day06-通用 Mapper 和 RESTful Web Service

课程大纲

1、 通用 Mapper
2、 Restful 概述
   a) 什么是 REST
   b) 什么是 RESTful
   c) REST 标准规范
3、 REST 最佳实践
4、 REST 实现 CRUD
5、 改造综合练习为 REST 风格

反馈意见

http://ifedorenko.github.com/m2e-extras/

教学笔记

1、 通用 Mapper

1.1、 通用 Mapper 概述

1.1.1、 什么是通用 Mapper？

开源中国的介绍页面：https://www.oschina.net/p/mybatis-mapper

GitHub 介绍页面：https://github.com/abel533/Mapper

作者：http://blog.csdn.net/isea533

1.1.2、 为什么要学习通用 Mapper

在我们的日常开发中，单表的增删改查是一个非常频繁的操作，而通常，我们都需要自己来定义每一张表的 Mapper.xml 文件，并且定义增删改查的 sql 语句，这些工作具有大量的重复性，意义不大，很浪费时间。

通用 Mapper 可以通过 Mybatis 的拦截器原理，动态的帮我们实现单表的增删改查功能，大

大降低了我们的开发成本，减少了我们的工作量。

1.2、 导入依赖

```
        <!-- 通用Mapper  -->
        <dependency>
            <groupId>tk.mybatis</groupId>
            <artifactId>mapper</artifactId>
            <version>4.0.1</version>
        </dependency>
```

1.3、 配置通用 Mapper 与 Spring 整合

通用 mapper 可以直接与 Spring 整合：
在 applicationContext-mybatis.xml 中原来的 mapper 包扫描修改为：

```
    <!-- mapper扫描包 -->
    <bean class="tk.mybatis.spring.mapper.MapperScannerConfigurer">
        <!-- mapper包的位置 -->
        <property name="basePackage"
value="cn.itcast.usermanage.mapper"/>
    </bean>
```

1.4、 使用通用 Mapper 接口来定义 UserMapper

```
/**
 * 定义新的UserMapper接口，继承通用的Mapper接口，并且指定泛型为User
 * @author 虎哥
 */
public interface AutoUserMapper extends Mapper<User>{

}
```

此时，我们的 Mapper 就已经具备了通用 Mapper 中提前定义好的方法：

```java
/**
 * 通用Mapper接口,其他接口继承该接口即可
 *
 * <p>这是一个例子，自己扩展时可以参考</p>
 *
 * <p>项目地址 : <a href="https://github.com/abel533/Mapper" target="_
 *
 * @param <T> 不能为空
 * @author liuzh
 */
public interface Mapper<T> {

    @SelectProvider(type = MapperProvider.class, method = "dynamicS
    T selectOne(T record);

    @SelectProvider(type = MapperProvider.class, method = "dynamicS
    List<T> select(T record);

    @SelectProvider(type = MapperProvider.class, method = "dynamicS
    int selectCount(T record);

    @SelectProvider(type = MapperProvider.class, method = "dynamicS
    T selectByPrimaryKey(Object key);

    @InsertProvider(type = MapperProvider.class, method = "dynamicS
    int insert(T record);

    @InsertProvider(type = MapperProvider.class, method = "dynamicS
    int insertSelective(T record);
```

现在，我们的 UserMapper 接口已经具备了单标的增删改查方法，但是并没有对应的 Mapper.xml 文件，我们不需要写。因为通用 Mapper 会帮我们动态生成这些方法对应的 Sql 和 statement

那么问题来了：通用 Mapper 如何知道 User 对应数据库中那张表？如何知道对应数据库哪些字段？

1.5、 设置实体类的字段

实体类按照如下规则和数据库表进行转换,注解全部是JPA中的注解:

1. 表名默认使用类名,驼峰转下划线(只对大写字母进行处理),如 `UserInfo` 默认对应的表名为 `user`
2. 表名可以使用 `@Table(name = "tableName")` 进行指定,对不符合第一条默认规则的可以通过这
3. 字段默认和 `@Column` 一样,都会作为表字段,表字段默认为Java对象的 `Field` 名字驼峰转下划线
4. 可以使用 `@Column(name = "fieldName")` 指定不符合第3条规则的字段名
5. 使用 `@Transient` 注解可以忽略字段,添加该注解的字段不会作为表字段使用.
6. **建议一定是有一个 `@Id` 注解作为主键的字段,可以有多个 `@Id` 注解的字段作为联合主键.**
7. **默认情况下,实体类中如果不存在包含 `@Id` 注解的字段,所有的字段都会作为主键字段进行使用(这**
8. 实体类可以继承使用,可以参考测试代码中的 `tk.mybatis.mapper.model.UserLogin2` 类.
9. 由于基本类型,如int作为实体类字段时会有默认值0,而且无法消除,所以实体类中建议不要使用基本

```java
@Table(name="tb_user")// 设置表名
public class User implements Serializable{

    private static final long serialVersionUID = -4697379238361843

    // 设置这个字段为主键，并且设置主键的自增策略
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 用户名
    // 设置这个字段对应的列名，这里默认会进行驼峰匹配，所以不配置也可以
    @Column(name="user_name")
    private String userName;

    // 密码
    private String password;

    // 姓名
    private String name;

    // 年龄
    private Integer age;

    // 性别，1男性，2女性
    private Integer sex;
```

