

arena teaching system

Spring+MyBatis开发实战



Java企业应用及互联网
高级工程师培训课程

达内集团教学研发部 编著

目 录

Unit01	1
1. 企业级项目环境搭建	2
1.1. Spring 和 MyBatis 整合方法	2
1.1.1. 【Spring 和 MyBatis 整合方法】导包	2
1.1.2. 【Spring 和 MyBatis 整合方法】配置 applicationContext.xml	2
1.1.3. 【Spring 和 MyBatis 整合方法】阶段性测试	3
1.2. 应用 Spring MVC	3
1.2.1. 【应用 Spring MVC】配置 web.xml	3
1.2.2. 【应用 Spring MVC】配置 applicationContext.xml	4
1.2.3. 【应用 Spring MVC】阶段性测试	5
经典案例	6
1. 使用 Spring 整合 MyBatis , 查询全部员工数据	6
2. 使用 Spring MVC , 完成员工查询功能	15
Unit02	22
1. MyBatis 动态 SQL	23
1.1. 动态 SQL 简介	23
1.1.1. 【动态 SQL 简介】动态 SQL 简介	23
1.2. 进行判断	23
1.2.1. 【进行判断】if 元素	23
1.2.2. 【进行判断】choose 元素	24
1.3. 拼关键字	24
1.3.1. 【拼关键字】where 元素	24
1.3.2. 【拼关键字】set 元素	25
1.3.3. 【拼关键字】trim 元素	26
1.4. 进行循环	26
1.4.1. 【进行循环】foreach 元素	26
经典案例	28
1. if 元素使用案例	28
2. choose 元素使用案例	33
3. where 元素使用案例	37
4. set 元素使用案例	42
5. trim 元素使用案例	47
6. foreach 元素使用案例	55

Unit03	63
1. Spring 与 Ajax	64
1.1. Spring 与 Ajax	64
1.1.1. 【Spring 与 Ajax】 Ajax 简介	64
1.1.2. 【Spring 与 Ajax】 Spring 对 Ajax 的支持	64
1.1.3. 【Spring 与 Ajax】 @ResponseBody 应用	65
经典案例	67
1. Spring 与 Ajax 应用案例	67
Unit04	73
1. MyBatis 关联映射	74
1.1. 主键映射	74
1.1.1. 【主键映射】 主键映射作用	74
1.1.2. 【主键映射】 自动递增	74
1.1.3. 【主键映射】 非自动递增	75
1.2. 关联映射	75
1.2.1. 【关联映射】 关联映射作用	75
1.2.2. 【关联映射】 嵌套查询映射	76
1.2.3. 【关联映射】 嵌套结果映射	76
1.3. 集合映射	77
1.3.1. 【集合映射】 集合映射作用	77
1.3.2. 【集合映射】 嵌套查询映射	78
1.3.3. 【集合映射】 嵌套结果映射	78
1.4. 鉴别器	79
1.4.1. 【鉴别器】 鉴别器的作用	79
1.4.2. 【鉴别器】 鉴别器的使用	80
1.4.3. 【鉴别器】 嵌套结果映射	80
经典案例	81
1. 主键映射使用案例	81
2. 多对一嵌套查询映射使用案例	85
3. 多对一嵌套结果映射使用案例	92
4. 一对多嵌套查询映射使用案例	98
5. 一对多嵌套结果映射使用案例	104
6. 鉴别器使用案例	109
Unit05	118
1. Spring AOP	119
1.1. AOP 介绍	119
1.1.1. 【AOP 介绍】 AOP 概念及优点	119
1.1.2. 【AOP 介绍】 什么是方面	119
1.1.3. 【AOP 介绍】 什么是目标	120

1.1.4. 【AOP 介绍】什么是切入点	120
1.1.5. 【AOP 介绍】什么是通知	120
1.1.6. 【AOP 介绍】AOP 实现原理.....	121
1.2. XML 配置实现 AOP	121
1.2.1. 【XML 配置实现 AOP】开发步骤	121
1.2.2. 【XML 配置实现 AOP】前置通知	122
1.2.3. 【XML 配置实现 AOP】环绕通知	122
1.2.4. 【XML 配置实现 AOP】异常通知	122
1.3. 注解实现 AOP	123
1.3.1. 【注解实现 AOP】开发步骤	123
1.3.2. 【注解实现 AOP】前置通知	123
1.3.3. 【注解实现 AOP】环绕通知	123
1.3.4. 【注解实现 AOP】异常通知	124
经典案例	125
1. Spring AOP 前置通知案例	125
2. Spring AOP 环绕通知案例	130
3. Spring AOP 异常通知案例	134
4. Spring AOP 注解使用案例	139
Unit06	145
1. Spring 事务处理.....	146
1.1. Spring 事务介绍	146
1.1.1. 【Spring 事务介绍】Spring 事务简介	146
1.1.2. 【Spring 事务介绍】编程式事务.....	146
1.1.3. 【Spring 事务介绍】声明式事务.....	147
1.2. 声明式事务.....	148
1.2.1. 【声明式事务】注解实现声明式事务	148
1.2.2. 【声明式事务】XML 配置实现声明式事务	149
1.3. Spring 事务控制	149
1.3.1. 【Spring 事务控制】事务回滚	149
1.3.2. 【Spring 事务控制】事务传播	150
1.3.3. 【Spring 事务控制】事务隔离级别.....	151
2. Spring 与 RESTful	152
2.1. Spring 与 RESTful	152
2.1.1. 【Spring 与 RESTful】RESTful 简介.....	152
2.1.2. 【Spring 与 RESTful】Spring 对 RESTful 的支持	153
2.1.3. 【Spring 与 RESTful】@RequestMapping 应用	153
2.1.4. 【Spring 与 RESTful】@PathVariable 应用	154
2.1.5. 【Spring 与 RESTful】客户端发送 PUT、DELETE 请求	154

2.1.6. 【Spring 与 RESTful】静态资源访问处理	154
经典案例	155
1. Spring 声明式事务-注解应用	155
2. Spring 声明式事务-XML 配置	162
3. Spring 声明式事务-回滚机制	165
4. RESTful 应用案例	170

Spring + MyBatis

开发实战 Unit01



知识体系.....Page 2

企业级项目环境搭建	Spring 和 MyBatis 整合方法	导包
		配置 applicationContext.xml
		阶段性测试
	应用 Spring MVC	配置 web.xml
		配置 applicationContext.xml
		阶段性测试

经典案例.....Page 6



使用 Spring 整合 MyBatis , 查询全部员工数据	阶段性测试
使用 Spring MVC , 完成员工查询功能	阶段性测试



1.1.3. 【Spring 和 MyBatis 整合方法】阶段性测试

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">阶段性测试</h4> <ul style="list-style-type: none"> • 写出查询全部的员工数据的案例，测试项目环境 <ul style="list-style-type: none"> – 创建员工DAO接口EmpDao，增加查询全部数据的方法，并使用@MyBatisRepository注解标识该接口。 – 创建员工表的MyBatis映射文件EmpMapper.xml，并实现查询全部员工数据。 – 创建JUnit测试类，并增加测试方法，测试查询全部员工数据的方法。 <div style="text-align: right;">  </div>
---	---

1.2. 应用 Spring MVC

1.2.1. 【应用 Spring MVC】配置 web.xml

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">配置web.xml</h4> <ul style="list-style-type: none"> • 在web.xml中配置如下内容 <ul style="list-style-type: none"> – 配置DispatcherServlet，充当Spring的前端控制器，用于分发请求 – 配置一个Filter，用于处理中文参数的乱码问题 <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">配置web.xml (续1)</h4> <ul style="list-style-type: none"> • 前端控制器配置代码如下 <pre> <!-- Spring前端控制器 --> <servlet> <servlet-name>SpringMVC</servlet-name> <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class> <init-param> <param-name>contextConfigLocation</param-name> <param-value>classpath:applicationContext.xml</param-value> </init-param> </servlet> <servlet-mapping> <servlet-name>SpringMVC</servlet-name> <url-pattern>*.do</url-pattern> </servlet-mapping> </pre> <div style="text-align: right;">  </div>
---	--

代码清单


配置web.xml (续2)

• 解决中文乱码Filter的配置代码如下

```

<!-- 使用Filter解决中文乱码问题 -->
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>*.do</url-pattern>
</filter-mapping>

```



+

1.2.2. 【应用 Spring MVC】配置 applicationContext.xml

代码清单

配置applicationContext.xml

• 在applicationContext.xml中追加如下配置

- 开启注解扫描
- 开启RequestMapping注解
- 配置ViewResolver，处理请求转发

+

代码清单

配置applicationContext.xml (续1)

• 配置代码如下


```

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 开启RequestMapping注解 -->
<mvc:annotation-driven />



<!-- 处理请求转发 -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/" />
  <property name="suffix" value=".jsp" />
</bean>

```



+

1.2.3. 【应用 Spring MVC】阶段性测试

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div><div>阶段性测试</div><div></div><div><ul style="list-style-type: none">• 完成员工查询功能，测试项目环境<ul style="list-style-type: none">- 创建员工模块业务控制器EmpController，并增加查询方法，调用EmpDao查询全部员工数据，发送至员工列表页面。- 创建员工列表页面emp_list.jsp，并将EmpController发送过来的全部员工数据循环输出在表格中。</div><div><div>知识讲解</div><div></div></div></div>
---	---

经典案例

1. 使用 Spring 整合 MyBatis , 查询全部员工数据

- 问题

使用 Spring 整合 MyBatis , 实现对数据库的访问。

- 方案

采用 MapperScannerConfigurer 方式整合 MyBatis。

- 步骤

步骤一：建表

本案例中采用的员工表，建表脚本如下：

--员工表

```
create table t_emp(  
  empno number(4) primary key,  
  ename varchar(20),  
  job varchar(10),  
  mgr number(4),  
  hiredate date,  
  sal number(9,2),  
  comm number(9,2),  
  deptno number(4)  
);
```

--员工表 sequence

```
create sequence emp_seq start with 100;
```

--预置的员工表数据

```
ALTER SESSION SET NLS DATE FORMAT = 'yyyy mm dd hh24:mi:ss';  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(1,'SMITH','CLERK',3,'1980-5-12',800,null,20);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(2,'ALLEN','SALESMAN',3,'1981-6-3',1600,300,30);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(3,'WARD','SALESMAN',4,'1990-3-15',1250,500,30);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(4,'JONES','MANAGER',5,'1985-4-8',2975,null,20);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(5,'MARTIN','SALESMAN',7,'1986-3-8',1250,1400,30);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(6,'BLAKE','MANAGER',9,'1989-6-1',2850,null,30);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(7,'CLARK','MANAGER',9,'1995-10-1',2450,null,10);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(8,'SCOTT','ANALYST',9,'1993-5-1',3000,null,20);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(9,'KING','PRESIDENT',null,'1988-8-8',5000,null,10);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values  
(10,'TURNER','SALESMAN',5,'1983-2-1',1500,0,30);  
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
```

```
{11,'ADAMS','CLERK',5,'1992-7-3',1100,null,20});
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(12,'JAMES','CLERK',1,'1996-9-10',950,null,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(13,'FORD','ANALYST',1,'1993-1-1',3000,null,20);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(14,'MILLER','CLERK',3,'1983-10-9',1300,null,10);
commit;
```

步骤二：导包

创建 WEB 项目 SpringUnit01，并导入如下开发包：

- 1) 数据库驱动包、连接池包、MyBatis 开发包
- 2) Spring 开发包
- 3) JSTL 开发包
- 4) Spring 整合 MyBatis 开发包

导入之后，项目的包结构如下图：

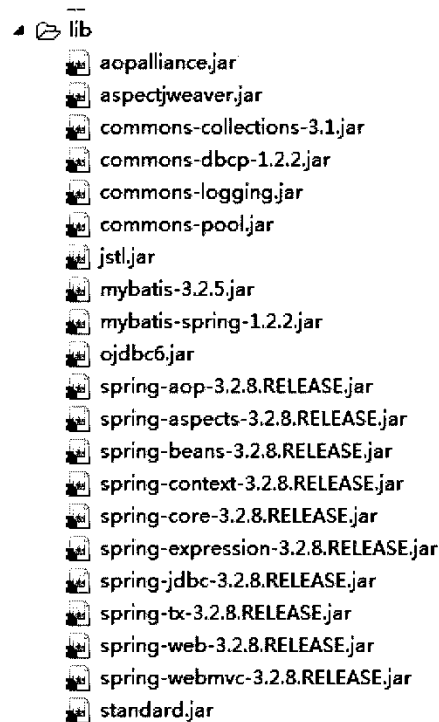


图-1

步骤三：配置 applicationContext.xml

在 src 下创建 applicationContext.xml，并配置如下内容：

- 1) 配置数据源
- 2) 配置 SqlSessionFactory
- 3) 配置 MyBatis 注解（该注解需要自定义）

完成上述配置后，applicationContext.xml 中代码如下：

```
<?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"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
    http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
        <property name="username" value="lhh" />
        <property name="password" value="123456" />
    </bean>

    <!-- 配置 SqlSessionFactory -->
    <bean
        class="org.mybatis.spring.SqlSessionFactoryBean"
        id="sqlSessionFactory"
        <property name="dataSource" ref="ds" />
        <property
            name="mapperLocations"
            value="classpath:com/tarena/entity/*.xml" />
    </bean>

    <!-- 配置 MyBatis 注解 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.tarena.dao" />
        <property
            name="annotationClass"
            value="com.tarena.annotation.MyBatisRepository" />
    </bean>

</beans>
```

上述配置中使用的 MyBatis 注解 `com.tarena.annotation.MyBatisRepository` 需要自定义，该注解的代码如下：

```
package com.tarena.annotation;

/**
 * MyBatis 使用该注解标识 DAO
 */
public @interface MyBatisRepository {
}
```

步骤四：创建实体类

创建员工实体类，代码如下：

```
package com.tarena.entity;

import java.sql.Date;

/**
 * 员工表的实体类
 */
public class Emp {

    private Integer empno;
    private String ename;
    private String job;
    private Integer mgr;
    private Date hiredate;
    private Double sal;
    private Double comm;
    private Integer deptno;

    public Integer getEmpno() {
        return empno;
    }

    public void setEmpno(Integer empno) {
        this.empno = empno;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public String getJob() {
        return job;
    }

    public void setJob(String job) {
        this.job = job;
    }

    public Integer getMgr() {
        return mgr;
    }

    public void setMgr(Integer mgr) {
        this.mgr = mgr;
    }

    public Date getHiredate() {
        return hiredate;
    }

    public void setHiredate(Date hiredate) {
        this.hiredate = hiredate;
    }

    public Double getSal() {
        return sal;
    }

}
```

```
public void setSal(Double sal) {
    this.sal = sal;
}

public Double getComm() {
    return comm;
}

public void setComm(Double comm) {
    this.comm = comm;
}

public Integer getDeptno() {
    return deptno;
}

public void setDeptno(Integer deptno) {
    this.deptno = deptno;
}
}
```

步骤五：创建 DAO 接口

创建员工 DAO 接口 EmpDao，并在接口中增加查询全部员工的方法，代码如下：

```
package com.tarena.dao;

import java.util.List;
import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

}
```

步骤六：创建 MyBatis 映射文件

创建员工表的 MyBatis 映射文件 EmpMapper.xml，并在该文件中实现查询全部员工数据所需的配置，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

</mapper>
```

步骤七：创建 Junit 测试类

创建 Junit 测试类，测试 EmpDao 中的 findAll 方法，代码如下：

```
package com.tarena.test;

import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Emp> list = dao.findAll();
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}
```

• 完整代码

员工表的建表脚本完整代码如下：

```
--员工表
create table t_emp(
empno number(4) primary key,
ename varchar(20),
job varchar(10),
mgr number(4),
hiredate date,
sal number(9,2),
comm number(9,2),
deptno number(4)
);

--员工表 sequence
create sequence emp_seq start with 100;

--预置的员工表数据
ALTER SESSION SET NLS_DATE_FORMAT = 'yyyy mm dd hh24:mi:ss';
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(1,'SMITH','CLERK',3,'1980-5-12',800,null,20);
```



```
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(2,'ALLEN','SALESMAN',3,'1981-6-3',1600,300,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(3,'WARD','SALESMAN',4,'1990-3-15',1250,500,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(4,'JONES','MANAGER',5,'1985-4-8',2975,null,20);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(5,'MARTIN','SALESMAN',7,'1986-3-8',1250,1400,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(6,'BLAKE','MANAGER',9,'1989-6-1',2850,null,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(7,'CLARK','MANAGER',9,'1995-10-1',2450,null,10);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(8,'SCOTT','ANALYST',9,'1993-5-1',3000,null,20);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(9,'KING','PRESIDENT',null,'1988-8-8',5000,null,10);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(10,'TURNER','SALESMAN',5,'1983-2-1',1500,0,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(11,'ADAMS','CLERK',5,'1992-7-3',1100,null,20);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(12,'JAMES','CLERK',1,'1996-9-10',950,null,30);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(13,'FORD','ANALYST',1,'1993-1-1',3000,null,20);
insert into t_emp (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO) values
(14,'MILLER','CLERK',3,'1983-10-9',1300,null,10);
commit;
```

MyBatis 注解完整代码如下：

```
package com.tarena.annotation;

/**
 * MyBatis 使用该注解标识 DAO
 */
public @interface MyBatisRepository {
}
```

员工实体类 Emp 完整代码如下：

```
package com.tarena.entity;
import java.sql.Date;

/**
 * 员工表的实体类
 */
public class Emp {

    private Integer empno;
    private String ename;
    private String job;
    private Integer mgr;
    private Date hiredate;
    private Double sal;
    private Double comm;
    private Integer deptno;

    public Integer getEmpno() {
        return empno;
    }

    public void setEmpno(Integer empno) {
        this.empno = empno;
    }
}
```

```
public String getEname() {
    return ename;
}

public void setEname(String ename) {
    this.ename = ename;
}

public String getJob() {
    return job;
}

public void setJob(String job) {
    this.job = job;
}

public Integer getMgr() {
    return mgr;
}

public void setMgr(Integer mgr) {
    this.mgr = mgr;
}

public Date getHiredate() {
    return hiredate;
}

public void setHiredate(Date hiredate) {
    this.hiredate = hiredate;
}

public Double getSal() {
    return sal;
}

public void setSal(Double sal) {
    this.sal = sal;
}

public Double getComm() {
    return comm;
}

public void setComm(Double comm) {
    this.comm = comm;
}

public Integer getDeptno() {
    return deptno;
}

public void setDeptno(Integer deptno) {
    this.deptno = deptno;
}
}
```

DAO 接口完整代码如下：

```
package com.tarena.dao;

import java.util.List;
import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
```

```
* 员工表的 DAO 组件
*/
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

}
```

MyBatis 映射文件 EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

</mapper>
```

Junit 测试类完整代码如下：

```
package com.tarena.test;

import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Emp> list = dao.findAll();
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}
```

2. 使用 Spring MVC，完成员工查询功能

- 问题

使用 Spring MVC，完成员工查询功能。

- 方案

应用 Spring MVC，处理页面请求。

- 步骤

步骤一：配置 web.xml

在 web.xml 中配置如下内容：

- 1) 配置 DispatcherServlet，充当 Spring 的前端控制器，用于分发请求。
- 2) 配置一个 Filter，用于处理中文参数的乱码问题。

配置后，web.xml 代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
  <display-name></display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <!-- Spring 前端控制器 -->
  <servlet>
    <servlet-name>SpringMVC</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>
        contextConfigLocation
      </param-name>
      <param-value>
        classpath:applicationContext.xml
      </param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>SpringMVC</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <!-- 使用 Filter 解决中文乱码问题 -->
  <filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>
      org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
```

```
<init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
</init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>

</web-app>
```

步骤二：配置 applicationContext.xml

在 applicationContext.xml 中追加如下配置：

- 1) 开启注解扫描
- 2) 开启 @RequestMapping 注解
- 3) 配置 ViewResolver，处理请求转发

配置后，applicationContext.xml 代码如下：

```
<?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"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
        <property name="username" value="lh" />
        <property name="password" value="123456" />
    </bean>

    <!-- 配置 SqlSessionFactory -->
    <bean
        class="org.mybatis.spring.SqlSessionFactoryBean"
        id="sqlSessionFactory"
        <property name="dataSource" ref="ds" />
```

```

        <property
value="classpath:com/tarena/entity/*.xml" />
    </bean>

    <!-- 配置 MyBatis 注解 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.tarena.dao" />
        <property
value="com.tarena.annotation.MyBatisRepository" />
        name="annotationClass"
    </bean>

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.tarena" />

    <!-- 开启 RequestMapping 注解 -->
    <mvc:annotation-driven />

    <!-- 处理请求转发 -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>

```

步骤三：创建业务控制器

创建业务控制器 EmpController，并增加查询方法实现查询业务，代码如下：

```

package com.tarena.controller;

import java.util.List;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
public class EmpController {

    @Resource
    private EmpDao empDao;

    @RequestMapping("/findEmp.do")
    public String find(Model model) {
        List<Emp> list = empDao.findAll();
        model.addAttribute("emps", list);
        return "emp/emp_list";
    }

}

```

步骤四：创建员工列表页面

创建员工列表页面 emp_list.jsp，将查询到的员工数据显示在表格中，代码如下：

```
<%@page pageEncoding="utf-8"%>
```

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
</head>
<body>
<table width="60%" border="1" cellpadding="2" cellspacing="0">
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
<c:forEach items="{emp}" var="emp">
<tr>
<td>{emp.empno}</td>
<td>{emp.ename}</td>
<td>{emp.job}</td>
<td>{emp.mgr}</td>
<td>{emp.hiredate}</td>
<td>{emp.sal}</td>
<td>{emp.comm}</td>
<td>{emp.deptno}</td>
</tr>
</c:forEach>
</table>
</body>
</html>
```

步骤五：测试

打开浏览器，访问员工列表，效果如下图：

← → C localhost:8088/SpringUnit01/emp/findEmp.do
 IE TarenaMail T 课程管理 TTS7 百度一下 hao123 NETCROSS

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12	800.0		20
2	ALLEN	SALESMAN	3	1981-06-03	1600.0	300.0	30
3	WARD	SALESMAN	4	1990-03-15	1250.0	500.0	30
4	JONES	MANAGER	5	1985-04-08	2975.0		20
5	MARTIN	SALESMAN	7	1986-03-08	1250.0	1400.0	30
6	BLAKE	MANAGER	9	1989-06-01	2850.0		30
7	CLARK	MANAGER	9	1995-10-01	2450.0		10
8	SCOTT	ANALYST	9	1993-05-01	3000.0		20
9	KING	PRESIDENT		1988-08-08	5000.0		10
10	TURNER	SALESMAN	5	1983-02-01	1500.0	0.0	30
11	ADAMS	CLERK	5	1992-07-03	1100.0		20
12	JAMES	CLERK	1	1996-09-10	950.0		30
13	FORD	ANALYST	1	1993-01-01	3000.0		20
14	Lee	CLERK	3	1983-10-09	1300.0		10
382	张三	SALESMAN	3	2014-09-07	4000.0	650.0	10

图-2

完整代码

web.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
    <display-name></display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <!-- Spring 前端控制器 -->
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>
                contextConfigLocation
            </param-name>
            <param-value>
                classpath:applicationContext.xml
            </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <!-- 使用 Filter 解决中文乱码问题 -->
    <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>
            org.springframework.web.filter.CharacterEncodingFilter
        </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>*.do</url-pattern>
    </filter-mapping>
</web-app>
```

applicationContext.xml 完整代码如下：

```
<?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"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
```



```

http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

```

```

<!-- 配置数据源 -->
<bean id="ds" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="username" value="lhh" />
    <property name="password" value="123456" />
</bean>

<!-- 配置 SqlSessionFactory -->
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="ds" />
    <property name="mapperLocations"
value="classpath:com/tarena/entity/*.xml" />
</bean>

<!-- 配置 MyBatis 注解 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.tarena.dao" />
    <property name="annotationClass"
value="com.tarena.annotation.MyBatisRepository" />
</bean>

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 开启 RequestMapping 注解 -->
<mvc:annotation-driven />

<!-- 处理请求转发 -->
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp" />
</bean>

</beans>

```

EmpController 完整代码如下：

```

package com.tarena.controller;

import java.util.List;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
public class EmpController {

```

```
@Resource
private EmpDao empDao;

@RequestMapping("/findEmp.do")
public String find(Model model) {
    List<Emp> list = empDao.findAll();
    model.addAttribute("emps", list);
    return "emp/emp_list";
}

