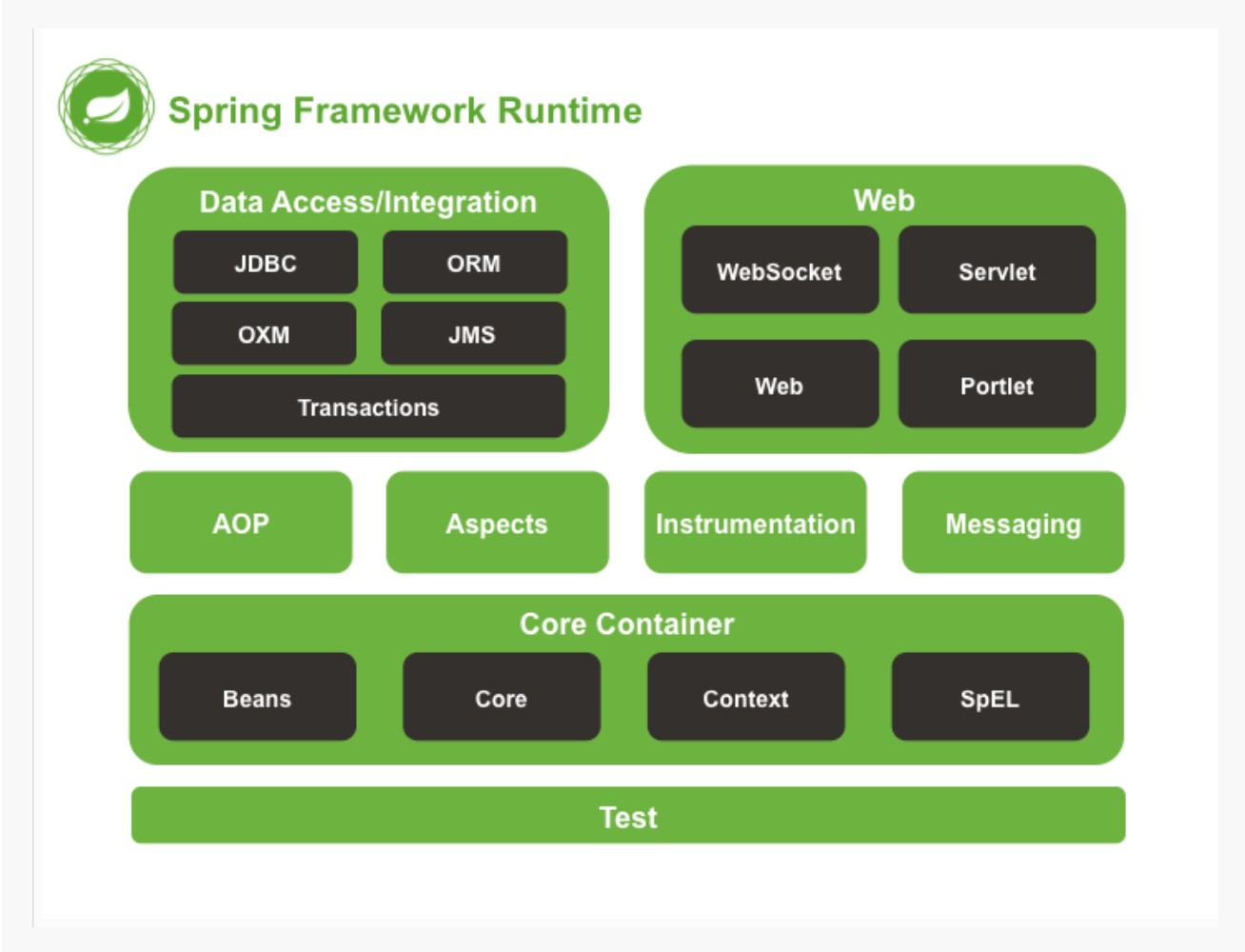
Spring的常见面试题

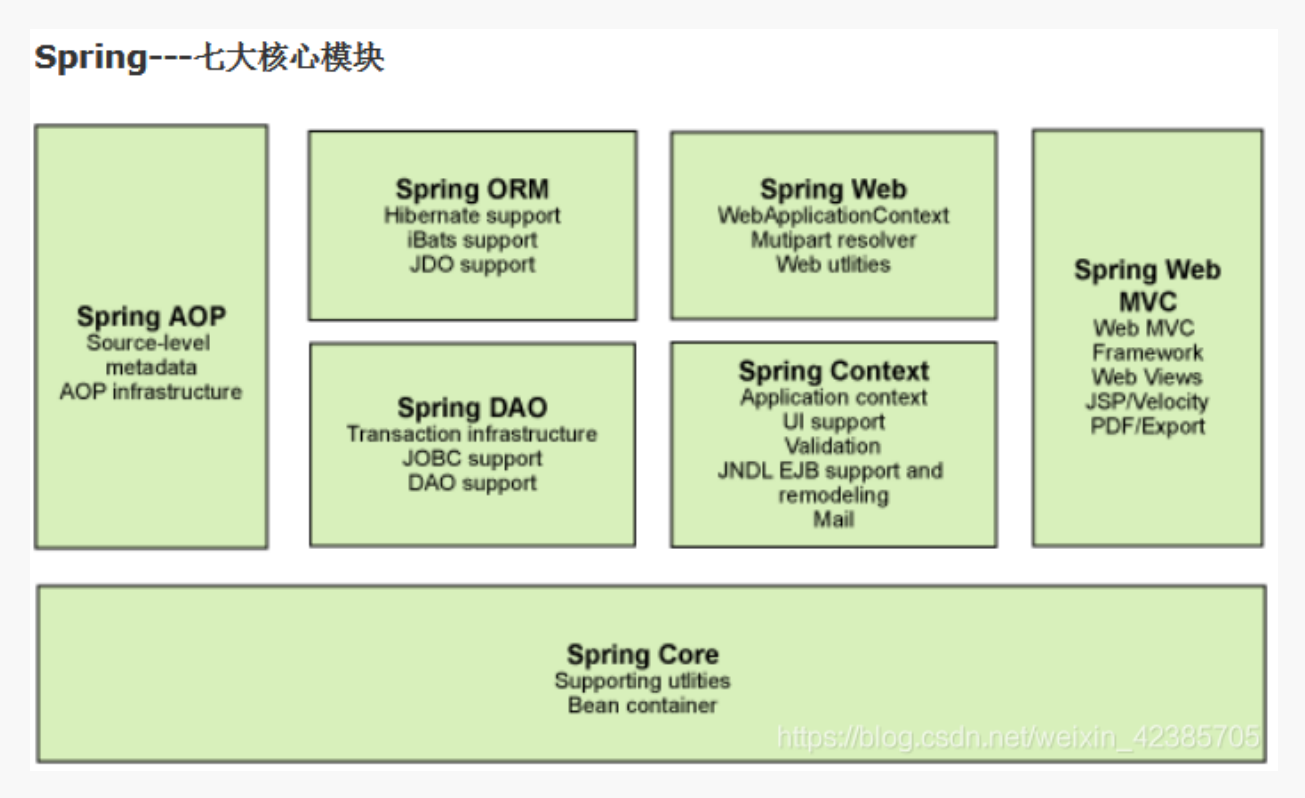
什么是Spring框架，Spring框架有那些主要模块

spring框架是一个为java应用程序的开发提供了综合，广泛的基础性支持的java平台。Spring帮助开发者解决了开发中基础性的问题，使得开发人员可以专注于应用程序的开发。Spring框架本身亦是按照设计模式精心打造，这时的我们可以在开发环境中安心得集成spring框架，不必担心Spring是如何在后台进行工作的。

Spring框架至今已建成了20多个模块，这些模块主要被分成：核心容器，数据访问/集成，web，Aop（面向切面编程），工具，消息和测试模块。



七大核心模块：



使用**Spring**框架能带来哪些好处？

- 统一管理bean，不需要自己在去创建对象
- 提供代码质量，可以让业务和程序的耦合度更低
- 默认情况下：单列的对象 （在内存中存在一份内存空间，防止bean频繁创建，造成内存消耗）
- 为后续对象的改造和增强提供了基础（AOP）
- spring提供系列的接口钩子函数，方便去扩展和集成第三方的框架比如：ORM比如日志。

什么是控制反转(IOC)？什么是依赖注入？（谈谈你对IOC认识和理解？）

1、控制反转是应用于软件工程领域中的，在运行时被装配器对象来绑定耦合对象的一种编程技巧，对象之间耦合关系在编译时通常是未知的。在传统的编程方式中，业务逻辑的流程是由应用程序中的早已被设定好关联关系的对象来决定的。在使用控制反转的情况下，业务逻辑的流程是由对象关系图来决定的，该对象关系图由装配器负责实例化，这种实现方式还可以将对象之间的关联关系的定义抽象化。而绑定的过程是通过“依赖注入”实现的。

