

## 第2天: Spring

### 一、学习目标

1. 能够使用Jdbc模板
2. 能够配置Spring的连接池
3. 能够使用JdbcTemplate完成增删查改操作
4. 使用注解改造CRUD工程
5. 基于纯注解改造CRUD工程
6. 能够实现Spring框架整合Junit
7. 理解代理设计模式
8. 掌握动态代理的两种实现方式
9. 完成动态代理案例

### 二、Spring整合JDBC

#### 2.1 JdbcTemplate概述

JdbcTemplate是Spring提供的一个模板类，它是对jdbc的封装。用于支持持久层的操作。它的特点是：简单、方便。

它简化了JDBC的使用并有助于避免常见错误。它执行核心的JDBC工作流程，留下应用程序代码以提供SQL并提取结果。此类执行SQL查询或更新，在ResultSets上启动迭代并捕获JDBC异常，并将其转换为 `org.springframework.dao` 程序包中定义的通用异常，信息量更大的异常层次结构。

#### 2.2 入门案例

##### 2.2.1 准备数据

- 数据库: db.properties

```
1 db.driver=com.mysql.jdbc.Driver
2 # db.url=jdbc:mysql:///mybatisdb?
  serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&
  ;characterEncoding=UTF-8
3 db.url=jdbc:mysql:///mybatisdb?
  serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&characterEn
  coding=UTF-8
4 db.username=root
5 db.password=root
```

- 数据表: `mybatis`.`account`

```
1 SET FOREIGN_KEY_CHECKS=0;
2
3 -----
4 -- Table structure for account
5 -----
```



```
8      `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键',
9      `uid` int(11) DEFAULT '1' COMMENT '用户编号',
10     `money` decimal(10,2) DEFAULT '0.00' COMMENT '余额',
11     PRIMARY KEY (`id`)
12 ) ENGINE=InnoDB AUTO_INCREMENT=114 DEFAULT CHARSET=utf8;
13
14 -----
15 -- Records of account
16 -----
17 INSERT INTO `account` VALUES ('1', '1', '10.00');
18 INSERT INTO `account` VALUES ('2', '10', '9.90');
19 INSERT INTO `account` VALUES ('3', '24', '99.00');
```

## 2.2.2 案例演示

- 工程名称: spring-day02-jdbc
- 添加依赖: pom.xml

```
1  <!-- 添加依赖 -->
2  <dependencies>
3      <!-- 1. Spring IOC 依赖-->
4      <dependency>
5          <groupId>org.springframework</groupId>
6          <artifactId>spring-context</artifactId>
7          <version>5.1.7.RELEASE</version>
8      </dependency>
9      <!-- 2. Spring Jdbc 依赖-->
10     <dependency>
11         <groupId>org.springframework</groupId>
12         <artifactId>spring-jdbc</artifactId>
13         <version>5.1.7.RELEASE</version>
14     </dependency>
15     <dependency>
16         <groupId>mysql</groupId>
17         <artifactId>mysql-connector-java</artifactId>
18         <version>8.0.17</version>
19     </dependency>
20     <!-- 3. Junit 单元测试 依赖 -->
21     <dependency>
22         <groupId>junit</groupId>
23         <artifactId>junit</artifactId>
24         <version>4.12</version>
25     </dependency>
26 </dependencies>
```

- 创建实体: com.itheima.jdbc.Account

```
1 package com.itheima.jdbc;
2
3 /**
4  * 账户实体.
5  *
6  * @author : Jason.lee
7  * @version : 1.0
```



```
10
11     private Integer id;
12     private Integer uid;
13     private Double money;
14
15     @Override
16     public String toString() {
17         return "Account{" +
18             "id=" + id +
19             ", uid=" + uid +
20             ", money=" + money +
21             '}';
22     }
23 }
```

- 案例代码: JdbcTests

```
1  import com.itheima.jdbc.Account;
2  import org.junit.Test;
3  import org.springframework.jdbc.core.BeanPropertyRowMapper;
4  import org.springframework.jdbc.core.JdbcTemplate;
5  import org.springframework.jdbc.datasource.DriverManagerDataSource;
6
7  import javax.sql.DataSource;
8  import java.util.List;
9
10 /**
11  * JdbcTemplate操作案例.
12  *
13  * @author : Jason.lee
14  * @version : 1.0
15  */
16 public class JdbcTests {
17
18     /**
19     * 演示：查询所有账户
20     */
21     @Test
22     public void testJdbcTemplate () {
23         DataSource ds = createDataSource();
24
25         // 1. 创建操作模板对象
26         JdbcTemplate jd = new JdbcTemplate(ds);
27         // 2. 定义Sql语句
28         String sql = "select * from account";
29         // 3. 处理结果
30         List<Account> all = jd.query(sql, new
31         BeanPropertyRowMapper<Account>(Account.class));
32         all.forEach(x -> System.out.println(x));
33     }
34
35     /**
36     * 创建数据源
37     * @return 数据源对象
38     */
39     private DataSource createDataSource() {
40         DriverManagerDataSource ds = new DriverManagerDataSource();
41         ds.setDriverClassName("com.mysql.jdbc.Driver");
42         ds.setUrl("jdbc:mysql://localhost:3306/test");
43         ds.setUsername("root");
44         ds.setPassword("123456");
45         return ds;
46     }
47 }
```



```
40         ds.setDriverClassName("com.mysql.jdbc.Driver");
41         ds.setUrl("jdbc:mysql:///mybatisdb?
serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&characterE
ncoding=UTF-8");
42         ds.setUsername("root");
43         ds.setPassword("root");
44         return ds;
45     }
46 }
```

