后者
* AutowiredAnnotationBeanPostProcessor和CommonAnnotationBeanPostProcessor也会被包含进来

**使用过滤器进行自定义扫描**

* 默认情况下，类被自动发现并 注册bean的条件是:使用@Component，@Repository，@Service，@Controller注解或者使用@Component的 自定义注解
* 可以通过过滤器修改上面的行为，如：下面例子的xml配置忽略所有的@Repository注解并用“stub”代替

<beans>

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

<context:include-filter type="regex" expression=".\*stub.\*Repository"/>

<context:exclude-filter type="annotation" expression ="org.springframework.stereotype.Repository"/>

<context:component-scan/>

</beans>

* 还可以使用use-default-filters=”false” 禁止自动发现与注册

**定义bean**

* 扫描过程中组件被自动检测，那么Bean名称是由BeanNameGeanertor生成的（@Component，@Repository，@Service，@Controller都会有个name属性用于显示设置Bean Name）
* 可以定义bean命名策略，实现BeanNameGenerator接口，并一定要包含一个无参数构造函数器

<beans>

<context:component-scan bese-package="org.example" name-generator="org.examle.MyNameGenerator"/>

</beans>

**作用域**

* 通常情况下自动查找的spring组件，器scope是singleton，Spring2.5 提供一个标识scope的注解@Scope

@Scope("prototype")

@Repository

public class MovieFinderImpl implement MovieFinder{

//....

}

* 也可以自定义scope策略，实现ScopeMatadataResolver接口并提供一个无参数构造器

<beans>

<context:component-scan bese-package="org.example" scope-resolve="org.example.MyScopeResolver"/>

</beans>

**代理方式**

* 可以使用scoped-proxy属性指定代理，有三个值可选：no，interface，targetClass

<beans>

<context:component-scan bese-package="org.example" scope-proxy="interfaces"/>

</beans>

理解就是会完成注解的扫描 base-p属性就是扫描哪个包下面的所有类

**@Required**

* @required注解适用于bean属性的setter方法
* 这个注解仅仅表示，受影响的bean属性必须在配置时被填充，通过在bean定义或通过自动装配一个明确的属性值

publi class SimpleMovieLister{

private MovieFinder movieFinder;

@Required //不常用

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder= movieFinder;

}

//.............

}

**@Autowired**

* 可以将@Autowired注解为”传统”的setter方法

private MovieFinder movieFinder;

@Autowired

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder= movieFinder;

}

* 可用于构造器或成员变量

@Autowired

private MovieCatalog movieCatalog;

@Autowired

public MovieRecommender(CustomerPreferenceDao dao){

this.dao=dao;

}

* 默认情况下，如果因找不到合适的bean将会导致autowiring 失败抛出异常，可以通过下面的方式避免

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Autowired(requird=false)

public void setMovieFinder(MovieFinder movieFinder){

this.movieFinder=movieFinder;

}

}

* 每个类只能有一个构造器被标记required=true
* @Aurowired的必要属性，建议使用@Required注解

**@Qualifier**

**基于java的容器注解**

* @Bean标识一个用于配置和初始化一个有SpringIoC容器管理   
  的新对象的方法，类似于XML配置文件中的
* 可以在spring的@Component注解的类中使用@Bean注解任何方法（仅仅是可以）
* 上一点中，通常使 用的是@Configuration(配置的意思)

@Resource   
- 如果没有显示指定@Resource 的name 默认的名称是从属性名 或者 setter方法得出

public class SimpleMovieLister{

private MovieFinder movieFinder;

@Resource

public void setMovieFinder ( MovieFinder movieFinder){

this.movieFinder=movieFinder;

}

}

* 注解提供的名字被解析为一个bean的名称，这是由ApplicationContext的中的CommonAnnotationBeanPostProcessor发现并处理的