}
```

emp_list.jsp 完整代码如下：

```
<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
</head>
<body>
<table width="60%" border="1" cellpadding="2" cellspacing="0">
    <tr>
        <th>EMPNO</th>
        <th>ENAME</th>
        <th>JOB</th>
        <th>MGR</th>
        <th>HIREDATE</th>
        <th>SAL</th>
        <th>COMM</th>
        <th>DEPTNO</th>
    </tr>
    <c:forEach items="${emps}" var="emp">
        <tr>
            <td>${emp.empno}</td>
            <td>${emp.ename}</td>
            <td>${emp.job}</td>
            <td>${emp.mgr}</td>
            <td>${emp.hiredate}</td>
            <td>${emp.sal}</td>
            <td>${emp.comm}</td>
            <td>${emp.deptno}</td>
        </tr>
    </c:forEach>
</table>
</body>
</html>
```

Spring + MyBatis

开发实战 Unit02

知识体系.....Page 23

MyBatis 动态 SQL	动态 SQL 简介	动态 SQL 简介
	进行判断	if 元素
		choose 元素
	拼关键字	where 元素
		set 元素
		trim 元素
	进行循环	foreach 元素


经典案例.....Page 28

if 元素使用案例	if 元素
choose 元素使用案例	choose 元素
where 元素使用案例	where 元素
set 元素使用案例	set 元素
trim 元素使用案例	trim 元素
foreach 元素使用案例	foreach 元素

1. MyBatis 动态 SQL


1.1. 动态 SQL 简介


1.1.1. 【动态 SQL 简介】动态 SQL 简介

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;">  </div> <h4 style="margin-top: 0;">动态SQL简介</h4> <ul style="list-style-type: none"> • 动态SQL是MyBatis框架中强大特性之一。在一些组合查询页面，需要根据用户输入的查询条件生成不同的查询SQL，这在JDBC或其他相似框架中需要在代码中拼写SQL，经常容易出错，但是在MyBatis中可以解决这种问题。 • 使用动态SQL元素与JSTL相似，它允许我们在XML中构建不同的SQL语句。常用的元素如下 <ul style="list-style-type: none"> – 判断元素：if, choose – 关键字元素：where, set, trim – 循环元素：foreach <div style="text-align: right; margin-top: 10px;">+</div>
---	--



1.2. 进行判断



1.2.1. 【进行判断】if 元素

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;">  </div> <h4 style="margin-top: 0;">if元素</h4> <ul style="list-style-type: none"> • if元素是简单的条件判断逻辑，满足指定条件时追加if元素内的SQL，不满足条件时不追加。使用格式如下 <pre style="margin-top: 10px;"> <select ...> SQL语句1 <if test="条件表达式"> SQL语句2 </if> </select> </pre> <div style="text-align: right; margin-top: 10px;">+</div>
---	---

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;">  </div> <h4 style="margin-top: 0;">if元素（续1）</h4> <ul style="list-style-type: none"> • if元素最常见的使用是在where子句部分，根据不同情况追加不同的SQL条件。示例代码如下 <pre style="margin-top: 10px;"> <select id="findByDeptNo" parameterType="Emp" resultType="Emp"> select * from EMP <if test="deptno != null"> where DEPTNO = #{deptno} </if> </select> </pre> <div style="text-align: right; margin-top: 10px;">+</div>
---	---



1.2.2. 【进行判断】choose 元素


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>choose元素</h3> <ul style="list-style-type: none"> choose元素的作用就相当于Java中的switch语句，基本上跟JSTL中的choose的作用和用法是一样的，通常与when和otherwise搭配使用。choose使用格式如下： <div style="border: 1px solid black; padding: 5px;"> <pre> <select ...> SQL语句1 <choose> <when test="条件表达式"> SQL语句2 </when> <otherwise> SQL语句3 </otherwise> </choose> </select> </pre> </div> <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>choose元素 (续1)</h3> <ul style="list-style-type: none"> choose元素的使用示例代码如下： <div style="border: 1px solid black; padding: 5px;"> <pre> <select id="findBySal" resultType="Emp" parameterType="Emp"> select * from EMP where <choose> <when test="sal > 2000"> SAL > #{sal} </when> <otherwise> SAL >= 2000 </otherwise> </choose> </select> </pre> </div> <div style="text-align: right;">  </div>
---	--


1.3. 拼关键字


1.3.1. 【拼关键字】where 元素

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>where元素</h3> <ul style="list-style-type: none"> where元素主要是用于简化查询语句中where部分的条件判断。where元素可以在<where>元素所在位置输出一个where关键字，而且还可以将后面条件多余的and或or关键字去除。 where使用格式如下： <div style="border: 1px solid black; padding: 5px;"> <pre> <select ...> select 字段 from 表 <where> 动态追加条件 </where> </select> </pre> </div> <div style="text-align: right;">  </div>
---	---



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>where元素 (续1)</h3> <ul style="list-style-type: none"> where元素的使用示例代码如下： <pre> <select id="findByCondition" resultType="Emp" parameterType="Emp"> select * from EMP <where> <if test="deptno != null"> DEPTNO = #{deptno} </if> <choose> <when test="sal > 2000"> and SAL >= #{sal} </when> <otherwise> and SAL >= 2000 </otherwise> </choose> </where> </select> </pre> <div style="position: absolute; left: 455px; top: 175px; writing-mode: vertical-rl; background-color: black; color: white; padding: 2px;">知识讲解</div> <div style="position: absolute; left: 455px; top: 280px;">+</div>
---	---



1.3.2. 【拼关键字】set 元素

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>set元素</h3> <ul style="list-style-type: none"> set元素主要是用在更新操作的时候，它的主要功能和where元素相似，主要是在<set>元素所在位置输出一个set关键字，而且还可以去除内容结尾中无关的逗号。有了set元素我们就可以动态的更新那些修改了的字段。 set使用格式如下： <pre> <update ...> update 表 <set> 动态追加更新字段 </set> </update> </pre> <div style="position: absolute; left: 455px; top: 415px; writing-mode: vertical-rl; background-color: black; color: white; padding: 2px;">知识讲解</div> <div style="position: absolute; left: 455px; top: 520px;">+</div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>set元素 (续1)</h3> <ul style="list-style-type: none"> set元素的使用示例代码如下： <pre> <update id="updateEmp" parameterType="Emp"> update EMP <set> <if test="ename != null"> ENAME = #{ename}, </if> <if test="sal != null"> SAL = #{sal}, </if> <if test="comm != null"> COMM = #{comm}, </if> <if test="deptno != null"> DEPTNO = #{deptno} </if> </set> where EMPNO=#{empno} </update> </pre> <div style="position: absolute; left: 455px; top: 650px; writing-mode: vertical-rl; background-color: black; color: white; padding: 2px;">知识讲解</div> <div style="position: absolute; left: 455px; top: 760px;">+</div>
---	---

1.3.3. 【拼关键字】trim 元素



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>trim元素</h4> <ul style="list-style-type: none"> trim元素的主要功能如下: <ul style="list-style-type: none"> 可以在自己包含的内容前加上某些前缀,也可以在其后加上某些后缀,与之对应的属性是prefix和suffix; 可以把包含内容的首部某些内容过滤,即忽略,也可以把尾部的某些内容过滤,对应的属性是prefixOverrides和suffixOverrides; 正因为trim有上述功能,所以我们可以非常简单的利用trim来代替where和set元素的功能 <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>trim元素 (续1)</h4> <ul style="list-style-type: none"> trim元素的应用示例如下: <pre> <!-- 等价于where元素 --> <trim prefix="WHERE" prefixOverrides="AND OR "> ... </trim> <!-- 等价于set元素 --> <trim prefix="SET" suffixOverrides=","> ... </trim> </pre> <div style="text-align: right;">  </div>
---	---

1.4. 进行循环

1.4.1. 【进行循环】foreach 元素

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>foreach 元素</h4> <ul style="list-style-type: none"> foreach元素实现了循环逻辑,可以进行一个集合的迭代,主要用在构建in条件中。foreach使用示例如下: <pre> <select ...> select 字段 from 表 where 字段 in <foreach collection="集合" item="迭代变量" open="{ " separator="," close="}"> #{迭代变量} </foreach> </select> </pre> foreach元素非常强大,它允许指定一个集合,声明集合项和索引变量,这些变量可以用在元素体内。它也允许指定开放和关闭的字符串,在迭代之间放置分隔符。 <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>foreach 元素 (续1)</h3> <ul style="list-style-type: none"> foreach元素的使用示例代码如下： <pre> <select id="findByDeptNos" resultType="Emp" parameterType="Emp" > select * from EMP <if test="deptnos != null"> where DEPTNO in <foreach collection="deptnos" item="dno" open="(" close=")" separator=","> #{dno} </foreach> </if> </select> </pre> <div style="text-align: right;">  </div>
---	--

经典案例

1. if 元素使用案例

- **问题**

使用 MyBatis 动态 SQL 的 if 元素，按部门做条件查询 EMP 员工信息表。

- **方案**

if 元素使用语法如下：

```
<select ...>
    SQL 语句 1
    <if test="条件表达式">
        SQL 语句 2
    </if>
</select>
```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：创建项目

复制项目 SpringUnit01，创建项目 SpringUnit02。

步骤二：增加根据部门查询员工的方法

创建封装查询条件的类 Condition，代码如下：

```
package com.tarena.entity;

import java.util.List;

public class Condition {

    private Integer deptno;

    private Double salary;

    private List<Integer> empnos;

    public Integer getDeptno() {
        return deptno;
    }

    public void setDeptno(Integer deptno) {
        this.deptno = deptno;
    }

    public Double getSalary() {
        return salary;
    }
}
```

```
public void setSalary(Double salary) {
    this.salary = salary;
}

public List<Integer> getEmpnos() {
    return empnos;
}

public void setEmpnos(List<Integer> empnos) {
    this.empnos = empnos;
}
}
```

在 EmpDao 接口中增加方法 findByDept，代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

}
```

步骤三：实现根据部门查询员工

在 EmpMapper.xml 中增加根据部门查询员工的 SQL，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">
<!-- 其他配置略 -->

<!-- if -->
<!-- 查询部门下的所有员工 -->
<select id="findByDept"
    parameterType="com.tarena.entity.Condition"
    resultType="com.tarena.entity.Emp">
    select * from t_emp
    <if test="deptno != null">
        where deptno=#{deptno}
    </if>
</select>

</mapper>
```

步骤四：测试根据部门查询员工的方法

在 TestEmpDao 中，增加测试方法 testFindByDept，代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    //其他方法略

    /**
     * 根据部门查询员工
     */
    @Test
    public void testFindByDept() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setDeptno(10);
        List<Emp> list = dao.findByDept(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}
```

- 完整代码

Condition 完整代码如下：

```
package com.tarena.entity;
import java.util.List;

public class Condition {

    private Integer deptno;
    private Double salary;
    private List<Integer> empnos;

    public Integer getDeptno() {
        return deptno;
    }
}
```

```
public void setDeptno(Integer deptno) {
    this.deptno = deptno;
}

public Double getSalary() {
    return salary;
}

public void setSalary(Double salary) {
    this.salary = salary;
}

public List<Integer> getEmpnos() {
    return empnos;
}

public void setEmpnos(List<Integer> empnos) {
    this.empnos = empnos;
}
}
```

EmpDao 完整代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org/DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

    <!-- if -->
    <!-- 查询部门下的所有员工 -->
    <select id="findByDept"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
```

```
<if test="deptno != null">
    where deptno=#{deptno}
</if>
</select>

</mapper>
```

TestEmpDao 完整代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Emp> list = dao.findAll();
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 根据部门查询员工
     */
    @Test
    public void testFindByDept() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setDeptno(10);
        List<Emp> list = dao.findByDept(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}
```

2. choose 元素使用案例

- **问题**

使用 MyBatis 动态 SQL 的 choose 元素，按工资做条件查询 EMP 员工信息表。

- **方案**

choose 元素使用语法如下：

```
<select ...>
    SQL 语句 1
    <choose>
        <when test="条件表达式">
            SQL 语句 2
        </when>
        <otherwise>
            SQL 语句 3
        </otherwise>
    </choose>
</select>
```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：增加根据工资查询员工的方法

在 EmpDao 中增加方法 findBySalary，代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

}
```

步骤二：实现根据工资查询员工

在 EmpMapper.xml 中增加根据工资查询员工的 SQL，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

<!-- 其他配置略 -->

<!-- choose -->
<!-- 查询大于当前收入的员工 -->
<select id="findBySalary"
    parameterType="com.tarena.entity.Condition"
    resultType="com.tarena.entity.Emp">
    select * from t_emp
    <choose>
        <when test="salary > 3000">
            where sal>#{salary}
        </when>
        <otherwise>
            where sal>=3000
        </otherwise>
    </choose>
</select>

</mapper>
```

步骤三：测试根据工资查询员工的方法

在 TestEmpDao 中增加方法 testFindBySalary，代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    //其他方法略

    /**
     * 查询大于当前收入的员工
     */
    @Test
    public void testFindBySalary() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
```

```

        cond.setSalary(4000.0);
        List<Emp> list = dao.findBySalary(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}

```

• 完整代码

EmpDao 完整代码如下：

```

package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

}

```

EmpMapper.xml 完整代码如下：

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">
    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

    <!-- if -->
    <!-- 查询部门下的所有员工 -->
    <select id="findByDept"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <if test="deptno != null">
            where deptno=#{deptno}
        </if>
    </select>

```



```
<!-- choose -->
<!-- 查询大于当前收入的员工 -->
<select id="findBySalary"
    parameterType="com.tarena.entity.Condition"
    resultType="com.tarena.entity.Emp">
    select * from t_emp
    <choose>
        <when test="salary > 3000">
            where sal>#{salary}
        </when>
        <otherwise>
            where sal>=3000
        </otherwise>
    </choose>
</select>

</mapper>
```

TestEmpDao 完整代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Emp> list = dao.findAll();
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 根据部门查询员工
     */
    @Test
    public void testFindByDept() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setDeptno(10);
    }
}
```

```

        List<Emp> list = dao.findByDept(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 查询大于当前收入的员工
     */
    @Test
    public void testFindBySalary() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setSalary(4000.0);
        List<Emp> list = dao.findBySalary(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}

```

3. where 元素使用案例

- 问题

使用 MyBatis 动态 SQL 的 where 元素，查询当前部门下大于指定工资的员工。

- 方案

where 元素的使用语法如下：

```

<select ...>
    select 字段 from 表
    <where>
        动态追加条件
    </where>
</select>

```

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加根据部门和工资查询员工的方法

在 EmpDao 中增加方法 findByDeptAndSalary，代码如下：

```
package com.tarena.dao;
```

```
import java.util.List;
import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

}
```

步骤二：实现根据部门和工资查询员工的方法

在 EmpMapper.xml 中增加根据部门和工资查询员工的 SQL，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">
    <!-- 其他配置略 -->

    <!-- where -->
    <!-- 查询当前部门下,大于当前收入的员工 -->
    <select id="findByDeptAndSalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <where>
            <if test="deptno != null">
                and deptno=#{deptno}
            </if>
            <if test="salary != null">
                and sal>#{salary}
            </if>
        </where>
    </select>

</mapper>
```

步骤三：测试根据部门和工资查询员工的方法

在 TestEmpDao 中，增加测试方法 testFindByDeptAndSalary，代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;
```

//其他方法略

```
/**
 * 查询当前部门下,大于当前收入的员工
 */
@Test
public void testFindByDeptAndSalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);
    cond.setSalary(2000.0);
    List<Emp> list = dao.findByDeptAndSalary(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}
```

• 完整代码

EmpDao 完整代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

    <!-- if -->
    <!-- 查询部门下的所有员工 -->
    <select id="findByDept"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <if test="deptno != null">
            where deptno=#{deptno}
        </if>
    </select>

    <!-- choose -->
    <!-- 查询大于当前收入的员工 -->
    <select id="findBySalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <choose>
            <when test="salary > 3000">
                where sal>#{salary}
            </when>
            <otherwise>
                where sal>=3000
            </otherwise>
        </choose>
    </select>

    <!-- where -->
    <!-- 查询当前部门下,大于当前收入的员工 -->
    <select id="findByDeptAndSalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <where>
            <if test="deptno != null">
                and deptno=#{deptno}
            </if>
            <if test="salary != null">
                and sal>#{salary}
            </if>
        </where>
    </select>

</mapper>
```

TestEmpDao 完整代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Emp> list = dao.findAll();
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 根据部门查询员工
     */
    @Test
    public void testFindByDept() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setDeptno(10);
        List<Emp> list = dao.findByDept(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 查询大于当前收入的员工
     */
    @Test
    public void testFindBySalary() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setSalary(4000.0);
        List<Emp> list = dao.findBySalary(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}
```

```
    }  
}  
  
/**  
 * 查询当前部门下,大于当前收入的员工  
 */  
@Test  
public void testFindByDeptAndSalary() {  
    ApplicationContext ctx =  
        new ClassPathXmlApplicationContext("applicationContext.xml");  
    EmpDao dao = ctx.getBean(EmpDao.class);  
    Condition cond = new Condition();  
    cond.setDeptno(20);  
    cond.setSalary(2000.0);  
    List<Emp> list = dao.findByDeptAndSalary(cond);  
    for(Emp e : list) {  
        System.out.println(  
            e.getEmpno() + " " +  
            e.getEname() + " " +  
            e.getJob()  
        );  
    }  
}
```

4. set 元素使用案例

- 问题

使用 MyBatis 动态 SQL 的 set 元素，实现更新员工。

- 方案

set 元素使用语法如下：

```
<update ...>  
    update 表  
    <set>  
        动态追加更新字段  
    </set>  
</update>
```

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加更新员工的方法

在 EmpDao 中增加更新员工的方法，代码如下：

```
package com.tarena.dao;  
  
import java.util.List;  
  
import com.tarena.annotation.MyBatisRepository;  
import com.tarena.entity.Condition;  
import com.tarena.entity.Emp;
```

```
/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

    void update(Emp emp);

}
```

步骤二：实现更新员工的方法

在 EmpMapper.xml 中增加更新员工的 SQL，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 其他配置略 -->

    <!-- set -->
    <!-- 更新员工 -->
    <update id="update"
        parameterType="com.tarena.entity.Emp">
        update t_emp
        <set>
            <if test="ename!=null">
                ename=#{ename},
            </if>
            <if test="job!=null">
                job=#{job},
            </if>
        </set>
        where empno=#{empno}
    </update>

</mapper>
```

步骤三：测试更新员工的方法

在 TestEmpDao 中增加测试更新员工的方法，代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```



```
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    //其他方法略

    /**
     * 更新员工
     */
    @Test
    public void testUpdate() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = new Emp();
        e.setEmpno(14);
        e.setName("Leo");
        dao.update(e);
    }

}
```

- **完整代码**

EmpDao 完整代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

    void update(Emp emp);

}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
"http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

<!-- 查询全部的员工 -->
<select id="findAll"
        resultType="com.tarena.entity.Emp">
    select * from t_emp
</select>

<!-- if -->
<!-- 查询部门下的所有员工 -->
<select id="findByDept"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
    select * from t_emp
    <if test="deptno != null">
        where deptno=#{deptno}
    </if>
</select>

<!-- choose -->
<!-- 查询大于当前收入的员工 -->
<select id="findBySalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
    select * from t_emp
    <choose>
        <when test="salary > 3000">
            where sal>#{salary}
        </when>
        <otherwise>
            where sal>=3000
        </otherwise>
    </choose>
</select>

<!-- where -->
<!-- 查询当前部门下,大于当前收入的员工 -->
<select id="findByDeptAndSalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
    select * from t_emp
    <where>
        <if test="deptno != null">
            and deptno=#{deptno}
        </if>
        <if test="salary != null">
            and sal>#{salary}
        </if>
    </where>
</select>

<!-- set -->
<!-- 更新员工 -->
<update id="update"
        parameterType="com.tarena.entity.Emp">
    update t_emp
    <set>
        <if test="ename!=null">
            ename=#{ename},
        </if>
        <if test="job!=null">
            job=#{job},
        </if>
    </set>
```

```
        where empno=#{empno}
    </update>

</mapper>
```

TestEmpDao 完整代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Emp> list = dao.findAll();
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 根据部门查询员工
     */
    @Test
    public void testFindByDept() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Condition cond = new Condition();
        cond.setDeptno(10);
        List<Emp> list = dao.findByDept(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

    /**
     * 查询大于当前收入的员工
     */
}
```

```

@Test
public void testFindBySalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setSalary(4000.0);
    List<Emp> list = dao.findBySalary(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 查询当前部门下,大于当前收入的员工
 */
@Test
public void testFindByDeptAndSalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);
    cond.setSalary(2000.0);
    List<Emp> list = dao.findByDeptAndSalary(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 更新员工
 */
@Test
public void testUpdate() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setEmpno(14);
    e.setEname("Leo");
    dao.update(e);
}
}

```

5. trim 元素使用案例

- 问题

使用 MyBatis 动态 SQL 的 trim 元素代替 where 和 set ,重写 findByDeptAndSalary 和 update 方法。

- 方案

trim 元素使用语法如下：

```
<!-- 等价于 where 元素 -->
<trim prefix="WHERE" prefixOverrides="AND |OR ">
...
</trim>
<!-- 等价于 set 元素 -->
<trim prefix="SET" suffixOverrides=",">
...
</trim>
```

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：重新增加根据部门和工资查询员工的方法，以及更新员工的方法

在 EmpDao 中增加 findByDeptAndSalary2 和 update2 方法，代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

    void update(Emp emp);

    List<Emp> findByDeptAndSalary2(Condition cond);

    void update2(Emp emp);

}
```

步骤二：重新实现根据部门和工资查询员工的方法，以及更新员工的方法

在 EmpMapper.xml 中，实现 findByDeptAndSalary2 和 update2，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">
```

```
<!-- 其他配置略 -->

<!-- 使用 trim 代替 where -->
<!-- 查询当前部门下,大于当前收入的员工 -->
<select id="findByDeptAndSalary2"
    parameterType="com.tarena.entity.Condition"
    resultType="com.tarena.entity.Emp">
    select * from t_emp
    <trim prefix="where" prefixOverrides="and">
        <if test="deptno != null">
            and deptno=#{deptno}
        </if>
        <if test="salary != null">
            and sal>#{salary}
        </if>
    </trim>
</select>

<!-- 使用 trim 代替 set -->
<update id="update2"
    parameterType="com.tarena.entity.Emp">
    update t_emp
    <trim prefix="set" suffixOverrides=",">
        <if test="ename!=null">
            ename=#{ename},
        </if>
        <if test="job!=null">
            job=#{job},
        </if>
    </trim>
    where empno=#{empno}
</update>

</mapper>
```

步骤三：重新测试根据部门和工资查询员工的方法，以及更新员工的方法

在 TestEmpDao 中，增加测试 findByDeptAndSalary2 和 update2 方法，代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {
    //其他方法略

    /**
     * 查询当前部门下,大于当前收入的员工
     */
}
```

```

@Test
public void testFindByDeptAndSalary2() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);
    cond.setSalary(2000.0);
    List<Emp> list = dao.findByDeptAndSalary2(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getName() + " " +
            e.getJob()
        );
    }
}

/**
 * 更新员工
 */
@Test
public void testUpdate2() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setEmpno(14);
    e.setName("Lee");
    dao.update2(e);
}
}

```

• 完整代码

EmpDao 完整代码如下：

```

package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

    void update(Emp emp);

    List<Emp> findByDeptAndSalary2(Condition cond);
}

```

```
void update2(Emp emp);

}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
"http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

    <!-- if -->
    <!-- 查询部门下的所有员工 -->
    <select id="findByDept"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <if test="deptno != null">
            where deptno=#{deptno}
        </if>
    </select>

    <!-- choose -->
    <!-- 查询大于当前收入的员工 -->
    <select id="findBySalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <choose>
            <when test="salary > 3000">
                where sal>#{salary}
            </when>
            <otherwise>
                where sal>=3000
            </otherwise>
        </choose>
    </select>

    <!-- where -->
    <!-- 查询当前部门下,大于当前收入的员工 -->
    <select id="findByDeptAndSalary"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
        <where>
            <if test="deptno != null">
                and deptno=#{deptno}
            </if>
            <if test="salary != null">
                and sal>#{salary}
            </if>
        </where>
    </select>

    <!-- set -->
    <!-- 更新员工 -->
    <update id="update"
        parameterType="com.tarena.entity.Emp">
```



```
update t emp
<set>
    <if test="ename!=null">
        ename=#{ename},
    </if>
    <if test="job!=null">
        job=#{job},
    </if>
</set>
where empno=#{empno}
</update>

<!-- 使用 trim 代替 where -->
<!-- 查询当前部门下,大于当前收入的员工 -->
<select id="findByDeptAndSalary2"
    parameterType="com.tarena.entity.Condition"
    resultType="com.tarena.entity.Emp">
    select * from t_emp
    <trim prefix="where" prefixOverrides="and">
        <if test="deptno != null">
            and deptno=#{deptno}
        </if>
        <if test="salary != null">
            and sal>#{salary}
        </if>
    </trim>
</select>

<!-- 使用 trim 代替 set -->
<update id="update2"
    parameterType="com.tarena.entity.Emp">
    update t_emp
    <trim prefix="set" suffixOverrides=",">
        <if test="ename!=null">
            ename=#{ename},
        </if>
        <if test="job!=null">
            job=#{job},
        </if>
    </trim>
    where empno=#{empno}
</update>

</mapper>
```

TestEmpDao 完整代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
}
```

```

*/
@Test
public void testFindAll() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext(
            "applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    List<Emp> list = dao.findAll();
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 根据部门查询员工
 */
@Test
public void testFindByDept() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(10);
    List<Emp> list = dao.findByDept(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 查询大于当前收入的员工
 */
@Test
public void testFindBySalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setSalary(4000.0);
    List<Emp> list = dao.findBySalary(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 查询当前部门下,大于当前收入的员工
 */
@Test
public void testFindByDeptAndSalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);

```

```
cond.setSalary(2000.0);
List<Emp> list = dao.findByDeptAndSalary(cond);
for(Emp e : list) {
    System.out.println(
        e.getEmpno() + " " +
        e.getEname() + " " +
        e.getJob()
    );
}

/**
 * 更新员工
 */
@Test
public void testUpdate() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setEmpno(14);
    e.setEname("Leo");
    dao.update(e);
}

/**
 * 查询当前部门下,大于当前收入的员工
 */
@Test
public void testFindByDeptAndSalary2() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);
    cond.setSalary(2000.0);
    List<Emp> list = dao.findByDeptAndSalary2(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 更新员工
 */
@Test
public void testUpdate2() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setEmpno(14);
    e.setEname("Lee");
    dao.update2(e);
}
}
```

6. foreach 元素使用案例

- 问题

使用 MyBatis 动态 SQL 的 foreach 元素，实现根据一组员工 ID 查询员工。

- 方案

foreach 元素使用语法如下：

```
<select ...>
    select 字段 from 表 where 字段 in
    <foreach collection="集合" item="迭代变量"
        open="(" separator="," close=")" ">
        #{迭代变量}
    </foreach>
</select>
```

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加根据一组员工 ID 查询员工的方法

在 EmpDao 中增加根据一组员工 ID 查询员工的方法，代码如下：

```
package com.tarena.dao;

import java.util.List;
import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

    void update(Emp emp);

    List<Emp> findByDeptAndSalary2(Condition cond);

    void update2(Emp emp);

    List<Emp> findByIds(Condition cond);

}
```

步骤二：实现根据一组员工 ID 查询员工的方法

在 EmpMapper.xml 中增加根据一组员工 ID 查询员工的 SQL，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

<!-- 其他配置略 -->

<!-- foreach -->
<!-- 根据 ID 查询员工 -->
<select id="findByIds"
    parameterType="com.tarena.entity.Condition"
    resultType="com.tarena.entity.Emp">
    select * from t_emp where empno in
    <foreach collection="empnos"
        open="(" close=")" separator="," item="id">
        #{id}
    </foreach>
</select>

</mapper>
```

步骤三：测试根据一组员工 ID 查询员工的方法

在 TestEmpDao 中增加测试 findByIds 的方法，代码如下：

```
package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    //其他方法略

    /**
     * 根据员工 ID 查询员工
     */
    @Test
    public void testFindByIds() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        List<Integer> ids = new ArrayList<Integer>();
        ids.add(3);
        ids.add(7);
    }
}
```

```

        ids.add(8);
        Condition cond = new Condition();
        cond.setEmpnos(ids);
        List<Emp> list = dao.findByIds(cond);
        for(Emp e : list) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}

```

• 完整代码

EmpDao 完整代码如下：

```

package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    List<Emp> findByDept(Condition cond);

    List<Emp> findBySalary(Condition cond);

    List<Emp> findByDeptAndSalary(Condition cond);

    void update(Emp emp);

    List<Emp> findByDeptAndSalary2(Condition cond);

    void update2(Emp emp);

    List<Emp> findByIds(Condition cond);

}

```

EmpMapper.xml 完整代码如下：

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">