## 2.3 使用IOC管理模板

### 2.3.1 添加配置

- 配置文件: applicationContext.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7      <!-- 1. 定义数据源 -->
8      <bean id="dataSource"
9            class="org.springframework.jdbc.datasource.DriverManagerDataSource">
10         <property name="driverClassName"
11             value="com.mysql.jdbc.Driver"/>
12         <property name="url" value="jdbc:mysql:///mybatisdb?
serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&
p;characterEncoding=UTF-8"/>
13         <property name="username" value="root"/>
14         <property name="password" value="root"/>
15     </bean>
16     <!-- 2. 创建操作模板 -->
17     <bean id="jdbcTemplate"
18           class="org.springframework.jdbc.core.JdbcTemplate">
19         <property name="dataSource" ref="dataSource"/>
20     </bean>
21 </beans>
```

### 2.3.2 使用容器

- 案例代码: JdbcTests

```
1  /**
2   * 演示：查询所有账户
3   */
4  ClassPathXmlApplicationContext ioc = new
ClassPathXmlApplicationContext("applicationContext.xml");
5  // 1. 获取操作模板对象
6  JdbcTemplate jd = ioc.getBean(JdbcTemplate.class);
7  @Test
8  public void testJdbcTemplate() {
```

```
11 // 3. 处理结果
12 List<Account> all = jd.query(sql, new
    BeanPropertyRowMapper<Account>(Account.class));
13 all.forEach(x -> System.out.println(x));
14 }
```

## 2.4 结果集映射

### 2.4.1 多记录映射器

- 查询多个账户

```
1 /**
2  * 演示：多记录查询（BeanProperty映射器）
3  */
4 @Test
5 public void testMany (){
6     // 2. 定义sql语句
7     String sql = "select * from account where id in(?,?)";
8     // 3. 处理结果
9     List<Account> all = jd.query(sql, new
    BeanPropertyRowMapper<Account>(Account.class), new Integer[]{1, 3});
10    all.forEach(x -> System.out.println(x));
11 }
```

### 2.4.2 自定义映射器

- 查询多个账户
- 自定义映射器: com.itheima.jdbc.AccountRowMapper

```
1 package com.itheima.jdbc;
2
3 import org.springframework.jdbc.core.RowMapper;
4
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7
8 /**
9  * 自定义账户属性与表字段映射器。
10  *
11  * @author : Jason.lee
12  * @version : 1.0
13  */
14 public class AccountRowMapper implements RowMapper<Account> {
15     /**
16      * 单行记录处理方法
17      * @param rs 结果集
18      * @param i 行号 (0开始)
19      * @return 行记录的封装对象
20      * @throws SQLException 异常信息
21      */
22     @Override
```



```
26         account.setId(rs.getInt("id"));
27         account.setUid(rs.getInt("uid"));
28         account.setMoney(rs.getDouble("money"));
29         return account;
30     }
31 }
```

- 单元测试: JdbcTests

```
1  /**
2   * 演示：多记录查询（自定义映射器）
3   */
4  @Test
5  public void testCustom () {
6      // 2. 定义sql语句
7      String sql = "select * from account where id in(?,?)";
8      // 3. 处理结果
9      List<Account> all = jd.query(sql, new AccountRowMapper(), new
Integer[]{1, 3});
10     all.forEach(x -> System.out.println(x));
11 }
```

### 2.4.3 单记录映射

- 根据编号查找账户

```
1  /**
2   * 演示：单记查询（自定义映射器）
3   */
4  @Test
5  public void testOne () {
6      // 2. 定义sql语句
7      String sql = "select * from account where id in(?)";
8      // 3. 处理结果
9      Account account = jd.queryForObject(sql, new RowMapper<Account>() {
10         /**
11          * 每行记录调用的封装方法
12          * 好处：可以自由映射表字段与对象属性的关系
13          * @param resultSet 结果集
14          * @param i 行号（0开始）
15          * @return 封装的对象
16          */
17         @Override
18         public Account mapRow(ResultSet resultSet, int i) throws
SQLException {
19             System.out.println("===== "+ i);
20             Account account = new Account();
21             account.setId(resultSet.getInt("id"));
22             account.setUid(resultSet.getInt("uid"));
23             account.setMoney(resultSet.getDouble("money"));
24             return account;
25         }
26     }, 1);
27     System.out.println(account);
```

- 添加依赖: pom.xml