2、控制反转是一种以给予应用程序中目标组件更多控制为目的设计范式，并在我们的实际工作中起到了有效的作用。

3、依赖注入是在编译阶段尚未知所需的功能是来自哪个的类的情况下，将其他对象所依赖的功能对象实例化的模式。这就需要一种机制用来激活相应的组件以提供特定的功能，所以依赖注入是控制反转的基础。否则如果在组件不受框架控制的情况下，框架又怎么知道要创建哪个组件？

> 回答：控制反转其实就是*IOC*的理念和概念，就是把传统应用创建对象方式交由容器来管理的一种机制。依赖注入其实就是指*DI*，就是指类和类之间会存在依赖耦合关系。如何把一个类的对象和另外一个类对象的初始化和实例化问题。而*DI*可以把类和类之间依赖的对象进行注入将其实例化。

Spring依赖注入（DI）的三种方式，

1． 接口注入 2． Setter方法注入 3． 构造方法注入

BeanFactory和Applicationcontext的区别

BeanFactory 可以理解为含有bean集合的工厂类。BeanFactory 包含了种bean的定义，以便在接收到客户端请求时将对应的bean实例化。

BeanFactory还能在实例化对象的时生成协作类之间的关系。此举将bean自身与bean客户端的配置中解放出来。BeanFactory还包含了bean生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。 --- FactoryBean和 BeanFactory

从表面上看，application context如同bean factory一样具有bean定义、bean关联关系的设置，根据请求分发bean的功能。但application context在此基础上还提供了其他的功能。

- 提供了支持国际化的文本消息
- 统一的资源文件读取方式
- 已在监听器中注册的bean的事件
- 对beanfactory进行增强，比如可以类型获取所有的容器的对应bean，也可以根据name获取所有的bean

以下是三种较常见的 **ApplicationContext** 实现方式：

```
1、ClassPathXmlApplicationContext：从classpath的XML配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。


ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);
2、FileSystemXmlApplicationContext ：由文件系统中的XML配置文件读取上下文。

ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);
3、XmlWebApplicationContext：由Web应用的XML文件读取上下文。
```

ApplicationContext是BeanFactory的子类， 通俗的说BeanFactory是一个Basic Container，ApplicationContext是一个Advanced Container以及对Transaction和AOP的支持等

两者装载**bean**的区别

BeanFactory：

BeanFactory在启动的时候不会去实例化Bean，中有从容器中拿Bean的时候才会去实例化；

ApplicationContext：

ApplicationContext在启动的时候就把所有的Bean全部实例化了。它还可以为Bean配置lazy-init=true来让Bean延迟实例化；

对**Spring**中**BeanFactory**和**FactoryBean**的理解

要讲清楚BeanFactory和FactoryBean的区别，首先就要搞清楚他们的含义以及为什么要使用它。

BeanFactory是什么

BeanFactory，根据其名字就知道它是一个Bean的工厂，它是Spring框架里最核心的接口，是Spring的IOC容器或对象工厂，为Spring容器定义核心功能的顶层规范，它定义了getBean()、containsBean()等管理Bean的通用方法，Spring中所有的Bean都是交给BeanFactory(就是所说的IoC容器)来管理的。Spring中有许多实现或者扩展了该接口的类，如ApplicationContext、XmlBeanFactory扩展并丰富了BeanFactory。
在Spring中BeanFactory 最重要的一个实现就是DefaultListableBeanFactory，它是BeanFactory的一个最全的实现。如果你看过源码就会知道，Spring初始化过程中使用的BeanFactory就是它。

FactoryBean是什么

FactoryBean，虽然名称里带了“Factory”，但它其实是一个Bean，只不过它是一个能够生产或者修饰对象的特殊Bean。FactoryBean是一个接口，Spring容器中有很多实现了该接口的类，在Spring初始化过程中，它做了重要的工作。
FactoryBean接口中定义了三个方法：

```
//返回自定义的泛型类型
T getObject() throws Exception;
//返回自定义类型
Class<?> getObjectType();
//FactoryBean定义的Bean默认作用域为singleton
default boolean isSingleton() {
    return true;
   }
```

当我们需要实现一个使用特殊方式返回或者需要对Bean做一些特殊修饰时可以使用这种方法。。例如，Mybatis框架正是使用FactoryBean的这一特性，在xxDao接口注册过程中，使用动态代理修改了BeanDefinition的BeanClass属性，使得我们明明注入的是一个接口，却可以注入成功。

如何用基于XML配置的方式配置Spring?