```

```
select * from t emp
</select>

<!-- if -->
<!-- 查询部门下的所有员工 -->
<select id="findByDept"
  parameterType="com.tarena.entity.Condition"
  resultType="com.tarena.entity.Emp">
  select * from t emp
  <if test="deptno != null">
    where deptno=#{deptno}
  </if>
</select>

<!-- choose -->
<!-- 查询大于当前收入的员工 -->
<select id="findBySalary"
  parameterType="com.tarena.entity.Condition"
  resultType="com.tarena.entity.Emp">
  select * from t emp
  <choose>
    <when test="salary > 3000">
      where sal>#{salary}
    </when>
    <otherwise>
      where sal>=3000
    </otherwise>
  </choose>
</select>

<!-- where -->
<!-- 查询当前部门下,大于当前收入的员工 -->
<select id="findByDeptAndSalary"
  parameterType="com.tarena.entity.Condition"
  resultType="com.tarena.entity.Emp">
  select * from t emp
  <where>
    <if test="deptno != null">
      and deptno=#{deptno}
    </if>
    <if test="salary != null">
      and sal>#{salary}
    </if>
  </where>
</select>

<!-- set -->
<!-- 更新员工 -->
<update id="update"
  parameterType="com.tarena.entity.Emp">
  update t_emp
  <set>
    <if test="ename!=null">
      ename=#{ename},
    </if>
    <if test="job!=null">
      job=#{job},
    </if>
  </set>
  where empno=#{empno}
</update>

<!-- 使用 trim 代替 where -->
<!-- 查询当前部门下,大于当前收入的员工 -->
<select id="findByDeptAndSalary2"
  parameterType="com.tarena.entity.Condition"
  resultType="com.tarena.entity.Emp">
```

```

        select * from t_emp
        <trim prefix="where" prefixOverrides="and">
            <if test="deptno != null">
                and deptno=#{deptno}
            </if>
            <if test="salary != null">
                and sal>#{salary}
            </if>
        </trim>
    </select>

    <!-- 使用 trim 代替 set -->
    <update id="update2"
        parameterType="com.tarena.entity.Emp">
        update t_emp
        <trim prefix="set" suffixOverrides=",">
            <if test="ename!=null">
                ename=#{ename},
            </if>
            <if test="job!=null">
                job=#{job},
            </if>
        </trim>
        where empno=#{empno}
    </update>

    <!-- foreach -->
    <!-- 根据 ID 查询员工 -->
    <select id="findByIds"
        parameterType="com.tarena.entity.Condition"
        resultType="com.tarena.entity.Emp">
        select * from t_emp where empno in
        <foreach collection="empnos"
            open="(" close=")" separator="," item="id">
            #{id}
        </foreach>
    </select>

</mapper>

```

TestEmpDao 完整代码如下：

```

package com.tarena.test;

import java.util.ArrayList;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Condition;
import com.tarena.entity.Emp;

/**
 * EmpDao 测试类
 */
public class TestEmpDao {

    /**
     * 测试查询全部员工
     */
    @Test
    public void testFindAll() {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext(

```



```
        "applicationContext.xml");
EmpDao dao = ctx.getBean(EmpDao.class);
List<Emp> list = dao.findAll();
for(Emp e : list) {
    System.out.println(
        e.getEmpno() + " " +
        e.getEname() + " " +
        e.getJob()
    );
}
}

/**
 * 根据部门查询员工
 */
@Test
public void testFindByDept() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(10);
    List<Emp> list = dao.findByDept(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 查询大于当前收入的员工
 */
@Test
public void testFindBySalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setSalary(4000.0);
    List<Emp> list = dao.findBySalary(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 查询当前部门下,大于当前收入的员工
 */
@Test
public void testFindByDeptAndSalary() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);
    cond.setSalary(2000.0);
    List<Emp> list = dao.findByDeptAndSalary(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
```

```

        e.getEname() + " " +
        e.getJob()
    );
    }
}

/**
 * 更新员工
 */
@Test
public void testUpdate() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setEmpno(14);
    e.setEname("Leo");
    dao.update(e);
}

/**
 * 查询当前部门下,大于当前收入的员工
 */
@Test
public void testFindByDeptAndSalary2() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Condition cond = new Condition();
    cond.setDeptno(20);
    cond.setSalary(2000.0);
    List<Emp> list = dao.findByDeptAndSalary2(cond);
    for(Emp e : list) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 更新员工
 */
@Test
public void testUpdate2() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setEmpno(14);
    e.setEname("Lee");
    dao.update2(e);
}

/**
 * 根据员工 ID 查询员工
 */
@Test
public void testFindByIds() {
    ApplicationContext ctx =
        new ClassPathXmlApplicationContext("applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    List<Integer> ids = new ArrayList<Integer>();
    ids.add(3);
    ids.add(7);
    ids.add(8);
}

```

```
Condition cond = new Condition();
cond.setEmpnos(ids);
List<Emp> list = dao.findByIds(cond);
for(Emp e : list) {
    System.out.println(
        e.getEmpno() + " " +
        e.getEname() + " " +
        e.getJob()
    );
}
}
}
```

Spring + MyBatis

开发实战 Unit03

知识体系.....Page 64

Spring 与 Ajax	Spring 与 Ajax	Ajax 简介
		Spring 对 Ajax 的支持
		@ResponseBody 应用

经典案例.....Page 67

Spring 与 Ajax 应用案例	@ResponseBody 应用
--------------------	------------------

1. Spring 与 Ajax

1.1. Spring 与 Ajax

1.1.1. 【Spring 与 Ajax】Ajax 简介


<div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div><div>Tarena 达内科技</div><h3>Ajax简介</h3><ul style="list-style-type: none">• Asynchronous JavaScript and Xml 异步的JavaScript和Xml• AJAX是一种用来改善用户体验的技术，具有异步处理和刷新部分页面内容的特点<div>返回并继续</div><div>++</div></div>
---	--


<div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div><div>Tarena 达内科技</div><h3>Ajax简介（续1）</h3><ul style="list-style-type: none">• Ajax技术在应用时，需要客户端JavaScript处理和服务端处理两部分。• 使用Ajax技术时，需要以下工作<ul style="list-style-type: none">– 编写JavaScript程序，发送Ajax请求– 编写服务器处理，返回处理结果（常用json格式）– 编写JavaScript程序，获取服务器结果，刷新部分页面<div>返回并继续</div><div>++</div></div>
---	--


1.1.2. 【Spring 与 Ajax】Spring 对 Ajax 的支持

<div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div><div>Tarena 达内科技</div><h3>Spring对Ajax的支持</h3><ul style="list-style-type: none">• 为了便于接收和处理Ajax请求，Spring MVC提供了JSON响应的支持，可以很方便地将数据自动转成JSON格式字符串给客户端JavaScript返回。• 在Spring MVC中，与JSON响应相关的注解为 @ResponseBody<div>返回并继续</div><div>++</div></div>
---	--

1.1.3. 【Spring 与 Ajax】@ResponseBody 应用

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>@ResponseBody应用</h4> <ul style="list-style-type: none"> • @ResponseBody注解主要用于Controller组件的处理方法前，具体使用方法如下： <ul style="list-style-type: none"> – 引入jackson开发包，后面示例采用的是jackson-annotations-2.4.1.jar、jackson-core-2.4.1.jar、jackson-databind-2.4.1.jar – 在Spring配置文件中定义<mvc:annotation-driven />，开启对@ResponseBody应用的支持 – 在Controller处理方法前定义@ResponseBody注解 <div style="text-align: right;">+</div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>@ResponseBody应用（续1）</h4> <ul style="list-style-type: none"> • 使用@ResponseBody，返回单个值的示例如下： <pre> @RequestMapping("/test1") @ResponseBody public boolean f1(){ //业务处理代码 return true; } </pre> <div style="text-align: right;">+</div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>@ResponseBody应用（续2）</h4> <ul style="list-style-type: none"> • 使用@ResponseBody，返回多个值的示例如下： <pre> @RequestMapping("/test2") @ResponseBody public Map<String, Object> f2(){ Map<String, Object> data = new HashMap<String, Object>(); data.put("id", 1001); data.put("name", "tom"); return data; } </pre> <div style="text-align: right;">+</div>
---	--


代码清单

@ResponseBody应用（续3）

• 使用@ResponseBody，返回List结果的示例如下：

```

@RequestMapping("/test3")
@ResponseBody
public List<String> f3(){
    List<String> list = new ArrayList<String>();
    list.add("spring");
    list.add("mybatis");
    list.add("struts");
    return list;
}
                    
```



++


代码清单

@ResponseBody应用（续4）

• 使用@ResponseBody，返回对象结果的示例如下：

```

@RequestMapping("/test4")
@ResponseBody
public Object f4(){
    Page page = new Page();
    return page;
}
                    
```



++

代码清单

@ResponseBody应用（续5）


• 使用@ResponseBody，返回对象集合结果的示例如下：

```

@RequestMapping("/test5")
@ResponseBody
public Object f5(){
    Cost c1 = new Cost();
    c1.setid(1);
    c1.setName("套餐");

    Cost c2 = new Cost();
    c2.setid(2);
    c2.setName("包月");

    List<Cost> list = new ArrayList<Cost>();
    list.add(c1);
    list.add(c2);
    return list;
}
                    
```



++

66

经典案例

1. Spring 与 Ajax 应用案例

- 问题

在 Spring 中使用@ResponseBody 注解标注业务方法，将业务方法的返回值做成 json 输出给页面。

- 方案

@ResponseBody 注解使用步骤

- 1) 导包
- 2) 开启@ResponseBody 注解
- 3) 使用@ResponseBody 标注业务方法

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：导包

复制项目 SpringUnit02 ,创建新项目 SpringUnit03 然后在新项目中导入如下的包：

- 1) jackson-annotations-2.4.1.jar
- 2) jackson-core-2.4.1.jar
- 3) jackson-databind-2.4.1.jar

完成之后，项目包结构如下图：



图-1

步骤二：开启@ResponseBody 注解

在 applicationContext.xml 中通过 <mvc:annotation-driven /> 开启 @ResponseBody 注解，由于在配置 Spring MVC 时已经增加了这段配置，因此这个步骤可以省略了。

步骤三：使用@ResponseBody 标注业务方法

创建业务控制器 JsonController，用于演示@ResponseBody 的多种用法，该注解可以将如下类型的数据做成 json：

- 1) 基本类型数据，如 boolean,String,int 等
- 2) Map 类型数据
- 3) 集合或数组
- 4) 实体对象
- 5) 实体对象集合

JsonController 代码如下：

```
package com.tarena.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
```

```
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/test")
public class JsonController {

    @RequestMapping("/test1.do")
    @ResponseBody
    public boolean test1() {
        return true;
    }

    @RequestMapping("/test2.do")
    @ResponseBody
    public Map<String, Object> test2() {
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("id", 16);
        map.put("name", "刘备");
        return map;
    }

    @RequestMapping("/test3.do")
    @ResponseBody
    public List<String> test3() {
        List<String> list = new ArrayList<String>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
        return list;
    }

    @RequestMapping("/test4.do")
    @ResponseBody
    public Emp test4() {
        Emp e = new Emp();
        e.setEmpno(1);
        e.setEname("刘苍松");
        e.setJob("老师");
        e.setMgr(1);
        e.setSal(10000.0);
        e.setDeptno(30);
        return e;
    }

    @RequestMapping("/test5.do")
    @ResponseBody
    public List<Emp> test5() {
        List<Emp> list = new ArrayList<Emp>();

        Emp e1 = new Emp();
        e1.setEmpno(1);
        e1.setEname("aaa");
        e1.setJob("aaa");
        e1.setMgr(1);
        e1.setSal(10000.0);
        e1.setDeptno(30);
        list.add(e1);

        Emp e2 = new Emp();
        e2.setEmpno(1);
        e2.setEname("bbb");
        e2.setJob("bbb");
        e2.setMgr(1);
        e2.setSal(20000.0);
        e2.setDeptno(30);
        list.add(e2);
    }
}
```

```

        Emp e3 = new Emp();
        e3.setEmpno(1);
        e3.setEname("ccc");
        e3.setJob("ccc");
        e3.setMgr(1);
        e3.setSal(30000.0);
        e3.setDeptno(30);
        list.add(e3);

        return list;
    }
}

```

步骤四：测试

部署项目并启动 tomcat，打开浏览器，依次访问 JsonResult 中的业务方法，效果如下图：

访问 test1.do 效果如下图：

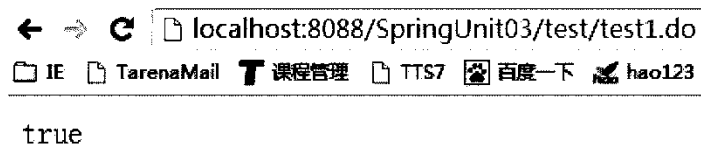


图-2

访问 test2.do 效果如下图：

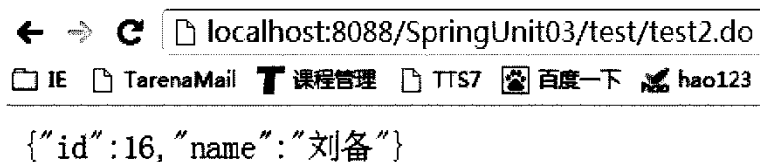


图-3

访问 test3.do 效果如下图：

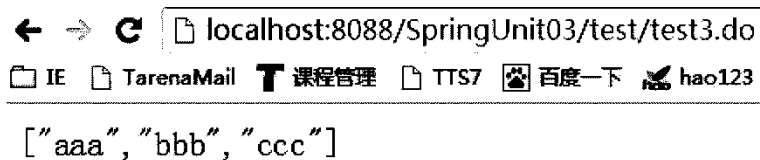


图-4

访问 test4.do 效果如下图：

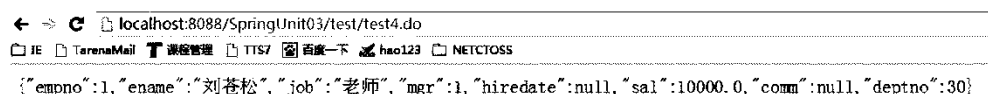
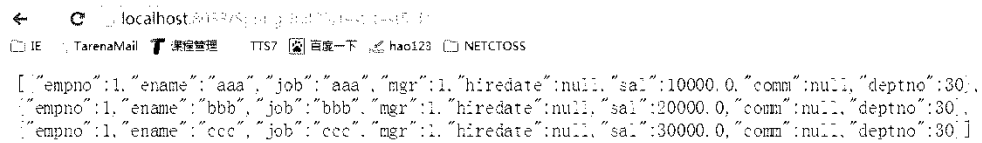


图-5

访问 test5.do 效果如下图：



```

[{"empno":1,"ename":"aaa","job":"aaa","mgr":1,"hiredate":null,"sal":10000.0,"comm":null,"deptno":30},
{"empno":1,"ename":"bbb","job":"bbb","mgr":1,"hiredate":null,"sal":20000.0,"comm":null,"deptno":30},
{"empno":1,"ename":"ccc","job":"ccc","mgr":1,"hiredate":null,"sal":30000.0,"comm":null,"deptno":30}]

```

图-6

• 完整代码

本案例的完整代码如下所示：

JsonController 完整代码如下：

```

package com.tarena.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/test")
public class JsonController {

    @RequestMapping("/test1.do")
    @ResponseBody
    public boolean test1() {
        return true;
    }

    @RequestMapping("/test2.do")
    @ResponseBody
    public Map<String, Object> test2() {
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("id", 16);
        map.put("name", "刘备");
        return map;
    }

    @RequestMapping("/test3.do")
    @ResponseBody
    public List<String> test3() {
        List<String> list = new ArrayList<String>();
        list.add("aaa");
        list.add("bbb");
        list.add("ccc");
        return list;
    }

    @RequestMapping("/test4.do")
    @ResponseBody
    public Emp test4() {
        Emp e = new Emp();
        e.setEmpno(1);
        e.setEname("刘苍松");
        e.setJob("老师");
    }
}

```

```
e.setMgr(1);
e.setSal(10000.0);
e.setDeptno(30);
return e;
}

@RequestMapping("/test5.do")
@ResponseBody
public List<Emp> test5() {
    List<Emp> list = new ArrayList<Emp>();

    Emp e1 = new Emp();
    e1.setEmpno(1);
    e1.setEname("aaa");
    e1.setJob("aaa");
    e1.setMgr(1);
    e1.setSal(10000.0);
    e1.setDeptno(30);
    list.add(e1);

    Emp e2 = new Emp();
    e2.setEmpno(1);
    e2.setEname("bbb");
    e2.setJob("bbb");
    e2.setMgr(1);
    e2.setSal(20000.0);
    e2.setDeptno(30);
    list.add(e2);

    Emp e3 = new Emp();
    e3.setEmpno(1);
    e3.setEname("ccc");
    e3.setJob("ccc");
    e3.setMgr(1);
    e3.setSal(30000.0);
    e3.setDeptno(30);
    list.add(e3);

    return list;
}
}
```

Spring + MyBatis

开发实战 Unit04

知识体系.....Page 74

MyBatis 关联映射	主键映射	主键映射作用
		自动递增
		非自动递增
	关联映射	关联映射作用
		嵌套查询映射
		嵌套结果映射
	集合映射	集合映射作用
		嵌套查询映射
		嵌套结果映射
	鉴别器	鉴别器的作用
		鉴别器的使用
		嵌套结果映射


经典案例.....Page 81

主键映射使用案例	非自动递增
多对一嵌套查询映射使用案例	多对一嵌套查询映射使用案例
多对一嵌套结果映射使用案例	多对一嵌套结果映射使用案例
一对多嵌套查询映射使用案例	嵌套查询映射
一对多嵌套结果映射使用案例	嵌套结果映射
鉴别器使用案例	鉴别器使用案例


1. MyBatis 关联映射


1.1. 主键映射

1.1.1. 【主键映射】主键映射作用



<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">主键映射作用</h4> <ul style="list-style-type: none"> • 在使用MyBatis做插入操作时，可以由MyBatis负责主键生成，主键字段部分的映射可以分成以下两种情况。 <ul style="list-style-type: none"> – 数据库支持自动递增，例如MySQL、SQLServer – 数据库不支持自动递增，例如Oracle <div style="text-align: right;">+</div>
---	--



1.1.2. 【主键映射】自动递增

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">自动递增</h4> <ul style="list-style-type: none"> • MySQL、SQLServer数据库都支持字段自动递增功能，在设计表时，可以为关键字段指定自动递增。这样在添加操作时，主键值数据库会自动生成，不需要指定。 • 在使用MyBatis时，有时需要返回MySQL和SQLServer数据库自动增长的主键值，可以采用下面方法定义： <pre><insert id="addDept" parameterType="Dept" keyProperty="deptno" useGeneratedKeys="true" > insert into T_DEPT (DNAME,LOC) values ({dname},{loc}) </insert></pre> <div style="text-align: right;">+</div>
---	---

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">自动递增（续1）</h4> <ul style="list-style-type: none"> • 在<insert>元素指定了自动递增属性设置后，MyBatis会在插入操作后将自动生成的主键值给keyProperty指定的属性赋值。 <div style="text-align: right;">+</div>
---	---



1.1.3. 【主键映射】非自动递增

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>非自动递增</h4> <ul style="list-style-type: none"> 在使用Oracle数据库时，由于不支持自动递增，经常采用序列生成主键值。 Oracle基于序列生成主键的映射方法如下： <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <pre> <insert id="addDept" parameterType="Dept"> <selectKey keyProperty="deptno" order="BEFORE" resultType="java.lang.Integer"> select dept_seq.nextval from dual </selectKey> insert into T_DEPT (DEPTNO,DNAME,LOC) values ({deptno},{dname},{loc}) </insert> </pre> </div> <div style="text-align: right;">  </div>
---	---



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>非自动递增（续1）</h4> <ul style="list-style-type: none"> 在<insert>元素指定了<selectKey>设置后，MyBatis会在插入操作前先执行<selectKey>获取主键值的SQL，然后再执行插入的SQL。 <div style="text-align: right;">  </div>
---	--


1.2. 关联映射

1.2.1. 【关联映射】关联映射作用



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>关联映射作用</h4> <ul style="list-style-type: none"> 在查询时经常需要获取两个或两个以上关联表的数据，通过关联映射可以由一个对象获取相关联对象的数据。例如查询一个Emp员工对象，可通过关联映射获取员工所在部门的Dept对象信息。 MyBatis的关联映射有以下两种不同的实现形式： <ul style="list-style-type: none"> 嵌套查询：通过执行另外一个SQL映射语句来返回关联数据结果（查两次）。 嵌套结果：执行一个表关联查询SQL，然后将查询结果映射成关联对象（查一次）。 <div style="text-align: right;">  </div>
---	---



1.2.2. 【关联映射】嵌套查询映射



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">嵌套查询映射</h4> <ul style="list-style-type: none"> 嵌套查询映射的应用示例如下： <pre> <select id="findById" parameterType="java.lang.Integer" resultMap="empMap"> select * from EMP where EMPNO=#{id} </select> <select id="selectDept" parameterType=" java.lang.Integer" resultType="Dept"> select * from DEPT where DEPTNO=#{id} </select> <resultMap type="Emp" id="empMap"> <association property="dept" column="DEPTNO" javaType="Dept" select="selectDept"> </association> </resultMap> </pre> <div style="text-align: right;">  </div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">嵌套查询映射（续1）</h4> <ul style="list-style-type: none"> 上面映射信息，当利用findById查询EMP时，会只查询EMP表返回一个Emp对象，当访问Emp对象的dept属性时，会调用selectDept查询操作获取DEPT表相关的记录。
---	--

1.2.3. 【关联映射】嵌套结果映射



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">嵌套结果映射</h4> <ul style="list-style-type: none"> 嵌套结果映射的应用示例如下： <pre> <select id="findById" parameterType="java.lang.Integer" resultMap="empMap"> select e.empno,e.ename,e.job,e.mgr,e.sal,e.comm, e.hiredate,e.deptno,d.dname,d.loc from EMP e join DEPT d on(d.deptno=e.deptno) where e.EMPNO=#{id} </select> </pre> <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>嵌套结果映射（续1）</h3> <ul style="list-style-type: none"> 上面resultMap属性指定的empMap的映射如下： <pre> <resultMap type="Emp" id="empMap"> <id property="empno" column="EMPNO"/> <result property="ename" column="ENAME"/> <result property="job" column="JOB" /> <result property="mgr" column="MGR"/> <result property="hireDate" column="HIREDATE"/> <result property="sal" column="SAL" /> <result property="comm" column="COMM"/> <association property="dept" column="DEPTNO" javaType="Dept" resultMap="deptResult"> </association> </resultMap> </pre> <div style="text-align: right;">  </div>
---	---


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>嵌套结果映射（续2）</h3> <ul style="list-style-type: none"> 上面映射信息，当利用findById查询时，会执行定义的关联查询SQL，然后MyBatis负责将结果数据提取，EMP字段值封装成Emp对象，DEPT字段值封装成Dept对象，然后将Dept对象给Emp对象的dept属性赋值。 <div style="text-align: right;">  </div>
---	--

1.3. 集合映射

1.3.1. 【集合映射】集合映射作用


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>集合映射作用</h3> <ul style="list-style-type: none"> 当查询某个表的记录信息时，如果关联表有多条相关记录，此时就可以使用集合映射。例如查询某个Dept部门对象信息，通过集合映射可以获得该部门中所有的Emp员工对象信息。 MyBatis的集合映射有以下两种不同的实现形式： <ul style="list-style-type: none"> 嵌套查询：通过执行另外一个SQL映射语句来返回关联数据结果（查两次）。 嵌套结果：执行一个表关联查询SQL，然后将查询结果映射成关联对象（查一次）。 <div style="text-align: right;">  </div>
---	--


1.3.2. 【集合映射】嵌套查询映射

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">嵌套查询映射</h4> <ul style="list-style-type: none"> 嵌套查询映射的应用示例如下： <pre> <select id="findById" parameterType="java.lang.Integer" resultMap="deptMap"> select DEPTNO,DNAME,LOC from DEPT where DEPTNO=#{deptno} </select> <select id="selectEmps" parameterType="java.lang.Integer" resultType="Emp"> select * from EMP where DEPTNO=#{deptno} </select> <resultMap id="deptMap" type="Dept"> <id column="DEPTNO" property="deptno"/> <collection ofType="Emp" property="emps" javaType="java.util.ArrayList" column="DEPTNO" select="selectEmps"> </collection> </resultMap> </pre> <div style="text-align: right;">+</div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">嵌套查询映射（续1）</h4> <ul style="list-style-type: none"> 上面映射信息，当利用findById查询DEPT时，会只查询DEPT表返回一个Dept对象，当访问Dept对象的emps属性时，会调用selectEmps查询操作获取EMP表相关的记录。 <div style="text-align: right;">+</div>
---	---

1.3.3. 【集合映射】嵌套结果映射

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">嵌套结果映射</h4> <ul style="list-style-type: none"> 嵌套结果映射的应用示例如下： <pre> <select id="findById" parameterType="java.lang.Integer" resultMap="deptEmpsResult"> select d.DEPTNO,d.DNAME,d.LOC, e.EMPNO,e.ENAME,e.SAL,e.MGR, e.COMM,e.HIREDATE,e.JOB from DEPT d join EMP e on(d.DEPTNO=e.DEPTNO) where d.DEPTNO=#{deptno} </select> </pre> <div style="text-align: right;">+</div>
---	---



嵌套结果映射（续1）

- 上面resultMap属性指定的deptEmpsResult的映射如下：


```

<resultMap id="deptEmpsResult" type="Dept">
  <id property="deptno" column="DEPTNO"/>
  <result property="dname" column="DNAME"/>
  <result property="loc" column="LOC"/>
  <collection property="emps" ofType="Emp">
    <id property="empno" column="EMPNO"/>
    <result property="ename" column="ENAME"/>
    <result property="job" column="JOB"/>
    <result property="mgr" column="MGR"/>
    <result property="hireDate" column="HIREDATE"/>
    <!-- 省略其他结果字段映射 -->
  </collection>
</resultMap>

```

代码清单

+



嵌套结果映射（续2）


- 上面映射信息，当利用findByld查询时，会执行定义的关联查询SQL，然后MyBatis负责将结果数据提取，DEPT字段值封装成Dept对象，EMP字段值封装成Emp对象集合，然后将Emp对象集合给Dept对象的emps属性赋值。

代码清单

+

1.4. 鉴别器

1.4.1. 【鉴别器】鉴别器的作用




鉴别器的作用

- 有时一个单独的数据库查询也许返回很多不同数据类型的结果集。例如一个表存储了单选题和多选题，查询时需要返回单选题对象和多选题对象。鉴别器就是用来处理这个情况的。鉴别器元素很像Java语言中的switch分支语句的结构。

代码清单

+

1.4.2. 【鉴别器】鉴别器的使用

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;"></div> <h3>鉴别器的使用</h3> <ul style="list-style-type: none">鉴别器的应用示例如下： <pre><select id="findAll" resultMap="vehicleMap"> select * from T_CAR </select> <resultMap id="vehicleMap" type="Vehicle"> <id property="id" column="ID"/> <result property="color" column="COLOR"/> <discriminator javaType="string" column="TYPE"> <case value="T" resultType="Truck"> <result property="boxSize" column="BOXSIZE"/> </case> <case value="C" resultType="Car"> <result property="doorSize" column="DOORSIZE"/> </case> </discriminator> </resultMap></pre> <div style="text-align: right;">++</div>
---	---

1.4.3. 【鉴别器】嵌套结果映射

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;"></div> <h3>嵌套结果映射</h3> <ul style="list-style-type: none">上面映射信息，当利用findAll查询时，会查询T_CAR表所有记录，然后MyBatis会根据鉴别器type字段值分别封装成Truck对象和Car对象放入返回的结果集中。 <div style="text-align: right;">++</div>
---	--