1.6、 测试

1.6.1、 查询数据

1.6.1.1、 查询 1 个用户

```
    /*
     * 查找1个用户，接收一个User对象。
     * 注意：
     *  1）会把User对象中的非空属性作为查询的条件，返回匹配到的一个用户！多个查询条件之间是AND关系
     *  2）如果查询的结果不止一个，会抛出异常。
     */
    @Test
    public void testSelectOne() {
        // 创建用户，里面的属性就是查询条件
        User = new User();
        // 根据用户名查找
        user.setUserName("liuyan");
        // 并且性别为女
        user.setSex(2);
        User result = this.uesrMapper.selectOne(user);
        System.out.println(result);
    }
```

通用 Mapper 动态生成的 SQL：

```
sion.defaults.DefaultSqlSession@182b1195] was not registered for sy
nnection from DataSource
ion [com.mysql.jdbc.JDBC4Connection@49b0380c] will not be managed b
 SELECT UPDATED,SEX,BIRTHDAY,USER_NAME USERNAME,CREATED,NAME,AGE,ID
 2(Integer), liuyan(String)
 1
ion [org.apache.ibatis.session.defaults.DefaultSqlSession@182b1195]
onnection to DataSource
```

1.6.1.2、 查询用户集合

```
/*
     * 查找用户，接收一个User对象。
     * 注意：
     *  1）会把User对象中的非空属性作为查询的条件，返回匹配到的多个用户的List集合！多个查询条件之间是AND关系
     *  2）如果想查询所有，可以传null
```

```
     */
    @Test
    public void testSelect() {
        List<User> users = this.uesrMapper.select(null);
        for (User user : users) {
            System.out.println(user);
        }
    }
```

```
G] JDBC Connection [com.mysql.jdbc.JDBC4Connection@4f7967de] will
  Preparing: SELECT BIRTHDAY,USER_NAME USERNAME,CREATED,SEX,UPDATE
 Parameters:
      Total: 9
ctional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSe
```

1.6.1.3、 查询总条数信息

```
    /*
     * 根据条件查询总条数信息,也是接收User对象作为查询条件
     * 返回符合条件的数据的条数
     */
    @Test
    public void testSelectCount() {
        // 根据条件查询总条数信息
        int count = this.userMapper.selectCount(null);
        System.out.println(count);
    }
```

```
hing JDBC Connection from DataSource
JDBC Connection [com.mysql.jdbc.JDBC4Connection@31a
=>  Preparing: SELECT COUNT(*) FROM tb_user
=> Parameters:
==      Total: 1
onal SqlSession [org.apache.ibatis.session.defaults
rning JDBC Connection to DataSource
```

1.6.1.4、 根据主键查找

```
    /*
     * 根据主键查找，注意这里的主键是指在Bean当中加了@ID注解的字段
     */
```

```
@Test
public void testSelectByPrimaryKey() {
    User user = this.userMapper.selectByPrimaryKey(1L);
    System.out.println(user);
}
```

```
C Connection from DataSource
nection [com.mysql.jdbc.JDBC4Connection@2a94b0d4] will not be manag
>  Preparing: SELECT UPDATED,BIRTHDAY,USER_NAME USERNAME,CREATED,S
> Parameters: 1(Long)
<==      Total: 1
Session [org.apache.ibatis.session.defaults.DefaultSqlSession@8432
BC Connection to DataSource
```

1.6.2、 添加数据

1）插入数据，不管字段是否为 null

```
/*
 * 添加一条数据。把接收到的对象插入数据库中，不管用户的字段是否为null。
 */
@Test
public void testInsert() {
    User user = new User();
    user.setUserName("tiebang");
    user.setName("铁棒");
    user.setAge(18);
    this.userMapper.insert(user);
}
```

```
onnection [com.mysql.jdbc.JDBC4Connection@10db448c] will not be managed by Spring
ng: INSERT INTO tb_user (CREATED,SEX,USER_NAME,AGE,UPDATED,NAME,BIRTHDAY,PASSWORD,ID)
rs: null, null, tiebang(String), 18(Integer), null, 铁棒(String), null, null, null
es: 1
>  Executing: SELECT LAST_INSERT_ID()
<      Total: 1
qlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@7017feb2]
```

