# Spring入门

## 什么是Spring

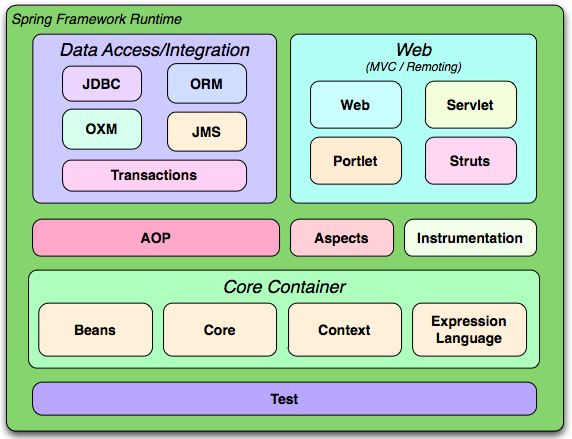
* Spring是分层的JavaSE/EE full-stack(一站式) 轻量级开源框架
* 以IoC（Inverse of Control 反转控制）和AOP（Aspect Oriented Programming 面向切面编程）为内核

## 应用Spring的好处

* 方便解耦，简化开发
  + Spring就是一个大工厂，可以将所有对象创建和依赖关系维护，交给Spring管理
* AOP编程的支持
  + Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能
* 声明式事务的支持
  + 只需要通过配置就可以完成对事务的管理，而无需手动编程
* 方便程序的测试
  + Spring对Junit4支持，可以通过注解方便的测试Spring程序
* 方便集成各种优秀框架
  + Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz等）的直接支持

## Spring体系结构

Spring 框架是一个分层架构，它包含一系列的功能要素并被分为大约20个模块。这些模块分为Core Container、Data Access/Integration、Web、AOP（Aspect Oriented Programming)、Instrumentation和测试部分



## Spring Ioc控制反转快速入门案例

* 下载Spring最新开发包
* 复制Spring开发 jar包到工程
* 理解IoC控制反转和DI依赖注入
* 编写Spring核心配置文件
* 在程序中读取Spring配置文件，通过Spring框架获得Bean，完成相应操作

### 下载Spring（最新）开发包

**还是使用最经典的Spring 3.X版本吧，注意Java6/7和Spring3兼容，Java8不兼容**

* Spring 3.2版本目录结构



### 导入Spring核心开发包到创建工程



* + spring-beans-3.2.0.RELEASE.jar
  + spring-context-3.2.0.RELEASE.jar
  + spring-core-3.2.0.RELEASE.jar
  + spring-expression-3.2.0.RELEASE.jar
* 还需要下载commons-logging日志包
  + commons-logging-1.1.1.jar
* 集成log4j 导入log4j jar包
* 提示：spring3.0.X 版本 asm jar包 已经被合并到 spring core包中

### 理解IoC控制反转和DI依赖注入

HelloTest类中使用 HelloService类对象

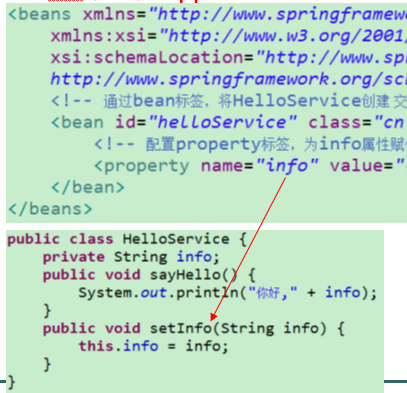
传统方式： HelloService helloService = new HelloService();

IoC Inverse of Control **反转控制**的概念，就是将原本在程序中手动创建HelloService对象的控制权，交由Spring框架管理，简单说，就是创建HelloService对象控制权被反转到了Spring框架

DI：Dependency Injection 依赖注入，在Spring框架负责创建Bean对象时，动态的将依赖对象注入到Bean组件

### 编写Spring核心配置文件

* 配置spring核心配置文件，将HelloService的创建交由容器管理
* **在src下创建applicationContext.xml (习惯上)**



### 在程序中读取Spring配置文件，通过Spring框架获得Bean，完成相应操作

ApplicationContext 应用上下文，加载Spring 框架配置文件

加载classpath：

new ClassPathXmlApplicationContext("applicationContext.xml");

加载磁盘路径：

new FileSystemXmlApplicationContext("applicationContext.xml");

通过 getBean方法获得Spring容器管理Bean对象



## BeanFactory和ApplicationContext



## MyEclipse 编写Spring配置文件无提示解决

* 方案一： 联网下载
  + http://www.springframework.org/schema/beans/spring-beans.xsd
* 方案二： 采用本地schema配置
  + Myeclipse window-preferences- 搜索xml catalog
  + 选中User Specified Entries -- Add 操作
  + location 浏览选中 解压spring 包中 schema\beans\spring-beans-3.2.xsd
  + 修改Key type 为 Schema location
  + key 修改为 http://www.springframework.org/schema/beans/spring-beans.xsd
  + 点击OK



## IOC装配Bean:

### Spring框架Bean实例化的方式:

提供了三种方式实例化Bean.