```
1 <!-- 4. Log4j 依赖 -->
2 <dependency>
3     <groupId>log4j</groupId>
4     <artifactId>log4j</artifactId>
5     <version>1.2.17</version>
6 </dependency>
7 <dependency>
8     <groupId>org.slf4j</groupId>
9     <artifactId>slf4j-api</artifactId>
10    <version>1.7.26</version>
11 </dependency>
12 <dependency>
13     <groupId>org.slf4j</groupId>
14     <artifactId>slf4j-log4j12</artifactId>
15     <version>1.7.7</version>
16 </dependency>
17
18 <!-- C3P0数据源 -->
19 <dependency>
20     <groupId>com.mchange</groupId>
21     <artifactId>c3p0</artifactId>
22     <version>0.9.5.4</version>
23 </dependency>
24 <!-- Druid数据源 -->
25 <dependency>
26     <groupId>com.alibaba</groupId>
27     <artifactId>druid</artifactId>
28     <version>1.1.12</version>
29 </dependency>
```

### 2.5.1 内置数据源

- applicationContext.xml

```
1 <!-- 内部数据源 -->
2 <bean id="dataSource"
3     class="org.springframework.jdbc.datasource.DriverManagerDataSource">
4     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
5     <property name="url" value="jdbc:mysql:///mybatisdb?
6         serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&
7         characterEncoding=UTF-8"/>
8     <property name="username" value="root"/>
9     <property name="password" value="root"/>
10 </bean>
```

### 2.5.2 C3P0数据源

- applicationContext.xml



```
3     <property name="driverClass" value="com.mysql.jdbc.Driver"/>
4     <property name="jdbcUrl" value="jdbc:mysql:///mybatisdb?
serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&
;characterEncoding=UTF-8"/>
5     <property name="user" value="root"/>
6     <property name="password" value="root"/>
7 </bean>
```

### 2.5.3 Druid数据源

- applicationContext.xml

```
1 <!-- druid数据源 -->
2 <bean id="druid" class="com.alibaba.druid.pool.DruidDataSource">
3     <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
4     <property name="url" value="jdbc:mysql:///mybatisdb?
serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true&
;characterEncoding=UTF-8"/>
5     <property name="username" value="root"/>
6     <property name="password" value="root"/>
7 </bean>
```

## 三、实现IOC的CRUD

- 需求: 使用Spring管理对象
- 技术: 使用jdbcTemplate实现crud

### 2.1 构建工程

- 工程名称: spring-day02-crud
- pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <parent>
6         <artifactId>spring2</artifactId>
7         <groupId>com.itheima</groupId>
8         <version>1.0-SNAPSHOT</version>
9     </parent>
10    <modelVersion>4.0.0</modelVersion>
11
12    <artifactId>spring-day02-crud</artifactId>
13
14    <dependencies>
15        <!-- Spring IOC依赖 -->
16        <dependency>
17            <groupId>org.springframework</groupId>
```





```
20     </dependency>
21     <!-- Spring JDBC依赖 -->
22     <dependency>
23         <groupId>org.springframework</groupId>
24         <artifactId>spring-jdbc</artifactId>
25         <version>5.1.7.RELEASE</version>
26     </dependency>
27     <!-- ali 连接池依赖 -->
28     <dependency>
29         <groupId>com.alibaba</groupId>
30         <artifactId>druid</artifactId>
31         <version>1.1.12</version>
32     </dependency>
33     <!-- Mysql 数据库驱动依赖 -->
34     <dependency>
35         <groupId>mysql</groupId>
36         <artifactId>mysql-connector-java</artifactId>
37         <version>8.0.15</version>
38     </dependency>
39     <!-- 单元测试 依赖 -->
40     <dependency>
41         <groupId>junit</groupId>
42         <artifactId>junit</artifactId>
43         <version>4.12</version>
44     </dependency>
45 </dependencies>
46
47 </project>
```

- Account.java

```
1 package com.itheima.crud.entity;
2
3 /**
4  * 账户类.
5  *
6  * @author : Jason.lee
7  * @version : 1.0
8  */
9 public class Account {
10     private Integer id;
11     private Integer uid;
12     private Double money;
13
14     public Integer getId() {
15         return id;
16     }
17
18     public void setId(Integer id) {
19         this.id = id;
20     }
21
22     public Integer getUid() {
23         return uid;
24     }
```



```
27         this.uid = uid;
28     }
29
30     public Double getMoney() {
31         return money;
32     }
33
34     public void setMoney(Double money) {
35         this.money = money;
36     }
37
38     @Override
39     public String toString() {
40         return "Account{" +
41             "id=" + id +
42             ", uid=" + uid +
43             ", money=" + money +
44             '}';
45     }
46 }
```