在Spring框架中，依赖和服务需要在专门的配置文件来实现，我常用的XML格式的配置文件。这些配置文件的格式通常用开头，然后一系列的bean定义和专门的应用配置选项组成。

SpringXML配置的主要目的时候是使所有的Spring组件都可以用xml文件的形式来进行配置。这意味着不会出现其他的Spring配置类型（比如声明的方式或基于Java Class的配置方式）

Spring的XML配置方式是使用被Spring命名空间的所支持的一系列的XML标签来实现的。Spring有以下主要的命名空间：context、beans、jdbc、tx、aop、mvc和aso。

```
<beans>

    <!-- JSON Support -->
    <bean name="viewResolver"
class="org.springframework.web.servlet.view.BeanNameViewResol
ver"/>
    <bean name="jsonTemplate"
class="org.springframework.web.servlet.view.json.MappingJacks
on2JsonView"/>

    <bean id="restTemplate"
class="org.springframework.web.client.RestTemplate"/>

</beans>

```

下面这个web.xml仅仅配置了DispatcherServlet，这件最简单的配置便能满足应用程序配置运行时组件的需求。

```
<web-app>
  <display-name>Archetype Created Web Application</display-
name>

  <servlet>
        <servlet-name>spring</servlet-name>
            <servlet-class>

org.springframework.web.servlet.DispatcherServlet
            </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>

```

```
    </servlet-mapping>

</web-app>
```

如何用基于Java配置的方式配置Spring?

Spring对Java配置的支持是由@Configuration注解和@Bean注解来实现的。由@Bean注解的方法将会实例化、配置和初始化一个新对象，这个对象将由Spring的IoC容器来管理。@Bean声明所起到的作用与 元素类似。被@Configuration所注解的类则表示这个类的主要目的是作为bean定义的资源。被@Configuration声明的类可以通过在同一个类的内部调用@bean方法来设置嵌入bean的依赖关系。

最简单的@Configuration 声明类请参考下面的代码：

```
@Configuration
public class AppConfig
{
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

对于上面的@Beans配置文件相同的XML配置文件如下：

```
<beans>
    <bean id="myService"
class="com.howtodoinjava.services.MyServiceImpl"/>
</beans>
```

上述配置方式的实例化方式如下：利用AnnotationConfigApplicationContext 类进行实例化

```
public static void main(String[] args) {
    ApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);
    MyService myService = ctx.getBean(MyService.class);
    myService.doStuff();
}
```

要使用组件组建扫描，仅需用@Configuration进行注解即可：

```
@Configuration
@ComponentScan(basePackages = "com.howtodoinjava")
public class AppConfig  {
    ...
}
```

在上面的例子中，com.acme包首先会被扫到，然后再容器内查找被@Component声明的类，找到后将这些类按照Sring bean定义进行注册。

如果你要在你的web应用开发中选用上述的配置的方式的话，需要用AnnotationConfigWebApplicationContext 类来读取配置文件，可以用来配置Spring的Servlet监听器ContrextLoaderListener或者Spring MVC的DispatcherServlet。

```
<web-app>
    <!-- Configure ContextLoaderListener to use
AnnotationConfigWebApplicationContext
        instead of the default XmlWebApplicationContext -->
    <context-param>
        <param-name>contextClass</param-name>
        <param-value>

org.springframework.web.context.support.AnnotationConfigWebAp
plicationContext
        </param-value>
    </context-param>

```

```
    <!-- Configuration locations must consist of one or more comma- or space-delimited
         fully-qualified @Configuration classes. Fully-qualified packages may also be
         specified for component-scanning -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>com.howtodoinjava.AppConfig</param-value>
    </context-param>

    <!-- Bootstrap the root application context as usual using ContextLoaderListener -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!-- Declare a Spring MVC DispatcherServlet as usual -->
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <!-- Configure DispatcherServlet to use AnnotationConfigWebApplicationContext
             instead of the default XmlWebApplicationContext -->
        <init-param>
            <param-name>contextClass</param-name>
            <param-value>

org.springframework.web.context.support.AnnotationConfigWebApplicationContext
            </param-value>
        </init-param>
```