**\* 构造方法实例化:(默认无参数)**

\* 静态工厂实例化:

\* 实例工厂实例化:

### 无参数构造方法的实例化:

<!-- 默认情况下使用的就是无参数的构造方法. -->

<bean id="bean1" class="cn.itcast.spring3.demo2.Bean1"></bean>

### 静态工厂实例化:

<!-- 第二种使用静态工厂实例化 -->

<bean id="bean2" class="cn.itcast.spring3.demo2.Bean2Factory" factory-method="getBean2"></bean>

### 实例工厂实例化:

<!-- 第三种使用实例工厂实例化 -->

<bean id="bean3" factory-bean="bean3Factory" factory-method="getBean3"></bean>

<bean id="bean3Factory" class="cn.itcast.spring3.demo2.Bean3Factory"/>

### Bean的其他配置:

#### id和name的区别:

id遵守XML约束的id的约束.id约束保证这个属性的值是唯一的,而且必须以字母开始，可以使用字母、数字、连字符、下划线、句话、冒号

name没有这些要求

\*\*\*\*\* 如果bean标签上没有配置id,那么name可以作为id.

\*\*\*\*\* 开发中Spring和Struts1整合的时候, /login.

<bean name=”/login” class=””>

现在的开发中都使用id属性即可.

#### 类的作用范围:

scope属性 :

**\* singleton :单例的.(默认的值.)**

**\* prototype :多例的.**

\* request :web开发中.创建了一个对象,将这个对象存入request范围,request.setAttribute();

\* session :web开发中.创建了一个对象,将这个对象存入session范围,session.setAttribute();

\* globalSession :一般用于Porlet应用环境.指的是分布式开发.不是porlet环境,globalSession等同于session;

实际开发中主要使用singleton,prototype

#### Bean的生命周期:

配置Bean的初始化和销毁的方法:

配置初始化和销毁的方法:

\* init-method=”setup”

\* destroy-method=”teardown”

执行销毁的时候,必须手动关闭工厂,而且只对scope=”**singleton**”有效.

Bean的生命周期的11个步骤:

1.instantiate bean对象实例化

2.populate properties 封装属性

3.如果Bean实现BeanNameAware 执行 setBeanName

4.如果Bean实现BeanFactoryAware 或者 ApplicationContextAware 设置工厂 setBeanFactory 或者上下文对象 setApplicationContext

**5.如果存在类实现 BeanPostProcessor（后处理Bean） ，执行postProcessBeforeInitialization**

6.如果Bean实现InitializingBean 执行 afterPropertiesSet

7.调用<bean init-method="init"> 指定初始化方法 init

**8.如果存在类实现 BeanPostProcessor（处理Bean） ，执行postProcessAfterInitialization**

9.执行业务处理

10.如果Bean实现 DisposableBean 执行 destroy

11.调用<bean destroy-method="customerDestroy"> 指定销毁方法 customerDestroy

在CustomerService类的add方法之前进行权限校验?

### Bean中属性注入:

Spring支持构造方法注入和setter方法注入:

#### 构造器注入:

<bean id="car" class="cn.itcast.spring3.demo5.Car">

<!-- <constructor-arg name="name" value="宝马"/>

<constructor-arg name="price" value="1000000"/> -->

<constructor-arg index="0" type="java.lang.String" value="奔驰"/>

<constructor-arg index="1" type="java.lang.Double" value="2000000"/>

</bean>

#### setter方法注入:

<bean id="car2" class="cn.itcast.spring3.demo5.Car2">

<!-- <property>标签中name就是属性名称,value是普通属性的值,ref:引用其他的对象 -->

<property name="name" value="保时捷"/>

<property name="price" value="5000000"/>

</bean>

#### setter方法注入对象属性:

<property name="car2" ref="car2"/>

#### 名称空间p:注入属性:

Spring2.5版本引入了名称空间p.

p:<属性名>="xxx" 引入常量值

p:<属性名>-ref="xxx" 引用其它Bean对象

引入名称空间:

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

**xmlns:p="http://www.springframework.org/schema/p"**

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

xsi:schemaLocation="

http://www.springframework.org/schema/beans [http://www.springframework.org/schema/beans/spring-beans.xsd">](http://www.springframework.org/schema/beans/spring-beans.xsd%22%3e)

<bean id="car2" class="cn.itcast.spring3.demo5.Car2" p:name="宝马" p:price="400000"/>