经典案例

1. 主键映射使用案例

- **问题**

新增员工时，自动生成主键，并且将主键值注入给传入的实体对象，从而可以通过实体对象得到该主键值。

- **方案**

主键映射有 2 种方式：

1) 自动递增

这种方式针对的是 Oracle 之外的数据库，语法如下：

```
<insert id="addDept" parameterType="Dept"
    keyProperty="deptno" useGeneratedKeys="true" >
    insert into T_DEPT (DNAME,LOC) values ({dname},{loc})
</insert>
```

2) 非自动递增

这种方式针对的是 Oracle 数据库，语法如下：

```
<insert id="addDept" parameterType="Dept">
    <selectKey keyProperty="deptno"
        order="BEFORE" resultType="java.lang.Integer">
        select dept_seq.nextval from dual
    </selectKey>
    insert into T_DEPT (DEPTNO,DNAME,LOC)
        values ({deptno},{dname},{loc})
</insert>
```

可见自动递增的方式十分简单，非自动递增的方式略微复杂，本案例只演示复杂的方式，即非自动递增的方式。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：创建项目

复制项目 SpringUnit03，创建新项目 SpringUnit04，删除 EmpDao 接口中的方法以及 EmpMapper.xml 中的 SQL，本案例中要写一些新内容。

步骤二：增加保存员工的方法

在 EmpDao 中增加保存员工的方法，代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    void save(Emp emp);

}
```

步骤三：实现保存员工的方法

在 EmpMapper.xml 中增加 SQL，实现保存员工，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        <selectKey keyProperty="empno"
            order="BEFORE" resultType="java.lang.Integer">
            select emp_seq.nextval from dual
        </selectKey>
        insert into t_emp values(
            #{empno},
            #{ename,jdbcType=VARCHAR},
            #{job,jdbcType=VARCHAR},
            #{mgr,jdbcType=NUMERIC},
            #{hiredate,jdbcType=DATE},
            #{sal,jdbcType=NUMERIC},
            #{comm,jdbcType=NUMERIC},
            #{deptno,jdbcType=NUMERIC}
        )
    </insert>

</mapper>
```

步骤四：测试保存员工的方法

创建 Junit 测试类 TestMapping，并增加测试保存员工的方法，代码如下：

```
package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
```

```
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试MyBatis 关联映射
 */
public class TestMapping {

    /**
     * 主键映射：新增一条员工数据
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = new Emp();
        e.setName("张三");
        e.setJob("SALESMAN");
        e.setMgr(3);
        e.setHiredate(
            new Date(System.currentTimeMillis()));
        e.setSal(4000.0);
        e.setComm(650.0);
        e.setDeptno(10);
        dao.save(e);
        System.out.println(e.getEmpno());
    }
}
```

执行该方法，控制台输出效果如下图：

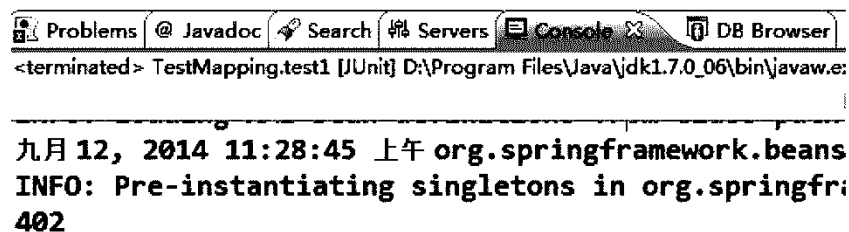


图-1

• 完整代码

本案例的完整代码如下所示：

EmpDao 完整代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {
```



```
void save(Emp emp);  
}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"  
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">  
  
<mapper namespace="com.tarena.dao.EmpDao">  
  
    <!-- 保存一条员工数据 -->  
    <insert id="save"  
        parameterType="com.tarena.entity.Emp">  
        <selectKey keyProperty="empno"  
            order="BEFORE" resultType="java.lang.Integer">  
            select emp_seq.nextval from dual  
        </selectKey>  
        insert into t_emp values(  
            #{empno},  
            #{ename,jdbcType=VARCHAR},  
            #{job,jdbcType=VARCHAR},  
            #{mgr,jdbcType=NUMERIC},  
            #{hiredate,jdbcType=DATE},  
            #{sal,jdbcType=NUMERIC},  
            #{comm,jdbcType=NUMERIC},  
            #{deptno,jdbcType=NUMERIC}  
        )  
    </insert>  
  
</mapper>
```

TestMapping 完整代码如下：

```
package com.tarena.test;  
  
import java.sql.Date;  
import java.util.List;  
import org.junit.Test;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import com.tarena.dao.DeptDao;  
import com.tarena.dao.EmpDao;  
import com.tarena.dao.VehicleDao;  
import com.tarena.entity.Dept;  
import com.tarena.entity.Emp;  
import com.tarena.entity.Vehicle;  
  
/**  
 * 测试MyBatis 关联映射  
 */  
public class TestMapping {  
  
    /**  
     * 主键映射：新增一条员工数据  
     */  
    @Test  
    public void test1() {  
        ApplicationContext ctx = new ClassPathXmlApplicationContext(  
            "applicationContext.xml");  
        EmpDao dao = ctx.getBean(EmpDao.class);  
        Emp e = new Emp();  
        e.setEname("张三");  
    }  
}
```

```
e.setJob("SALESMAN");
e.setMgr(3);
e.setHiredate(
    new Date(System.currentTimeMillis()));
e.setSal(4000.0);
e.setComm(650.0);
e.setDeptno(10);
dao.save(e);
System.out.println(e.getEmpno());
}

}
```

2. 多对一嵌套查询映射使用案例

- **问题**

查询员工时，自动查询出他所在的部门。

- **方案**

使用嵌套查询映射，分别写出查询员工和查询部门的 SQL，然后通过配置自动的将查询到的部门装配给员工。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：增加关联属性

在 Emp 中增加关联属性，用于封装员工对应的部门信息，代码如下：

```
package com.tarena.entity;

import java.sql.Date;

/**
 * 员工表的实体类
 */
public class Emp {

    private Integer empno;
    private String ename;
    private String job;
    private Integer mgr;
    private Date hiredate;
    private Double sal;
    private Double comm;
    private Integer deptno;

    /**
     * 关联属性，用于封装员工对应的部门信息
     */
    private Dept dept;

    public Dept getDept() {
        return dept;
    }
}
```

```
}

public void setDept(Dept dept) {
    this.dept = dept;
}

public Integer getEmpno() {
    return empno;
}

public void setEmpno(Integer empno) {
    this.empno = empno;
}

public String getName() {
    return ename;
}

public void setName(String ename) {
    this.ename = ename;
}

public String getJob() {
    return job;
}

public void setJob(String job) {
    this.job = job;
}

public Integer getMgr() {
    return mgr;
}

public void setMgr(Integer mgr) {
    this.mgr = mgr;
}

public Date getHiredate() {
    return hiredate;
}

public void setHiredate(Date hiredate) {
    this.hiredate = hiredate;
}

public Double getSal() {
    return sal;
}

public void setSal(Double sal) {
    this.sal = sal;
}

public Double getComm() {
    return comm;
}

public void setComm(Double comm) {
    this.comm = comm;
}

public Integer getDeptno() {
    return deptno;
}

public void setDeptno(Integer deptno) {
```

```
        this.deptno = deptno;
    }

}
```

步骤二：增加根据 ID 查询员工的方法

在 EmpDao 中增加根据 ID 查询员工的方法，代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    void save(Emp emp);

    Emp findById(int id);

}
```

步骤三：实现根据 ID 查询员工的方法

在 EmpMapper.xml 中增加 SQL，实现根据 ID 查询员工，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        <selectKey keyProperty="empno"
            order="BEFORE" resultType="java.lang.Integer">
            select emp seq.nextval from dual
        </selectKey>
        insert into t_emp values(
            #{empno},
            #{ename,jdbcType=VARCHAR},
            #{job,jdbcType=VARCHAR},
            #{mgr,jdbcType=NUMERIC},
            #{hiredate,jdbcType=DATE},
            #{sal,jdbcType=NUMERIC},
            #{comm,jdbcType=NUMERIC},
            #{deptno,jdbcType=NUMERIC}
        )
    </insert>

    <!-- 使用嵌套查询，在查询一条员工数据时，关联查询出对应的部门 -->
    <select id="findById"
        parameterType="java.lang.Integer"
        resultMap="empMap">
        select * from t_emp where empno=#{id}
```

```

</select>
<select id="findDept"
    parameterType="java.lang.Integer"
    resultType="com.tarena.entity.Dept">
    select * from t_dept where deptno=#{deptno}
</select>
<resultMap type="com.tarena.entity.Emp" id="empMap">
    <association property="dept" column="deptno"
        javaType="com.tarena.entity.Dept" select="findDept">
    </association>
</resultMap>

</mapper>

```

步骤四：测试根据 ID 查询员工的方法

在 TestMapping 中增加方法，测试根据 ID 查询员工，代码如下：

```

package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试 MyBatis 关联映射
 */
public class TestMapping {

    //其他方法略

    /**
     * 多对一嵌套查询映射：
     * 根据 ID 查询一条员工数据，并查询他所在的部门。
     */
    @Test
    public void test2() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById(1);
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob() + " " +
            e.getDept().getDeptno() + " " +
            e.getDept().getDname()
        );
    }
}

```

执行该方法，控制台输出效果如下图：

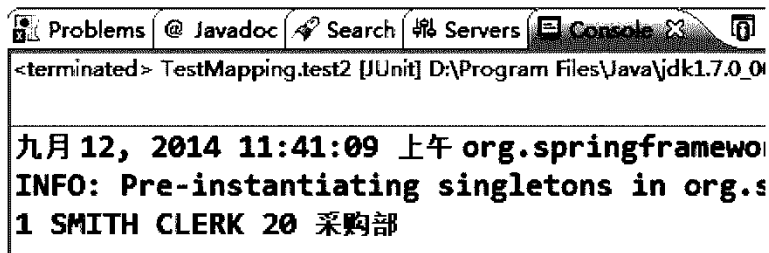


图-2

• 完整代码

本案例的完整代码如下所示：

Emp 完整代码如下：

```
package com.tarena.entity;
import java.sql.Date;

/**
 * 员工表的实体类
 */
public class Emp {
    private Integer empno;
    private String ename;
    private String job;
    private Integer mgr;
    private Date hiredate;
    private Double sal;
    private Double comm;
    private Integer deptno;

    /**
     * 关联属性，用于封装员工对应的部门信息
     */
    private Dept dept;

    public Dept getDept() {
        return dept;
    }

    public void setDept(Dept dept) {
        this.dept = dept;
    }

    public Integer getEmpno() {
        return empno;
    }

    public void setEmpno(Integer empno) {
        this.empno = empno;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }
}
```

```
public String getJob() {
    return job;
}

public void setJob(String job) {
    this.job = job;
}

public Integer getMgr() {
    return mgr;
}

public void setMgr(Integer mgr) {
    this.mgr = mgr;
}

public Date getHiredDate() {
    return hiredDate;
}

public void setHiredDate(Date hiredDate) {
    this.hiredDate = hiredDate;
}

public Double getSal() {
    return sal;
}

public void setSal(Double sal) {
    this.sal = sal;
}

public Double getComm() {
    return comm;
}

public void setComm(Double comm) {
    this.comm = comm;
}

public Integer getDeptno() {
    return deptno;
}

public void setDeptno(Integer deptno) {
    this.deptno = deptno;
}
}
```

EmpDao 完整代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    void save(Emp emp);

    Emp findById(int id);
}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
"http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        <selectKey keyProperty="empno"
            order="BEFORE" resultType="java.lang.Integer">
            select emp_seq.nextval from dual
        </selectKey>
        insert into t_emp values(
            #{empno},
            #{ename,jdbcType=VARCHAR},
            #{job,jdbcType=VARCHAR},
            #{mgr,jdbcType=NUMERIC},
            #{hiredate,jdbcType=DATE},
            #{sal,jdbcType=NUMERIC},
            #{comm,jdbcType=NUMERIC},
            #{deptno,jdbcType=NUMERIC}
        )
    </insert>

    <!-- 使用嵌套查询，在查询一条员工数据时，关联查询出对应的部门 -->
    <select id="findById"
        parameterType="java.lang.Integer"
        resultMap="empMap">
        select * from t_emp where empno=#{id}
    </select>
    <select id="findDept"
        parameterType="java.lang.Integer"
        resultType="com.tarena.entity.Dept">
        select * from t_dept where deptno=#{deptno}
    </select>
    <resultMap type="com.tarena.entity.Emp" id="empMap">
        <association property="dept" column="deptno"
            javaType="com.tarena.entity.Dept" select="findDept">
        </association>
    </resultMap>

</mapper>
```

TestMapping 完整代码如下：

```
package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试 MyBatis 关联映射
```



```
*/
public class TestMapping {

    /**
     * 主键映射：新增一条员工数据
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = new Emp();
        e.setName("张三");
        e.setJob("SALESMAN");
        e.setMgr(3);
        e.setHiredate(
            new Date(System.currentTimeMillis()));
        e.setSal(4000.0);
        e.setComm(650.0);
        e.setDeptno(10);
        dao.save(e);
        System.out.println(e.getEmpno());
    }

    /**
     * 多对一嵌套查询映射：
     * 根据 ID 查询一条员工数据，并查询他所在的部门。
     */
    @Test
    public void test2() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById(1);
        System.out.println(
            e.getEmpno() + " " +
            e.getName() + " " +
            e.getJob() + " " +
            e.getDept().getDeptno() + " " +
            e.getDept().getDname()
        );
    }
}
```

3. 多对一嵌套结果映射使用案例

- 问题

查询员工时，自动查询出他所在的部门。

- 方案

使用嵌套结果映射，写出员工关联部门的 SQL，然后通过配置自动的将查询到的部门装配给员工。

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加根据 ID 查询员工的方法

在 EmpDao 中增加根据 ID 查询员工的方法，代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    void save(Emp emp);

    Emp findById(int id);

    Emp findById2(int id);

}
```

步骤二：实现根据 ID 查询员工的方法

在 EmpMapper.xml 中增加 SQL，实现根据 ID 查询员工，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        <selectKey keyProperty="empno"
            order="BEFORE" resultType="java.lang.Integer">
            select emp_seq.nextval from dual
        </selectKey>
        insert into t_emp values(
            #{empno},
            #{ename,jdbcType=VARCHAR},
            #{job,jdbcType=VARCHAR},
            #{mgr,jdbcType=NUMERIC},
            #{hiredate,jdbcType=DATE},
            #{sal,jdbcType=NUMERIC},
            #{comm,jdbcType=NUMERIC},
            #{deptno,jdbcType=NUMERIC}
        )
    </insert>

    <!-- 使用嵌套查询，在查询一条员工数据时，关联查询出对应的部门 -->
    <select id="findById"
        parameterType="java.lang.Integer"
        resultMap="empMap">
        select * from t_emp where empno=#{id}
    </select>
    <select id="findDept"
        parameterType="java.lang.Integer"
        resultType="com.tarena.entity.Dept">
        select * from t_dept where deptno=#{deptno}
```

```

</select>
<resultMap type="com.tarena.entity.Emp" id="empMap">
    <association property="dept" column="deptno"
        javaType="com.tarena.entity.Dept" select="findDept">
    </association>
</resultMap>

<!-- 使用嵌套结果，在查询一条员工数据时，关联查询出对应的部门 -->
<select id="findById2"
    parameterType="java.lang.Integer"
    resultMap="empMap">
    select e.*,d.* from t_emp e
    inner join t_dept d on e.deptno=d.deptno
    where e.empno=#{id}
</select>
<resultMap type="com.tarena.entity.Emp" id="empMap2">
    <id property="empno" column="empno"/>
    <result property="ename" column="ename"/>
    <result property="job" column="job"/>
    <result property="mgr" column="mgr"/>
    <result property="hiredate" column="hiredate"/>
    <result property="sal" column="sal"/>
    <result property="comm" column="comm"/>
    <result property="deptno" column="deptno"/>

    <association property="dept"
        column="deptno"
        javaType="com.tarena.entity.Dept">
        <id property="deptno" column="deptno"/>
        <result property="dname" column="dname"/>
        <result property="loc" column="loc"/>
    </association>
</resultMap>

</mapper>

```

步骤三：测试根据 ID 查询员工的方法

在 TestMapping 中增加方法，测试根据 ID 查询员工，代码如下：

```

package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试 MyBatis 关联映射
 */
public class TestMapping {

    //其他方法略

```

```
/**
 * 多对一嵌套结果映射：
 * 根据 ID 查询一条员工数据，并查询他所在的部门。
 */
@Test
public void test3() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = dao.findById2(1);
    System.out.println(
        e.getEmpno() + " " +
        e.getName() + " " +
        e.getJob() + " " +
        e.getDept().getDeptno() + " " +
        e.getDept().getDname()
    );
}
}
```

• 完整代码

本案例的完整代码如下所示：

EmpDao 完整代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    void save(Emp emp);

    Emp findById(int id);

    Emp findById2(int id);

}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        <selectKey keyProperty="empno"
            order="BEFORE" resultType="java.lang.Integer">
            select emp_seq.nextval from dual
        </selectKey>
```

```

insert into t_emp values(
    #{empno},
    #{ename,jdbcType=VARCHAR},
    #{job,jdbcType=VARCHAR},
    #{mgr,jdbcType=NUMERIC},
    #{hiredate,jdbcType=DATE},
    #{sal,jdbcType=NUMERIC},
    #{comm,jdbcType=NUMERIC},
    #{deptno,jdbcType=NUMERIC}
)
</insert>

<!-- 使用嵌套查询，在查询一条员工数据时，关联查询出对应的部门 -->
<select id="findById"
    parameterType="java.lang.Integer"
    resultMap="empMap">
    select * from t_emp where empno=#{id}
</select>
<select id="findDept"
    parameterType="java.lang.Integer"
    resultType="com.tarena.entity.Dept">
    select * from t_dept where deptno=#{deptno}
</select>
<resultMap type="com.tarena.entity.Emp" id="empMap">
    <association property="dept"
        column="deptno"
        javaType="com.tarena.entity.Dept"
        select="findDept">
    </association>
</resultMap>

<!-- 使用嵌套结果，在查询一条员工数据时，关联查询出对应的部门 -->
<select id="findById2"
    parameterType="java.lang.Integer"
    resultMap="empMap">
    select e.*,d.* from t_emp e
    inner join t_dept d on e.deptno=d.deptno
    where e.empno=#{id}
</select>
<resultMap type="com.tarena.entity.Emp" id="empMap2">
    <id property="empno" column="empno"/>
    <result property="ename" column="ename"/>
    <result property="job" column="job"/>
    <result property="mgr" column="mgr"/>
    <result property="hiredate" column="hiredate"/>
    <result property="sal" column="sal"/>
    <result property="comm" column="comm"/>
    <result property="deptno" column="deptno"/>

    <association property="dept"
        column="deptno"
        javaType="com.tarena.entity.Dept">
        <id property="deptno" column="deptno"/>
        <result property="dname" column="dname"/>
        <result property="loc" column="loc"/>
    </association>
</resultMap>

</mapper>

```

TestMapping 完整代码如下：

```

package com.tarena.test;

import java.sql.Date;
import java.util.List;

```

```
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试MyBatis 关联映射
 */
public class TestMapping {

    /**
     * 主键映射：新增一条员工数据
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = new Emp();
        e.setName("张三");
        e.setJob("SALESMAN");
        e.setMgr(3);
        e.setHiredate(
            new Date(System.currentTimeMillis()));
        e.setSal(4000.0);
        e.setComm(650.0);
        e.setDeptno(10);
        dao.save(e);
        System.out.println(e.getEmpno());
    }

    /**
     * 多对一嵌套查询映射：
     * 根据 ID 查询一条员工数据，并查询他所在的部门。
     */
    @Test
    public void test2() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById(1);
        System.out.println(
            e.getEmpno() + " " +
            e.getName() + " " +
            e.getJob() + " " +
            e.getDept().getDeptno() + " " +
            e.getDept().getDname()
        );
    }

    /**
     * 多对一嵌套结果映射：
     * 根据 ID 查询一条员工数据，并查询他所在的部门。
     */
    @Test
    public void test3() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById2(1);
        System.out.println(
            e.getEmpno() + " " +

```

```
        e.getName() + " " +  
        e.getJob() + " " +  
        e.getDept().getDeptno() + " " +  
        e.getDept().getDname()  
    );  
}  
}
```

4. 一对多嵌套查询映射使用案例

- **问题**

查询部门时，自动查询出该部门下所有的员工。

- **方案**

使用嵌套查询映射，分别写出查询部门和查询员工的 SQL，然后通过配置自动的将查询到的员工装配给部门。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：增加关联属性

创建部门实体类 Dept，并增加关联属性，用于封装该部门对应的一组员工数据，代码如下：

```
package com.tarena.entity;  
  
import java.util.List;  
  
public class Dept {  
  
    private Integer deptno;  
    private String dname;  
    private String loc;  
  
    /**  
     * 关联属性，用于封装部门对应的员工信息  
     */  
    private List<Emp> emps;  
  
    public List<Emp> getEmps() {  
        return emps;  
    }  
  
    public void setEmps(List<Emp> emps) {  
        this.emps = emps;  
    }  
  
    public Integer getDeptno() {  
        return deptno;  
    }  
  
    public void setDeptno(Integer deptno) {  
        this.deptno = deptno;  
    }  
}
```

```

    }

    public String getDname() {
        return dname;
    }

    public void setDname(String dname) {
        this.dname = dname;
    }

    public String getLoc() {
        return loc;
    }

    public void setLoc(String loc) {
        this.loc = loc;
    }
}

```

步骤二：增加根据 ID 查询部门的方法

创建部门 DAO 接口，并增加根据 ID 查询部门的方法，代码如下：

```

package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Dept;

@MyBatisRepository
public interface DeptDao {

    Dept findById(int id);

}

```

步骤三：实现根据 ID 查询部门的方法

创建部门映射文件 DeptMapper.xml，并增加 SQL，实现根据 ID 查询部门的方法，代码如下：

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.DeptDao">

    <!-- 嵌套查询 -->
    <select id="findById"
        parameterType="java.lang.Integer"
        resultMap="deptMap">
        select * from t_dept where deptno=#{id}
    </select>
    <select id="findEmps"
        parameterType="java.lang.Integer"
        resultType="com.tarena.entity.Emp">
        select * from t_emp where deptno=#{deptno}
    </select>
    <resultMap type="com.tarena.entity.Dept" id="deptMap">
        <id property="deptno" column="deptno"/>
        <collection property="emps" column="deptno"
            javaType="java.util.ArrayList"

```



```
        select="findEmps" ofType="com.tarena.entity.Emp">
    </collection>
</resultMap>

</mapper>
```

步骤四：测试根据 ID 查询部门的方法

在 TestMapping 中增加方法，测试根据 ID 查询部门的方法，代码如下：

```
package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试MyBatis 关联映射
 */
public class TestMapping {

    //其他方法略

    /**
     * 一对多嵌套查询映射：
     * 查询部门及部门下所有的员工。
     */
    @Test
    public void test4() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        DeptDao dao = ctx.getBean(DeptDao.class);
        Dept d = dao.findById(10);
        System.out.println(
            d.getDeptno() + " " +
            d.getDname() + " " +
            d.getLoc()
        );
        List<Emp> emps = d.getEmps();
        for(Emp e : emps) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }
}
```

- 完整代码

本案例的完整代码如下所示：

Dept 完整代码如下：

```
package com.tarena.entity;

import java.util.List;

public class Dept {

    private Integer deptno;
    private String dname;
    private String loc;

    /**
     * 关联属性，用于封装部门对应的员工信息
     */
    private List<Emp> emps;

    public List<Emp> getEmps() {
        return emps;
    }

    public void setEmps(List<Emp> emps) {
        this.emps = emps;
    }

    public Integer getDeptno() {
        return deptno;
    }

    public void setDeptno(Integer deptno) {
        this.deptno = deptno;
    }

    public String getDname() {
        return dname;
    }

    public void setDname(String dname) {
        this.dname = dname;
    }

    public String getLoc() {
        return loc;
    }

    public void setLoc(String loc) {
        this.loc = loc;
    }

}
```

DeptDao 完整代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Dept;

@MyBatisRepository
public interface DeptDao {

    Dept findById(int id);
}
```

DeptMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.DeptDao">

    <!-- 嵌套查询 -->
    <select id="findById"
        parameterType="java.lang.Integer"
        resultMap="deptMap">
        select * from t_dept where deptno=#{id}
    </select>
    <select id="findEmps"
        parameterType="java.lang.Integer"
        resultType="com.tarena.entity.Emp">
        select * from t_emp where deptno=#{deptno}
    </select>
    <resultMap type="com.tarena.entity.Dept" id="deptMap">
        <id property="deptno" column="deptno"/>
        <collection property="emps" column="deptno"
            javaType="java.util.ArrayList"
            select="findEmps" ofType="com.tarena.entity.Emp">
        </collection>
    </resultMap>

</mapper>
```

TestMapping 完整代码如下：

```
package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试MyBatis 关联映射
 */
public class TestMapping {

    /**
     * 主键映射：新增一条员工数据
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = new Emp();
        e.setName("张三");
        e.setJob("SALESMAN");
        e.setMgr(3);
        e.setHiredate(
            new Date(System.currentTimeMillis()));
    }
}
```

```

        e.setSal(4000.0);
        e.setComm(650.0);
        e.setDeptno(10);
        dao.save(e);
        System.out.println(e.getEmpno());
    }

    /**
     * 多对一嵌套查询映射:
     * 根据 ID 查询一条员工数据, 并查询他所在的部门。
     */
    @Test
    public void test2() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById(1);
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob() + " " +
            e.getDept().getDeptno() + " " +
            e.getDept().getDname()
        );
    }

    /**
     * 多对一嵌套结果映射:
     * 根据 ID 查询一条员工数据, 并查询他所在的部门。
     */
    @Test
    public void test3() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById2(1);
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob() + " " +
            e.getDept().getDeptno() + " " +
            e.getDept().getDname()
        );
    }

    /**
     * 一对多嵌套查询映射:
     * 查询部门及部门下所有的员工。
     */
    @Test
    public void test4() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        DeptDao dao = ctx.getBean(DeptDao.class);
        Dept d = dao.findById(10);
        System.out.println(
            d.getDeptno() + " " +
            d.getDname() + " " +
            d.getLoc()
        );
        List<Emp> emps = d.getEmps();
        for(Emp e : emps) {
            System.out.println(
                e.getEmpno() + " " +
                e.getEname() + " " +
                e.getJob()
            );
        }
    }

```

```
}  
}  
}
```

5. 一对多嵌套结果映射使用案例

- 问题

查询部门时，自动查询出该部门下所有的员工。

- 方案

使用嵌套结果映射，写出部门关联员工的 SQL，然后通过配置自动的将查询到的员工装配给部门。

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加根据 ID 查询部门的方法

在 DeptDao 中增加根据 ID 查询部门的方法，代码如下：

```
package com.tarena.dao;  
  
import com.tarena.annotation.MyBatisRepository;  
import com.tarena.entity.Dept;  
  
@MyBatisRepository  
public interface DeptDao {  
  
    Dept findById(int id);  
  
    Dept findById2(int id);  
  
}
```