2）插入数据，只插入非 null 字段

```
/*
 * 添加一条数据。把接收到的对象插入数据库中，但是仅仅插入的是非null字段
 */
@Test
```

```
public void testInsertSelective() {
    User user = new User();
    user.setUserName("wukong");
    user.setName("悟空");
    user.setAge(20);
    this.userMapper.insertSelective(user);
}
```

```
nnection [com.mysql.jdbc.JDBC4Connection@2b714bf9] will not be mana
 Preparing: INSERT INTO tb_user ( USER_NAME,AGE,NAME,ID ) VALUES (
Parameters: wukong(String), 20(Integer), 悟空(String), null
   Updates: 1
EBUG] ==>  Executing: SELECT LAST_INSERT_ID()
EBUG] <==      Total: 1
lSession [org.apache.ibatis.session.defaults.DefaultSqlSession@3752
```

1.6.3、 删除数据

1.6.3.1、 根据组合条件删除

```
/*
 * 根据条件删除。接收一个Use对象，把User的非空字段作为条件
 */
@Test
public void testDelete() {
    // 设置要删除的条件
    User user = new User();
    user.setUserName("wukong");
    // 删除用户
    this.userMapper.delete(user);
}
```

```
EBUG] JDBC Connection [com.mysql.jdbc.JDBC4Connection@1b4daa75]
==>  Preparing: DELETE FROM tb_user WHERE USER_NAME = ?
==> Parameters: wukong(String)
<==    Updates: 1
nsactional SqlSession [org.apache.ibatis.session.defaults.Defaul
] Returning JDBC Connection to DataSource
```

1.6.3.2、 根据主键删除

```
@Test
public void testDeleteByPrimaryKey() {
    // 根据主键删除
    this.userMapper.deleteByPrimaryKey(16L);
}
```

```
ction [com.mysql.jdbc.JDBC4Connection@1d7b3c8] will nc
 Preparing: DELETE FROM tb_user WHERE (ID = ? )
Parameters: 16(Long)
   Updates: 1
ssion [org.apache.ibatis.session.defaults.DefaultSqlSe
 Connection to DataSource
```

1.6.4、 修改用户

```
/*
 * 根据主键进行修改数据。
 *  1）接收一个User对象，其中的的主键ID作为修改的条件
 *  2）User的其它非null属性都会被修改
 */
@Test
public void testUpdateByPrimaryKeySelective() {
    User user = new User();
    user.setId(10L);// 用主键来确定要修改哪条数据
    user.setAge(35);// 修改年龄属性

    this.userMapper.updateByPrimaryKeySelective(user);
}
```

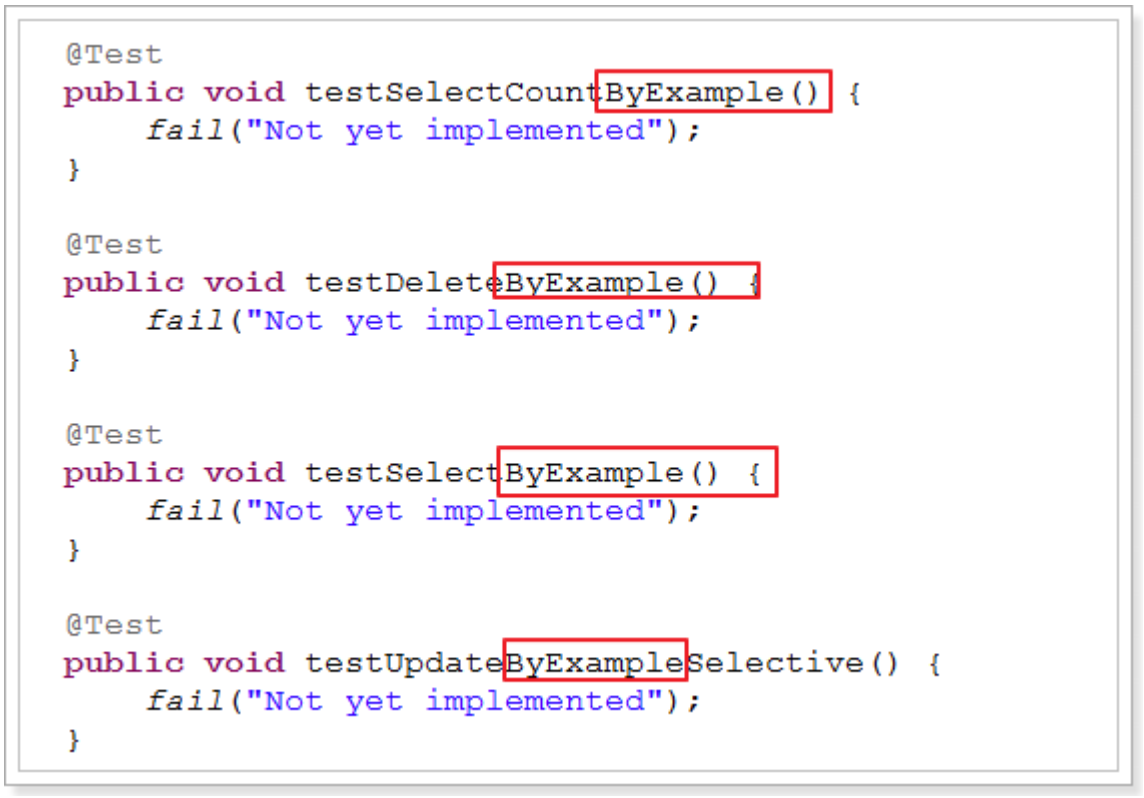
```
 [com.mysql.jdbc.JDBC4Connection@1e2161df] will not be managed by
==>  Preparing: UPDATE tb_user SET AGE = ? WHERE ID = ?
==> Parameters: 35(Integer), 10(Long)
<==    Updates: 1
 [org.apache.ibatis.session.defaults.DefaultSqlSession@3752919c]
```