```
        <!-- Again, config locations must consist of one or more comma- or space-delimited
            and fully-qualified @Configuration classes -->
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>com.howtodoinjava.web.MvcConfig</param-value>
        </init-param>
    </servlet>

    <!-- map all requests for /app/* to the dispatcher servlet -->
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/app/*</url-pattern>
    </servlet-mapping>
</web-app>
```

怎样用注解的方式配置Spring?

Spring在2.5版本以后开始支持用注解的方式来配置依赖注入。可以用注解的方式来替代XML方式的bean描述，可以将bean描述转移到组件类的内部，只需要在相关类上、方法上或者字段声明上使用注解即可。注解注入将会被容器在XML注入之前被处理，所以后者会覆盖掉前者对于同一个属性的处理结果。

注解装配在Spring中是默认关闭的。所以需要在Spring文件中配置一下才能使用基于注解的装配模式。如果你想要在你的应用程序中使用关于注解的方法的话，请参考如下的配置。

```
<beans>
   <context:annotation-config/>
   <!-- bean definitions go here -->
</beans>
```

在 context:annotation-config/标签配置完成以后，就可以用注解的方式在Spring中向属性、方法和构造方法中自动装配变量。

下面是几种比较重要的注解类型：

@Required：该注解应用于设值方法。
@Autowired：该注解应用于有值设值方法、非设值方法、构造方法和变量。
@Qualifier：该注解和@Autowired注解搭配使用，用于消除特定bean自动装配的歧义。
JSR-250 Annotations：Spring支持基于JSR-250 注解的以下注解，@Resource、@PostConstruct 和 @PreDestroy。

请解释**Spring Bean**的生命周期？

Spring Bean的生命周期简单易懂。在一个bean实例被初始化时，需要执行一系列的初始化操作以达到可用的状态。同样的，当一个bean不在被调用时需要进行相关的析构操作，并从bean容器中移除。

Spring bean factory 负责管理在spring容器中被创建的bean的生命周期。Bean的生命周期由两组回调（call back）方法组成。

- 初始化之后调用的回调方法。
- 销毁之前调用的回调方法。

Spring框架提供了以下四种方式来管理bean的生命周期事件：

- InitializingBean和DisposableBean回调接口
- 针对特殊行为的其他Aware接口
- Bean配置文件中的Custom init()方法和destroy()方法
- @PostConstruct和@PreDestroy注解方式使用customInit()和customDestroy()方法管理bean生

生命周期的代码样例如下：

```
<beans>
    <bean id="demoBean"
class="com.howtodoinjava.task.DemoBean"
            init-method="customInit" destroy-
method="customDestroy"></bean>
</beans>
```

Spring框架中的单例Beans是线程安全的么

结论： 不是线程安全的

Spring容器中的Bean是否线程安全，容器本身并没有提供Bean的线程安全策略，因此可以说Spring容器中的Bean本身不具备线程安全的特性，但是具体还是要结合具体scope的Bean去研究。

Spring 的 bean 作用域（scope）类型

1、singleton:单例，默认作用域。

2、prototype:原型，每次创建一个新对象。

3、request:请求，每次Http请求创建一个新对象，适用于WebApplicationContext环境下。

4、session:会话，同一个会话共享一个实例，不同会话使用不用的实例。

5、global-session:全局会话，所有会话共享一个实例。

线程安全这个问题，要从单例与原型Bean分别进行说明。

原型Bean

对于原型Bean,每次创建一个新对象，也就是线程之间并不存在Bean共享，自然是不会有线程安全的问题。

单例Bean

对于单例Bean,所有线程都共享一个单例实例Bean,因此是存在资源的竞争。

如果单例Bean,是一个无状态Bean，也就是线程中的操作不会对Bean的成员执行查询以外的操作，那么这个单例Bean是线程安全的。比如Spring mvc 的Controller、Service、Dao等，这些Bean大多是无状态的，只关注于方法本身。

Spring单例，为什么controller、service和dao确能保证线程安全？

Spring中的Bean默认是单例模式的，框架并没有对bean进行多线程的封装处理。
实际上大部分时间Bean是无状态的（比如Dao） 所以说在某种程度上来说Bean其实是安全的。
但是如果Bean是有状态的 那就需要开发人员自己来进行线程安全的保证，最简单的办法就是改变bean的作用域 把 "singleton"改为’‘protopyte’ 这样每次请求Bean就相当于是 new Bean() 这样就可以保证线程的安全了。

- 有状态就是有数据存储功能
- 无状态就是不会保存数据

controller、service和dao层本身并不是线程安全的，只是如果只是调用里面的方法，而且多线程调用一个实例的方法，会在内存中复制变量，这是自己的线程的工作内存，是安全的。

在 Spring 中如何注入一个 Java 集合？

Spring 提供以下几种集合的配置元素：

- 类型用于注入一列值，允许有相同的值。

- 类型用于注入一组值，不允许有相同的值。
- **类型用于注入一组键值对，键和值都可以为任意类型。**
- **类型用于注入一组键值对，键和值都只能为 String 类型。

