# 1 Spring介绍

## 1.1 Spring概述

**Spring是一个开源框架，Spring是于2003 年兴起的一个轻量级的Java 开发框架**，由Rod Johnson 在其著作Expert One-On-One J2EE Development and Design中阐述的部分理念和原型衍生而来。**它是为了解决企业应用开发的复杂性而创建的**。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益

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

## 1.2 Spring好处

**方便解耦，简化开发:**

* **Spring就是一个大工厂，专门负责生成Bean**，可以将所有对象创建和依赖关系维护由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）的支持

**降低JavaEE API的使用难度 Spring:**

* 对JavaEE开发中一些难用的API（JDBC、JavaMail、远程调webservice用等），都提供了封装，使这些API应用难度大大降低

## 1.3 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,如下图所示：

|  |
| --- |
|  |

## 1.4 在项目中的架构

web层：Struts,SpringMVC

dao层：Hibernate,mybatis

|  |
| --- |
|  |

# 2 Spring 快速入门

## 目标：

掌握web中集成Spring需要哪些包

掌握IOC是什么

## 2.1 编写流程

* 下载Spring 开发包
* 导入Spring的jar包
* 配置Spring的核心xml文件
* 在程序中读取Spring的配置文件来获取Bean**【Bean其实就是一个new好的对象】**

## 2.2 下载官网

* Spring现在有很多框架，如Spring SpringMVC,Spring-Data,Sprint-Boot

|  |
| --- |
|  |

## 2.3 Spring的核心jar包

|  |
| --- |
| **spring-core-3.2.2.RELEASE.jar**  包含Spring框架基本的核心工具类，Spring其它组件要都要使用到这个包里的类,是其它组件的基本核心。 |
| **spring-beans-3.2.2.RELEASE.jar**  所有应用都要用到的，它包含访问配置文件、创建和管理bean  以及进行Inversion of Control(IoC) / Dependency Injection(DI)操作相关的所有类 |
| **spring-context-3.2.2.RELEASE.jar**  Spring提供在基础IoC功能上的扩展服务，此外还提供许多企业级服务的支持,  如邮件服务、任务调度、JNDI定位、EJB集成、远程访问、缓存以及各种视图层框架的封装等。 |
| **spring-expression-3.2.2.RELEASE.jar**  Spring表达式语言 |
| **com.springsource.org.apache.commons.logging-1.1.1.jar**  第三方的主要用于处理日志 |

## 2.4 Spring的入门案例

### 第一步：

* 下载Spring的包

|  |
| --- |
|  |
| spring-framework-3.2.0.RELEASE-dist.zip 【Spring的核心包】  spring-framework-3.0.2.RELEASE-dependencies.zip 【Spring的依赖包】 |

### 第二步：

* 创建Web项目，导Spring的jar包
  + **导入4核心(beans、core、context,expression) + 1个依赖(common-logging.jar)**
  + **注意导入时，不要导入带sources的源文件了**

|  |
| --- |
|  |
|  |

### 第三步：

* 写个简单的Service 并在main方法中调用

|  |  |
| --- | --- |
|  |  |

### 第四步：

* Spring IoC控制反转创建实例

1. 写了个配置文件beans.xml，配置文件的约束可以访问

spring-framework-3.2.0.RC2-docs/reference/html/xsd-config.html网页查看

|  |
| --- |
|  |

2> 离线配置文件约束提示的配置

|  |
| --- |
| **xsd可以在schema目录中找** |
|  |

3> Beans.xml文件配置

|  |
| --- |
|  |

### 第五步：

* 从beans.xml获取bean

|  |
| --- |
|  |

### 总结：【IoC】

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

## 2.5 DI解释

* **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。

### 例子：

在UserService中提供一个get/set的name方法，在beans.xml中通过property去注入

|  |
| --- |
|  |
|  |

# 加载Spring容器的三种方式

## 3.1 类路径获得配置文件

|  |
| --- |
| ApplicationContext context =  **new** ClassPathXmlApplicationContext("com/gyf/spring/demo01/beans.xml"); |

## 3.2 文件系统路径获得配置文件

|  |
| --- |
|  |

## 3.3使用BeanFactory(了解)

|  |
| --- |
|  |

## 3.4BeanFactory和ApplicationContext对比

* BeanFactory 采取延迟加载，第一次getBean时才会初始化Bean
* ApplicationContext是对BeanFactory扩展，提供了更多功能
* 国际化处理
* 事件传递
* **Bean自动装配**
* 各种不同应用层的Context实现

# 装配Bean(xml)

## 4.1 实例化Bean的三种方式

### 4.1.1 使用构造方法实例化

### 4.1.2 使用静态工厂方法实例化

|  |
| --- |
| Bean.xml |
| *<!--装配bean的三种方式，所谓的装配bean就是在xml写一个bean标签-->   <!-- 第一种方式: new 实现类-->* <**bean id="userService1" class="com.gyf.service.UserServiceImpl"**></**bean**>   *<!-- 第二种方式：通过静态工厂方法  spring的版本过低，3.0版本,把jdk改成1.7  -->* <**bean id="userService2" class="com.gyf.service.UserSereviceFactory1" factory-method="createUserService"**></**bean**>   *<!--第三种方式：通过实例工厂方法 -->  <!-- 创建实例factory2 bean-->* <**bean id="factory2" class="com.gyf.service.UserSereviceFactory2"**></**bean**>   <**bean id="userService3" factory-bean="factory2" factory-method="createUserService"**></**bean**> |
| test |
| @Test **public void** test1(){  ApplicationContext context = **new** ClassPathXmlApplicationContext(**"beans3.xml"**);   *//new 对象  //IUserService userService1 = (IUserService) context.getBean("userService1");  //userService1.add();    //静态工厂  //IUserService userService2 = UserSereviceFactory1.createUserService();  //IUserService userService2 = (IUserService) context.getBean("userService2");  //userService2.add();    //实例工厂  //1.创建工厂  // UserSereviceFactory2 factory2 = new UserSereviceFactory2();  //IUserService userService3 = factory2.createUserService();* IUserService userService3 = (IUserService) context.getBean(**"userService3"**);  userService3.add(); } |