- AccountDaoImpl.java

```
1 package com.itheima.crud.dao;
2
3 import com.itheima.crud.entity.Account;
4 import org.springframework.jdbc.core.BeanPropertyRowMapper;
5 import org.springframework.jdbc.core.JdbcTemplate;
6
7 import java.util.List;
8
9 /**
10  * 账户持久层操作类.
11  *
12  * @author : Jason.lee
13  * @version : 1.0
14  */
15 public class AccountDaoImpl implements AccountDao {
16
17     JdbcTemplate jdbcTemplate;
18
19     public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
20         this.jdbcTemplate = jdbcTemplate;
21     }
22
23     @Override
24     public List<Account> findAll() {
25         String sql = "select * from account";
26         return jdbcTemplate.query(
27             sql, new BeanPropertyRowMapper(Account.class)
28         );
29     }
30
31     @Override
32     public void add(Account account) {
```



```
34         jdbcTemplate.update(sql, account.getId(), account.getUid(),
35         account.getMoney());
36     }
37     @Override
38     public void update(Account account) {
39         String sql = "update account set uid=?, money=? where id=?";
40         jdbcTemplate.update(sql, account.getUid(), account.getMoney(),
41         account.getId());
42     }
43     @Override
44     public void del(Account account) {
45         String sql = "delete from account where id=?";
46         jdbcTemplate.update(sql, account.getId());
47     }
48 }
49
```

- AccountServiceImpl.java

```
1 package com.itheima.crud.service;
2
3 import com.itheima.crud.dao.AccountDao;
4 import com.itheima.crud.entity.Account;
5
6 import java.util.List;
7
8 /**
9  * 账户业务类.
10  *
11  * @author : Jason.lee
12  * @version : 1.0
13  */
14 public class AccountServiceImpl implements AccountService {
15
16     AccountDao accountDao;
17
18     public void setAccountDao(AccountDao accountDao) {
19         this.accountDao = accountDao;
20     }
21
22     @Override
23     public List<Account> findAll() {
24         return accountDao.findAll();
25     }
26
27     @Override
28     public void add(Account account) {
29         accountDao.add(account);
30     }
31
32     @Override
33     public void update(Account account) {
34         accountDao.update(account);
35     }
36
37     @Override
38     public void del(Account account) {
39         accountDao.del(account);
40     }
41 }
```

```
37     @Override
38     public void del(Account account) {
39         accountDao.del(account);
40     }
41 }
```

## 2.2 文件配置

- db.properties

```
1 db.driver=com.mysql.cj.jdbc.Driver
2 # 针对MySQL 8.x数据库添加的参数
3 #   serverTimezone: 指定时区(UTC)
4 #   useSSL: 指定是否使用加密安全连接(false)
5 #   allowPublicKeyRetrieval: 是否允许检索公钥(true)
6 db.url=jdbc:mysql:///mybatisdb?
   serverTimezone=UTC&useSSL=false&allowPublicKeyRetrieval=true
7 db.username=root
8 db.password=root
```

- applicationContext.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
6       http://www.springframework.org/schema/beans/spring-beans.xsd
7       http://www.springframework.org/schema/context
8       https://www.springframework.org/schema/context/spring-context.xsd">
9
10    <!-- 加载配置文件 -->
11    <context:property-placeholder location="classpath:db.properties"/>
12
13    <!-- 定义数据源对象 -->
14    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
15        <property name="driverClassName" value="${db.driver}"/>
16        <property name="url" value="${db.url}"/>
17        <property name="username" value="${db.username}"/>
18        <property name="password" value="${db.password}"/>
19    </bean>
20
21    <!-- 定义JDBC操作模板对象 -->
22    <bean id="jdbcTemplate"
23          class="org.springframework.jdbc.core.JdbcTemplate">
24        <property name="dataSource" ref="dataSource"/>
25    </bean>
26
27    <!-- 定义账户操作类对象 -->
28    <bean id="accountDao" class="com.itheima.crud.dao.AccountDaoImpl">
29        <property name="jdbcTemplate" ref="jdbcTemplate"/>
```



```
class="com.itheima.crud.service.AccountServiceImpl">
33     <property name="accountDao" ref="accountDao"/>
34 </bean>
35
36 </beans>
```

## 2.3 单元测试

- 测试类

```
1  import com.itheima.crud.entity.Account;
2  import com.itheima.crud.service.AccountService;
3  import org.junit.Test;
4  import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6  import java.util.List;
7
8  /**
9   * IOC容器的CRUD单元测试.
10  */
11  public class CrudTests {
12
13      // 创建IOC容器
14      ClassPathXmlApplicationContext context = new
15      ClassPathXmlApplicationContext("applicationContext.xml");
16      // 获取对象
17      AccountService service = (AccountService)
18      context.getBean("accountService");
19
20      @Test
21      public void testFind (){
22          List<Account> all = service.findAll();
23          all.stream().forEach(x-> System.out.println(x));
24      }
25
26      @Test
27      public void testAdd (){
28          Account account = new Account();
29          account.setId(11);
30          account.setUid(1);
31          account.setMoney(999.0);
32          service.add(account);
33      }
34
35      @Test
36      public void testUpdate (){
37          Account account = new Account();
38          account.setId(11);
39          account.setUid(1);
40          account.setMoney(0.0);
41          service.update(account);
42      }
43
44      @Test
45      public void testDel (){
46          Account account = new Account();
47      }
```

