SSM框架:

1.Spring:用来管理对象的容器

2.SPringMVC:

3.MyBati

一.Spring的特点:

1.控制反转(IoC,inversion of Control):对象的控制权由程序员转向容器

2.依赖注入(DI,Dependency Injection):配置对象间依赖关系,方便容器创建对象时,把这个对象需要的其他对象创建出来

注:控制反转是思想,依赖注入是做法

3.面向切面编程(AOP,Aspect Oriented Programming)

框架的学习

1.导包

2.配置

3.使用

二.Spring的使用:

1.在pom.xml导入spring-context

```
<dependency>
 <groupId>org.springframework</groupId>
 <artifactId>spring-context</artifactId>
 <version>5.1.5.RELEASE</version>
</dependency>
```

2.Spring的配置:

a.xml

b.注解

c.Java

di:

依赖注入的方式

1属性注入:私有属性+setter

//会找到所有写设置器的方法,对属性进行修改

2.构造方法注入:有参构造方法

3.工厂方法注入

a.非静态工厂方法注入

b.静态工厂方法注

对象作用域

1.singleton:单例模式,容器有且仅生成一个该类型对象

2.prototype:原型模式,每次创建一个新对象

对象的生命周期`init destroy`

类:

```
public class Cat {

 public void init() {
 System.out.println("Cat init");
 }

 public void destroy() {
 System.out.println("Cat destroy");
 }
}
```

```
<!--
init-method:当对象初始化调用该方法
```

```
3 destroy-method:当对象销毁时调用的方法
4 -->
5
6 <bean id="cat2" class="com.lanou.xml.Cat" init-method="init" destroy-meth
od="destroy"/>
```

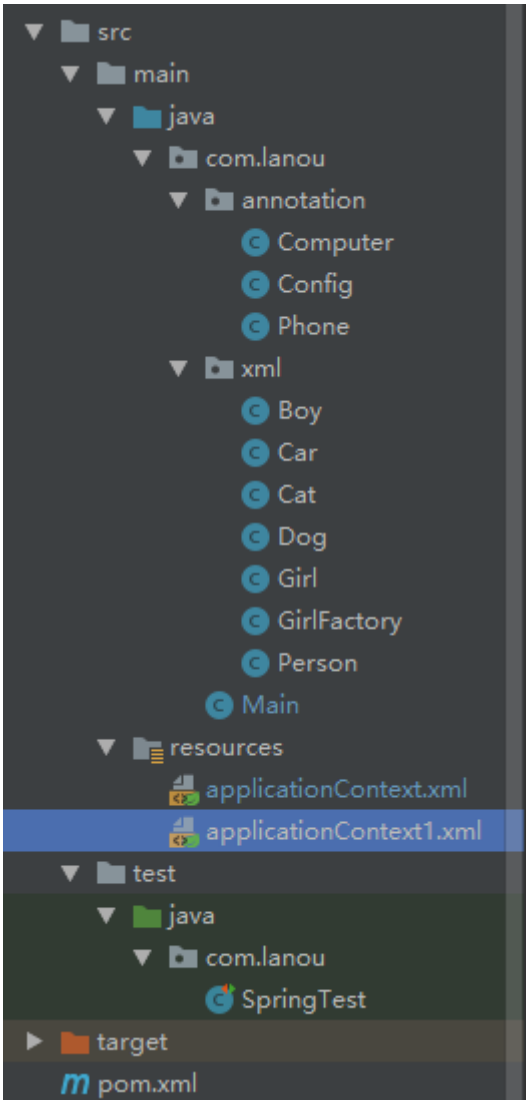
测试方法:

```
1 @Test
2 void test3() {
3  //
4  ClassPathXmlApplicationContext context=new ClassPathXmlApplicationContex
t("applicationContext.xml");
5
6
7  Object cat = context.getBean("cat");
8  System.out.println(cat);
9  Object cat1 = context.getBean("cat");
10  System.out.println(cat1);
11
12  Car car=Car.getInstance();
13  Object cat2 = context.getBean("cat1");
14  System.out.println(cat2);
15  Object cat3 = context.getBean("cat1");
16  System.out.println(cat3);
17
18
19  Object cat4 = context.getBean("cat2");
20  System.out.println(cat4);
21  Object cat5 = context.getBean("cat2");
22  System.out.println(cat5);
23
24  context.close();
25
26 }
```

输出结果:

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
Cat init
com.lanou.xml.Cat@3d680b5a
com.lanou.xml.Cat@3d680b5a
com.lanou.xml.Cat@4b5d6a01
com.lanou.xml.Cat@4a22f9e2
com.lanou.xml.Cat@3c419631
com.lanou.xml.Cat@3c419631
Cat destroy
```

详细代码：



Spring的配置：1.xml注入

在resource中新建applicationContext.xml(名字不唯一)

```
1 <!--
```

```
bean:指定容器管理对象
id:唯一标识符,用于获取对象
class:对象类型
-->
<bean id="girl" class="com.lanou.xml.Girl"/>
<bean id="dog" class="com.lanou.xml.Dog"/>
<bean id="boy" class="com.lanou.xml.Boy"/>
```

a.属性注入

```
<bean id="girl2" class="com.lanou.xml.Girl">
 <!--
 property:用于指定属性的值
 name:属性名,比如:基本数据类型,字符串
 value:属性值,比如:引用类型
 -->
 <property name="boyFrirnd" ref="boy"/>
 <property name="dog" ref="dog"/>
</bean>
```

b.构造方法注入

