Spring的两大核心机制，IoC（控制反转），AOP（面向切面编程）

IoC

property标签：name对应属性名，value是属性的值，如果属性值包含特殊字符，比如name="<张三>"，使用]]>

```
<bean id="student" class="com.southwind.entity.Student">
    <property name="id" value="1"></property>
    <property name="name">
        <value><![CDATA[<张三>]]></value>
    </property>
    <property name="age" value="22"></property>
</bean>
```

Spring的IoC容器通过调用每个属性的setter方法来完成属性赋值的，所以实体类必须有setter方法，否则无法完成属性的赋值。

通过运行时类获取对象。

```
//1.加载spring.xml配置文件
ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("spring.xml");
//2.通过运行时类获取对象
Student student2 = (Student) applicationContext.getBean(Student.class);
System.out.println(student2);
```

通过运行时类获取对象有一个弊端，当spring.xml中配置两个Student的bean时程序会抛异常，因为此时两个bean都是由Student类生成的，IoC容器无法讲两个bean返回，必须指定一个唯一的bean。

以上是IoC容器通过无参构造函数创建对象的方式，同时IoC容器也可以通过有参构造函数来创建对象。

```
<bean id="student3" class="com.southwind.entity.Student">
    <constructor-arg name="id" value="3" ></constructor-arg>
    <constructor-arg name="name" value="王五"></constructor-arg>
    <constructor-arg name="age" value="18"></constructor-arg>
</bean>
```

IoC容器会根据constructor-arg标签去加载对应的有参构造函数，创建对象并完成属性赋值。

name的值需要与有参构造函数的形参名对应，value是对应的值。

除了使用name对应参数之外，还可以通过下标index来对应参数。

```
<bean id="student4" class="com.southwind.entity.Student">
    <constructor-arg index="0" value="4"></constructor-arg>
    <constructor-arg index="1" value="小明"></constructor-arg>
    <constructor-arg index="2" value="33"></constructor-arg>
</bean>
```

以上是IoC容器通过有参构造函数创建对象的方式，获取对象有两种方式可以选择：id和运行时类。

IoC容器管理多个对象，并且对象之间有及联关系，如何实现?

```
package com.southwind.entity;

public class Classes {
    private int id;
    private String name;

    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;
    }

    @Override
    public String toString() {
        return "Classes{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

```
package com.southwind.entity;

public class Student {
    private int id;
    private String name;
```

```
    private int age;
    private Classes classes;

    public Classes getClasses() {
        return classes;
    }

    public void setClasses(Classes classes) {
        this.classes = classes;
    }

    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;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                ", classes=" + classes +
                '}';
    }

    public Student() {

    }
```

```
    public Student(int id, String name, int age) {
        System.out.println("通过有参构造函数创建了Student对象");
        this.id = id;
        this.name = name;
        this.age = age;
    }
}
```

```
<!-- 创建Classes对象 -->
<bean id="classes" class="com.southwind.entity.Classes">
    <property name="id" value="1"></property>
    <property name="name" value="一班"></property>
</bean>

<!-- 创建学生对象 -->
<bean id="stu" class="com.southwind.entity.Student">
    <property name="id" value="1"></property>
    <property name="name" value="小红"></property>
    <property name="age" value="16"></property>
    <!-- 将IoC容器中的classes对象赋给stu对象的classes -->
    <property name="classes" ref="classes"></property>
</bean>
```

在spring.xml中，通过ref属性将其他bean赋给当前bean对象的方式叫做依赖注入（DI），是Spring框架非常重要的一种机制，DI是将不同的对象进行关联的一种方式，是IoC的具体实现方式，通过DI和IoC是结合起来使用，所以一般来将IoC包括来DI，IoC是自动创建对象的一种机制，DI是完成对象之间关联关系的方式。

通过有参构造函数来创建对象。

```
public Classes(int id, String name) {
    this.id = id;
    this.name = name;
}
```

```
public Student(int id, String name, int age, Classes classes) {
    this.id = id;
    this.name = name;
    this.age = age;
    this.classes = classes;
}
```

```
<!-- 有参构造创建Classes对象 -->
<bean id="classes" class="com.southwind.entity.Classes">
    <constructor-arg name="id" value="1"></constructor-arg>
    <constructor-arg name="name" value="一班"></constructor-arg>
</bean>