```
44     }  
45 }
```

## 四、使用注解改造案例

- 配置+注解是企业开发的常用方式

### 4.1 工程改造

- 工程名称: spring-day02-ax

### 4.2 注解改造

- AccountDaoImpl.java

```
1  package com.itheima.crud.dao;  
2  
3  import com.itheima.crud.entity.Account;  
4  import org.springframework.beans.factory.annotation.Autowired;  
5  import org.springframework.jdbc.core.BeanPropertyRowMapper;  
6  import org.springframework.jdbc.core.JdbcTemplate;  
7  import org.springframework.stereotype.Repository;  
8  
9  import java.util.List;  
10  
11 /**  
12  * 使用@Repository代替  
13  * <bean id="accountDaoImpl" class="com.itheima.crud.dao.AccountDaoImpl">.  
14  */  
15 @Repository  
16 public class AccountDaoImpl implements AccountDao {  
17  
18     // 使用@Autowired代替  
19     // <property name="jdbcTemplate" ref="jdbcTemplate"/>  
20     @Autowired  
21     JdbcTemplate jdbcTemplate;  
22  
23     // 使用注解后无需提供set方法  
24     // public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {  
25     //     this.jdbcTemplate = jdbcTemplate;  
26     // }  
27  
28     @Override  
29     public List<Account> findAll() {  
30         String sql = "select * from account";  
31         return jdbcTemplate.query(  
32             sql, new BeanPropertyRowMapper(Account.class)  
33         );  
34     }  
35  
36     @Override
```



```
39         jdbcTemplate.update(sql, account.getId(), account.getUid(),
40         account.getMoney());
41     }
42     @Override
43     public void update(Account account) {
44         String sql = "update account set uid=?, money=? where id=?";
45         jdbcTemplate.update(sql, account.getUid(), account.getMoney(),
46         account.getId());
47     }
48     @Override
49     public void del(Account account) {
50         String sql = "delete from account where id=?";
51         jdbcTemplate.update(sql, account.getId());
52     }
53 }
54
```

- AccountServiceImpl.java

```
1 package com.itheima.crud.service;
2
3 import com.itheima.crud.dao.AccountDao;
4 import com.itheima.crud.entity.Account;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9
10 /**
11  * 使用@Service代替
12  * <bean id="accountServiceImpl"
13  * class="com.itheima.crud.dao.AccountDaoImpl">.
14  */
15 @Service
16 public class AccountServiceImpl implements AccountService {
17     /**
18     * 使用@Autowired代替
19     * <bean id="accountService"
20     * class="com.itheima.crud.service.AccountServiceImpl">
21     */
22     @Autowired
23     AccountDao accountDao;
24
25     // 使用注解不需要提供set方法
26     // public void setAccountDao(AccountDao accountDao) {
27     //     this.accountDao = accountDao;
28     // }
29
30     @Override
31     public List<Account> findAll() {
32         return accountDao.findAll();
33     }
34 }
```



```
35     public void add(Account account) {
36         accountDao.add(account);
37     }
38
39     @Override
40     public void update(Account account) {
41         accountDao.update(account);
42     }
43
44     @Override
45     public void del(Account account) {
46         accountDao.del(account);
47     }
48 }
```

### 4.3 配置改造

- applicationContext.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xmlns:context="http://www.springframework.org/schema/context"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans
6          http://www.springframework.org/schema/beans/spring-beans.xsd
7          http://www.springframework.org/schema/context
8          https://www.springframework.org/schema/context/spring-context.xsd">
9
10     <!-- 【重点】 -->
11     <!-- 使用注解需要开启注解扫描 -->
12     <context:component-scan base-package="com.itheima.crud"/>
13
14
15     <!-- 加载配置文件 -->
16     <context:property-placeholder location="classpath:db.properties"/>
17
18     <!-- 定义数据源对象 -->
19     <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
20         <property name="driverClassName" value="${db.driver}"/>
21         <property name="url" value="${db.url}"/>
22         <property name="username" value="${db.username}"/>
23         <property name="password" value="${db.password}"/>
24     </bean>
25
26     <!-- 定义JDBC操作模板对象 -->
27     <bean id="jdbcTemplate"
28         class="org.springframework.jdbc.core.JdbcTemplate">
29         <property name="dataSource" ref="dataSource"/>
30     </bean>
31
32     <!-- 定义账户操作类对象 -->
33     <!-- 使用@Repository代替了 -->
34     <!-- <bean id="accountDao" class="com.itheima.crud.dao.AccountDaoImpl">
```



```
36
37     <!-- 定义账户业务类对象 -->
38     <!-- 使用@Service代替了 -->
39     <!--<bean id="accountService"
class="com.itheima.crud.service.AccountServiceImpl">
40         <property name="accountDao" ref="accountDao"/>
41     </bean>-->
42
43 </beans>
```