步骤二：实现根据 ID 查询部门的方法

在 DeptMapper.xml 中增加 SQL，实现根据 ID 查询部门，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"  
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">  
  
<mapper namespace="com.tarena.dao.DeptDao">  
  
    <!-- 嵌套查询 -->  
    <select id="findById"  
        parameterType="java.lang.Integer"  
        resultMap="deptMap">  
        select * from t_dept where deptno=#{id}  
    </select>  
    <select id="findEmps"  
        parameterType="java.lang.Integer"
```

```

        resultType="com.tarena.entity.Emp">
        select * from t_emp where deptno=#{deptno}
    </select>
    <resultMap type="com.tarena.entity.Dept" id="deptMap">
        <id property="deptno" column="deptno"/>
        <collection property="emps" column="deptno"
            javaType="java.util.ArrayList"
            select="findEmps" ofType="com.tarena.entity.Emp">
        </collection>
    </resultMap>

    <!-- 嵌套结果 -->
    <select id="findById2"
        parameterType="java.lang.Integer"
        resultMap="deptMap2">
        select d.*,e.* from t_dept d
        inner join t_emp e on d.deptno=e.deptno
        where d.deptno=#{id}
    </select>
    <resultMap type="com.tarena.entity.Dept" id="deptMap2">
        <id property="deptno" column="deptno"/>
        <result property="dname" column="dname"
            jdbcType="VARCHAR" javaType="string"/>
        <result property="loc" column="loc"
            jdbcType="VARCHAR" javaType="string"/>
        <collection property="emps" ofType="com.tarena.entity.Emp"
            javaType="java.util.ArrayList" column="deptno">
            <id property="empno" column="empno"/>
            <result property="ename" column="ename"/>
            <result property="job" column="job"/>
            <result property="mgr" column="mgr"/>
            <result property="hiredate" column="hiredate"/>
            <result property="sal" column="sal"/>
            <result property="comm" column="comm"/>
            <result property="deptno" column="deptno"/>
        </collection>
    </resultMap>

</mapper>

```

步骤三：测试根据 ID 查询部门的方法

在 TestMapping 中增加方法，测试根据 ID 查询部门，代码如下：

```

package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试 MyBatis 关联映射
 */
public class TestMapping {

```

//其他方法略

```
/**
 * 一对多嵌套结果映射：
 * 查询部门及部门下所有的员工。
 */
@Test
public void test5() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    DeptDao dao = ctx.getBean(DeptDao.class);
    Dept d = dao.findById2(10);
    System.out.println(
        d.getDeptno() + " " +
        d.getDname() + " " +
        d.getLoc()
    );
    List<Emp> emps = d.getEmps();
    for(Emp e : emps) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}
```

• 完整代码

本案例的完整代码如下所示：

DeptDao 完整代码如下：

```
package com.tarena.dao;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Dept;

@MyBatisRepository
public interface DeptDao {

    Dept findById(int id);

    Dept findById2(int id);

}
```

DeptMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.DeptDao">

    <!-- 嵌套查询 -->
    <select id="findById"
        parameterType="java.lang.Integer"
```

```

        resultMap="deptMap">
            select * from t_dept where deptno=#{id}
        </select>
        <select id="findEmps"
            parameterType="java.lang.Integer"
            resultType="com.tarena.entity.Emp">
            select * from t_emp where deptno=#{deptno}
        </select>
        <resultMap type="com.tarena.entity.Dept" id="deptMap">
            <id property="deptno" column="deptno"/>
            <collection property="emps" column="deptno"
                javaType="java.util.ArrayList"
                select="findEmps" ofType="com.tarena.entity.Emp">
            </collection>
        </resultMap>

        <!-- 嵌套结果 -->
        <select id="findById2"
            parameterType="java.lang.Integer"
            resultMap="deptMap2">
            select d.*,e.* from t_dept d
            inner join t_emp e on d.deptno=e.deptno
            where d.deptno=#{id}
        </select>
        <resultMap type="com.tarena.entity.Dept" id="deptMap2">
            <id property="deptno" column="deptno"/>
            <result property="dname" column="dname"
                jdbcType="VARCHAR" javaType="string"/>
            <result property="loc" column="loc"
                jdbcType="VARCHAR" javaType="string"/>
            <collection property="emps" ofType="com.tarena.entity.Emp"
                javaType="java.util.ArrayList" column="deptno">
            <id property="empno" column="empno"/>
            <result property="ename" column="ename"/>
            <result property="job" column="job"/>
            <result property="mgr" column="mgr"/>
            <result property="hiredate" column="hiredate"/>
            <result property="sal" column="sal"/>
            <result property="comm" column="comm"/>
            <result property="deptno" column="deptno"/>
            </collection>
        </resultMap>
    </mapper>

```

TestMapping 完整代码如下：

```

package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试 MyBatis 关联映射
 */
public class TestMapping {

```



```
/**
 * 主键映射：新增一条员工数据
 */
@Test
public void test1() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = new Emp();
    e.setName("张三");
    e.setJob("SALESMAN");
    e.setMgr(3);
    e.setHiredate(
        new Date(System.currentTimeMillis()));
    e.setSal(4000.0);
    e.setComm(650.0);
    e.setDeptno(10);
    dao.save(e);
    System.out.println(e.getEmpno());
}

/**
 * 多对一嵌套查询映射：
 * 根据 ID 查询一条员工数据，并查询他所在的部门。
 */
@Test
public void test2() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = dao.findById(1);
    System.out.println(
        e.getEmpno() + " " +
        e.getName() + " " +
        e.getJob() + " " +
        e.getDept().getDeptno() + " " +
        e.getDept().getDname()
    );
}

/**
 * 多对一嵌套结果映射：
 * 根据 ID 查询一条员工数据，并查询他所在的部门。
 */
@Test
public void test3() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = dao.findById2(1);
    System.out.println(
        e.getEmpno() + " " +
        e.getName() + " " +
        e.getJob() + " " +
        e.getDept().getDeptno() + " " +
        e.getDept().getDname()
    );
}

/**
 * 一对多嵌套查询映射：
 * 查询部门及部门下所有的员工。
 */
@Test
public void test4() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
}
```

```

DeptDao dao = ctx.getBean(DeptDao.class);
Dept d = dao.findById(10);
System.out.println(
    d.getDeptno() + " " +
    d.getDname() + " " +
    d.getLoc()
);
List<Emp> emps = d.getEmps();
for(Emp e : emps) {
    System.out.println(
        e.getEmpno() + " " +
        e.getEname() + " " +
        e.getJob()
    );
}
}

/**
 * 一对多嵌套结果映射：
 * 查询部门及部门下所有的员工。
 */
@Test
public void test5() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    DeptDao dao = ctx.getBean(DeptDao.class);
    Dept d = dao.findById2(10);
    System.out.println(
        d.getDeptno() + " " +
        d.getDname() + " " +
        d.getLoc()
    );
    List<Emp> emps = d.getEmps();
    for(Emp e : emps) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}
}

```

6. 鉴别器使用案例

- **问题**

查询汽车表时，将不同类型的汽车自动封装成不同的对象。

- **方案**

使用鉴别器 根据表中某个字段区别数据 将查询出的数据自动封装成不同类型的对象。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：建表

创建汽车表，并插入预置数据，脚本如下：

```
CREATE TABLE t_car (  
    id NUMBER(11) NOT NULL,  
    type VARCHAR2(1) default NULL,  
    doorSize NUMBER(11) default NULL,  
    boxSize NUMBER(11) default NULL,  
    color VARCHAR2(20) default NULL,  
    PRIMARY KEY (id)  
);  
  
insert into t_car values (1,'C',2,null,'红色');  
insert into t_car values (2,'C',4,null,'黑色');  
insert into t_car values (3,'T',null,1,'蓝色');  
insert into t_car values (4,'T',null,2,'蓝色');  
commit;
```

步骤二：创建实体类

创建交通工具实体类 Vehicle，封装汽车表中的公用字段，代码如下：

```
package com.tarena.entity;  
  
/**  
 * 交通工具实体类，封装汽车表中公用的字段  
 */  
public class Vehicle {  
  
    private int id;  
    private String type;  
    private String color;  
  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getType() {  
        return type;  
    }  
  
    public void setType(String type) {  
        this.type = type;  
    }  
  
    public String getColor() {  
        return color;  
    }  
  
    public void setColor(String color) {  
        this.color = color;  
    }  
}
```

创建小汽车实体类，继承于 Vehicle，并封装小汽车相关的字段，代码如下：

```
package com.tarena.entity;

/**
 * 小汽车实体类，封装小汽车相关字段
 */
public class Car extends Vehicle {

    private int doorSize;

    public int getDoorSize() {
        return doorSize;
    }

    public void setDoorSize(int doorSize) {
        this.doorSize = doorSize;
    }
}
```

创建卡车实体类，继承于 Vehicle，并封装卡车相关的字段，代码如下：

```
package com.tarena.entity;

/**
 * 卡车实体类，封装卡车相关的字段
 */
public class Truck extends Vehicle {

    private int boxSize;

    public int getBoxSize() {
        return boxSize;
    }

    public void setBoxSize(int boxSize) {
        this.boxSize = boxSize;
    }
}
```

步骤三：创建 DAO 接口

创建 DAO 接口 VehicleDao，并增加查询全部汽车数据的方法，代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Vehicle;

@MyBatisRepository
public interface VehicleDao {

    List<Vehicle> findAll();

}
```

步骤四：创建映射文件

创建映射文件 VehicleMapper.xml，并增加 SQL 实现查询全部汽车，代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
"http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">
<mapper namespace="com.tarena.dao.VehicleDao">

    <resultMap id="vehicleMap" type="com.tarena.entity.Vehicle">
        <id property="id" column="ID" />
        <result property="color" column="COLOR" />
        <discriminator javaType="java.lang.String" column="TYPE">
            <case value="T" resultType="com.tarena.entity.Truck">
                <result property="boxSize" column="BOXSIZE" />
            </case>
            <case value="C" resultType="com.tarena.entity.Car">
                <result property="doorSize" column="DOORSIZE" />
            </case>
        </discriminator>
    </resultMap>

    <select id="findAll" resultMap="vehicleMap">
        select * from T CAR
    </select>

</mapper>
```

步骤五：增加测试方法

在 TestMapping 中增加方法，测试查询全部汽车的方法，代码如下：

```
package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试MyBatis 关联映射
 */
public class TestMapping {

    //其他方法略

    /**
     * 鉴别器：
     * 查询汽车表，根据类型封装成不同的对象。
     */
    @Test
    public void test6() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        VehicleDao dao = ctx.getBean(VehicleDao.class);
        List<Vehicle> list = dao.findAll();
        for (Vehicle v : list) {
            System.out.println(v);
        }
    }
}
```

- **完整代码**

本案例的完整代码如下所示：

Vehicle 完整代码如下：

```
package com.tarena.entity;

/**
 * 交通工具实体类，封装汽车表中公用的字段
 */
public class Vehicle {

    private int id;
    private String type;
    private String color;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

}
```

Car 完整代码如下：

```
package com.tarena.entity;

/**
 * 小汽车实体类，封装小汽车相关字段
 */
public class Car extends Vehicle {

    private int doorSize;

    public int getDoorSize() {
        return doorSize;
    }

    public void setDoorSize(int doorSize) {
        this.doorSize = doorSize;
    }

}
```

Truck 完整代码如下：

```
package com.tarena.entity;

/**
 * 卡车实体类，封装卡车相关的字段
 */
public class Truck extends Vehicle {

    private int boxSize;

    public int getBoxSize() {
        return boxSize;
    }

    public void setBoxSize(int boxSize) {
        this.boxSize = boxSize;
    }
}
```

VehicleDao 完整代码如下：

```
package com.tarena.dao;

import java.util.List;

import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Vehicle;

@MyBatisRepository
public interface VehicleDao {

    List<Vehicle> findAll();

}
```

VehicleMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">
<mapper namespace="com.tarena.dao.VehicleDao">

    <resultMap id="vehicleMap" type="com.tarena.entity.Vehicle">
        <id property="id" column="ID" />
        <result property="color" column="COLOR" />
        <discriminator javaType="java.lang.String" column="TYPE">
            <case value="T" resultType="com.tarena.entity.Truck">
                <result property="boxSize" column="BOXSIZE" />
            </case>
            <case value="C" resultType="com.tarena.entity.Car">
                <result property="doorSize" column="DOORSIZE" />
            </case>
        </discriminator>
    </resultMap>

    <select id="findAll" resultMap="vehicleMap">
        select * from T CAR
    </select>

</mapper>
```

TestMapping 完整代码如下：

```
package com.tarena.test;

import java.sql.Date;
import java.util.List;
import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.dao.DeptDao;
import com.tarena.dao.EmpDao;
import com.tarena.dao.VehicleDao;
import com.tarena.entity.Dept;
import com.tarena.entity.Emp;
import com.tarena.entity.Vehicle;

/**
 * 测试MyBatis 关联映射
 */
public class TestMapping {

    /**
     * 主键映射：新增一条员工数据
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = new Emp();
        e.setEname("张三");
        e.setJob("SALESMAN");
        e.setMgr(3);
        e.setHiredate(
            new Date(System.currentTimeMillis()));
        e.setSal(4000.0);
        e.setComm(650.0);
        e.setDeptno(10);
        dao.save(e);
        System.out.println(e.getEmpno());
    }

    /**
     * 多对一嵌套查询映射：
     * 根据 ID 查询一条员工数据，并查询他所在的部门。
     */
    @Test
    public void test2() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpDao dao = ctx.getBean(EmpDao.class);
        Emp e = dao.findById(1);
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob() + " " +
            e.getDept().getDeptno() + " " +
            e.getDept().getDname()
        );
    }

    /**
     * 多对一嵌套结果映射：
     * 根据 ID 查询一条员工数据，并查询他所在的部门。
     */
}
```



```
@Test
public void test3() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    EmpDao dao = ctx.getBean(EmpDao.class);
    Emp e = dao.findById2(1);
    System.out.println(
        e.getEmpno() + " " +
        e.getEname() + " " +
        e.getJob() + " " +
        e.getDept().getDeptno() + " " +
        e.getDept().getDname()
    );
}

/**
 * 一对多嵌套查询映射:
 * 查询部门及部门下所有的员工。
 */
@Test
public void test4() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    DeptDao dao = ctx.getBean(DeptDao.class);
    Dept d = dao.findById(10);
    System.out.println(
        d.getDeptno() + " " +
        d.getDname() + " " +
        d.getLoc()
    );
    List<Emp> emps = d.getEmps();
    for(Emp e : emps) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 一对多嵌套结果映射:
 * 查询部门及部门下所有的员工。
 */
@Test
public void test5() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    DeptDao dao = ctx.getBean(DeptDao.class);
    Dept d = dao.findById2(10);
    System.out.println(
        d.getDeptno() + " " +
        d.getDname() + " " +
        d.getLoc()
    );
    List<Emp> emps = d.getEmps();
    for(Emp e : emps) {
        System.out.println(
            e.getEmpno() + " " +
            e.getEname() + " " +
            e.getJob()
        );
    }
}

/**
 * 鉴别器:
```

```
* 查询汽车表，根据类型封装成不同的对象。
*/
@Test
public void test6() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    VehicleDao dao = ctx.getBean(VehicleDao.class);
    List<Vehicle> list = dao.findAll();
    for (Vehicle v : list) {
        System.out.println(v);
    }
}
}
```

Spring + MyBatis

开发实战 Unit05

知识体系.....Page 119

Spring AOP	AOP 介绍	AOP 概念及优点
		什么是方面
		什么是目标
		什么是切入点
		什么是通知
		AOP 实现原理
	XML 配置实现 AOP	开发步骤
		前置通知
		环绕通知
		异常通知
	注解实现 AOP	开发步骤
		前置通知
		环绕通知
		异常通知


经典案例.....Page 125


Spring AOP 前置通知案例	前置通知
Spring AOP 环绕通知案例	环绕通知
Spring AOP 异常通知案例	异常通知
Spring AOP 注解使用案例	前置通知
	环绕通知
	异常通知

1. Spring AOP


1.1. AOP 介绍

1.1.1. 【AOP 介绍】AOP 概念及优点


<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">AOP概念及优点</h4> <ul style="list-style-type: none"> • AOP为Aspect Oriented Programming的缩写，被称为面向切面编程。 • AOP主要用于处理共通逻辑，例如日志记录，性能统计，安全控制，事务处理，异常处理等等。AOP可以将这些共通逻辑从普通业务逻辑代码中分离出来，这样在日后修改这些逻辑的时候就不会影响普通业务逻辑的代码。 • 利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。 <div style="text-align: right;">+</div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">AOP概念及优点（续1）</h4> <ul style="list-style-type: none"> • AOP、OOP在字面上虽然非常类似，但却是面向不同领域的两种设计思想。 • OOP（面向对象编程）针对业务处理过程的实体及其属性和行为进行抽象封装，以获得更加清晰高效的逻辑单元划分。 • AOP则是针对业务处理过程中的切面进行提取，它所面对的是处理过程中的某个步骤或阶段，以获得逻辑过程中各部分之间低耦合性的隔离效果。 • AOP需要以OOP为前提和基础。 <div style="text-align: right;">+</div>
---	---

1.1.2. 【AOP 介绍】什么是方面

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">什么是方面</h4> <ul style="list-style-type: none"> • 方面是指封装共通处理的组件，该组件被作用到其他目标组件方法上。 <div style="text-align: right;">+</div>
---	--


1.1.3. 【AOP 介绍】什么是目标



<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;"></div> <h4>什么是目标</h4> <ul style="list-style-type: none">• 目标是指被一个或多个方面所作用的对象。 <div style="position: absolute; left: 455px; top: 195px; background-color: black; color: white; padding: 2px 5px; writing-mode: vertical-rl;">知识讲解</div> <div style="text-align: right; margin-top: 20px;">++</div>
---	--

1.1.4. 【AOP 介绍】什么是切入点



<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;"></div> <h4>什么是切入点</h4> <ul style="list-style-type: none">• 切入点是用于指定哪些组件和方法使用方面功能，在Spring中利用一个表达式指定切入目标。• Spring提供了以下常用的切入点表达式<ul style="list-style-type: none">– 方法限定表达式 execution(修饰符? 返回类型 方法名(参数) throws 异常类型?)– 类型限定表达式 within(包名, 类型)– Bean名称限定表达式 bean("Bean的id或name属性值") <div style="position: absolute; left: 455px; top: 435px; background-color: black; color: white; padding: 2px 5px; writing-mode: vertical-rl;">知识讲解</div> <div style="text-align: right; margin-top: 20px;">++</div>
---	---

1.1.5. 【AOP 介绍】什么是通知

<div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div> <div style="border-bottom: 1px solid black; height: 15px; margin-bottom: 5px;"></div>	<div style="text-align: right;"></div> <h4>什么是通知</h4> <ul style="list-style-type: none">• 通知是用于指定方面组件和目标组件作用的时机。例如方面功能在目标方法之前或之后执行等时机。• Spring框架提供以下几种类型的通知：<ul style="list-style-type: none">– 前置通知：先执行方面功能再执行目标功能– 后置通知：先执行目标功能再执行方面功能（目标无异常才执行方面）– 最终通知：先执行目标功能再执行方面功能（目标有无异常都执行方面）– 异常通知：先执行目标，抛出后执行方面– 环绕通知：先执行方面前置部分，然后执行目标，最后再执行方面后置部分 <div style="position: absolute; left: 455px; top: 675px; background-color: black; color: white; padding: 2px 5px; writing-mode: vertical-rl;">知识讲解</div> <div style="text-align: right; margin-top: 20px;">++</div>
---	---



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>什么是通知（续1）</h3> <ul style="list-style-type: none"> Spring框架提供5种通知，可以按下面try-catch-finally结构理解。 <pre> try{ //前置通知--执行方面 //环绕通知前置部分 //执行目标组件方法 //环绕通知后置部分 //后置通知--执行方面 }catch(){ //异常通知--执行方面 }finally{ //最终通知--执行方面 } </pre> <div style="text-align: right;">  </div>
---	--

1.1.6. 【AOP 介绍】AOP 实现原理



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>AOP实现原理</h3> <ul style="list-style-type: none"> Spring AOP实现主要是基于动态代理技术。当Spring采用AOP配置后, Spring容器返回的目标对象，实质上是Spring利用动态代理技术生成一个代理类型。代理类重写了原目标组件方法的功能，在代理类中调用方面对象功能和目标对象功能。 Spring框架采用了两种动态代理实现： <ul style="list-style-type: none"> 利用cglib工具包 目标没有接口时采用此方法，代理类是利用继承方法生成一个目标子类。 利用JDK Proxy API 目标有接口时采用此方法，代理类是采用实现目标接口方法生成一个类。 <div style="text-align: right;">  </div>
---	--

1.2. XML 配置实现 AOP



1.2.1. 【XML 配置实现 AOP】开发步骤

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>开发步骤</h3> <ul style="list-style-type: none"> 创建方面组件 <ul style="list-style-type: none"> 创建一个类，充当方面组件，实现通用业务逻辑。 声明方面组件 <ul style="list-style-type: none"> 在applicationContext.xml中，声明方面组件。 使用方面组件 <ul style="list-style-type: none"> 在applicationContext.xml中，将方面组件作用到目标组件的方法上，并设置通知类型以确认方面组件调用的时机。 <div style="text-align: right;">  </div>
---	--

1.2.2. 【XML 配置实现 AOP】前置通知

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>前置通知</h4> <ul style="list-style-type: none"> 使用前置通知，方面组件中方法格式如下： <pre>public void log() { }</pre> applicationContext.xml中配置代码如下： <pre><aop:config> <aop:aspect ref="optLogger"> <aop:before method="log" pointcut="within(controller..*)"/> </aop:aspect> </aop:config></pre> 后置通知、最终通知使用方式与前置通知一致，只需将配置代码中的aop:before改为aop:after-returning、after即可。 <div style="text-align: right;">  </div>
---	---

1.2.3. 【XML 配置实现 AOP】环绕通知



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>环绕通知</h4> <ul style="list-style-type: none"> 使用环绕通知，方面组件中方法格式如下： <pre>public Object log(ProceedingJoinPoint p) throws Throwable { // 此处代码在目标组件前执行 Object obj = p.proceed(); // 执行目标组件方法 // 此处代码在目标组件后执行 return obj; }</pre> applicationContext.xml中配置代码如下： <pre><aop:aspect ref="optLogger"> <aop:around method="log" pointcut="within(controller..*)"/> </aop:aspect></pre> <div style="text-align: right;">  </div>
---	--

1.2.4. 【XML 配置实现 AOP】异常通知



<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4>异常通知</h4> <ul style="list-style-type: none"> 使用异常通知，方面组件中方法格式如下： <pre>public void log(Exception e) { }</pre> applicationContext.xml中配置代码如下： <pre><aop:aspect ref="optLogger"> <aop:after-throwing method="log" throwing="e" pointcut="within(controller..*)"/> </aop:aspect></pre> <div style="text-align: right;">  </div>
---	--

1.3. 注解实现 AOP

1.3.1. 【注解实现 AOP】开发步骤

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">开发步骤</h4> <ul style="list-style-type: none"> • 创建方面组件 <ul style="list-style-type: none"> - 创建一个类，充当方面组件，实现通用业务逻辑。 • 声明方面组件 <ul style="list-style-type: none"> - 在 applicationContext.xml 中开启 AOP 注解扫描： <pre><aop:aspectj-autoproxy proxy-target-class="true"/></pre> - 使用 @Component 注解标识这个类，将其声明为组件。 - 使用 @Aspect 注解标识这个类，将其声明为方面组件。 • 使用方面组件 <ul style="list-style-type: none"> - 在组件的方法上，使用注解将方面组件作用到目标组件的方法上，并设置通知类型以确认方面组件调用的时机。 <div style="text-align: right;">  </div>
---	---

1.3.2. 【注解实现 AOP】前置通知

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">前置通知</h4> <ul style="list-style-type: none"> • 使用前置通知，在方面组件方法上增加注解： <pre>@Before("within(controller..*)") public void log() { }</pre> • 后置通知、最终通知使用方式与前置通知一致，只需将注解改为 @AfterReturning、@After 即可。 <div style="text-align: right;">  </div>
---	--

1.3.3. 【注解实现 AOP】环绕通知

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">环绕通知</h4> <ul style="list-style-type: none"> • 使用环绕通知，在方面组件方法上增加注解： <pre>@Around("within(controller..*)") public Object log(ProceedingJoinPoint p) throws Throwable { // 此处代码在目标组件前执行 Object obj = p.proceed(); // 执行目标组件方法 // 此处代码在目标组件后执行 return obj; }</pre> <div style="text-align: right;">  </div>
---	---

1.3.4. 【注解实现 AOP】异常通知

异常通知

- 使用异常通知，在方面组件方法上增加注解：
`@AfterThrowing(pointcut="within(controller..*)",throwing="e")`
`public void log(Exception e) {`
`}`

代码编辑

经典案例

1. Spring AOP 前置通知案例

- **问题**

使用 Spring AOP 前置通知 ,在访问 Controller 中每个方法前 ,记录用户的操作日志。

- **方案**

Spring AOP 使用步骤 :

- 1) 创建方面组件
- 2) 声明方面组件
- 3) 将方面组件作用到目标组件上

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：创建 Controller

复制项目 SpringUnit04 ,创建新项目 SpringUnit05。

创建员工业务控制器 EmpController ,并实现员工查询 ,代码如下 :

```
package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/emp")
public class EmpController {

    /**
     * 查询员工
     */
    @RequestMapping("/findEmp.do")
    public String find() {
        // 模拟查询员工数据
        System.out.println("查询员工数据 , 发送至列表页面.");
        return "emp/emp_list.jsp";
    }
}
```

步骤二：创建方面组件

创建方面组件 OperateLogger ,并在该类中创建记录用户操作日志的方法 ,代码如下 :

```
package com.tarena.aspect;
```

```
import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
public class OperateLogger {

    /**
     * 前置通知、后置通知、最终通知使用的方法
     */
    public void log1() {
        // 记录日志
        System.out.println("-->记录用户操作信息");
    }
}
```

步骤三：声明方面组件

在 applicationContext.xml 中，声明该方面组件，关键代码如下：

```
<!-- 声明方面组件 -->
<bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>
```

步骤四：将方面组件作用到目标组件上

在 applicationContext.xml 中 将声明的方面组件作用到 com.tarena.controller 包下所有类的所有方法上，关键代码如下：

```
<!-- 声明方面组件 -->
<bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>

<!-- 配置 AOP -->
<aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)"/>
    </aop:aspect>
</aop:config>
```

步骤五：测试

创建 Junit 测试类 TestEmpController，并增加测试查询员工的方法，代码如下：