```java
package com.LHB.collection;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;
public class Department {
    private String name;
    private String[] empName;
    private List<Employee> empList;    //List集合
    private Set<Employee> empSets;     //Set集合
    private Map<String,Employee> empMap; //map集合
    private Properties pp;    //Properties的使用

    public Properties getPp() {
        return pp;
    }
    public void setPp(Properties pp) {
        this.pp = pp;
    }
    public Map<String, Employee> getEmpMap() {
        return empMap;
    }
    public void setEmpMap(Map<String, Employee> empMap) {
        this.empMap = empMap;
    }
    public Set<Employee> getEmpSets() {
        return empSets;
    }
    public void setEmpSets(Set<Employee> empSets) {
        this.empSets = empSets;
    }
    public List<Employee> getEmpList() {
```

```
        return empList;
    }
    public void setEmpList(List<Employee> empList) {
        this.empList = empList;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String[] getEmpName() {
        return empName;
    }
    public void setEmpName(String[] empName) {
        this.empName = empName;
    }
}
```

2.继续在包中创建Employee类

```
package com.LHB.collection;
public class Employee {
    private String name;
    private int id;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

```

```
}
```

3.创建applicationContext.xml配置文件，配置重点在数组，List，Set，Map，propertes装载值的环节

```
<?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"

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">

   <bean id="department" class="com.LHB.collection.Department">
       <property name="name" value="财务部门" />
       <!-- 给数组注入值 -->
       <property name="empName">
           <list>
               <value>小米</value>
               <value>小明</value>
               <value>小四</value>
           </list>
       </property>

       <!-- 给list注入值 可以有相同的多个对象  -->
       <property name="empList">
           <list>
               <ref bean="emp1" />
               <ref bean="emp2"/>
           </list>
```

```
27          </property>
28          <!-- 给set注入值 不能有相同的对象 -->
29          <property name="empSets">
30              <set>
31                  <ref bean="emp1" />
32                  <ref bean="emp2"/>
33              </set>
34          </property>
35
36          <!-- 给map注入值 只要map中的key值不一样就可以装配value -->
37          <property name="empMap">
38              <map>
39                  <entry key="1" value-ref="emp1" />
40                  <entry key="2" value-ref="emp2" />
41              </map>
42          </property>
43
44          <!-- 给属性集合配置 -->
45          <property name="pp">
46              <props>
47                  <prop key="pp1">hello</prop>
48                  <prop key="pp2">world</prop>
49              </props>
50          </property>
51      </bean>
52      <bean id="emp1" class="com.LHB.collection.Employee">
53          <property name="name">
54              <value>北京</value>
55          </property>
56      </bean>
57       <bean id="emp2" class="com.LHB.collection.Employee">
58          <property name="name">
59              <value>天津</value>
60          </property>
61      </bean>
62
63  </beans>
```

请解释Spring Bean的自动装配？

在Spring框架中，在配置文件中设定bean的依赖关系是一个很好的机制，Spring容器还可以自动装配合作关系bean之间的关联关系。这意味着Spring可以通过向Bean Factory中注入的方式自动搞定bean之间的依赖关系。自动装配可以设置在每个bean上，也可以设定在特定的bean上。
下面的XML配置文件表明了如何根据名称将一个bean设置为自动装配：

```
<bean id="employeeDAO"
class="com.howtodoinjava.EmployeeDAOImpl" autowire="byName" />
```

除了bean配置文件中提供的自动装配模式，还可以使用@Autowired注解来自动装配指定的bean。在使用@Autowired注解之前需要在按照如下的配置方式在Spring配置文件进行配置才可以使用。