## 4.4 单元测试

```
1  import com.itheima.crud.entity.Account;
2  import com.itheima.crud.service.AccountService;
3  import org.junit.Test;
4  import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6  import java.util.List;
7
8  /**
9   * 改造内容(根据类型获取service对象).
10  * 创建对象注解创建的对象默认以首字母小写的类名为对象名 (accountServiceImpl)
11  *
12  * @author : Jason.lee
13  * @version : 1.0
14  */
15  public class CrudTests {
16
17      // 创建IOC容器
18      ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
19      // 获取对象
20      AccountService service =
context.getBean(/*accountService*/AccountService.class);
21
22      @Test
23      public void testFind (){
24          List<Account> all = service.findAll();
25          all.stream().forEach(x-> System.out.println(x));
26      }
27      @Test
28      public void testAdd (){
29          Account account = new Account();
30          account.setId(11);
31          account.setUid(1);
32          account.setMoney(999.0);
33          service.add(account);
34      }
35      @Test
36      public void testUpdate (){
37          Account account = new Account();
38          account.setId(11);
39          account.setUid(1);
```



```
42     }
43     @Test
44     public void testDel () {
45         Account account = new Account();
46         account.setId(11);
47         service.del(account);
48     }
49 }
```

## 五、Spring纯注解开发

- 纯注解也可以理解为 **零配置**
- 改造目标: 去除所有配置文件
- 核心思想: 找到相应的注解 **代替** 配置文件已经配置文件中的配置项

### 5.1 替代注解

- 先睹为快 (使用示例)

```
1 package com.itheima.crud;
2
3 import com.alibaba.druid.pool.DruidDataSource;
4 import com.itheima.crud.dao.AccountDao;
5 import org.springframework.beans.factory.annotation.Value;
6 import org.springframework.context.annotation.*;
7 import org.springframework.jdbc.core.JdbcTemplate;
8
9 import javax.sql.DataSource;
10
11 /**
12  * Spring 配置类.
13  * @Configuration: 代替Spring配置文件 (applicationContext.xml)
14  * @ComponentScan: 代替<context:component-scan...> value=base-package
15  * @PropertySource 代替<context:property-placeholder...> value=location
16  * @Import: 代替<import resource="..."> value=配置类字节码
17  */
18 @Configuration
19 @ComponentScan("com.itheima.crud")
20 @PropertySource("classpath:db.properties")
21 @Import(OtherConfig.class)
22 public class Config {
23
24     /**
25      * @Value: 用于注入配置文件中的参数值
26      * value: 属性值 | 基本数据类型
27      * ${}: 注入配置文件中参数值
28      * #{ }: 注入SpEL表达式结果
29      */
30     @Value("${db.driver}")
31     String driver;
32     @Value("${db.url}")
```



```
35     String username;  
36     @Value("${db.password}")  
37     String password;  
38  
39     /**  
40     * @Bean: 用于将方法返回值对象添加到IOC容器  
41     * value: 对象名称  
42     */  
43     @Bean  
44     public DataSource druidDataSource(){  
45         DruidDataSource source = new DruidDataSource();  
46         source.setDriverClassName(driver);  
47         source.setUrl(url);  
48         source.setUsername(username);  
49         source.setPassword(password);  
50         return source;  
51     }  
52  
53     @Bean  
54     public JdbcTemplate jdbcTemplate(DataSource dataSource){  
55         return new JdbcTemplate(dataSource);  
56     }  
57 }
```

### @Configuration

- 位置: 类
- 作用: 修饰类识别为配置类
- 意义: 代替Spring配置文件 (applicationContext.xml)

### @ComponentScan

- 位置: 类
- 作用: 配置注解扫描包
- 意义: 代替<context:component-scan..
- value: 与base-package属性相同

### @PropertySource

- 位置: 类
- 作用: 加载配置文件
- 意义: 代替<context:property-placeholder..
- value: 与location属性相同

### @Bean

- 位置: 方法
- 作用: 将返回的对象添加到IOC容器
- 意义: 代替<bean标签弥补第三方无法注解缺陷
- value: 对象名称

### @Import

- 位置: 类



- value: 其他配置类字节码

## 5.2 创建容器

- 单元测试

```
1  import com.itheima.crud.Config;
2  import com.itheima.crud.entity.Account;
3  import com.itheima.crud.service.AccountService;
4  import org.junit.Test;
5  import
    org.springframework.context.annotation.AnnotationConfigApplicationContext;
6  import org.springframework.context.support.ClassPathXmlApplicationContext;
7
8  import java.util.List;
9
10 /**
11  * 改造内容(根据类型获取service对象).
12  * 创建对象注解创建的对象默认以首字母小写的类名为对象名 (accountServiceImpl)
13  *
14  * @author : Jason.lee
15  * @version : 1.0
16  */
17 public class AnnoTests {
18
19     // 创建IOC容器
20     AnnotationConfigApplicationContext context = new
    AnnotationConfigApplicationContext(Config.class);
21     // 获取对象
22     AccountService service =
    context.getBean(/*accountService*/AccountService.class);
23
24     @Test
25     public void testFind (){
26         List<Account> all = service.findAll();
27         all.stream().forEach(x-> System.out.println(x));
28     }
29     @Test
30     public void testAdd (){
31         Account account = new Account();
32         account.setId(11);
33         account.setUId(1);
34         account.setMoney(999.0);
35         service.add(account);
36     }
37     @Test
38     public void testUpdate (){
39         Account account = new Account();
40         account.setId(11);
41         account.setUId(1);
42         account.setMoney(0.0);
43         service.update(account);
44     }
```



```
47     Account account = new Account();
48     account.setId(11);
49     service.del(account);
50 }
51 }
```