1.6.5、 复杂条件的查询、删除和修改

我们发现，上面在进行删除、修改、查询时，where 后面跟的条件，都是 AND 关系，一些复杂的，例如：OR 、IN、ORDER BY 等等都无法实现

这时就要用到通用 Mapper 中提供的 Example 类了

删、改、查都有对应的 xxxByExample()方法



```
@Test
public void testSelectCountByExample() {
    fail("Not yet implemented");
}

@Test
public void testDeleteByExample() {
    fail("Not yet implemented");
}

@Test
public void testSelectByExample() {
    fail("Not yet implemented");
}

@Test
public void testUpdateByExampleSelective() {
    fail("Not yet implemented");
}
```

我们以查询为例，来介绍这个 Example 的使用

```
    // 通过Example实现特殊条件的查询
    @Test
    public void testSelectByExample() {
        // 创建Example对象，并且指定要操作的实体类的Class对象
        Example example = new Example(User.class);
        // 创建查询条件对象,默认是and关系
        example.createCriteria().andEqualTo("sex",
2).andBetween("age", 16, 24);// 查询女性,并且年龄在16到24
        // 添加查询条件，or关系
        example.or(example.createCriteria().andEqualTo("userName",
"lisi"));// 或者用户名是lisi的
        // 实现排序，多个排序规则以','隔开
        example.setOrderByClause("age desc");

        List<User> users =
```

```
this.userMapper.selectByExample(example);
        for (User user : users) {
            System.out.println(user);
        }
    }
```

生成的 SQL 语句：

```
FROM tb_user WHERE ( SEX = ? and AGE between ? and ? ) or ( USER_NAME
```

其中的参数：

```
SELECT SEX,USER_NAME USERNAME,PASSWORD,ID,AGE,NAME,CREATED,UPDATED,
2(Integer), 16(Integer), 24(Integer), lisi(String)
5
```

查询的结果：

```
2017-04-07 19:43:41,254 [main] [org.mybatis.spring.SqlSessionUtils]
2017-04-07 19:43:41,254 [main] [org.springframework.jdbc.datasource
User [id=11, userName=zhengshuang, password=123456, name=郑爽, age=23
User [id=3, userName=wangwu, password=123456, name=王五, age=22, sex=
User [id=2, userName=lisi, password=123456, name=李四, age=21, sex=2,
User [id=8, userName=liuyan, password=123456, name=柳岩, age=21, sex=
User [id=9, userName=liuyifei, password=123456, name=刘亦菲, age=18,
2017-04-07 19:43:41,256 [main] [org.springframework.test.context.su
```

1.7、 使用通用 Mapper 改造综合练习

- 改造 UserService

```
/**
 * 使用通用Mapper来实现UserService的功能
 * @author 虎哥
 */
@Service
public class AutoUserService {
    @Autowired
    private AutoUserMapper userMapper;
```

```
    /**
     * 分页查询方法
     * @param page 当前页码
     * @param rows 每页条数
     * @return
     */
    public DataGridResult<User> queryUserListByPage(Integer page,
Integer rows) {
        // 使用分页助手开始分页,指定两个参数：当前页码，每页条数
        PageHelper.startPage(page, rows);
        // 然后分页拦截器会自动对接下来的查询进行分页
        List<User> users = this.userMapper.select(null);// 不传查询条
件
        // 封装分页信息对象
        PageInfo<User> pageInfo = new PageInfo<>(users);
        // 封装页面数据对象
        DataGridResult<User> result = new
DataGridResult<>(pageInfo.getTotal(), users);

        return result;
    }

    /**
     * 新增用户的功能:
     * @param user
     */
    public void saveUser(User user) {
        user.setCreated(new Date);
        user.setUpdated(user.getCreated());
        this.userMapper.insertSelective(user);
    }
    /**
     * 删除用户功能
     * @param ids
     */
    public void deleteUserByIds(Long[] ids) {
        // 把数组变为集合
        List<Object> list = new ArrayList<>();
        // 添加元素
        Collections.addAll(list, ids);

        // 因为是删除多条，需要用到in,因此我们使用Example
        Example example = new Example(User.class);
        // 创建条件，使用in
```

```
        example.createCriteria().andIn("id", list);

        this.userMapper.deleteByExample(example);
    }
}
```