```
<context:annotation-config />
```

也可以通过在配置文件中配置AutowiredAnnotationBeanPostProcessor 达到相同的效果。

```
<bean class
="org.springframework.beans.factory.annotation.AutowiredAnnota
tionBeanPostProcessor"/>
```

配置好以后就可以使用@Autowired来标注了。

```
@Autowired
public EmployeeDAOImpl ( EmployeeManager manager ) {
    this.manager = manager;
}
```

自动装配有哪些局限性?

自动装配的局限性是：

重写：你仍需用 和 配置来定义依赖，意味着总要重写自动装配。

基本数据类型：你不能自动装配简单的属性，如基本数据类型，String字符串，和类。

模糊特性：自动装配不如显式装配精确，如果有可能，建议使用显式装配。

请解释各种自动装配模式的区别

在Spring中共有5种自动装配模式，让我们逐一分析。

（1）no：这是Spring的默认设置，在该设置下自动装配是关闭的，开发者需要自行在Bean定义中用标签明确地设置依赖关系。
（2）byName：该模式可以根据Bean名称设置依赖关系。当向一个Bean中自动装配一个属性时，容器将根据Bean的名称自动在配置文件中查询一个匹配的Bean。如果找到就装配这个属性，如果没找到就报错。
（3）byType：该模式可以根据Bean类型设置依赖关系。当向一个Bean中自动装配一个属性时，容器将根据Bean的类型自动在配置文件中查询一个匹配的Bean。如果找到就装配这个属性，如果没找到就报错。
（4）constructor：和byType模式类似，但是仅适用于有与构造器相同参数类型的Bean，如果在容器中没有找到与构造器参数类型一致的Bean，那么将会抛出异常。
（5）autodetect：该模式自动探测使用constructor自动装配或者byType自动装配。首先会尝试找合适的带参数的构造器，如果找到就是用构造器自动装配，如果在Bean内部没有找到相应的构造器或者构造器是无参构造器，容器就会自动选择byType模式。

Spring中构造方法注入和设值注入有什么区别

设值注入的优势：

1. 设值注入写法直观便于理解，使各种关系清晰明了。
2. 设值注入可以避免因复杂的依赖实例化时所造成的性能问题。
3. 设值注入的灵活性较强。

构造方法注入的优势：

1. 构造方法注入可以决定依赖关系的注入顺序，有限依赖的优先注入。
2. 对于依赖关系无需变化的Bean，构造方法注入使所有的依赖关系全部在构造器内设定，可避免后续代码对依赖关系的破坏。
3. 构造方法注入中只有组建的创建者才能改变组建的依赖关系，更符合高内聚原则。

建议：

采用设值注入为主，构造注入为辅的注入策略。依赖关系无需变化时，尽量采用构造注入，而其它的依赖关系的注入，则考虑设值注入。

Spring框架中有哪些不同类型的事件？

Spring 提供了以下5种标准的事件：

Spring的ApplicationContext 提供了支持事件和代码中监听器的功能。
我们可以创建bean用来监听在ApplicationContext 中发布的事件。
ApplicationEvent类和在ApplicationContext接口中处理的事件，如果一个bean实现了ApplicationListener接口，当一个ApplicationEvent 被发布以后，bean会自动被通知。

```
public class AllApplicationEventListener implements
ApplicationListener < ApplicationEvent >{
    @Override
    public void onApplicationEvent(ApplicationEvent
applicationEvent){
        //process event
    }
}
```

Spring 提供了以下5中标准的事件：

- 上下文更新事件（ContextRefreshedEvent）：该事件会在ApplicationContext被初始化或者更新时发布。也可以在调用ConfigurableApplicationContext 接口中的refresh()方法时被触发。
- 上下文开始事件（ContextStartedEvent）：当容器调用ConfigurableApplicationContext的Start()方法开始/重新开始容器时触发该事件。
- 上下文停止事件（ContextStoppedEvent）：当容器调用ConfigurableApplicationContext的Stop()方法停止容器时触发该事件。
- 上下文关闭事件（ContextClosedEvent）：当ApplicationContext被关闭时触发该事件。容器被关闭时，其管理的所有单例Bean都被销毁。
- 请求处理事件（RequestHandledEvent）：在Web应用中，当一个http请求（request）结束触发该事件。
- 除了上面介绍的事件以外，还可以通过扩展ApplicationEvent 类来开发自定义的事件。