```
<bean id="girl3" class="com.lanou.xml.Girl">
 <!--
 constructor-arg:配置构造方法的参数
 name:参数名(形参)
 ref:参数值(实参)
 -->
 <constructor-arg name="dog" ref="dog"/>
 <constructor-arg name="boyFrirnd" ref="boy"/>
</bean>

```

c.非静态工厂方法

```
<bean name="girlFactory" class="com.lanou.xml.GirlFactory"/>
<bean id="girl4" factory-bean="girlFactory" factory-method="createGirl">
</bean>
```

d.静态工厂方法

```
<bean id="girl5" class="com.lanou.xml.GirlFactory" factory-method="makeGi
rl"/>
```

e.不同类型的属性注入

类型：

```
private int a;
private String b;
private Dog c;
private int[] d;
private List e;
private Set f;
private Map g;
private Properties h;
```

```
<bean id="person" class="com.lanou.xml.Person">
 <property name="a" value="1"/>
 <!--<property name="b"><value>abc</value></property>-->
 <property name="b"><null/></property>
 <property name="c" ref="dog"/>
 <property name="d">
 <array>
 <value>123</value>
 <value>173</value>
  <value>198</value>
 </array>
 </property>

 <property name="e">
 <list>
 <value>ab</value>
 <value>cfd</value>
 <value>adb</value>
 </list>
 </property>

 <property name="f">
 <set>
 <value>123</value>
 <value>193</value>
 <value>103</value>
```

```
27  </set>
28  </property>
29
30  <property name="g">
31  <map>
32  <entry key="a" value="1"/>
33  <entry key="a" value="1"/>
34  <entry key="a" value="1"/>
35  </map>
36  </property>
37  <property name="h">
38  <props>
39  <prop key="a">1</prop>
40  <prop key="b">2</prop>
41  <prop key="c">3</prop>
42  </props>
43  </property>
44
45 </bean>
```

2.注解注入:

```
1 <context:component-scan base-package="包路径"
```

(包扫描)

@Component（定义）

```
1 扫描包中的类
2 通常指定bean的id例如@Componet（"my"）则该bean的id就是my
3 @Repository 用于对DAO实现类进行注解
4 @Service 用于对Service实现类进行注解
5 @Controller 用于对Controller实现类进行注解
6 @Repository
7 -->
8 <context:component-scan base-package="com.lanou.annotation"/>
```

@Scope（作用域）

```
1 1 在类上使用@Scope，value默认singleton可以修改prototype
```

@Value（注入基本类型属性）

```
1 1 在属性上使用@Value，指定要注入的值，使用时可以不写setter方法
```

```
2 2 若写了可以在seter方法上使用
```

@Autowired（按类型注入域属性）

```
1 1 在域属性上使用@Autowried,即在被@Scope修饰过的属性上使用
2 2 使用时，类中无需setter。若写了可以在seter方法上使用
```

@Qualifer（按名称注入域属性）

```
1 1 使用时需要配合@Autowired使用@Qualifer
2 2 使用时，类中无需setter。若写了可以在seter方法上使用
3 3 value是bean中的id 即指定id为XXX的域属性装配
4 4 此时@Autowried通常会使用required属性 参数是布尔类型，代表是否忽略
5 5 默认值为true，表示当匹配失败后，会终止程序运行。
6 6 若将其值设置为false，则匹配失败，将被忽略，未匹配的属性值为null。
```

@Recource（域属性注解）

```
1 1 这是一个jdk中的注解，版本为Java6以上
2 2 @Resource注解既可以按名称匹配Bean，也可以按类型匹配Bean。
3 3 注解若不带任何参数，则会按照类型进行匹配注入。
4 4 若加了属性 name="xx"，则会按照bean中的id匹配注入。
```

@PostConstruct（生命周期的开始）

```
1 1 使用在方法上，无参数
```

@PreDestroy（生命周期的结束）

```
1 1 使用在方法上，无参数
```

Resource和Autowired区别:

1.Autowired先找类型,后找id,包搜索;Resource,java提供的注解,先找id名字.后找类型

在类中配置@Component

新建手机类,电脑类,要求打印手机类可以显示电脑的属性和方法

手机类中:

```
1 @Component
2 public class Phone {
3  @Value("1")
4  private int a;
5
6
```

```
7   //Resource和Autowired区别:
8   //1.先找类型,后找id,包搜索;Resource,java提供的注解,先找id名字.后找类型
9   //先找类型,再找id,(required=false)
10   //只有一个类型.可以匹配
11  // @Autowired
12  // @Qualifier("computer")
13   //指定id找的是谁
14   @Resource
15   private Computer computer;
16
17   @Override
18   public String toString() {
19   return "Phone{" +
20   "a=" + a +
21   ", computer=" + computer +
22   '}';
23
24   }
25
26   //初始化执行
27   @PostConstruct
28   public void init(){
29   System.out.println("init");
30   }
31   //销毁执行
32   @PreDestroy
33   public void destroy(){
34   System.out.println("destroy");
35   }
36  }
```

电脑类中无内容

测试类中：

```
1  @Test
2  void test4() {
3   ClassPathXmlApplicationContext context=new ClassPathXmlApplicationContex
t("applicationContext1.xml");
4
```

```
5   Object phone = context.getBean("phone");
6   System.out.println(phone);
7
8   Object computer = context.getBean("computer");
9
10  }
```

执行结果:

```
init
Phone{a=1, computer=com.lanou.annotation.Computer@4da4253}
destroy
```

3.java代码注入.

新建一个配置类:

```
1  //配置类
2  @ComponentScan("com.lanou.*")
3  @Configuration
4  public class Config {
5   @Bean
6   public Computer getComputer(){
7   return new Computer();
8   }
9
10  }
```