Spring，day02，主讲：汤小洋

一、实例化bean的方式

1. 简介

可以通过多种方式创建对象：

- 构造方法：无参、有参
- 静态工厂：无参、有参（静态方法）
- 实例工厂：无参、有参（非静态方法）

2. 构造方法

```
<!-- 无参-->
<!--<bean id="springBean" class="ioc09.SpringBean">-->
<!--<property name="username" value="tom"/>-->
<!--<property name="password" value="123"/>-->
<!--<property name="age" value="21"/>-->
<!--</bean>-->

<!-- 带参 -->
<bean id="springBean2" class="ioc09.SpringBean">
  <constructor-arg name="name" value="alice"/>
  <constructor-arg name="pwd" value="456"/>
  <constructor-arg name="age" value="18"/>
       </bean>
```

3. 静态工厂

```
<!-- 无参 -->
<!--<bean id="springBean" class="ioc10.SpringBeanFactory" factory-method="getSpringBean">-->
<!--<property name="name" value="jack"/>-->
<!--</bean>-->

<!-- 带参 -->
<bean id="springBean2" class="ioc10.SpringBeanFactory" factory-method="getSpringBean">
  <constructor-arg name="name" value="lucy"/>
</bean>
```

4.实例工厂

```
<!-- 无参 -->
<!--<bean id="springBean" factory-bean="springBeanFactory"  factory-method="getSpringBean">-->
<!--<property name="name" value="tom"/>-->
<!--</bean>-->

<!-- 带参 -->
<bean id="springBean2" factory-bean="springBeanFactory" factory-method="getSpringBean">
  <constructor-arg name="name" value="alice"/>
</bean>
```

二、实例化bean的时机

1. ApplicationContext容器

默认预先实例化，即在容器启动时实例化

可以设置为懒实例化，即在第一次使用bean时实例化

```
<bean id="springBean" class="ioc14.SpringBean" lazy-init="true">
  <property name="name" value="alice"/>
</bean>
```

实际应用中都是使用预先实例化，虽然启动时较慢，但用户访问时速度较慢

2.BeanFactory容器

只能懒实例化，使用bean时才会实例化

三、Bean的作用域

1.简介

在IoC容器中bean默认是单例的，存在的问题：

单例bean中的属性是线程不安全的，多线程并发访问时数据不安全

设置scope属性来指定作用域，配置为非单例的

2. 用法

```
<!--
                scope作用域，取值：
                       singleton：单例
                       prototype：非单例
                       request：同一个请求中单例
                       session：同一个会话中单例
-->

<bean id="springBean" class="ioc15.SpringBean" scope="prototype">
  <property name="name" value="alice"/>
</bean>
```

四、继承配置

1. 简介

用来简化代码，减少配置

2. 用法

- 用法1：不同的子bean类继承自父bean
- 用法2：相同的子bean类继承自父bean

五、自动装配

1.简介

IoC容器可以根据bean的名称、类型或构造方法自动进行注入，称为自动装配

只针对于**其他bean的引用**

2. 配置方式

```
<!--
        autowire，取值：
                default，不进行自动装配,等同于no
                byName，根据属性名自动装配，查找同名的bean
                byType，根据属性类型自动装配，查找同类型的bean（推荐）
                        如果刚好找到一个，则注入
                        如果找到多个，则抛出异常
                constructor，根据构造方法自动装配
                        同时根据byName和byType自动装配，先按byName，再按byType
                        注：此时不是通过setter方法进行装配的，所以可以不写对应的
setter方法


-->

<bean id="springBean" class="ioc17.SpringBean" autowire="constructor">
        <!--<property name="otherBean" ref="otherBean"/>-->
</bean>

<bean id="otherBean" class="ioc17.OtherBean">
        <property name="name" value="tom"/>
</bean>
```