```
package com.tarena.test;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.controller.EmpController;

public class TestEmpController {
```

```
/**
 * 测试查询员工
 */
@Test
public void test1() {
    ApplicationContext ctx = new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    EmpController ctl = ctx.getBean(EmpController.class);
    ctl.find();
}
}
```

执行该测试方法，控制台输出效果如下图：



图-1

可见，在执行 `EmpController.find()` 方法之前，执行了方面组件的记录日志的方法，由于该方法采用 AOP 面向对象的思想实现的，因此不需要对 `Controller` 类做任何改动。

步骤六：扩展

后置通知、最终通知的用法与前置通知完全一致，只需要在配置 AOP 时将 `aop:before` 改为 `aop:after-returning` 和 `aop:after`。请自己尝试将前置通知类型改为后置通知、最终通知，并执行测试方法，观察控制台的输出情况。

• 完整代码

本案例的完整代码如下所示：

`EmpController` 完整代码如下：

```
package com.tarena.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/emp")
public class EmpController {

    /**
     * 查询员工
     */
    @RequestMapping("/findEmp.do")
    public String find() {
        // 模拟查询员工数据
        System.out.println("查询员工数据，发送至列表页面。");

        return "emp/emp_list.jsp";
    }
}
```

OperateLogger 完整代码如下：

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
public class OperateLogger {

    /**
     * 前置通知、后置通知、最终通知使用的方法
     */
    public void log1() {
        // 记录日志
        System.out.println("-->记录用户操作信息");
    }

}
```

applicationContext.xml 完整代码如下：

```
<?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"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="url"
            value="jdbc:oracle:thin:@localhost:1521:xe"/>
    </bean>
```

```

        <property name="driverClassName"
            value="oracle.jdbc.OracleDriver"/>
        <property name="username" value="lhk"/>
        <property name="password" value="123456"/>
    </bean>

    <!-- 配置 SqlSessionFactory -->
    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="ds" />
        <property name="mapperLocations"
            value="classpath:com/tarena/entity/*.xml"/>
    </bean>

    <!-- 配置 MyBatis 注解 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.tarena.dao" />
        <property name="annotationClass"
            value="com.tarena.annotation.MyBatisRepository"/>
    </bean>

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.tarena" />

    <!-- 支持 @RequestMapping 请求和 Controller 映射 -->
    <mvc:annotation-driven />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".jsp"/>
    </bean>

    <!-- 声明方面组件 -->
    <bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>

    <!-- 配置 AOP -->
    <aop:config>
        <aop:aspect ref="operateLogger">
            <aop:before method="log1"
                pointcut="within(com.tarena.controller..*)" />
        </aop:aspect>
    </aop:config>

</beans>

```

TestEmpController 完整代码如下：

```

package com.tarena.test;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.controller.EmpController;

public class TestEmpController {

    /**
     * 测试查询员工
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpController ctl = ctx.getBean(EmpController.class);
    }
}

```

```
        ctl.find();
    }

}
```

2. Spring AOP 环绕通知案例

- 问题

使用 Spring AOP 环绕通知 在访问 Controller 中每个方法前 记录用户的操作日志。

- 方案

Spring AOP 使用步骤：

- 1) 创建方面组件
- 2) 声明方面组件
- 3) 将方面组件作用到目标组件上

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建方面组件

复用方面组件 OperateLogger，在该类中创建新的记录日志的方法 log2，代码如下：

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
public class OperateLogger {

    //其他方法略

    /**
     * 环绕通知使用的方法
     */
    public Object log2(ProceedingJoinPoint p) throws Throwable {
        // 目标组件的类名
        String className = p.getTarget().getClass().getName();
        // 调用的方法名
        String method = p.getSignature().getName();
        // 当前系统时间
```

```
String date = new SimpleDateFormat(
    "yyyy-MM-dd hh:mm:ss").format(new Date());
// 拼日志信息
String msg = "-->用户在" + date + ", 执行了"
    + className + "." + method + "()";
// 记录日志
System.out.println(msg);

// 执行目标组件的方法
Object obj = p.proceed();

// 在调用目标组件业务方法后也可以做一些业务处理
System.out.println("-->调用目标组件业务方法后...");

return obj;
}
}
```

步骤二：声明方面组件

由于复用的方面组件已经声明，因此该步骤可以省略。

步骤三：将方面组件作用到目标组件上

在 applicationContext.xml 中，声明方面组件的 log2 方法，关键代码如下：

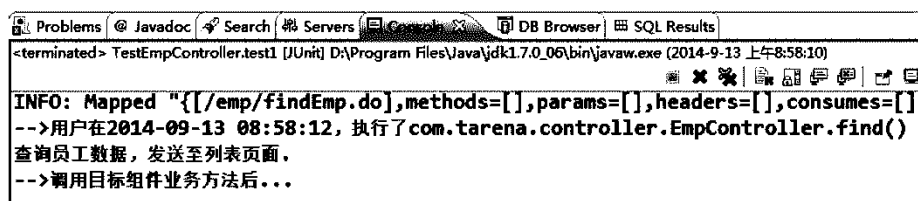
```
<!-- 声明方面组件 -->
<bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>

<!-- 配置 AOP -->
<aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>

    <aop:aspect ref="operateLogger">
        <aop:around method="log2"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
</aop:config>
```

步骤四：测试

执行测试方法 TestEmpController.test1()，控制台输出效果如下图：



```
<terminated> TestEmpController.test1 [JUnit] D:\Program Files\Java\jdk1.7.0_06\bin\javaw.exe (2014-9-13 上午8:58:10)
INFO: Mapped "[{/emp/findEmp.do],methods=[],params=[],headers=[],consumes=[]]
-->用户在2014-09-13 08:58:12, 执行了com.tarena.controller.EmpController.find()
查询员工数据，发送至列表页面。
-->调用目标组件业务方法后...
```

图-2

- **完整代码**

本案例的完整代码如下所示：

OperateLogger 完整代码如下：

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
public class OperateLogger {

    /**
     * 前置通知、后置通知、最终通知使用的方法
     */
    public void log1() {
        // 记录日志
        System.out.println("-->记录用户操作信息");
    }

    /**
     * 环绕通知使用的方法
     */
    public Object log2(ProceedingJoinPoint p) throws Throwable {
        // 目标组件的类名
        String className = p.getTarget().getClass().getName();
        // 调用的方法名
        String method = p.getSignature().getName();
        // 当前系统时间
        String date = new SimpleDateFormat(
            "yyyy-MM-dd hh:mm:ss").format(new Date());
        // 拼日志信息
        String msg = "-->用户在" + date + ", 执行了"
            + className + "." + method + "()";
        // 记录日志
        System.out.println(msg);

        // 执行目标组件的方法
        Object obj = p.proceed();

        // 在调用目标组件业务方法后也可以做一些业务处理
        System.out.println("-->调用目标组件业务方法后...");

        return obj;
    }
}
```

applicationContext.xml 完整代码如下：

```
<?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"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

<!-- 配置数据源 -->
<bean id="ds"
    class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="url"
        value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <property name="driverClassName"
        value="oracle.jdbc.OracleDriver"/>
    <property name="username" value="lhh"/>
    <property name="password" value="123456"/>
</bean>

<!-- 配置 SqlSessionFactory -->
<bean id="sqlSessionFactory"
    class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="ds" />
    <property name="mapperLocations"
        value="classpath:com/tarena/entity/*.xml"/>
</bean>

<!-- 配置 MyBatis 注解 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.tarena.dao" />
    <property name="annotationClass"
        value="com.tarena.annotation.MyBatisRepository"/>
</bean>

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 支持@RequestMapping 请求和 Controller 映射 -->
<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp"/>
</bean>

<!-- 声明方面组件 -->
<bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>

```

```
<!-- 配置 AOP -->
<aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:around method="log2"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
</aop:config>

</beans>
```

3. Spring AOP 异常通知案例

- 问题

使用 Spring AOP 异常通知，在每个 Controller 的方法发生异常时，记录异常日志。

- 方案

Spring AOP 使用步骤：

- 1) 创建方面组件
- 2) 声明方面组件
- 3) 将方面组件作用到目标组件上

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：创建方面组件

复用方面组件 OperateLogger，在该类中创建新的记录日志的方法 log3，代码如下：

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
public class OperateLogger {

    //其他方法略

}
```

```

    * 异常通知使用的方法
    */
    public void log3(Exception e) {
        StackTraceElement[] elems = e.getStackTrace();
        // 将异常信息记录
        System.out.println("-->" + elems[0].toString());
    }
}

```

步骤二：声明方面组件

由于复用的方面组件已经声明，因此该步骤可以省略。

步骤三：将方面组件作用到目标组件上

在 applicationContext.xml 中，声明方面组件的 log3 方法，关键代码如下：

```

<!-- 声明方面组件 -->
<bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>

<!-- 配置 AOP -->
<aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:around method="log2"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>

    <aop:aspect ref="operateLogger">
        <aop:after-throwing method="log3" throwing="e"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
</aop:config>

```

步骤四：测试

为了便于测试，在 EmpController.find()方法中制造一个异常，代码如下：

```

package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/emp")
public class EmpController {

    /**
     * 查询员工
     */
    @RequestMapping("/findEmp.do")
    public String find() {
        // 模拟查询员工数据
    }
}

```

```
System.out.println("查询员工数据，发送至列表页面。");
```

```
// 制造一个异常，便于测试异常通知
Integer.valueOf("abc");
```

```
return "emp/emp_list.jsp";
```

执行测试方法 TestEmpController.test1()，控制台输出效果如下图：

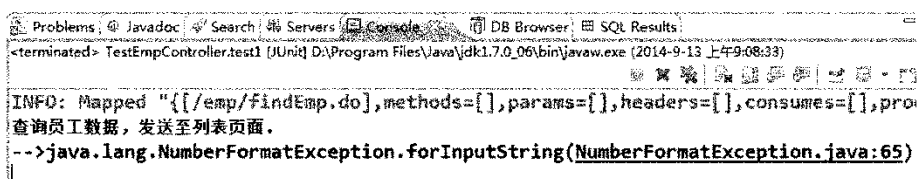


图-3

• 完整代码

本案例的完整代码如下所示：

OperateLogger 完整代码如下：

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
public class OperateLogger {

    /**
     * 前置通知、后置通知、最终通知使用的方法
     */
    public void log1() {
        // 记录日志
        System.out.println("-->记录用户操作信息");
    }

    /**
     * 环绕通知使用的方法
     */
    public Object log2(ProceedingJoinPoint p) throws Throwable {
        // 目标组件的类名
        String className = p.getTarget().getClass().getName();
        // 调用的方法名
        String method = p.getSignature().getName();
        // 当前系统时间
```

```
String date = new SimpleDateFormat(
    "yyyy-MM-dd hh:mm:ss").format(new Date());
// 拼日志信息
String msg = "-->用户在" + date + ", 执行了"
    + className + "." + method + "()";
// 记录日志
System.out.println(msg);

// 执行目标组件的方法
Object obj = p.proceed();

// 在调用目标组件业务方法后也可以做一些业务处理
System.out.println("-->调用目标组件业务方法后...");

return obj;
}

/**
 * 异常通知使用的方法
 */
public void log3(Exception e) {
    StackTraceElement[] elems = e.getStackTrace();
    // 将异常信息记录
    System.out.println("-->" + elems[0].toString());
}
}
```

applicationContext.xml 完整代码如下：

```
<?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"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="url"
            value="jdbc:oracle:thin:@localhost:1521:xe"/>
    </bean>
</beans>
```

```

    <property name="driverClassName"
        value="oracle.jdbc.OracleDriver"/>
    <property name="username" value="lhh"/>
    <property name="password" value="123456"/>
</bean>

<!-- 配置 SqlSessionFactory -->
<bean id="sqlSessionFactory"
    class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="ds" />
    <property name="mapperLocations"
        value="classpath:com/tarena/entity/*.xml"/>
</bean>

<!-- 配置 MyBatis 注解 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.tarena.dao" />
    <property name="annotationClass"
        value="com.tarena.annotation.MyBatisRepository"/>
</bean>

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 支持 @RequestMapping 请求和 Controller 映射 -->
<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp"/>
</bean>

<!-- 声明方面组件 -->
<bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/>

<!-- 配置 AOP -->
<aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:around method="log2"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:after-throwing method="log3" throwing="e"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
</aop:config>

</beans>

```

EmpController 完整代码如下：

```

package com.tarena.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/emp")
public class EmpController {

```

```
/**
 * 查询员工
 */
@RequestMapping("/findEmp.do")
public String find() {
    // 模拟查询员工数据
    System.out.println("查询员工数据，发送至列表页面。");
    // 制造一个异常，便于测试异常通知
    Integer.valueOf("abc");
    return "emp/emp_list.jsp";
}
}
```

4. Spring AOP 注解使用案例

• 问题

使用 Spring AOP 注解替代 XML 配置，重构上面的 3 个案例。

• 方案

Spring AOP 相关注解及含义如下：

- @Aspect：用于声明方面组件
- @Before：用于声明前置通知
- @AfterReturning：用于声明后置通知
- @After：用于声明最终通知
- @Around：用于声明环绕通知
- @AfterThrowing：用于声明异常通知

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：开启 AOP 注解扫描

在 applicationContext.xml 中，去掉方面组件声明及作用的 XML 配置，并开启 AOP 注解扫描，关键代码如下：

```
<!-- 声明方面组件 -->
<!-- <bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/> -->

<!-- 配置 AOP -->
<!-- <aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)"/>
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:around method="log2"
            pointcut="within(com.tarena.controller..*)"/>
    </aop:aspect>
</aop:config>
```



```
<aop:aspect ref="operateLogger">
    <aop:after-throwing method="log3" throwing="e"
        pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
</aop:config> -->

<!-- 开启 AOP 注解扫描 -->
<aop:aspectj-autoproxy proxy-target-class="true"/>
```

步骤二：使用注解声明方面组件

在 OperateLogger 中,使用@Aspect 注解声明方面组件,并分别用@Before、@Around、@AfterThrowing 注解声明 log1、log2、log3 方法,将方面组件作用到目标组件上,代码如下:

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件,演示 Spring AOP 的各种通知类型。
 */

@Component
@Aspect

public class OperateLogger {

    /**
     * 前置通知、后置通知、最终通知使用的方法
     */

    @Before("within(com.tarena.controller..*)")

    public void log1() {
        // 记录日志
        System.out.println("-->记录用户操作信息");
    }

    /**
     * 环绕通知使用的方法
     */

    @Around("within(com.tarena.controller..*)")

    public Object log2(ProceedingJoinPoint p) throws Throwable {
        // 目标组件的类名
        String className = p.getTarget().getClass().getName();
        // 调用的方法名
        String method = p.getSignature().getName();
    }
}
```

```
// 当前系统时间
String date = new SimpleDateFormat(
    "yyyy-MM-dd hh:mm:ss").format(new Date());

// 拼日志信息
String msg = "-->用户在" + date + ", 执行了"
    + className + "." + method + "{}";

// 记录日志
System.out.println(msg);

// 执行目标组件的方法
Object obj = p.proceed();

// 在调用目标组件业务方法后也可以做一些业务处理
System.out.println("-->调用目标组件业务方法后...");

return obj;
}

/**
 * 异常通知使用的方法
 */

@AfterThrowing(pointcut="within(com.tarena.controller.*)",throwing="e")

public void log3(Exception e) {
    StackTraceElement[] elems = e.getStackTrace();
    // 将异常信息记录
    System.out.println("-->" + elems[0].toString());
}
}
```

步骤三：测试

执行测试方法 TestEmpController.test1()，控制台输出效果如下图：

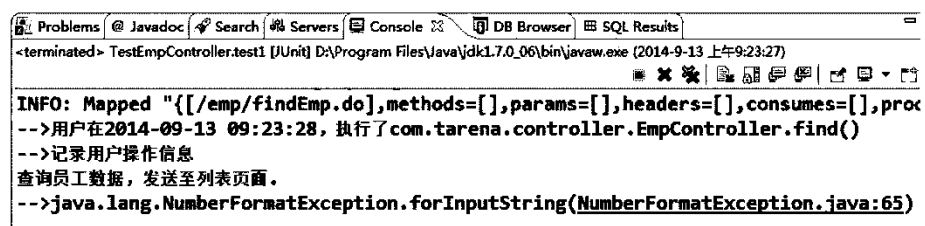


图-4

• 完整代码

本案例的完整代码如下所示：

applicationContext.xml 完整代码如下：

```
<?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"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
```

```

xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

<!-- 配置数据源 -->
<bean id="ds"
    class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <property name="url"
        value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <property name="driverClassName"
        value="oracle.jdbc.OracleDriver"/>
    <property name="username" value="lhh"/>
    <property name="password" value="123456"/>
</bean>

<!-- 配置 SqlSessionFactory -->
<bean id="sqlSessionFactory"
    class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="ds" />
    <property name="mapperLocations"
        value="classpath:com/tarena/entity/*.xml"/>
</bean>

<!-- 配置 MyBatis 注解 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.tarena.dao" />
    <property name="annotationClass"
        value="com.tarena.annotation.MyBatisRepository"/>
</bean>

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 支持@RequestMapping 请求和 Controller 映射 -->
<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp"/>
</bean>

<!-- 声明方面组件 -->
<!-- <bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/> -->

<!-- 配置 AOP -->
<!-- <aop:config>

```

```
<aop:aspect ref="operateLogger">
    <aop:before method="log1"
        pointcut="within(com.tarena.controller..*)" />
</aop:aspect>
<aop:aspect ref="operateLogger">
    <aop:around method="log2"
        pointcut="within(com.tarena.controller..*)" />
</aop:aspect>
<aop:aspect ref="operateLogger">
    <aop:after-throwing method="log3" throwing="e"
        pointcut="within(com.tarena.controller..*)" />
</aop:aspect>
</aop:config> -->

<!-- 开启 AOP 注解扫描 -->
<aop:aspectj-autoproxy proxy-target-class="true"/>

</beans>
```

OperateLogger 完整代码如下：

```
package com.tarena.aspect;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

/**
 * 用于记录日志的方面组件，演示 Spring AOP 的各种通知类型。
 */
@Component
@Aspect
public class OperateLogger {

    /**
     * 前置通知、后置通知、最终通知使用的方法
     */
    @Before("within(com.tarena.controller..*)")
    public void log1() {
        // 记录日志
        System.out.println("-->记录用户操作信息");
    }

    /**
     * 环绕通知使用的方法
     */
    @Around("within(com.tarena.controller..*)")
    public Object log2(ProceedingJoinPoint p) throws Throwable {
        // 目标组件的类名
        String className = p.getTarget().getClass().getName();
        // 调用的方法名
        String method = p.getSignature().getName();
        // 当前系统时间
        String date = new SimpleDateFormat(
            "yyyy-MM-dd hh:mm:ss").format(new Date());
        // 拼日志信息
        String msg = "-->用户在" + date + ", 执行了"
            + className + "." + method + "()";
        // 记录日志
    }
}
```

```
System.out.println(msg);

// 执行目标组件的方法
Object obj = p.proceed();

// 在调用目标组件业务方法后也可以做一些业务处理
System.out.println("-->调用目标组件业务方法后...");

return obj;
}

/**
 * 异常通知使用的方法
 */
@AfterThrowing(pointcut="within(com.tarena.controller..*)",throwing="e")
public void log3(Exception e) {
    StackTraceElement[] elems = e.getStackTrace();
    // 将异常信息记录
    System.out.println("-->" + elems[0].toString());
}
}
```

Spring + MyBatis

开发实战 Unit06

知识体系.....Page 146

Spring 事务处理	Spring 事务介绍	Spring 事务简介
		编程式事务
		声明式事务
	声明式事务	注解实现声明式事务
		XML 配置实现声明式事务
	Spring 事务控制	事务回滚
		事务传播
		事务隔离级别
Spring 与 RESTful	Spring 与 RESTful	RESTful 简介
		Spring 对 RESTful 的支持
		@RequestMapping 应用
		@PathVariable 应用
		客户端发送 PUT、DELETE 请求
		静态资源访问处理

经典案例.....Page 155

Spring 声明式事务-注解应用	注解实现声明式事务
Spring 声明式事务-XML 配置	XML 配置实现声明式事务
Spring 声明式事务-回滚机制	事务回滚
RESTful 应用案例	@RequestMapping 应用
	@PathVariable 应用
	客户端发送 PUT、DELETE 请求
	静态资源访问处理

1. Spring 事务处理

1.1. Spring 事务介绍


1.1.1. 【Spring 事务介绍】Spring 事务简介


<div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div><div>Spring事务简介</div><div><div>Tarena 达内科技</div></div><div><ul style="list-style-type: none">• Spring框架引入的重要因素之一是它全面的事务支持。Spring框架提供了一致的事务管理方式，给程序带来以下好处：<ul style="list-style-type: none">– 提供简单易用的编程式事务管理API– 支持声明式事务管理– 便于Spring整合各种数据访问技术</div><div><div>知识扩展</div><div></div></div><div><div>+</div><div>+</div></div></div>
---	---

1.1.2. 【Spring 事务介绍】编程式事务


<div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div><div>编程式事务</div><div><div>Tarena 达内科技</div></div><div><ul style="list-style-type: none">• 使用编程式事务时，Spring提供了以下两种事务管理的API<ul style="list-style-type: none">– 使用 TransactionTemplate– 直接使用一个 PlatformTransactionManager 实现• 如果采用编程式事务管理，推荐使用TransactionTemplate</div><div><div>知识扩展</div><div></div></div><div><div>+</div><div>+</div></div></div>
---	---

<div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div> <div></div>	<div><div>编程式事务（续1）</div><div><div>Tarena 达内科技</div></div><div><ul style="list-style-type: none">• TransactionTemplate与Spring中JdbcTemplate等模板类使用风格相似，它也使用回调机制，将事务代码和业务代码分离，这样便于开发者把精力集中在具体的业务编程上。</div><div><div>知识扩展</div><div></div></div><div><div>+</div><div>+</div></div></div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>编程式事务（续2）</h3> <ul style="list-style-type: none"> TransactionTemplate应用的示例代码如下： <pre> public class SimpleService implements Service { private final TransactionTemplate transactionTemplate; public SimpleService(PlatformTransactionManager tm) { this.transactionTemplate = new TransactionTemplate(tm); } public Object someServiceMethod() { return transactionTemplate.execute(new TransactionCallback() { public Object doInTransaction(TransactionStatus status) { updateOperation1(); return resultOfUpdateOperation2(); } }); } } </pre> <div style="text-align: right;">++</div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>编程式事务（续3）</h3> <ul style="list-style-type: none"> 如果不需要返回值，可以创建一个 TransactionCallbackWithoutResult的匿名类，示例代码如下： <pre> transactionTemplate.execute(new TransactionCallbackWithoutResult() { protected void doInTransactionWithoutResult(TransactionStatus status) { try { updateOperation1(); updateOperation2(); } catch (SomeBusinessException ex) { status.setRollbackOnly(); } } }); </pre> <div style="text-align: right;">++</div>
---	---

1.1.3. 【Spring 事务介绍】声明式事务

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3>声明式事务</h3> <ul style="list-style-type: none"> Spring的声明式事务管理是通过Spring AOP实现的。使用时不需要修改原有的业务代码，只需通过简单配置就可以追加事务控制功能。 大多数Spring用户选择声明式事务管理。这是对程序代码影响最小的选择，因此也最符合非侵入式的理念。 <div style="text-align: right;">++</div>
---	---


1.2. 声明式事务

1.2.1. 【声明式事务】注解实现声明式事务

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">注解实现声明式事务</h4> <ul style="list-style-type: none"> 步骤一，在applicationContext.xml中声明事务组件，开启事务注解扫描，示例代码如下： <pre> <!-- 声明事务管理组件 --> <bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"> <property name="dataSource" ref="ds"/> </bean> <!-- 开启事务注解扫描 --> <tx:annotation-driven transaction-manager="txManager" proxy-target-class="true"/> </pre> <ul style="list-style-type: none"> transaction-manager指定的事务管理器txManager，需要根据数据库访问技术不同选择不同的实现。例如JDBC、MyBatis技术选用DataSourceTransactionManager，而Hibernate技术则选用HibernateTransactionManager。 <div style="text-align: right;">  </div>
---	--

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">注解实现声明式事务（续1）</h4> <ul style="list-style-type: none"> 步骤二，使用@Transactional注解声明事务，使用方法如下： <pre> @Transactional public class DefaultFooService implements FooService { // @Transactional public void insertFoo(Foo foo){...} public void updateFoo(Foo foo){...} } </pre> <ul style="list-style-type: none"> @Transactional注解标记可以用在类定义和方法定义前，方法的事务设置将优先于类级别注解的事务设置。 <div style="text-align: right;">  </div>
---	---

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h4 style="text-align: center;">注解实现声明式事务（续2）</h4> <ul style="list-style-type: none"> @Transactional注解标记有以下属性，在使用时可以根据需要做特殊设定。 <ul style="list-style-type: none"> - propagation：设置事务传播 - isolation：设置事务隔离级别 - readOnly：设置为只读，还是可读写 - rollbackFor：设置遇到哪些异常必须回滚 - noRollbackFor：设置遇到哪些异常不回滚 <div style="text-align: right;">  </div>
---	---




注解实现声明式事务（续3）

- @Transactional注解属性默认设置如下：
 - 事务传播设置是 PROPAGATION_REQUIRED
 - 事务隔离级别是 ISOLATION_DEFAULT
 - 事务是 读/写
 - 事务超时默认是依赖于事务系统的，或者事务超时没有被支持。
 - 任何 RuntimeException 将触发事务回滚，但是任何 Checked Exception 将不触发事务回滚

知识讲解

+

1.2.2. 【声明式事务】XML 配置实现声明式事务



XML配置实现声明式事务

- 在applicationContext.xml中声明事务组件，配置事务作用的范围及类型，示例代码如下：

```


<!-- 声明事务管理组件 -->
<bean id="txManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="ds"/>
</bean>
<!-- XML配置声明事务范围及类型 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="add*" propagation="REQUIRED" />
        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>
<aop:config proxy-target-class="true">
    <aop:advisor advice-ref="txAdvice" pointcut="within(controller..*)" />
</aop:config>
            
```

知识讲解

+

1.3. Spring 事务控制

1.3.1. 【Spring 事务控制】事务回滚



事务回滚

- @Transactional默认情况下RuntimeException异常将触发事务回滚，但是任何Checked Exception将不触发事务回滚
- 常见的RuntimeException和Checked Exception如表所示：

RuntimeException	Checked Exception
ArithmeticException	ClassNotFoundException
NullPointerException	NoSuchMethodException
ClassCastException	NoSuchFieldException
NumberFormatException	CloneNotSupportedException
IndexOutOfBoundsException	IOException
NegativeArraySizeException	
UnsupportedOperationException	

知识讲解

+

Tarena 达内科技

事务回滚（续1）

- 对于Checked Exception，需要手动指定异常类型，才能实现事务回滚。
- 使用注解实现声明式事务时，按如下方式指定异常：
@Transactional(rollbackFor=ClassNotFoundException.class)
- 使用XML配置实现声明式事务时，按如下方式指定异常：
`<tx:method name="update" propagation="REQUIRED" rollback-for="java.lang.ClassNotFoundException"/>`

+

知识拓展

Tarena 达内科技

事务回滚（续2）

- 当使用自定义异常时，异常类必须从RuntimeException继承才能自动回滚事务，否则需要指定事务回滚的异常类型

+

知识拓展

1.3.2. 【Spring 事务控制】事务传播

Tarena 达内科技

事务传播

- 事务传播是指一个方法调用了另一个带有事务控制的方法，这种复杂情况就需要指定事务传播的处理方案
- Spring中事务的传播类型有以下几种：

事务传播类型	作用描述
REQUIRED	支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择
SUPPORTS	支持当前事务，如果当前没有事务，就以非事务方式执行
MANDATORY	支持当前事务，如果当前没有事务，就抛出异常
REQUIRES_NEW	新建事务，如果当前存在事务，把当前事务挂起
NOT_SUPPORTED	以非事务方式执行操作，如果当前存在事务，就把当前事务挂起
NEVER	以非事务方式执行，如果当前存在事务，则抛出异常
NESTED	如果当前存在事务，则在嵌套事务内执行。拥有多个可以回滚的保存点，内部回滚不会对外部事务产生影响。只对DataSourceTransactionManager有效

+

知识拓展

代码讲解


事务传播（续1）


• 利用@Transactional注解的propagation属性可以控制事务的传播行为，示例代码如下：