<!-- 有参构造创建Student对象 -->
<bean id="stu" class="com.southwind.entity.Student">
    <constructor-arg name="id" value="1"></constructor-arg>
    <constructor-arg name="name" value="小红"></constructor-arg>
    <constructor-arg name="age" value="16"></constructor-arg>
    <constructor-arg name="classes" ref="classes"></constructor-arg>
</bean>
```

集合属性的DI，通过无参构造函数创建对象。

```
<!-- Student对象 -->
<bean id="student" class="com.southwind.entity.Student">
    <property name="id" value="1"></property>
    <property name="name" value="张三"></property>
    <property name="age" value="22"></property>
</bean>

<bean id="student2" class="com.southwind.entity.Student">
    <property name="id" value="2"></property>
    <property name="name" value="李四"></property>
    <property name="age" value="23"></property>
</bean>

<bean id="student3" class="com.southwind.entity.Student">
    <property name="id" value="3"></property>
    <property name="name" value="王五"></property>
    <property name="age" value="24"></property>
</bean>

<!-- Classes对象 -->
<bean id="classes" class="com.southwind.entity.Classes">
    <property name="id" value="1"></property>
    <property name="name" value="一班"></property>
    <property name="students">
        <list>
            <ref bean="student"></ref>
            <ref bean="student2"></ref>
            <ref bean="student3"></ref>
        </list>
    </property>
</bean>
```

通过有参构造函数创建对象。

```
<bean id="student" class="com.southwind.entity.Student">
    <constructor-arg name="id" value="1"></constructor-arg>
    <constructor-arg name="name" value="张三"></constructor-arg>
    <constructor-arg name="age" value="22"></constructor-arg>
</bean>

<bean id="student2" class="com.southwind.entity.Student">
    <constructor-arg name="id" value="2"></constructor-arg>
    <constructor-arg name="name" value="李四"></constructor-arg>
    <constructor-arg name="age" value="23"></constructor-arg>
</bean>

<bean id="student3" class="com.southwind.entity.Student">
    <constructor-arg name="id" value="3"></constructor-arg>
    <constructor-arg name="name" value="王五"></constructor-arg>
    <constructor-arg name="age" value="24"></constructor-arg>
</bean>

<bean id="classes" class="com.southwind.entity.Classes">
    <constructor-arg name="id" value="1"></constructor-arg>
    <constructor-arg name="name" value="一班"></constructor-arg>
    <constructor-arg name="students">
        <list>
            <ref bean="student"></ref>
            <ref bean="student2"></ref>
            <ref bean="student3"></ref>
        </list>
    </constructor-arg>
</bean>
```

集合属性通过list标签和ref标签完成注入，ref的bean属性指向需要注入的bean对象。

Spring中bean是根据scope来生成的，scope表示bean的作用域。

scope有4种类型：

- singleton：单例，表示通过IoC容器获取的对象是唯一的。
- prototype：原型，表示通过IoC容器获取的对象是不同的。
- request：请求，表示通过IoC容器获取的对象在一次HTTP请求内有效。
- session：会话，表示通过IoC容器获取的对象在一个用户会话内有效。

request和session只适用于web项目，大多数情况下，我们只会使用singleton和prototype两种scope，并且scope的默认值是singleton。

scope=singleton时，一旦加载spring.xml配置文件就会创建bean对象，并且只创建一次，单例模式。

scope=prototype时，加载spring.xml配置文件时不会创建bean对象，每次获取对象时再来创建，获取多少次就创建多少个对象，原型模式。

Spring的继承，与Java中的继承不一样，但是思想是类似的，Spring中的继承表示子bean可以继承父bean中的属性，对象层面的继承，Java中的继承表示子类可以继承父类的结构，类层面的继承。

```
<bean id="user" class="com.southwind.entity.User">
    <property name="id" value="1"></property>
    <property name="name" value="张三"></property>
</bean>

<bean id="user2" class="com.southwind.entity.User" parent="user">
    <property name="name" value="李四"></property>
</bean>

<bean id="student" class="com.southwind.entity.Student">
    <property name="id" value="1"></property>
    <property name="name" value="小明"></property>
</bean>

<bean id="user1" class="com.southwind.entity.User" parent="student"></bean>
```

子bean在继承父bean的属性值的基础之上，可以对属性值进行覆盖，两个不同类型的对象之间也可以实现继承，前提是两个对象的属性完全一致，User中有int id和String name，Student中有int id和String name，同样可以完成继承，读取父bean的属性值，赋给对应的子bean的属性。