- 在 Controller 中注入新的 Service

```
@Controller
@RequestMapping("/user")
public class UserController {

    @Autowired
    private AutoUserService userService;

    // 页面跳转功能，进入用户列表页面
```

2、 RESTful 概述

2.1、 什么是 REST

作者：

- Roy Thomas Fielding博士2000年提出的
- REST是英文Representational State Transfer的缩写
  – 表象化状态转变 或者 表述性状态转移
- REST是Web服务的一种架构风格
- 使用HTTP、URI等广泛流行的标准和协议
- 轻量级、跨平台、跨语言的架构设计。

REST 仅仅是一种架构的风格，并不是真正的架构，也不是一个软件，而是一种思想。

我们可以基于现有的 HTTP、URI、XML、等现有技术来实现 REST 的风格。而不用去学习任何新的技术。

而学习 REST 的关键，不是任何的 API 或者实现方式，而是这种思想。

2.2、 什么是 RESTful

- RESTful对应的中文是 REST式的。
- RESTful Web Service 翻译为REST式的WEB服务，是遵守了REST风格的web服务。
- REST式的web服务是一种ROA（面向资源的架构）

2.3、 REST 标准规范

JAX-WS

在 REST 提出后，到了 2008 年，Java 才具有了完善的官方 REST 式的 WEB 服务标准规范：
JAX-RS 标准：

表 1-1　JAX-RS 标准和 Jersey 版本信息

JAX-RS 标准	JAX-RS 名称	标准发布时间	
311	JAX-RS 1.0	2008 年 9 月 8 日	
311	JAX-RS 1.1	2009 年 9 月 17 日	
339	JAX-RS 2.0	2013 年 5 月 22 日	

Jersey 是 Java 的 JAX-RS 标准的一种实现框架，就是一种 RESTful 的框架，基于 Jersey 开发的 WEB 应用自然就是 Restful 的 WEB 服务

SpringMVC 天生也是支持 REST 标准的。

2.3.1、 REST 风格的 WEB 设计原则

从上面的定义中，我们可以发现 REST 其实是一种组织 Web 服务的架构风格，而并不是我们想象的那样是实现 Web 服务的一种新的技术，更没有要求一定要使用 HTTP。其目标是为了创建具有良好扩展性的分布式系统。它提出了一系列架构级约束。这些约束有：

- **使用客户/服务器模型**。
  客户和服务器之间通过一个统一的接口来互相通讯。
- **层次化的系统**
  在一个 REST 系统中，客户端并不会固定地与一个服务器打交道。
- **无状态**
  在一个 REST 系统中，服务端并不会保存有关客户的任何状态。也就是说，客户端自身负责用户状态的维持，并在每次发送请求时都需要提供足够的信息。
  Http 协议本身就是无状态的协议。

- **可缓存**
  REST 系统需要能够恰当地缓存请求，以尽量减少服务端和客户端之间的信息传输，以提高性能。
- **统一的接口**
  一个 REST 系统需要使用一个统一的接口来完成子系统之间以及服务与用户之间的交互。这使得 REST 系统中的各个子系统可以独自完成演化。