```
@Transactional(propagation=Propagation.REQUIRED)
public void f1(int id){

    //业务处理逻辑

}
```






1.3.3. 【Spring 事务控制】事务隔离级别


代码讲解

事务隔离级别

• 在读取数据库的过程中，如果两个事务并发执行，那么彼此之间的数据会发生影响。为了避免这种并发冲突，需要将两个事务隔离开，根据隔离程度不同分为以下几种级别：

隔离级别	描述
READ_UNCOMMITTED	这是事务最低的隔离级别，它允许令外一个事务可以看到这个事务未提交的数据
READ_COMMITTED	大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，该级别适用于大多数系统
REPEATABLE_READ	保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据
SERIALIZABLE	这是花费代价最高,但最可靠的事务隔离级别。事务被处理为顺序执行。除了防止脏读，不可重复读外，还避免了幻读
DEFAULT	默认的隔离级别，使用数据库默认的事务隔离级别





代码讲解


事务隔离级别（续1）

• 利用@Transactional注解的isolation属性可以控制事务的隔离级别，示例代码如下

```
@Transactional(isolation=Isolation.READ_COMMITTED)
public void f1(int id){

    //业务处理逻辑

}
```





2. Spring 与 RESTful

2.1. Spring 与 RESTful

2.1.1. 【Spring 与 RESTful】RESTful 简介



RESTful简介

- 什么是REST ?
 - REST是英文Representational State Transfer的缩写，中文翻译为“表述性状态转移”，他是由Roy Thomas Fielding博士在他的论文 中提出的一个术语。
 - REST本身只是为分布式超媒体系统设计的一种架构风格，适合开发Web Service服务，并没有任何技术和语言上的限制。
 - REST软件架构之所以适合超媒体系统，是因为它可以把网络上所有资源进行唯一的URL定位，资源类型没有限制，例如图片、Word文件、视频文件，txt文件、xml文件、html文件等。

项目来源






RESTful简介（续1）

- REST架构是一个抽象的概念，目前主要是基于HTTP协议实现，其目的是为了提高系统的可伸缩性、降低应用之间的耦合度、便于架构分布式处理程序。
- REST主要对以下两个方面进行了规范
 - 定位资源的URL风格，例如
<http://tarena.com/customers/1234>
<http://tarena.com/orders/2007/10/776654>
 - 如何对资源操作
 采用HTTP协议规定的GET、POST、PUT、DELETE动作处理资源的增删改查操作

项目来源






RESTful简介（续2）

Method	CRUD
POST	Create update, delete
GET	Read
PUT	Update create
DELETE	Delete

项目来源



知识技能

RESTful简介 (续3)

Tarena 达内科技

- 什么是RESTful ?
 - 符合REST约束风格和原则的应用程序或设计就是RESTful

```

/blog/1 HTTP GET => 查询id=1的blog
/blog/1 HTTP DELETE => 删除id=1的blog
/blog/1 HTTP PUT => 更新blog
/blog/add HTTP POST => 新增BLOG
          
```

++

2.1.2. 【Spring 与 RESTful】Spring 对 RESTful 的支持

知识技能

Spring对RESTful的支持

Tarena 达内科技

- Spring MVC对RESTful应用提供了以下支持
 - 利用@RequestMapping指定要处理请求的URI模板和HTTP请求的动作类型
 - 利用@PathVariable将URI请求模板中的变量映射到处理方法的参数上
 - 利用Ajax, 在客户端发出PUT、DELETE动作的请求

++

2.1.3. 【Spring 与 RESTful】@RequestMapping 应用

知识技能

@RequestMapping应用

Tarena 达内科技

- @RequestMapping可以定义在Controller类前和处理方法前, 主要用于指定Controller的方法处理哪些请求。
- 在RESTful应用中, @RequestMapping可以采用以下使用格式:

```



@RequestMapping(value="/emp/{id}", method = RequestMethod.GET)
@RequestMapping (value="/emp/add", method = RequestMethod.POST)
@RequestMapping(value="/emp/{id}", method = RequestMethod.DELETE)
@RequestMapping(value="/emp/{id}", method = RequestMethod.PUT)
          
```

++

2.1.4. 【Spring 与 RESTful】@PathVariable 应用

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3 style="text-align: center;">@PathVariable应用</h3> <ul style="list-style-type: none"> • @PathVariable作用是将URI请求模板中的变量解析出来，映射到处理方法的参数上，使用示例如下： <pre> @RequestMapping(value="/emp/{id}",method=RequestMethod.GET) public String execute(@PathVariable("id") int id){ //查询操作处理 return ""; } </pre> <ul style="list-style-type: none"> • 上述URI请求模板匹配/emp/1、emp/2等格式请求 <div style="text-align: right;">  </div>
---	--

2.1.5. 【Spring 与 RESTful】客户端发送 PUT、DELETE 请求

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3 style="text-align: center;">客户端发送PUT、DELETE请求</h3> <ul style="list-style-type: none"> • 可以采用Ajax方式发送PUT和DELETE请求，示例如下 <pre> \$.ajax({ type:"PUT", url:"/SpringRestful/emp/update", data:JSON.stringify(\$("#myform").serializeObject()), dataType:"json", contentType:"application/json", success:function(data){ location.href = "/SpringRestful/emp/find"; } }); </pre> <div style="text-align: right;">  </div>
---	--

2.1.6. 【Spring 与 RESTful】静态资源访问处理

<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	<div style="text-align: right;">  </div> <h3 style="text-align: center;">静态资源访问处理</h3> <ul style="list-style-type: none"> • 采用RESTful架构后，需要将web.xml中控制器拦截的请求设置为/，这样会将css.js等静态资源进行拦截，发生404错误。 • 解决上述问题的方法如下： <ul style="list-style-type: none"> – 配置<mvc:resources/>，使用示例如下 <pre> <mvc:resources mapping="/请求URI" location="/资源位置" /> </pre> <ul style="list-style-type: none"> – 配置<mvc:default-servlet-handler/>，使用示例如下 <pre> <mvc:default-servlet-handler/> </pre> <div style="text-align: right;">  </div>
---	---

经典案例

1. Spring 声明式事务-注解应用

- 问题

使用 Spring 注解实现声明式事务，管理 Controller 中各业务方法的事务。

- 方案

Spring 事务注解使用步骤：

- 1) 声明事务组件
- 2) 开启事务注解扫描
- 3) 使用注解标识类/方法

- 步骤

实现此案例需要按照如下步骤进行。

步骤一：测试未管理事务的情况

复制项目 SpringUnit05，创建新项目 SpringUnit06。

在 EmpController 中，增加批量添加员工的方法，并模拟实现，代码如下：

```
package com.tarena.controller;

import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.RequestMapping;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
public class EmpController {

    @Resource
    private EmpDao empDao;

    //其他方法略

    /**
     * 模拟批量添加员工
     */
    @RequestMapping("/addEmps.do")
    public String addBatch() {
        // 插入第一个员工
        Emp e1 = new Emp();
        e1.setEname("刘备");
```



```
e1.setJob("皇叔");
e1.setSal(1000.0);
e1.setEmpno(10);
empDao.save(e1);

// 模拟异常
Integer.valueOf("abc"); //ClassCastException

// 插入第二个员工
Emp e2 = new Emp();
e2.setEname("关羽");
e2.setJob("候");
e2.setSal(1000.0);
e2.setEmpno(10);
empDao.save(e2);

return "redirect:findEmp.do";
}

}
```

在 TestEmpController 中，增加测试方法，测试批量添加员工，代码如下：

```
package com.tarena.test;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.controller.EmpController;

public class TestEmpController {

    //其他方法略

    /**
     * 测试批量添加员工
     */
    @Test
    public void test2() throws ClassNotFoundException {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpController ctl = ctx.getBean(EmpController.class);
        ctl.addBatch();
    }

}
```

没有执行该测试方法前，员工表中的数据如下图：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12 00:00:00.0	800	<NULL>	20
2	ALLEN	SALESMAN	3	1981-06-03 00:00:00.0	1600	300	30
3	WARD	SALESMAN	4	1990-03-15 00:00:00.0	1250	500	30
4	JONES	MANAGER	5	1985-04-08 00:00:00.0	2975	<NULL>	20
5	MARTIN	SALESMAN	7	1986-03-08 00:00:00.0	1250	1400	30
6	BLAKE	MANAGER	9	1989-06-01 00:00:00.0	2850	<NULL>	30
7	CLARK	MANAGER	9	1995-10-01 00:00:00.0	2450	<NULL>	10
8	SCOTT	ANALYST	9	1993-05-01 00:00:00.0	3000	<NULL>	20
9	KING	PRESIDENT	<NULL>	1988-08-08 00:00:00.0	5000	<NULL>	10
10	TURNER	SALESMAN	5	1983-02-01 00:00:00.0	1500	0	30
11	ADAMS	CLERK	5	1992-07-03 00:00:00.0	1100	<NULL>	20
12	JAMES	CLERK	1	1996-09-10 00:00:00.0	950	<NULL>	30
13	FORD	ANALYST	1	1993-01-01 00:00:00.0	3000	<NULL>	20
14	Lee	CLERK	3	1983-10-09 00:00:00.0	1300	<NULL>	10

图-1

执行该测试方法后，员工表中的数据如下图：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12 00:00:00.0	800	<NULL>	20
2	ALLEN	SALESMAN	3	1981-06-03 00:00:00.0	1600	300	30
3	WARD	SALESMAN	4	1990-03-15 00:00:00.0	1250	500	30
4	JONES	MANAGER	5	1985-04-08 00:00:00.0	2975	<NULL>	20
5	MARTIN	SALESMAN	7	1986-03-08 00:00:00.0	1250	1400	30
6	BLAKE	MANAGER	9	1989-06-01 00:00:00.0	2850	<NULL>	30
7	CLARK	MANAGER	9	1995-10-01 00:00:00.0	2450	<NULL>	10
8	SCOTT	ANALYST	9	1993-05-01 00:00:00.0	3000	<NULL>	20
9	KING	PRESIDENT	<NULL>	1988-08-08 00:00:00.0	5000	<NULL>	10
10	TURNER	SALESMAN	5	1983-02-01 00:00:00.0	1500	0	30
11	ADAMS	CLERK	5	1992-07-03 00:00:00.0	1100	<NULL>	20
12	JAMES	CLERK	1	1996-09-10 00:00:00.0	950	<NULL>	30
13	FORD	ANALYST	1	1993-01-01 00:00:00.0	3000	<NULL>	20
14	Lee	CLERK	3	1983-10-09 00:00:00.0	1300	<NULL>	10
433	刘备	老板	<NULL>	<NULL>	1000	<NULL>	<NULL>

图-2

可见，员工刘备添加成功。但 EmpController.addBatch()方法是业务方法，体现了完整的业务逻辑，发生异常时应该整个业务失败，而不是一部分成功一部分失败，因此添加刘备成功是不符合要求的，应该引入事务管理机制来解决此问题。

步骤二：声明事务组件

在 applicationContext.xml 中声明事务组件，关键代码如下：

```
<!-- 声明事务管理组件 -->
<bean id="txManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="ds"/>
</bean>
```

步骤三：开启事务注解扫描

在 applicationContext.xml 中，开启事务注解扫描，关键代码如下：

```
<!-- 声明事务管理组件 -->
<bean id="txManager"
```

```
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="ds"/>
</bean>
```

```
<!-- 开启事务注解扫描 -->
```

```
<tx:annotation-driven
    transaction-manager="txManager" proxy-target-class="true"/>
```

步骤四：使用注解标识类/方法

在 EmpController 中，使用注解对该方法启用事务管理，代码如下：

```
package com.tarena.controller;

import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.RequestMapping;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")

@Transactional

public class EmpController {
    @Resource
    private EmpDao empDao;

    //其他方法略

    /**
     * 模拟批量添加员工
     */
    @RequestMapping("/addEmps.do")
    public String addBatch() {
        // 插入第一个员工
        Emp e1 = new Emp();
        e1.setName("刘备");
        e1.setJob("皇叔");
        e1.setSal(1000.0);
        e1.setEmpno(10);
        empDao.save(e1);

        // 模拟异常
        Integer.valueOf("abc"); //ClassCastException

        // 插入第二个员工
        Emp e2 = new Emp();
        e2.setName("关羽");
        e2.setJob("候");
        e2.setSal(1000.0);
        e2.setEmpno(10);
        empDao.save(e2);

        return "redirect:findEmp.do";
    }
}
```

步骤五：测试

手动删除员工表中的刘备，执行测试方法 `TestEmpController.test2()`，然后查询员工表数据如下图：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12 00:00:00.0	800	<NULL>	20
2	ALLEN	SALESMAN	3	1981-06-03 00:00:00.0	1600	300	30
3	WARD	SALESMAN	4	1990-03-15 00:00:00.0	1250	500	30
4	JONES	MANAGER	5	1985-04-08 00:00:00.0	2975	<NULL>	20
5	MARTIN	SALESMAN	7	1986-03-08 00:00:00.0	1250	1400	30
6	BLAKE	MANAGER	9	1989-06-01 00:00:00.0	2850	<NULL>	30
7	CLARK	MANAGER	9	1995-10-01 00:00:00.0	2450	<NULL>	10
8	SCOTT	ANALYST	9	1993-05-01 00:00:00.0	3000	<NULL>	20
9	KING	PRESIDENT	<NULL>	1988-08-08 00:00:00.0	5000	<NULL>	10
10	TURNER	SALESMAN	5	1983-02-01 00:00:00.0	1500	0	30
11	ADAMS	CLERK	5	1992-07-03 00:00:00.0	1100	<NULL>	20
12	JAMES	CLERK	1	1996-09-10 00:00:00.0	950	<NULL>	30
13	FORD	ANALYST	1	1993-01-01 00:00:00.0	3000	<NULL>	20
14	Lee	CLERK	3	1983-10-09 00:00:00.0	1300	<NULL>	10

图-3

可见，在引入了事务管理机制后，业务方法发生异常时，由于事务回滚，所以没有成功插入员工刘备，从而保证了业务的完整性。

• 完整代码

本案例的完整代码如下所示：

`EmpController` 完整代码如下：

```
package com.tarena.controller;

import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.RequestMapping;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
@Transactional
public class EmpController {

    @Resource
    private EmpDao empDao;

    /**
     * 查询员工
     */
    @RequestMapping("/findEmp.do")
    @Transactional(readonly=true)
    public String find() {
        // 模拟查询员工数据
        System.out.println("查询员工数据，发送至列表页面。");
        // 制造一个异常，便于测试异常通知
        Integer.valueOf("abc");
        return "emp/emp_list.jsp";
    }
}
```

```
}

/**
 * 模拟批量添加员工
 */
@RequestMapping("/addEmps.do")
public String addBatch() {
    // 插入第一个员工
    Emp e1 = new Emp();
    e1.setName("刘备");
    e1.setJob("皇叔");
    e1.setSal(1000.0);
    e1.setEmpno(10);
    empDao.save(e1);

    // 模拟异常
    Integer.valueOf("abc"); //ClassCastException

    // 插入第二个员工
    Emp e2 = new Emp();
    e2.setName("关羽");
    e2.setJob("候");
    e2.setSal(1000.0);
    e2.setEmpno(10);
    empDao.save(e2);

    return "redirect:findEmp.do";
}

}
```

TestEmpController 完整代码如下：

```
package com.tarena.test;

import org.junit.Test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.tarena.controller.EmpController;

public class TestEmpController {

    /**
     * 测试查询员工
     */
    @Test
    public void test1() {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpController ctl = ctx.getBean(EmpController.class);
        ctl.find();
    }

    /**
     * 测试批量添加员工
     */
    @Test
    public void test2() throws ClassNotFoundException {
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
            "applicationContext.xml");
        EmpController ctl = ctx.getBean(EmpController.class);
        ctl.addBatch();
    }
}
```

applicationContext.xml 完整代码如下：

```
<?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"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
    http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="url"
            value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="driverClassName"
            value="oracle.jdbc.OracleDriver"/>
        <property name="username" value="lhh"/>
        <property name="password" value="123456"/>
    </bean>

    <!-- 配置 SqlSessionFactory -->
    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="ds" />
        <property name="mapperLocations"
            value="classpath:com/tarena/entity/*.xml"/>
    </bean>

    <!-- 配置 MyBatis 注解 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.tarena.dao" />
        <property name="annotationClass"
            value="com.tarena.annotation.MyBatisRepository"/>
    </bean>

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.tarena" />

    <!-- 支持 @RequestMapping 请求和 Controller 映射 -->
    <mvc:annotation-driven />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/" />
    </bean>
</beans>
```

```

        <property name="suffix" value=".jsp"/>
    </bean>

    <!-- 声明方面组件 -->
    <!-- <bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/> -->

    <!-- 配置 AOP -->
    <!-- <aop:config>
        <aop:aspect ref="operateLogger">
            <aop:before method="log1"
                pointcut="within(com.tarena.controller..*)" />
        </aop:aspect>
        <aop:aspect ref="operateLogger">
            <aop:around method="log2"
                pointcut="within(com.tarena.controller..*)" />
        </aop:aspect>
        <aop:aspect ref="operateLogger">
            <aop:after-throwing method="log3" throwing="e"
                pointcut="within(com.tarena.controller..*)" />
        </aop:aspect>
    </aop:config> -->

    <!-- 开启 AOP 注解扫描 -->
    <aop:aspectj-autoproxy proxy-target-class="true"/>

    <!-- 声明事务管理组件 -->
    <bean id="txManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="ds"/>
    </bean>

    <!-- 开启事务注解扫描 -->
    <tx:annotation-driven
        transaction-manager="txManager" proxy-target-class="true"/>

</beans>

```

2. Spring 声明式事务-XML 配置

• 问题

使用 XML 配置代替注解，实现 Spring 声明式事务。

• 方案

XML 配置声明式事务，关键代码如下：

```

<!-- XML 配置声明事务范围及类型 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="add*" propagation="REQUIRED"
            rollback-for="java.lang.Exception"/>
        <tx:method name="update*" propagation="REQUIRED"
            rollback-for="java.lang.Exception"/>
        <tx:method name="delete*" propagation="REQUIRED"
            rollback-for="java.lang.Exception"/>
    </tx:attributes>
</tx:advice>
<aop:config proxy-target-class="true">

```

```
<aop:advisor advice-ref="txAdvice"
    pointcut="within(com.tarena.controller..*)" />
</aop:config>
```

• 步骤

实现此案例需要按照如下步骤进行。

步骤一：取消事务注解

在 applicationContext.xml 中，去掉开启事务注解扫描的代码，使事务注解失效，关键代码如下：

```
<!-- 开启事务注解扫描 -->
<!-- <tx:annotation-driven
    transaction-manager="txManager" proxy-target-class="true"/> -->
```

步骤二：使用 XML 配置声明式事务

在 applicationContext.xml 中，使用 XML 配置声明式事务，关键代码如下：

```
<!-- XML 配置声明事务范围及类型 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="add*" propagation="REQUIRED" />
        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>
<aop:config proxy-target-class="true">
    <aop:advisor advice-ref="txAdvice"
        pointcut="within(com.tarena.controller..*)" />
</aop:config>
```

步骤三：测试

执行测试方法 TestEmpController.test2()，然后查询员工表数据如下图：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12 00:00:00.0	800	<NULL>	20
2	ALLEN	SALESMAN	3	1981-06-03 00:00:00.0	1600	300	30
3	WARD	SALESMAN	4	1990-03-15 00:00:00.0	1250	500	30
4	JONES	MANAGER	5	1985-04-08 00:00:00.0	2975	<NULL>	20
5	MARTIN	SALESMAN	7	1986-03-08 00:00:00.0	1250	1400	30
6	BLAKE	MANAGER	9	1989-06-01 00:00:00.0	2850	<NULL>	30
7	CLARK	MANAGER	9	1995-10-01 00:00:00.0	2450	<NULL>	10
8	SCOTT	ANALYST	9	1993-05-01 00:00:00.0	3000	<NULL>	20
9	KING	PRESIDENT	<NULL>	1988-08-08 00:00:00.0	5000	<NULL>	10
10	TURNER	SALESMAN	5	1983-02-01 00:00:00.0	1500	0	30
11	ADAMS	CLERK	5	1992-07-03 00:00:00.0	1100	<NULL>	20
12	JAMES	CLERK	1	1996-09-10 00:00:00.0	950	<NULL>	30
13	FORD	ANALYST	1	1993-01-01 00:00:00.0	3000	<NULL>	20
14	Lee	CLERK	3	1983-10-09 00:00:00.0	1300	<NULL>	10

图-4

可见，XML 配置的声明式事务与注解方式效果一样。

- 完整代码

本案例的完整代码如下所示：

applicationContext.xml 完整代码如下：

```
<?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"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
    http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds"
        class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="url"
            value="jdbc:oracle:thin:@localhost:1521:xe"/>
        <property name="driverClassName"
            value="oracle.jdbc.OracleDriver"/>
        <property name="username" value="lhk"/>
        <property name="password" value="123456"/>
    </bean>

    <!-- 配置 SqlSessionFactory -->
    <bean id="sqlSessionFactory"
        class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="ds" />
        <property name="mapperLocations"
            value="classpath:com/tarena/entity/*.xml"/>
    </bean>

    <!-- 配置 MyBatis 注解 -->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.tarena.dao" />
        <property name="annotationClass"
            value="com.tarena.annotation.MyBatisRepository"/>
    </bean>

    <!-- 开启注解扫描 -->
    <context:component-scan base-package="com.tarena" />

    <!-- 支持@RequestMapping 请求和 Controller 映射 -->
```

```

<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp" />
</bean>

<!-- 声明方面组件 -->
<!-- <bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/> -->

<!-- 配置 AOP -->
<!-- <aop:config>
    <aop:aspect ref="operateLogger">
        <aop:before method="log1"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:around method="log2"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
    <aop:aspect ref="operateLogger">
        <aop:after-throwing method="log3" throwing="e"
            pointcut="within(com.tarena.controller..*)" />
    </aop:aspect>
</aop:config> -->

<!-- 开启 AOP 注解扫描 -->
<aop:aspectj-autoproxy proxy-target-class="true"/>

<!-- 声明事务管理组件 -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="ds" />
</bean>

<!-- 开启事务注解扫描 -->
<!-- <tx:annotation-driven
    transaction-manager="txManager" proxy-target-class="true"/> -->

<!-- XML 配置声明事务范围及类型 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="add*" propagation="REQUIRED"
            rollback-for="java.lang.Exception" />
        <tx:method name="update*" propagation="REQUIRED"
            rollback-for="java.lang.Exception" />
        <tx:method name="delete*" propagation="REQUIRED"
            rollback-for="java.lang.Exception" />
    </tx:attributes>
</tx:advice>
<aop:config proxy-target-class="true">
    <aop:advisor advice-ref="txAdvice"
        pointcut="within(com.tarena.controller..*)" />
</aop:config>

</beans>

```

3. Spring 声明式事务-回滚机制

- 问题

默认情况下，声明式事务只能捕获 `RuntimeException`，并使事务回滚，请处理非 `RuntimeException` 情况下的事务回滚。

- **方案**

使用注解实现声明式事务时，按如下方式指定异常：

```
@Transactional(rollbackFor=ClassNotFoundException.class)
```

使用 XML 配置实现声明式事务时，按如下方式指定异常：

```
<tx:method name="update*" propagation="REQUIRED"
          rollback-for="java.lang.ClassNotFoundException"/>
```

本案例中演示第二种方式。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：测试非 `RuntimeException` 异常的情况

修改 `EmpController.addBatch()` 方法，将模拟的异常换成非 `RuntimeException`，代码如下：

```
package com.tarena.controller;

import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.RequestMapping;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
@Transactional
public class EmpController {

    @Resource
    private EmpDao empDao;

    //其他方法略

    /**
     * 模拟批量添加员工
     */
    @RequestMapping("/addEmps.do")
    public String addBatch() throws ClassNotFoundException {
        // 插入第一个员工
        Emp e1 = new Emp();
        e1.setName("刘备");
        e1.setJob("皇叔");
        e1.setSal(1000.0);
```

```

e1.setEmpno(10);
empDao.save(e1);

// 模拟异常

// Integer.valueOf("abc"); //ClassCastException
// Class.forName("BadClass"); //ClassNotFoundException

// 插入第二个员工
Emp e2 = new Emp();
e2.setEname("关羽");
e2.setJob("候");
e2.setSal(1000.0);
e2.setEmpno(10);
empDao.save(e2);

return "redirect:findEmp.do";
}
}

```

执行测试方法 `TestEmpController.test2()`，然后查询员工表数据如下图：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12 00:00:00.0	800	<NULL>	20
2	ALLEN	SALESMAN	3	1981-06-03 00:00:00.0	1600	300	30
3	WARD	SALESMAN	4	1990-03-15 00:00:00.0	1250	500	30
4	JONES	MANAGER	5	1985-04-08 00:00:00.0	2975	<NULL>	20
5	MARTIN	SALESMAN	7	1986-03-08 00:00:00.0	1250	1400	30
6	BLAKE	MANAGER	9	1989-06-01 00:00:00.0	2850	<NULL>	30
7	CLARK	MANAGER	9	1995-10-01 00:00:00.0	2450	<NULL>	10
8	SCOTT	ANALYST	9	1993-05-01 00:00:00.0	3000	<NULL>	20
9	KING	PRESIDENT	<NULL>	1988-08-08 00:00:00.0	5000	<NULL>	10
10	TURNER	SALESMAN	5	1983-02-01 00:00:00.0	1500	0	30
11	ADAMS	CLERK	5	1992-07-03 00:00:00.0	1100	<NULL>	20
12	JAMES	CLERK	1	1996-09-10 00:00:00.0	950	<NULL>	30
13	FORD	ANALYST	1	1993-01-01 00:00:00.0	3000	<NULL>	20
14	Lee	CLERK	3	1983-10-09 00:00:00.0	1300	<NULL>	10
433	刘备	皇叔	<NULL>	<NULL>	1000	<NULL>	<NULL>

图-5

可见，对于非 `RuntimeException` 异常，默认的声明式事务是处理不了的。

步骤二：注解上指定处理的异常类型

修改 `applicationContext.xml`，在通知配置上指定异常类型，关键代码如下：