```
public class CustomApplicationEvent extends ApplicationEvent{
    public CustomApplicationEvent ( Object source, final
String msg ){
        super(source);
        System.out.println("Created a Custom event");
    }
}

```

为了监听这个事件，还需要创建一个监听器：

```
public class CustomEventListener implements
ApplicationListener < CustomApplicationEvent >{
    @Override
    public void onApplicationEvent(CustomApplicationEvent
applicationEvent) {
        //handle event
    }
```

之后通过applicationContext接口的publishEvent()方法来发布自定义事件。

```
CustomApplicationEvent customEvent = new
CustomApplicationEvent(applicationContext, "Test message");
applicationContext.publishEvent(customEvent);
```

Spring 框架中都用到了哪些设计模式?

(1)工厂模式：BeanFactory 就是简单工厂模式的体现，用来创建对象的实例;

(2)单例模式：Bean 默认为单例模式。

(3)代理模式：Spring 的AOP 功能用到了JDK 的动态代理和CGLIB 字节码生成技术;

(4)模板方法：用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。

(5)观察者模式：定义对象键一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知被制动更新，如Spring 中listener 的实现--ApplicationListener。

(5)委派者模式：Spring提供了DispatcherServlet来堆请求进行分发

请解释下**Spring** 框架中的**IOC** 容器？

Spring 中的org.springframework.beans 包和org.springframework.context 包构成了Spring 框架IOC 容器的基础。

BeanFactory 接口提供了一个先进的配置机制， 使得任何类型的对象的配置成为可能。ApplicationContex 接口对BeanFactory（是一个子接口）进行了扩展，在BeanFactory 的基础上添加了其他功能，比如与Spring 的AOP 更容易集成，也提供了处理message resource 的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对Web 应用的WebApplicationContext。

在**Spring** 中可以注入**null** 或空字符串吗？

完全可以。

谈谈你对**SpringAop**的认识和理解

1、Aop(Aspect OrientedProgramming)面向切面编程。它不是一种新的技术，也不是框架。而是一种编程思想，可以理解为OOP（object Oriented Programing）面向对象编程的补充和完善

2、OOP引入了封装，基础和多态的等概念来简历一种对象层次结构，用以模拟攻共行为的一个集合。

3、当我们需要为分散的对象引入公共行为的时候，OOP就显得无能为力。也就是说OOP允许你定义从上到下的关系，但是不适合定义从左到右的关系。

- 比如日志功能，日志代码往往水平地散布在所有对象层次中，而与它所散布的对象的核心功能毫无关系，对于其他类型的代码，如安全性、异常处理也都是如此。这种散布的在各处的无关代码被称为横切逻辑代码（cross-cutting）。在OOP设计中，它导致了大量代码的重复，布利于各个模块的重用。
- AOP技术则恰恰相反，它利用一种横切面技术，剖解开封装的对象内部，并将哪些影响了多个类的工共行为封装到一个可重用的模块。并将其命名为：Aspect (切面)。
- 简单地说就是将那些与业务无关却为业务模块所共同调用的逻辑和责任封装起来。便于减少系统的重复代码。降低模块的耦合度。并且有利于未来的可操作性和可维护性。
- AOP代表的是一个横向的关系，如果说对象是一个空心的圆柱体，其中封装的是对象的属性和行为、那么面向切面编程的方法就像一把利刃，将这些空心圆柱体剖开，以获得其内部消息，而剖开的切面也就是所谓的Aspect、

AOP设计的基本概念：

- Aspect（切面）：通常是一个类，里面定义切入点和通知。
- Joinpoint(连接点)：程序执行过程中明确的点，一般是方法的调用
- Advice（通知）：AOP在特定的切入点上执行的增强处理逻辑，有before,after,afterreturing,afterthrowing,around。
- Pointcut（切入点）:带有通知的连接点，在程序中主要体现为书写切入点白哦打赏，
- AOP代理：AOP框架创建的对象，代理就是目标对象增强，Spring中的AOP代理可以是JDK动态代理，也可以是CGLIB代理。前者基于接口，后者基于子类。

SpringAop

Spring中的AOP代理是离不开Spring的IOC容器的。代理的生成，管理机器依赖关系都是由IOC容器负责的。Spring默认是jdk动态代理。在需要代理类而不是代理接口的时候，Spring会自动自由切换为使用CGLIB代理。布过现在项目都是面向接口编程，所以JDK动态代理相对用的多一些。