

初识spring

day01:IOC控制反转 DI依赖注入 整合JUnit 整合web

day02 : AOP切面编程 jdbcTemplate

day03:事务管理

jar包

4+1 : 4个核心 : (bean , core,context,expression) +1个依赖 (commons-logging.jar)

配置文件

- 位置 : 任意 , 开发中一般在classpath下 (即src)
- 名称 : 任意 , 开发中常用applicationContext.xml
- 内容 : 添加schema约束

beans.xml中的约束

```
<?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:util="http://www.springframework.org/schema/util"
  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/util
    http://www.springframework.org/schema/util/spring-util.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
<!-- bean definitions here -->
</beans>
```

入门案例 DI

- DI 依赖注入

is a : 继承

has a: 有一个 , 成员变量 , 依赖

```
class B{
    private A a; //B类依赖A类
}
```

依赖 : 一个对象需要使用另一个对象

注入 : 通过setter方法进行另一个对象实例的设置。

- 例如

```
class BookService{
//之前开发 : 接口=实现类
    private BookDao bookDao = new BookDaoImpl();
//spring之后
    private BookDao bookDao
    setter方法
}
```

模拟spring执行过程

创建service实例 : BookService bookService = new BookServiceImpl() -->loc <bean>

创建dao实例 : BookDao bookDao = new BookDaoImpl(); -->loc

将dao设置给service : bookService.setBookDao(bookDao); -->DI <property>

创建<bean>，注入的类写在<property>中，name为注入的属性，ref为依赖的<bean>的id

```
<bean id="BookDao" class="com.mccken.di.BookDaoImpl"></bean>

<bean id="BookService" class="com.mccken.di.BookServiceImpl">
    <property name="bookDao" ref="BookDao"></property>
</bean>
```

```
private BookDao bookDao;
public void setBookDao(BookDao bookDao) {
    this.bookDao = bookDao;
}
@Override
public void addBook() {
    this.bookDao.save();
}
}
```

注入的依赖
BookDao类，bookDao为
BookServiceImpl的属性
要写入<property>的
name中

测试代码在此：

```
public class TestDI {
```

```
@Test
```

```
public void Test01(){
```

```
String xmlPath = "com/mccken/di/beans.xml";
```

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
```

```
BookService bookService = (BookService) applicationContext.getBean("BookService");
```

```
bookService.addBook();
```

```
}
```

```
}
```

```
7 public class TestDI {
```

```
8
```

```
9 @Test
```

```
0 public void Test01(){
```

```
1     String xmlPath = "com/mccken/di/beans.xml";
```

```
2     ApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
```

```
3     BookService bookService = (BookService) applicationContext.getBean("BookService");
```

```
4     bookService.addBook();
```

```
5 }
```

```
6 }
```

```
7
```

注意：创建对象时，new接口

- 编写流程(基于xml)

- 1.导入jar包：4+1 -->beans/core/context/pression | commons-logging
- 2.编写目标类：dao和服务
- 3.spring配置文件
 - IoC：<bean id="" class="">
 - DI:<bean><property name="" value=""|ref="">
 - 实例化方式：

- 默认构造
- 静态工厂：<bean id="" class="工厂类" factory-method="静态方法">
- 实例工厂:<bean id="工厂id" class="工厂类"> <bean id="" factory-bean="工厂id" factory-method="方法">
- 作用域：<bean id="" class="" scope="singleton | prototype" >
- 生命周期：<bean id="" class="" init-method="" destroy-method="">

后处理bean BeanPostProcessor接口，<bean class="注册"> 对容器所有的bean都生效

- 属性注入
 - 构造方法注入:<bean> <constructor-arg index="" type="">
 - setter方法注入
 - p命名空间
 - SpEL
 - 集合注入
 - 数组<array>
 - List<list>
 - Set<set>
 - Map<map><entry key="" value="">
 - Properties<props><prop key="">...
- 4.核心api
 - BeanFactory,延时实例化bean，第一次调用getBean
 - applicationContext 一般常用，功能更强
 - ClassPathXmlApplicationContext 加载classpath xml文件
 - FileSystemXmlApplicationContext加载指定盘符文件， ServletContext.getRealPath()

• 基于注解

◦ 1.扫描含有注解的类

<context:component-scan base-package="包名">

◦ 2.常见的注解

- @Component 组件，任意bean
- @Controller web层
- @Service service层
- @Repository dao层
- 注入--->字段或者setter方法
 - 普通值：@Value
 - 引用值：
 - 类型：@Autowired
 - 名称1：@Autowired @Qualifier("名称")
 - 名称2：@Resoutce("名称")
- 作用域：@Scope("prototype")
- 生命周期：
 - 初始化：@PostConstruct
 - 销毁方法：@PreDestroy

• 注解和xml混合使用

- 1.将所有的bean都配置到xml中
- 2.将所有的依赖都使用注解
 - @Autowired

默认不生效，为了生效，需要在xml配置：<context:annotation-config>