### 4.1.3 使用实例工厂方法实例化

## 4.2 bean的作用域

**掌握红色两个常用的即可**

|  |  |
| --- | --- |
| 类别 | 说明 |
| **singleton** | 在Spring IoC容器中仅存在一个Bean实例，Bean以单例方式存在，默认值 |
| **prototype** | 每次从容器中调用Bean时，都返回一个新的实例，即每次调用getBean()时 ，相当于执行new XxxBean() |
| **request** | 每次HTTP请求都会创建一个新的Bean，该作用域仅适用于WebApplicationContext环境 |
| **session** | 同一个HTTP Session 共享一个Bean，不同Session使用不同Bean，仅适用于WebApplicationContext 环境 |
| **globalSession** | 一般用于Portlet应用环境，该作用域仅适用于WebApplicationContext 环境 |

案例：

|  |
| --- |
|  |
|  |

# bean的生命周期【了解】

## 生命周期图

|  |
| --- |
|  |

## 生命周期图解释

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 执行setBeanFactory ，**获取Spring容器**

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

**执行业务处理**

9.如果Bean实现 DisposableBean 执行 destroy

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

## 演示

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# 依赖注入Bean属性(xml)

## 6.1手动装配，使用xml配置

### 构造方法注入

|  |
| --- |
|  |
|  |

### 属性setter方法注入

**setter方法有两种注入，一般使用第一种直观**

|  |
| --- |
| <bean id=*"user"* class=*"com.gyf.spring.demo04.User"*>  <property name=*"username"* value=*"zhangsan"*></property>  <property name=*"password"* value=*"123456"*></property>  </bean> |
|  |

### p命名空间注入【了解】

|  |
| --- |
|  |
|  |
|  |

## 6.2 SpEL表达式【了解】

Spring 表达式

* **对<property>进行统一编程，所有的内容都使用value**
* **<property name="" value="#{表达式}">**

**#{123}、#{'jack'} ： 数字、字符串**

**#{beanId} ：另一个bean引用**

**#{beanId.propName} ：操作数据**

**#{beanId.toString()} ：执行方法**

**#{T(类).字段|方法} ：静态方法或字段**

|  |
| --- |
|  |
|  |

## 6.3 集合注入

集合的注入都是给<property>添加子标签

数组：<array>

List：<list>

Set：<set>

Map：<map> ，map存放k/v 键值对，使用<entry>描述

Properties：<props> <prop key=""></prop> 【】

普通数据：<value>

引用数据：<ref>

### List

|  |
| --- |
|  |
|  |

### Set

|  |
| --- |
|  |
|  |
|  |

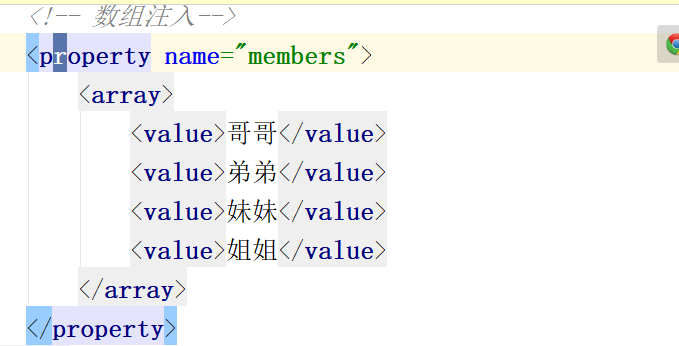
### Map

|  |
| --- |
|  |
|  |

### Properties

|  |
| --- |
|  |
|  |

### 数组



## 6.4 注解注入

* **注解：就是一个类，使用@注解名称**
* **开发中：使用注解 取代 xml配置文件。**

### 6.4.1 @Component

@component取代<bean class="">

### 6.4.2 @Component("id")

取代 <bean id="" class="">

### 6.4.4 web开发，提供3个@Component注解衍生注解（功能一样）取代<bean class="">

@Repository(“名称”)：dao层

@Service(“名称”)：service层

@Controller(“名称”)：web层

@Autowired：自动根据类型注入

@Qualifier(“名称”):指定自动注入的id名称

@Resource(“名称”)

@ PostConstruct 自定义初始化

@ PreDestroy 自定义销毁

### 6.4.5 使用案例

#### 第一个案例

|  |
| --- |
|  |
|  |
|  |
|  |

#### 第二个案例

|  |
| --- |
|  |
|  |

#### 第三个案例

|  |
| --- |
|  |
|  |

#### 第四个案例

|  |
| --- |
|  |
|  |

#### 第五个案例

|  |
| --- |
|  |

#### 第六个案例

|  |
| --- |
|  |
|  |