如果一个系统满足了上面所列出的五条约束，那么该系统就被称为是 RESTful 的。

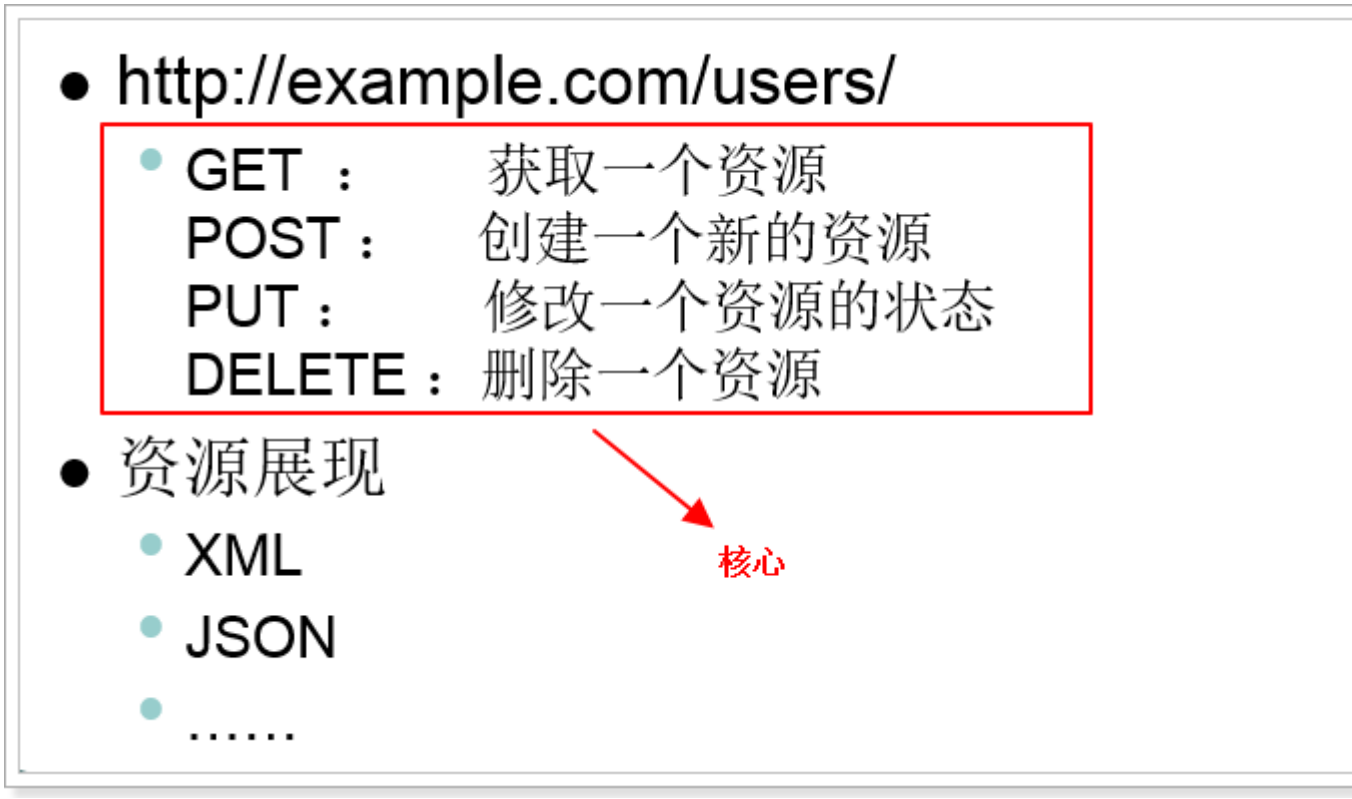
2.3.2、 如何统一接口规则

要实现统一接口，必须满足一下条件：

- 每个资源都拥有一个资源标识。每个资源的资源标识可以用来唯一地标明该资源
  REST 是围绕资源为核心的，任何接口的设计都是围绕着资源进行，而不再是围绕着业务功能
  一个典型的 URI：protocol://host/resourceName
  一般资源的 URI 中，用名词来表示一个资源，不允许出现动词

- 消息的自描述性
  在 REST 系统中所传递的消息需要能够提供自身如何被处理的足够信息。例如该消息所使用的 MIME 类型，是否可以被缓存，返回结果包含哪些字段等

- 资源操作动作
  不使用动词来描述要对资源进行的操作，那么 REST 中该如何表示 CRUD 呢？
  一般我们会借助于 HTTP 协议中的请求方法来表明对资源的操作：



举例，
以前非 REST 时，我们的 URI

查询用户： http://localhost/user/query?id=1 - GET
添加用户： http://localhost/user/insert - POST
修改用户： http://localhost/user/update - POST
删除用户： http://localhost/user/delete?id=1 - GET

遵循 REST 规范的 URI 定义：

查询用户： http://localhost/user/{id} - GET
添加用户： http://localhost/user - POST
修改用户： http://localhost/user - PUT
删除用户： http://localhost/user - DELETE

3、 REST 最佳实践

3.1、 接口设计

REST 中的最佳实践：

3.2、 响应设计

3.2.1、 响应规则

- Content body 仅仅用来传输数据。
- 数据要做到拿来就可用的原则，不需要“拆箱”的过程。
- 用来描述数据或者请求的元数据放Header中，例如X-Result-Fields。

示例：

错误的做法

```
{
    "status": 200,
    "data": {
        "trade_id": 1234,
        "trade_name": "Bala bala"
    }
}
```

正确的做法

```
Response Headers:
    Status: 200
Response Body:
    {
        "trade_id": 1234,
        "trade_name": "Bala bala"
    }
```

3.2.2、 响应字段

无状态服务器应该允许客户端对数据按需提取。在请求头使用`X-Result-Fields`指定数据返回的字段
例如：trade 有`trade_id`, `trade_name`, `created_at` 三个属性，客户端只需其中的`trade_id`与`tr`