<bean id="person" class="cn.itcast.spring3.demo5.Person" p:name="童童" p:car2-ref="car2"/>

#### SpEL:属性的注入:

Spring3.0提供注入属性方式:

语法：#{表达式}

<bean id="" value="#{表达式}">

<bean id="car2" class="cn.itcast.spring3.demo5.Car2">

<property name="name" value="#{'大众'}"></property>

<property name="price" value="#{'120000'}"></property>

</bean>

<bean id="person" class="cn.itcast.spring3.demo5.Person">

<!--<property name="name" value="#{personInfo.name}"/>-->

<property name="name" value="#{personInfo.showName()}"/>

<property name="car2" value="#{car2}"/>

</bean>

<bean id="personInfo" class="cn.itcast.spring3.demo5.PersonInfo">

<property name="name" value="张三"/>

</bean>

### 集合属性的注入:

<bean id="collectionBean" class="cn.itcast.spring3.demo6.CollectionBean">

<!-- 注入List集合 -->

<property name="list">

<list>

<value>童童</value>

<value>小凤</value>

</list>

</property>

<!-- 注入set集合 -->

<property name="set">

<set>

<value>杜宏</value>

<value>如花</value>

</set>

</property>

<!-- 注入map集合 -->

<property name="map">

<map>

<entry key="刚刚" value="111"/>

<entry key="娇娇" value="333"/>

</map>

</property>

<property name="properties">

<props>

<prop key="username">root</prop>

<prop key="password">123</prop>

</props>

</property>

</bean>

### 加载配置文件:

一种写法:

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("bean1.xml",”bean2.xml”);

二种方法:

<import resource="applicationContext2.xml"/>

## IOC装配Bean(注解方式)

### Spring的注解装配Bean

Spring2.5 引入使用注解去定义Bean

@Component 描述Spring框架中Bean

Spring的框架中提供了与@Component注解等效的三个注解:

@Repository 用于对DAO实现类进行标注

@Service 用于对Service实现类进行标注

@Controller 用于对Controller实现类进行标注

\*\*\*\*\* 三个注解为了后续版本进行增强的.

### Bean的属性注入:

普通属性;

@Value(value="itcast")

private String info;

对象属性:

@Autowired:自动装配默认使用类型注入.

@Autowired

@Qualifier("userDao") --- 按名称进行注入.

@Autowired

@Qualifier("userDao")

private UserDao userDao;

等价于

@Resource(name="userDao")

private UserDao userDao;

### Bean其他的属性的配置:

配置Bean初始化方法和销毁方法:

\* init-method 和 destroy-method.

@PostConstruct 初始化

@PreDestroy 销毁

配置Bean的作用范围:

@Scope

### Spring3.0提供使用Java类定义Bean信息的方法

@Configuration

public class BeanConfig {

@Bean(name="car")

public Car showCar(){

Car car = new Car();

car.setName("长安");

car.setPrice(40000d);

return car;

}

@Bean(name="product")

public Product initProduct(){

Product product = new Product();

product.setName("空调");

product.setPrice(3000d);

return product;

}

}

### 实际开发中使用XML还是注解?

XML:

\* bean管理

注解;

\* 注入属性的时候比较方便.

两种方式结合;一般使用XML注册Bean,使用注解进行属性的注入.

<context:annotation-config/>

s

@Autowired

@Qualifier("orderDao")

private OrderDao orderDao;

## Spring整合web开发:

正常整合Servlet和Spring没有问题的

但是每次执行Servlet的时候加载Spring配置,加载Spring环境.

\* 解决办法:在Servlet的init方法中加载Spring配置文件?

\* 当前这个Servlet可以使用,但是其他的Servlet的用不了了!!!

\* 将加载的信息内容放到ServletContext中.ServletContext对象是全局的对象.服务器启动的时候创建的.在创建ServletContext的时候就加载Spring的环境.

\* ServletContextListener:用于监听ServletContext对象的创建和销毁的.

导入;spring-web-3.2.0.RELEASE.jar

在web.xml中配置:

<listener>

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

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

修改程序的代码:

WebApplicationContext applicationContext = WebApplicationContextUtils.getWebApplicationContext(getServletContext());

WebApplicationContext applicationContext = (WebApplicationContext) getServletContext().getAttribute(WebApplicationContext.ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE);

## Spring集成JUnit测试:

1. 程序中有Junit环境.
2. 导入一个jar包.spring与junit整合jar包.

\* spring-test-3.2.0.RELEASE.jar

1. 测试代码:

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations="classpath:applicationContext.xml")

public class SpringTest {

@Autowired

private UserService userService;

@Test

public void demo1(){

userService.sayHello();

}

}