六、在Bean中获取IoC容器

1.定义一个IoC容器工具类

步骤：

1. 定义一个类，实现ApplicationContextAware接口、
2. 将该工具bean添加到IoC容器
3. 调用工具类，获取IoC容器中的bean

2.基本用法

```java
/**
 * Author: 汤小洋
 * Date: 2018-03-23 15:32
 * Description: IoC容器的工具类，用于获取并操作IoC容器
 */
public class ApplicationContextHolder implements
ApplicationContextAware {

    private static ApplicationContext ac;

    @Override
    public void setApplicationContext(ApplicationContext
applicationContext) throws BeansException {

System.out.println("ApplicationContextHolder.setApplicationContext");
        ac=applicationContext;
    }

    //一般不直接提供获取ApplicationContext的方法，不安全
    //public static ApplicationContext getApplicationContext(){
    //    return ac;
    //}

    public static Object getBean(String beanName){
       return ac.getBean(beanName);
    }

    public static <T> T getBean(Class<T> clazz){
        return ac.getBean(clazz);
    }

}
```

七、FactoryBean

1. 简介

Spring中有两种类型的Bean：

- 普通Bean，返回的是Bean本身的对象
- 工厂Bean，即FactoryBean

  应用场合：如果普通bean的配置比较复杂，在配置文件中定义时步骤比较多，此时可以使用FactoryBean

2. 定义FactoryBean

步骤：

1. 定义一个类，实现FactoryBean接口
2. 将该bean添加到IoC容器中
3. 从容器中获取该bean，返回的是该FactoryBean中的getObject()方法返回的对象

八、Resource类

1.简介

本质上就是java.io.File的封装

根据资源位置不的同，提供了不同的实现类，用来快速获取文件资源

- FileSystemResource
- ClassPathResource
- UrlResource
- InputStreamResource

2. 基本用法

```
//Resource resource=new FileSystemResource("e:/create.sql");
Resource resource=new ClassPathResource("ioc21/spring.xml");

System.out.println(resource.getFilename());
System.out.println(resource.contentLength());
System.out.println(resource.exists());
InputStream inputStream = resource.getInputStream();
StreamUtils.copy(inputStream,new FileOutputStream("e:/itany.xml"));
```

3. 装配Resource

```
<bean id="springBean" class="ioc21.SpringBean">
  <!--<property name="resource" value="file:e:/create.sql"/>-->
  <property name="resource" value="classpath:ioc21/spring.xml"/>
</bean>
```

九、后(置)处理器

1.两种后处理器

- Bean后处理器，实现BeanPostProcessor接口
- BeanFactory后处理器，实现BeanFactoryPostProcessor接口，也称为容器后处理器

2. BeanPostProcessor

2.1 简介

Bean后处理器用来对bean的功能进行扩展增强，**对IoC容器中的所有bean都有效**

时机：执行初始化方法之前和之后

bean的生命周期：

代码块——>实例化——>数据装配——>初始化之前——>初始化方法——>初始化之后——>就绪——>使用——>销毁方法——>从容器销毁

2.2 实现步骤

1. 定义一个类，实现BeanPostProcessor接口
2. 将该后处理器添加到IoC容器中

2.3 练习

需求：读取properties文件，为所有bean注入值

3.BeanFactoryPostProcessor

3.1 简介

容器后处理器在bean创建之前，修改bean的定义属性

bean的生命周期：

BeanFactoryPostProcessor——>代码块——>实例化——>数据装配——>初始化之前——>初始化方法——>初始化之后——>就绪——>使用——>销毁方法——>从容器销毁

3.2 实现步骤

1. 定义一个类，实现BeanFactoryPostProcessor
2. 将该bean添加到IoC容器中
3. 定义属性编辑器PropertyEditor（转换器），实现PropertyEditor接口或继承PropertyEditorSupport父类
4. 在容器后处理器中注册属性编辑器

3.3 练习

通过容器后处理器，完成birthday、sex的转换和装配

3.4 内置容器后处理器

Spring预定义了容器后处理器

- CustomEditorConfigurer 用来注册自定义的属性编辑器
- PropertyPlaceholderConfigurer 用来读取属性文件，同时内置了常用的属性编辑器