## 六、Spring测试框架

### 6.1 单元测试的问题

- 每次都需要手动创建IOC容器

### 6.2 Spring测试框架

- Spring-test整合了JUnit框架
- 仅需在测试类上添加两个注解即可自动创建容器
- pom.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5                               http://maven.apache.org/xsd/maven-4.0.0.xsd">
6      <parent>
7          <artifactId>spring2</artifactId>
8          <groupId>com.itheima</groupId>
9          <version>1.0-SNAPSHOT</version>
10     </parent>
11     <modelVersion>4.0.0</modelVersion>
12     <artifactId>spring-day02-ax</artifactId>
13
14     <dependencies>
15         <!-- Spring IOC依赖 -->
16         <dependency>
17             <groupId>org.springframework</groupId>
18             <artifactId>spring-context</artifactId>
19             <version>5.1.7.RELEASE</version>
20         </dependency>
21         <!-- Spring JDBC依赖 -->
22         <dependency>
23             <groupId>org.springframework</groupId>
24             <artifactId>spring-jdbc</artifactId>
25             <version>5.1.7.RELEASE</version>
26         </dependency>
27         <!-- ali 连接池依赖 -->
28         <dependency>
29             <groupId>com.alibaba</groupId>
30             <artifactId>druid</artifactId>
31             <version>1.1.12</version>
32         </dependency>
33         <!-- MySQL 数据库驱动依赖 -->
34         <dependency>
```



```
37         <version>8.0.15</version>
38     </dependency>
39     <!-- 单元测试 依赖 -->
40     <dependency>
41         <groupId>junit</groupId>
42         <artifactId>junit</artifactId>
43         <version>4.12</version>
44     </dependency>
45     <!-- 添加Spring测试框架 依赖 -->
46     <dependency>
47         <groupId>org.springframework</groupId>
48         <artifactId>spring-test</artifactId>
49         <version>5.1.7.RELEASE</version>
50     </dependency>
51 </dependencies>
52
53 </project>
```

### @RunWith

- 位置: 类
- 作用: 指定测试框架启动类
- 意义: 代替ApplicationContext实现类

### @ContextConfiguration

- 位置: 类
- 作用: 指定配置文件或配置类
- 意义: 配合@RunWith使用代理ApplicationContext实现类

## 5.3 单元测试示例

```
1  import com.itheima.crud.Config;
2  import com.itheima.crud.entity.Account;
3  import com.itheima.crud.service.AccountService;
4  import org.junit.Test;
5  import org.junit.runner.RunWith;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import
    org.springframework.context.annotation.AnnotationConfigApplicationContext;
8  import org.springframework.test.context.ContextConfiguration;
9  import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
10
11 import java.util.List;
12
13 /**
14  * @RunWith: 指定启动器
15  * @ContextConfiguration: 指定启动配置
16  * value: 配置文件路径
17  * classes: 配置类路径
18  */
19 @RunWith(SpringJUnit4ClassRunner.class)
20 @ContextConfiguration(classes = {Config.class})
```

```
24      * 使用Spring-test框架
25      * 可以直接使用注入注解给属性赋值
26      */
27      @Autowired
28      AccountService service;
29
30      @Test
31      public void testFind (){
32          List<Account> all = service.findAll();
33          all.stream().forEach(x-> System.out.println(x));
34      }
35  }
```

## 七、代理模式

- 为目标对象提供一个代理对象以便控制外部对目标对象的访问

### 2.1 静态代理

- 代理类是程序运行前准备好的代理方式被成为 静态代理
- interface: Star.java

```
1  package com.itheima.proxy;
2
3  /**
4   * 明星接口.
5   *
6   * @author : Jason.lee
7   */
8  public interface Star {
9      /**
10     * 唱歌.
11     *
12     * @param money the money
13     */
14     void sing(Integer money);
15
16     /**
17     * 拍戏.
18     *
19     * @param money the money
20     */
21     void act(Integer money);
22 }
```

- target: LiuStar.java

```
1  package com.itheima.proxy;
```



```
5  *
6  * @author : Jason.lee
7  * @version : 1.0
8  */
9  public class LiuStar implements Star {
10     @Override
11     public void sing(Integer money) {
12         System.out.println("刘德华唱歌真好听，收费: "+money);
13     }
14
15     @Override
16     public void act(Integer money) {
17         System.out.println("刘德华拍戏棒棒哒，收费: "+money);
18     }
19 }
```

- proxy: LiuStarProxy.java

