Spring02

注解配置-控制反转 IOC

```
@Repository("accountDao1")
public class AccountDaoImpl implements AccountDao
```

```
@Service("accountService")
public class AccountServiceImpl implements AccountService
```

第三步：applicationContext.xml

```
<?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">

<!--告知spring在创建容器时要扫描的包，配置所需要的标签不是在beans的约束中，而是一个名称为
context名称空间和约束中-->
<context:component-scan base-package="com.itheima"></context:component-scan>
</beans>
```

注解配置-依赖注入 DI

1：@Autowired、@Qualifier（对象的注入）

2：@Resource（对象的注入）

3：@Value（值的注入）

```
@Value(value = "张三")
private String name;
@Value(value = "18")
private Integer age;
```

```java
@Service("accountService")
public class AccountServiceImpl implements AccountService {

    @Autowired
    @Qualifier("accountDao1")
    //@Resource(name = "accountDao2")
    private AccountDao accountDao;

    public void  saveAccount(){
        accountDao.saveAccount();
    }
}
```

注解配置-作用域

```java
@Service("accountService")
@Scope("prototype")
public class AccountServiceImpl implements AccountService
```

注解配置-初始化和销毁（单例）（了解）

```java
@Service("accountService")
//@Scope("prototype")// 测试初始化和销毁，不能使用多例
public class AccountServiceImpl implements AccountService {
    @Autowired
    @Qualifier("accountDao1")
    //@Resource(name = "accountDao2")
    private AccountDao accountDao = null;
    public AccountServiceImpl(){
        System.out.println("构造方法...");
    }
    @PostConstruct
    public void  init(){
        System.out.println("初始化方法执行了");
    }
    @PreDestroy
    public void  destroy(){
        System.out.println("销毁方法执行了");
    }
    public void  saveAccount(){
        accountDao.saveAccount();
    }
}
```

使用 XML+注解配置 IOC（常用）

```
@Repository("accountDao")
public class AccountDaoImpl implements AccountDao {

    @Autowired
    private QueryRunner runner;
```

```
@Service("accountService")
public class AccountServiceImpl implements AccountService {

    @Autowired
    private AccountDao accountDao;
```

编写 applicationContext.xml

```
<?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">
    <!-- 告知spring在创建容器时要扫描的包 -->
    <context:component-scan base-package="com.itheima"></context:component-scan>
    <!--配置QueryRunner-->
    <bean id="runner" class="org.apache.commons.dbutils.QueryRunner" scope="prototype">
        <!--注入数据源-->
        <constructor-arg name="ds" ref="dataSource"></constructor-arg>
    </bean>
    <!-- 配置数据源 -->
    <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <!--连接数据库的必备信息-->
        <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
        <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/itcastspring">
        </property>
        <property name="user" value="root"></property>
        <property name="password" value="root"></property>
    </bean>
</beans>
```

注解和 XML 比较

	基于XML配置	基于注解配置
Bean定义	<bean id="..." class="..." />	@Component 衍生类@Repository @Service @Controller
Bean名称	通过 id或name 指定	@Component("person")
Bean注入	<property> 或者通过p命名空间	@Autowired 按类型注入 @Qualifier按名称注入
生命过程、Bean作用范围	init-method destroy-method 范围 scope属性	@PostConstruct 初始化 @PreDestroy 销毁 @Scope设置作用范围
适合场景	Bean来自第三方，使用其它	Bean的实现类由用户自己开发