Request Header

```
X-Result-Fields: trade_id,trade_name
```

3.2.3、 响应状态码

Code	HTTP Operation	Body Contents	Descri
200	GET,PUT	资源	操作成功
201	POST	资源,元数据	对象创建成功
202	POST,PUT,DELETE,PATCH	N/A	请求已经被接受
204	DELETE,PUT,PATCH	N/A	操作已经执行成功，但
301	GET	link	资源已被移除
303	GET	link	重定向
304	GET	N/A	资源没有被修改
400	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	参数列表错误(缺少，格
401	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	未授权
403	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	访问受限，授权过期
404	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	资源，服务未找到
405	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	不允许的http方法
409	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	资源冲突，或者资源被
415	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	不支持的数据(媒体)类型
429	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	请求过多被限制
500	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	系统内部错误
501	GET,PSOT,PUT,DELETE,PATCH	错误提示(消息)	接口未实现

4、 REST 实现用户 CRUD

4.1、 根据 ID 查询用户

根据 REST 的原则，我们查询后需要根据结果返回不同的状态码：

资源存在：200
资源不存在：404
内部异常：500

这里通过 ResponseEntity 来封装响应的结果，包含两部分内容：数据和状态码

- 定义 Controller

```
/**
 * 根据ID查询用户
 * @param id
 * @return
  */
    @RequestMapping(value="{id}",method=RequestMethod.GET)
    public ResponseEntity<User> queryUserById(@PathVariable("id")
Long id){
        try {
            // 查询用户
            User user = this.userService.queryUserById(id);
            if(user != null){
                // 资源存在，返回200
                return new ResponseEntity<>(user, HttpStatus.OK);
            }
            // 资源不存在，返回404
```

```
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        } catch (Exception e) {
            e.printStackTrace();
            // 出现异常，返回500
            return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

4.2、 新增用户

在原来的新增用户功能中，我们返回的结果包含了 status，也就是状态，这不符合 REST 风格。响应状态应该包含在 header 中
并且新增成功，应返回 201 状态码

Code	HTTP Operation	Body Contents	Desc
200	GET,PUT	资源	操作成功
201	POST	资源,元数据	对象创建成功
202	POST,PUT,DELETE,PATCH	N/A	请求已经被接受
204	DELETE,PUT,PATCH	N/A	操作已经执行成功，

```
    /**
     * 新增用户
     * @param user
     * @return
     */
    @RequestMapping(method=RequestMethod.POST)
    public ResponseEntity<Void> saveUser(User user){
        try {
            // 新增用户
            this.userService.saveUser(user);
            // 新增成功，返回201
            return new ResponseEntity<Void>(HttpStatus.CREATED);
        } catch (Exception e) {
            e.printStackTrace();
            // 出现异常，返回500
            return new
```