```
1  package com.itheima.proxy;
2
3  /**
4   * 刘德华代理人(经纪人).
5   * 静态代理:
6   * 特点:
7   *     1. 需要实现与目标对象相同的接口
8   *     2. 需要在内部维护目标对象
9   * 缺点:
10    *     1. 接口方法较多时，代理实现较麻烦（可能只需要代理个别方法）
11    *     2. 接口扩展方法后，目标对象与代理对象都需要更改
12    *
13    * @author : Jason.lee
14    * @version : 1.0
15    */
16  public class LiuStarProxy implements Star {
17
18      Star star = new LiuStar();
19
20      @Override
21      public void sing(Integer money) {
22          if (money > 10000) {
23              star.sing(money);
24          } else {
25              System.out.println("档期忙!");
26          }
27      }
28
29      @Override
30      public void act(Integer money) {
31          if (money > 20000) {
32              star.act(money);
33          } else {
34              System.out.println("档期忙!");
35          }
36      }
37  }
```





```
1 package com.itheima.proxy;
2
3 import org.junit.Test;
4
5 public class LiuStarProxyTest {
6
7     @Test
8     public void sing() {
9         new LiuStarProxy().sing(10001);
10        new LiuStarProxy().act(10001);
11    }
12 }
```

- 静态代理的缺点
  - 接口方法较多时,代理实现较麻烦 (可能只需要代理个别方法)
  - 接口扩展方法后,目标对象与代理对象都需要更改

## 2.2 动态代理

- 代理类在程序运行时创建的代理方式被称为 动态代理

### 2.2.1 Proxy动态代理

- JDK提供的动态代理API

```
1 @Test
2 public void testProxy() {
3     // 目标对象
4     LiuStar ls = new LiuStar();
5
6     /**
7      * 动态创建代理对象:
8      * Proxy: JDK动态代理的Api也是生成的代理对象的超类
9      * loader: 类加载器
10     * interfaces: 代理对象需要实现的接口组
11     * h: 代理对象方法调用的处理程序
12     */
13     Star star = (Star) Proxy.newProxyInstance(
14         // 默认类加载器
15         this.getClass().getClassLoader(),
16         // 实现的接口组
17         new Class[]{Star.class},
18         // 调用处理程序
19         new InvocationHandler() {
20             @Override
21             public Object invoke(Object proxy, Method method, Object[]
22             args) throws Throwable {
23                 // 获取方法名
24                 String name = method.getName();
25                 // 获取参数值
26                 Integer arg = (Integer) args[0];
27                 // 拦截方法
```



```
30         return method.invoke(ls, args);
31     }
32     return null;
33 }
34 }
35 );
36
37 // 代理类继承了Proxy
38 System.out.println(star instanceof Proxy);
39 // 代理类实现了Star
40 System.out.println(star instanceof Star);
41 // 查看类字节码
42 System.out.println(star.getClass()); // class com.sun.proxy.$Proxy4
43
44 // 调用代理方法
45 star.sing(10001);
46 star.act(10001);
47
48 }
```

- JDK动态代理也称之为 **接口代理**：代理类实现了与目标对象相同的接口 (\$Proxy4 implement Star)

### 2.2.2 Cglib动态代理

- 第三方框架实现的动态代理

```
1  @Test
2  public void testCglib() {
3
4      // 目标对象
5      LiuStar ls = new LiuStar();
6
7      /**
8       * 动态创建代理对象：
9       *  Enhancer: Cglib动态代理的Api
10      *    superclass: 超类字节码
11      *    interfaces: 代理对象需要实现的接口组
12      *    callback: 代理对象方法调用的拦截器
13      */
14      Star star = (Star) Enhancer.create(
15          LiuStar.class,
16          new Class[]{ Star.class },
17          new MethodInterceptor() {
18              @Override
19              public Object intercept(Object o, Method method, Object[]
20              objects, MethodProxy methodProxy) throws Throwable {
21                  // 获取方法名
22                  String name = method.getName();
23                  // 获取参数值
24                  Integer arg = (Integer) objects[0];
25                  // 拦截方法
26                  if ((name.equals("sing") && arg > 10000)
27                      || (name.equals("act") && arg > 20000)) {
```

```
30         return null;
31     }
32 }
33 );
34
35 // 代理类继承了LiuStar
36 System.out.println(star instanceof LiuStar);
37 // 代理类实现了Star
38 System.out.println(star instanceof Star);
39 // 查看类字节码
40 System.out.println(star.getClass()); // class
com.itheima.proxy.LiuStar$$EnhancerByCGLIB$$13a93339
41 // 调用代理方法
42 star.sing(10001);
43 star.act(10001);
44 }
```

- Cglib动态代理也称之为 **子类代理**：代理类继承了目标对象 (\$\$EnhancerByCGLIB extends LiuStar)

### 2.2.3 动态代理对比

	JDK	CGLIB
实现原理	利用 <b>实现</b> 接口方法来拦截目标方法	利用 <b>继承</b> 覆写方法来拦截目标方法
实现前提	<b>需要目标对象实现接口</b>	需要目标方法未使用final/static修饰
实现效率	JDK1.6以前较Cglib慢; 1.6以及1.7大量调用时较慢	1.8时被JDK代理超越