```

<!-- XML 配置声明事务范围及类型 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />

        <tx:method name="add*" propagation="REQUIRED"
            rollback-for="java.lang. ClassNotFoundException"/>

        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
    </tx:attributes>

```

```
</tx:advice>
<aop:config proxy-target-class="true">
  <aop:advisor advice-ref="txAdvice"
    pointcut="within(com.tarena.controller..*)" />
</aop:config>
```

步骤三：测试

手动删除员工表中的刘备，执行测试方法 `TestEmpController.test2()`，然后查询员工表数据如下图：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	SMITH	CLERK	3	1980-05-12 00:00:00.0	800	<NULL>	20
2	ALLEN	SALESMAN	3	1981-06-03 00:00:00.0	1600	300	30
3	WARD	SALESMAN	4	1990-03-15 00:00:00.0	1250	500	30
4	JONES	MANAGER	5	1985-04-08 00:00:00.0	2975	<NULL>	20
5	MARTIN	SALESMAN	7	1986-03-08 00:00:00.0	1250	1400	30
6	BLAKE	MANAGER	9	1989-06-01 00:00:00.0	2850	<NULL>	30
7	CLARK	MANAGER	9	1995-10-01 00:00:00.0	2450	<NULL>	10
8	SCOTT	ANALYST	9	1993-05-01 00:00:00.0	3000	<NULL>	20
9	KING	PRESIDENT	<NULL>	1988-08-08 00:00:00.0	5000	<NULL>	10
10	TURNER	SALESMAN	5	1983-02-01 00:00:00.0	1500	0	30
11	ADAMS	CLERK	5	1992-07-03 00:00:00.0	1100	<NULL>	20
12	JAMES	CLERK	1	1996-09-10 00:00:00.0	950	<NULL>	30
13	FORD	ANALYST	1	1993-01-01 00:00:00.0	3000	<NULL>	20
14	Lee	CLERK	3	1983-10-09 00:00:00.0	1300	<NULL>	10

图-6

可见，在注解上指定了要处理的异常类型后，它就可以处理非 `RuntimeException` 了。

• 完整代码

本案例的完整代码如下所示：

`applicationContext.xml` 完整代码如下：

```
<?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"
  xmlns:jdbc="http://www.springframework.org/schema/jdbc"
  xmlns:jee="http://www.springframework.org/schema/jee"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:jpa="http://www.springframework.org/schema/data/jpa"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/jdbc
    http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
    http://www.springframework.org/schema/data/jpa
    http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
    http://www.springframework.org/schema/aop
```

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
 http://www.springframework.org/schema/mvc
 http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

```
<!-- 配置数据源 -->
<bean id="ds"
      class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
  <property name="url"
    value="jdbc:oracle:thin:@localhost:1521:xe"/>
  <property name="driverClassName"
    value="oracle.jdbc.OracleDriver"/>
  <property name="username" value="lh"/>
  <property name="password" value="123456"/>
</bean>

<!-- 配置 SqlSessionFactory -->
<bean id="sqlSessionFactory"
      class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="ds" />
  <property name="mapperLocations"
    value="classpath:com/tarena/entity/*.xml"/>
</bean>

<!-- 配置 MyBatis 注解 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
  <property name="basePackage" value="com.tarena.dao" />
  <property name="annotationClass"
    value="com.tarena.annotation.MyBatisRepository"/>
</bean>

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 支持@RequestMapping 请求和 Controller 映射 -->
<mvc:annotation-driven />

<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/" />
  <property name="suffix" value=".jsp"/>
</bean>

<!-- 声明方面组件 -->
<!-- <bean id="operateLogger" class="com.tarena.aspect.OperateLogger"/> -->

<!-- 配置 AOP -->
<!-- <aop:config>
  <aop:aspect ref="operateLogger">
    <aop:before method="log1"
      pointcut="within(com.tarena.controller..*)" />
  </aop:aspect>
  <aop:aspect ref="operateLogger">
    <aop:around method="log2"
      pointcut="within(com.tarena.controller..*)" />
  </aop:aspect>
  <aop:aspect ref="operateLogger">
    <aop:after-throwing method="log3" throwing="e"
      pointcut="within(com.tarena.controller..*)" />
  </aop:aspect>
</aop:config> -->

<!-- 开启 AOP 注解扫描 -->
<aop:aspectj-autoproxy proxy-target-class="true"/>

<!-- 声明事务管理组件 -->
```

```
<bean id="txManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="ds"/>
</bean>

<!-- 开启事务注解扫描 -->
<!-- <tx:annotation-driven
    transaction-manager="txManager" proxy-target-class="true"/> -->

<!-- XML 配置声明事务范围及类型 -->
<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="find*" read-only="true" />
        <tx:method name="add*" propagation="REQUIRED"
            rollback-for="java.lang.ClassNotFoundException"/>
        <tx:method name="update*" propagation="REQUIRED" />
        <tx:method name="delete*" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>
<aop:config proxy-target-class="true">
    <aop:advisor advice-ref="txAdvice"
        pointcut="within(com.tarena.controller..*)" />
</aop:config>

</beans>
```

4. RESTful 应用案例

- **问题**

使用 RESTful 实现员工模块的增删改查功能。

- **方案**

在 Spring+MyBatis 框架基础上实现员工模块的增删改查功能，然后再使用 RESTful 来重构请求 URL。

- **步骤**

实现此案例需要按照如下步骤进行。

步骤一：实现员工模块增删改查功能

复制项目 SpringUnit01，创建新项目 SpringRestful，在新项目中完成员工的增删改查功能。

EmpDao 代码如下：

```
package com.tarena.dao;

import java.util.List;
import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
```

```
*/
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    void save(Emp emp);

    Emp findById(int id);

    void update(Emp emp);

    void delete(int id);

}
```

EmpMapper.xml 代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        insert into t_emp values(
            emp seq.nextval,
            #{ename,jdbcType=VARCHAR},
            #{job,jdbcType=VARCHAR},
            #{mgr,jdbcType=NUMERIC},
            #{hiredate,jdbcType=DATE},
            #{sal,jdbcType=NUMERIC},
            #{comm,jdbcType=NUMERIC},
            #{deptno,jdbcType=NUMERIC}
        )
    </insert>

    <!-- 根据 ID 查询员工 -->
    <select id="findById"
        parameterType="java.lang.Integer"
        resultType="com.tarena.entity.Emp">
        select * from t_emp where empno=#{id}
    </select>

    <!-- 修改员工 -->
    <update id="update" parameterType="com.tarena.entity.Emp">
        update t_emp set
            ename=#{ename,jdbcType=VARCHAR},
            job=#{job,jdbcType=VARCHAR},
            sal=#{sal,jdbcType=NUMERIC}
        where empno=#{empno,jdbcType=NUMERIC}
    </update>

    <!-- 删除员工 -->
    <delete id="delete" parameterType="java.lang.Integer">
```



```
        delete from t_emp where empno=#{id}
    </delete>

</mapper>
```

EmpController 代码如下：

```
package com.tarena.controller;

import java.util.List;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
public class EmpController {

    @Resource
    private EmpDao empDao;

    @RequestMapping("/findEmp.do")
    public String find(Model model) {
        List<Emp> list = empDao.findAll();
        model.addAttribute("emps", list);
        return "emp/emp_list";
    }

    /**
     * 打开新增页面
     */
    @RequestMapping("/toAddEmp.do")
    public String toAdd() {
        return "emp/add_emp";
    }

    /**
     * 新增保存
     */
    @RequestMapping("/addEmp.do")
    public String add(Emp emp) {
        empDao.save(emp);
        return "redirect:findEmp.do";
    }

    /**
     * 打开修改页面
     */
    @RequestMapping("/toUpdateEmp.do")
    public String toUpdate(
        @RequestParam("id") int id,
        Model model) {
        Emp e = empDao.findById(id);
        model.addAttribute("emp", e);
        return "emp/update_emp";
    }

    /**
     * 修改保存
     */
}
```

```

@RequestMapping("/updateEmp.do")
public String update(Emp emp) {
    empDao.update(emp);
    return "redirect:findEmp.do";
}

/**
 * 删除
 */
@RequestMapping("/deleteEmp.do")
public String delete(@RequestParam("id") int id) {
    empDao.delete(id);
    return "redirect:findEmp.do";
}
}

```

员工列表页面 emp_list.jsp 代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<script type="text/javascript" src="../js/jquery-1.11.1.js"></script>
<script type="text/javascript">
    function delete_emp(id) {
        var r = window.confirm("确定要删除此数据吗?");
        if(r) {
            location.href = "deleteEmp.do?id="+id;
        }
    }
</script>
</head>
<body>
<div align="center">
    <input type="button" value="新 增"
onclick="location.href='toAddEmp.do'"/>
</div>
<table width="60%" border="1" cellpadding="2" cellspacing="0"
align="center">
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
<th></th>
</tr>
<c:forEach items="${emps}" var="emp">
<tr>
<td>${emp.empno}</td>
<td>${emp.ename}</td>
<td>${emp.job}</td>
<td>${emp.mgr}</td>
<td>${emp.hiredate}</td>
<td>${emp.sal}</td>
<td>${emp.comm}</td>
<td>${emp.deptno}</td>
<td>
<input type="button" value="修 改"
onclick="location.href='toUpdateEmp.do?id=${emp.empno}'"/>

```

```

        <input type="button" value="删除"
onclick="delete_emp({emp.empno});"/>
    </td>
</tr>
</c:forEach>
</table>
</body>
</html>

```

员工新增页面 add_emp.jsp 代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
</head>
<body>
    <form action="addEmp.do" method="post">
        <table width="40%" border="1" cellpadding="2" cellspacing="0"
align="center">
            <tr>
                <td>姓名:</td>
                <td><input type="text" name="ename"/></td>
            </tr>
            <tr>
                <td>岗位:</td>
                <td><input type="text" name="job"/></td>
            </tr>
            <tr>
                <td>工资:</td>
                <td><input type="text" name="sal"/></td>
            </tr>
            <tr>
                <td colspan="2"><input type="submit" value="保存"/></td>
            </tr>
        </table>
    </form>
</body>
</html>

```

员工修改页面 update_emp.jsp 代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
</head>
<body>
    <form action="updateEmp.do" method="post">
        <table width="40%" border="1" cellpadding="2" cellspacing="0"
align="center">
            <tr>
                <td>EMPNO:</td>
                <td><input type="text" name="empno"
value="${emp.empno}"/></td>
            </tr>
            <tr>
                <td>姓名:</td>
                <td><input type="text" name="ename"
value="${emp.ename}"/></td>
            </tr>

```

```

    </tr>
    <tr>
        <td>岗位:</td>
        <td><input type="text" name="job" value="${emp.job }"/></td>
    </tr>
    <tr>
        <td>工资:</td>
        <td><input type="text" name="sal" value="${emp.sal }"/></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit" value="保存"/></td>
    </tr>
</table>
</form>
</body>
</html>

```

部署项目并启动 tomcat，测试通过员工的增删改查功能。

步骤二：启用 RESTful

修改 web.xml，将*.do 改为/，代码如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app 2.5.xsd">
    <display-name></display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <!-- Spring 前端控制器 -->
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>
                contextConfigLocation
            </param-name>
            <param-value>
                classpath:applicationContext.xml
            </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>

        <url-pattern>/</url-pattern>

    </servlet-mapping>
    <!-- 使用 Filter 解决中文乱码问题 -->
    <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>
            org.springframework.web.filter.CharacterEncodingFilter
        </filter-class>
        <init-param>

```

```
<param-name>encoding</param-name>
<param-value>UTF-8</param-value>
</init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
```

```
<url-pattern>/*</url-pattern>
```

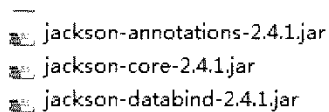
```
</filter-mapping>
```

```
</web-app>
```

修改 applicationContext.xml，增加支持 RESTful 访问静态资源的配置，关键代码如下：

```
<!-- 支持 RESTful 访问静态资源 -->
<mvc:default-servlet-handler />
```

导入 Spring 中使用 Ajax 所需的 3 个开发包，如下图：



```
jackson-annotations-2.4.1.jar
jackson-core-2.4.1.jar
jackson-databind-2.4.1.jar
```

图-7

步骤三：使用 RESTful 重构 Controller

修改 EmpController，使用 RESTful 设计访问 URI，代码如下：

```
package com.tarena.controller;

import java.util.List;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
public class EmpController {

  @Resource
  private EmpDao empDao;

  @RequestMapping(value="/find",method=RequestMethod.GET)

  public String find(Model model) {
    List<Emp> list = empDao.findAll();
    model.addAttribute("emps", list);
    return "emp/emp_list";
  }
}
```

```

/**
 * 打开新增页面
 */

@RequestMapping(value="/toAdd",method=RequestMethod.GET)

public String toAdd() {
    return "emp/add_emp";
}

/**
 * 新增保存
 */

@RequestMapping(value="/add",method=RequestMethod.POST)

public String add(Emp emp) {
    empDao.save(emp);

    return "redirect:find";
}

/**
 * 打开修改页面
 */

@RequestMapping(value="/toUpdate/{id}",method=RequestMethod.GET)

public String toUpdate(

    @PathVariable("id") int id,

    Model model) {
    Emp e = empDao.findById(id);
    model.addAttribute("emp", e);
    return "emp/update_emp";
}

/**
 * 修改保存
 */

@RequestMapping(value="/update",method=RequestMethod.PUT)
@ResponseBody
public boolean update(@RequestBody Emp emp) {
    empDao.update(emp);
    return true;
}

/**
 * 删除
 */

@RequestMapping(value="/{id}",method=RequestMethod.DELETE)
@ResponseBody

```

```
public boolean delete(@PathVariable("id") int id) {
    empDao.delete(id);
    return true;
}

}
```

步骤四：修改页面发请求的方式

在 WebRoot/js 文件夹下创建 json.js 用于将表单数据转换成 json 对象 代码如下：

```
// 将表单数据转换成 json 对象
$.fn.serializeObject = function() {
    var o = {};
    var a = this.serializeArray();
    $.each(a, function() {
        if (o[this.name]) {
            if (!o[this.name].push) {
                o[this.name] = [o[this.name]];
            }
            o[this.name].push(this.value || '');
        } else {
            o[this.name] = this.value || '';
        }
    });
    return o;
};
```

在 emp_list.jsp 中,对新增、修改按钮的 URL 进行修改,并将删除按钮改为 RESTful 的删除提交方式,代码如下：

```
<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<script type="text/javascript" src="../../js/jquery-1.11.1.js"></script>
<script type="text/javascript">
    function deleteEmp(id) {
        var r = window.confirm("确定要删除此数据吗?");
        if(r) {

            //location.href = "deleteEmp.do?id="+id;
            $.ajax({
                type:"DELETE",
                url:"/SpringRestful/emp/"+id,
                dataType:"json",
                success:function(data){
                    location.href = "/SpringRestful/emp/find";
                }
            });

        }
    }
</script>
</head>
<body>
<div align="center">

    <input type="button" value="新增" onclick="location.href='toAdd'"/>
```

```

</div>
<table width="60%" border="1" cellpadding="2" cellspacing="0"
align="center">
  <tr>
    <th>EMPNO</th>
    <th>ENAME</th>
    <th>JOB</th>
    <th>MGR</th>
    <th>HIREDATE</th>
    <th>SAL</th>
    <th>COMM</th>
    <th>DEPTNO</th>
    <th></th>
  </tr>
  <c:forEach items="${emps}" var="emp">
    <tr>
      <td>${emp.empno}</td>
      <td>${emp.ename}</td>
      <td>${emp.job}</td>
      <td>${emp.mgr}</td>
      <td>${emp.hiredate}</td>
      <td>${emp.sal}</td>
      <td>${emp.comm}</td>
      <td>${emp.deptno}</td>
      <td>
        <input type="button" value="修改"
onclick="location.href='toUpdate/${emp.empno}'"/>
        <input type="button" value="删除"
onclick="delete_emp(${emp.empno});"/>
      </td>
    </tr>
  </c:forEach>
</table>
</body>
</html>

```

修改 add_emp.jsp 表单的提交路径，代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
</head>
<body>

```

<form action="add" method="post">

```

  <table width="40%" border="1" cellpadding="2" cellspacing="0"
align="center">
    <tr>
      <td>姓名:</td>
      <td><input type="text" name="ename"/></td>
    </tr>
    <tr>
      <td>岗位:</td>
      <td><input type="text" name="job"/></td>
    </tr>
    <tr>
      <td>工资:</td>

```



```

        <td><input type="text" name="sal"/></td>
    </tr>
    <tr>
        <td colspan="2"><input type="submit" value="保存" /></td>
    </tr>
</table>
</form>
</body>
</html>

```

修改 update_emp.jsp 中表单提交的方式和路径，代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>

<script type="text/javascript" src="../../js/jquery-1.11.1.js"></script>
<script type="text/javascript" src="../../js/json.js"></script>
<script type="text/javascript">
    function save() {
        $.ajax({
            type:"PUT",
            url:"/SpringRestful/emp/update",
            data:JSON.stringify($("#myform").serializeObject()),
            dataType:"json",
            contentType:"application/json",
            success:function(data){
                location.href = "/SpringRestful/emp/find";
            }
        });
    }
</script>

</head>
<body>
    <form action="updateEmp.do" method="post" id="myform">
        <table width="40%" border="1" cellpadding="2" cellspacing="0"
align="center">
            <tr>
                <td>EMPNO :</td>
                <td><input type="text" name="empno"
value="${emp.empno }"/></td>
            </tr>
            <tr>
                <td>姓名 :</td>
                <td><input type="text" name="ename"
value="${emp.ename }"/></td>
            </tr>
            <tr>
                <td>岗位 :</td>
                <td><input type="text" name="job" value="${emp.job }"/></td>
            </tr>
            <tr>
                <td>工资 :</td>
                <td><input type="text" name="sal" value="${emp.sal }"/></td>
            </tr>
            <tr>
                <td colspan="2"><input type="button" value=" 保 存 "

```

```
onclick="save()"/></td>
```

```
        </tr>
    </table>
</form>
</body>
</html>
```

步骤五：测试

重新部署并启动 tomcat，测试通过员工的增删改查功能。

• 完整代码

本案例的完整代码如下所示：

EmpDao 完整代码如下：

```
package com.tarena.dao;

import java.util.List;
import com.tarena.annotation.MyBatisRepository;
import com.tarena.entity.Emp;

/**
 * 员工表的 DAO 组件
 */
@MyBatisRepository
public interface EmpDao {

    List<Emp> findAll();

    void save(Emp emp);

    Emp findById(int id);

    void update(Emp emp);

    void delete(int id);

}
```

EmpMapper.xml 完整代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//ibatis.apache.org//DTD Mapper 3.0//EN"
    "http://ibatis.apache.org/dtd/ibatis-3-mapper.dtd">

<mapper namespace="com.tarena.dao.EmpDao">

    <!-- 查询全部的员工 -->
    <select id="findAll"
        resultType="com.tarena.entity.Emp">
        select * from t_emp
    </select>

    <!-- 保存一条员工数据 -->
    <insert id="save"
        parameterType="com.tarena.entity.Emp">
        insert into t_emp values(
            emp_seq.nextval,
```

```

        #{ename,jdbcType=VARCHAR},
        #{job,jdbcType=VARCHAR},
        #{mgr,jdbcType=NUMERIC},
        #{hiredate,jdbcType=DATE},
        #{sal,jdbcType=NUMERIC},
        #{comm,jdbcType=NUMERIC},
        #{deptno,jdbcType=NUMERIC}
    )
</insert>

<!-- 根据 ID 查询员工 -->
<select id="findById"
    parameterType="java.lang.Integer"
    resultType="com.tarena.entity.Emp">
    select * from t emp where empno=#{id}
</select>

<!-- 修改员工 -->
<update id="update" parameterType="com.tarena.entity.Emp">
    update t emp set
        ename=#{ename,jdbcType=VARCHAR},
        job=#{job,jdbcType=VARCHAR},
        sal=#{sal,jdbcType=NUMERIC}
    where empno=#{empno,jdbcType=NUMERIC}
</update>

<!-- 删除员工 -->
<delete id="delete" parameterType="java.lang.Integer">
    delete from t emp where empno=#{id}
</delete>

</mapper>

```

web.xml 完整代码如下：

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <display-name></display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <!-- Spring 前端控制器 -->
    <servlet>
        <servlet-name>SpringMVC</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>
                contextConfigLocation
            </param-name>
            <param-value>
                classpath:applicationContext.xml
            </param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>SpringMVC</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

```

```
<!-- 使用 Filter 解决中文乱码问题 -->
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

</web-app>
```

applicationContext.xml 完整代码如下：

```
<?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"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:jpa="http://www.springframework.org/schema/data/jpa"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd
        http://www.springframework.org/schema/jdbc
        http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd
        http://www.springframework.org/schema/jee
        http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa-1.3.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd">

    <!-- 配置数据源 -->
    <bean id="ds" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
        <property name="username" value="lhh" />
        <property name="password" value="123456" />
    </bean>

    <!-- 配置 SqlSessionFactory -->
    <bean
        class="org.mybatis.spring.SqlSessionFactoryBean"
        value="classpath:com/tarena/entity/*.xml" />
        id="sqlSessionFactory"
        name="mapperLocations"

    <!-- 配置 MyBatis 注解 -->
```

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.tarena.dao" />
    <property name="annotationClass"
value="com.tarena.annotation.MyBatisRepository" />
</bean>

<!-- 开启注解扫描 -->
<context:component-scan base-package="com.tarena" />

<!-- 开启 RequestMapping 注解 -->
<mvc:annotation-driven />

<!-- 处理请求转发 -->
<bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp" />
</bean>

<!-- 支持 RESTful 访问静态资源 -->
<mvc:default-servlet-handler />

</beans>
```

EmpController 完整代码如下：

```
package com.tarena.controller;

import java.util.List;
import javax.annotation.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import com.tarena.dao.EmpDao;
import com.tarena.entity.Emp;

@Controller
@RequestMapping("/emp")
public class EmpController {

    @Resource
    private EmpDao empDao;

    @RequestMapping(value="/find",method=RequestMethod.GET)
    public String find(Model model) {
        List<Emp> list = empDao.findAll();
        model.addAttribute("emps", list);
        return "emp/emp_list";
    }

    /**
     * 打开新增页面
     */
    @RequestMapping(value="/toAdd",method=RequestMethod.GET)
    public String toAdd() {
        return "emp/add_emp";
    }

    /**
     * 新增保存
     */
}
```

```

@RequestMapping(value="/add",method=RequestMethod.POST)
public String add(Emp emp) {
    empDao.save(emp);
    return "redirect:find";
}

/**
 * 打开修改页面
 */
@RequestMapping(value="/toUpdate/{id}",method=RequestMethod.GET)
public String toUpdate(
    @PathVariable("id") int id,
    Model model) {
    Emp e = empDao.findById(id);
    model.addAttribute("emp", e);
    return "emp/update_emp";
}

/**
 * 修改保存
 */
@RequestMapping(value="/update",method=RequestMethod.PUT)
@ResponseBody
public boolean update(@RequestBody Emp emp) {
    empDao.update(emp);
    return true;
}

/**
 * 删除
 */
@RequestMapping(value="/{id}",method=RequestMethod.DELETE)
@ResponseBody
public boolean delete(@PathVariable("id") int id) {
    empDao.delete(id);
    return true;
}
}

```

json.js 完整代码如下：

```

// 将表单数据转换成 json 对象
$.fn.serializeObject = function() {
    var o = {};
    var a = this.serializeArray();
    $.each(a, function() {
        if (o[this.name]) {
            if (!o[this.name].push) {
                o[this.name] = [o[this.name]];
            }
            o[this.name].push(this.value || '');
        } else {
            o[this.name] = this.value || '';
        }
    });
    return o;
};

```

emp_list.jsp 完整代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

```

```

<html>
<head>
<script type="text/javascript" src="../../js/jquery-1.11.1.js"></script>
<script type="text/javascript">
function delete_emp(id) {
    var r = window.confirm("确定要删除此数据吗? ");
    if(r) {
        //location.href = "deleteEmp.do?id="+id;
        $.ajax({
            type:"DELETE",
            url:"/SpringRestful/emp/"+id,
            dataType:"json",
            success:function(data){
                location.href = "/SpringRestful/emp/find";
            }
        });
    }
}
</script>
</head>
<body>
<div align="center">
    <input type="button" value="新增" onclick="location.href='toAdd'"/>
</div>
<table width="60%" border="1" cellpadding="2" cellspacing="0"
align="center">
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
<th></th>
</tr>
<c:forEach items="${emps}" var="emp">
<tr>
<td>${emp.empno}</td>
<td>${emp.ename}</td>
<td>${emp.job}</td>
<td>${emp.mgr}</td>
<td>${emp.hiredate}</td>
<td>${emp.sal}</td>
<td>${emp.comm}</td>
<td>${emp.deptno}</td>
<td>
                <input type="button" value="修 改"
onclick="location.href='toUpdate/${emp.empno}'"/>
                <input type="button" value="删 除"
onclick="delete_emp(${emp.empno});"/>
            </td>
</tr>
</c:forEach>
</table>
</body>
</html>

```

add_emp.jsp 完整代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>

```

```

</head>
<body>
  <form action="add" method="post">
    <table width="40%" border="1" cellpadding="2" cellspacing="0"
align="center">
      <tr>
        <td>姓名: </td>
        <td><input type="text" name="ename"/></td>
      </tr>
      <tr>
        <td>岗位: </td>
        <td><input type="text" name="job"/></td>
      </tr>
      <tr>
        <td>工资: </td>
        <td><input type="text" name="sal"/></td>
      </tr>
      <tr>
        <td colspan="2"><input type="submit" value="保存" /></td>
      </tr>
    </table>
  </form>
</body>
</html>

```

update_emp.jsp 完整代码如下：

```

<%@page pageEncoding="utf-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<script type="text/javascript" src="../../js/jquery-1.11.1.js"></script>
<script type="text/javascript" src="../../js/json.js"></script>
<script type="text/javascript">
  function save() {
    $.ajax({
      type:"PUT",
      url:"/SpringRestful/emp/update",
      data:JSON.stringify($("#myform").serializeObject()),
      dataType:"json",
      contentType:"application/json",
      success:function(data){
        location.href = "/SpringRestful/emp/find";
      }
    });
  }
</script>
</head>
<body>
  <form action="updateEmp.do" method="post" id="myform">
    <table width="40%" border="1" cellpadding="2" cellspacing="0"
align="center">
      <tr>
        <td>EMPNO: </td>
        <td><input type="text" name="empno"
value="${emp.empno}"/></td>
      </tr>
      <tr>
        <td>姓名: </td>
        <td><input type="text" name="ename"
value="${emp.ename}"/></td>
      </tr>
      <tr>
        <td>岗位: </td>
        <td><input type="text" name="job" value="${emp.job}"/></td>

```



```
        </tr>
        <tr>
            <td>工资: </td>
            <td><input type="text" name="sal" value="{emp.sal }"/></td>
        </tr>
        <tr>
            <td colspan="2"><input type="button" value=" 保 存 "
onclick="save();" /></td>
        </tr>
    </table>
</form>
</body>
</html>
```