```
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

4.3、 修改用户

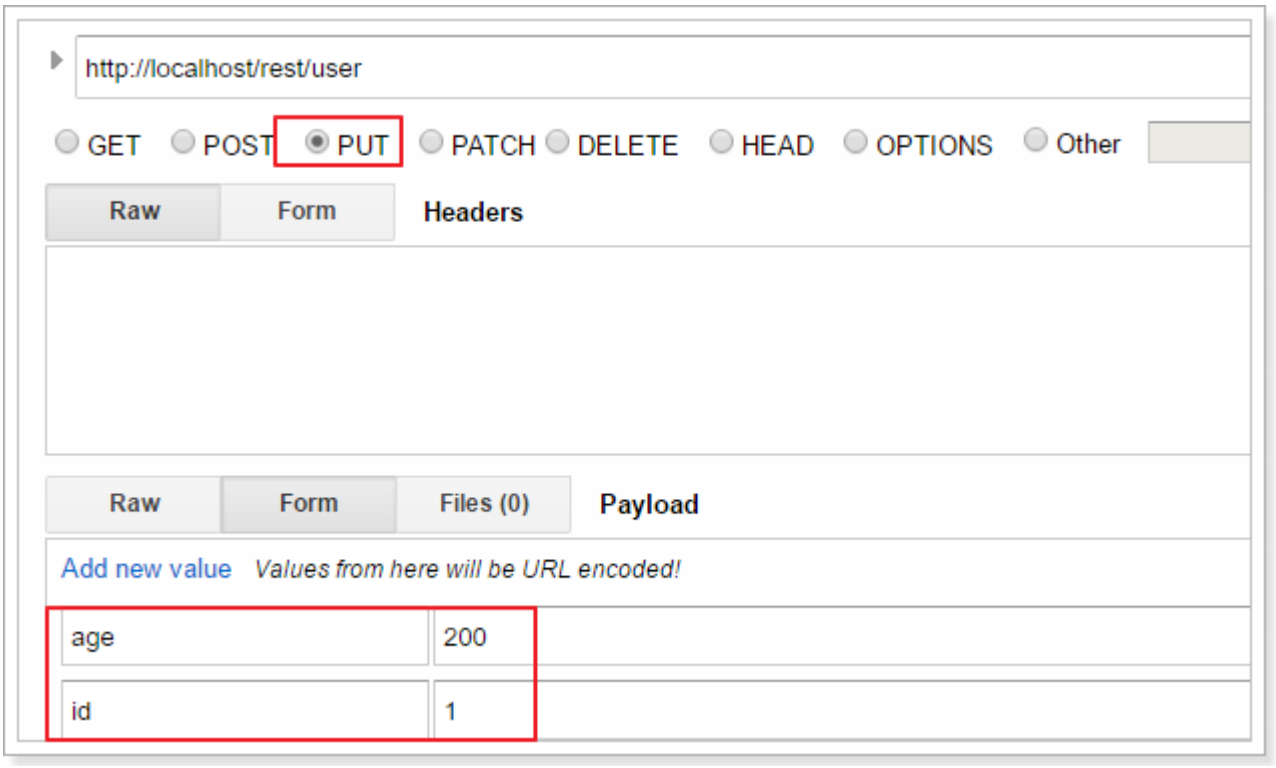
4.3.1、 编写 Controller

修改成功，但是没有结果返回，根据 REST 原则，返回 204：

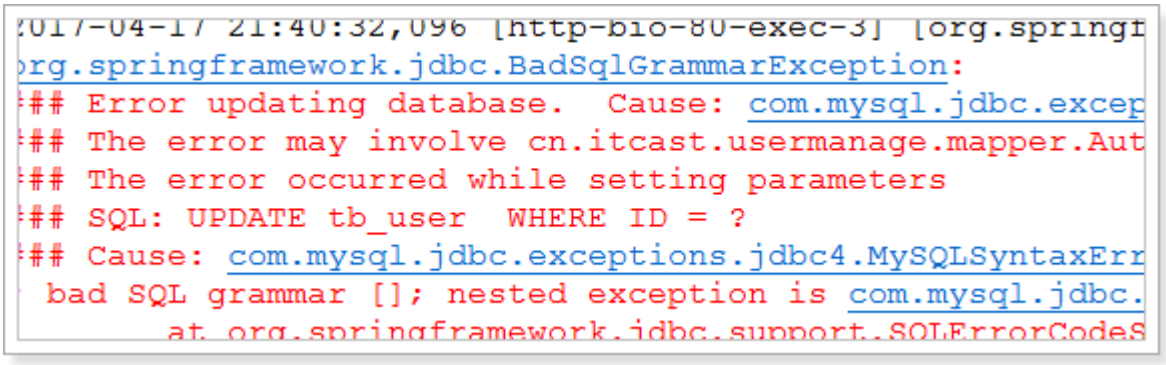
Code	HTTP Operation	Body Contents	Desc
200	GET,PUT	资源	操作成功
201	POST	资源,元数据	对象创建成功
202	POST,PUT,DELETE,PATCH	N/A	请求已经被接受
204	DELETE,PUT,PATCH	N/A	操作已经执行成功，

```
    /**
     * 修改用户
     * @param user
     * @return
     */
    @RequestMapping(method=RequestMethod.PUT)
    public ResponseEntity<Void> updateUser(User user){
        try {
            // 修改用户
            this.userService.updateUser(user);
            // 成功，返回204 ,操作成功，但是不返回数据
            return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
        } catch (Exception e) {
            e.printStackTrace();
            // 出现异常，返回500
            return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

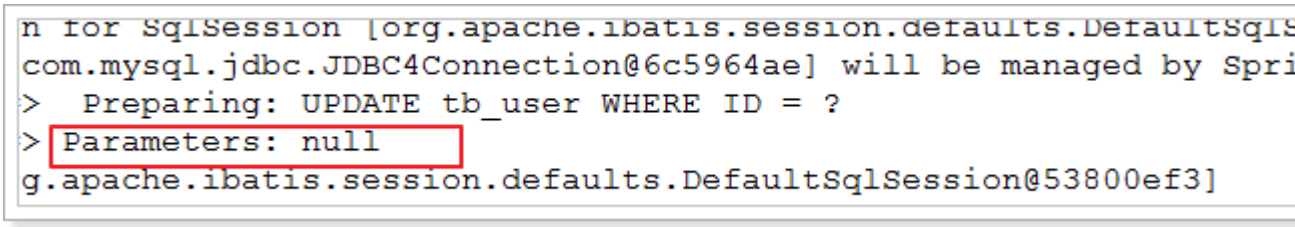
4.3.2、 模拟 Put 请求进行测试



发生异常：



发现我们的参数都是 null：



原因：

Java 本身是不支持 PUT 和 DELTE 请求，当发起这些请求时，可以接收，但是请求参数接

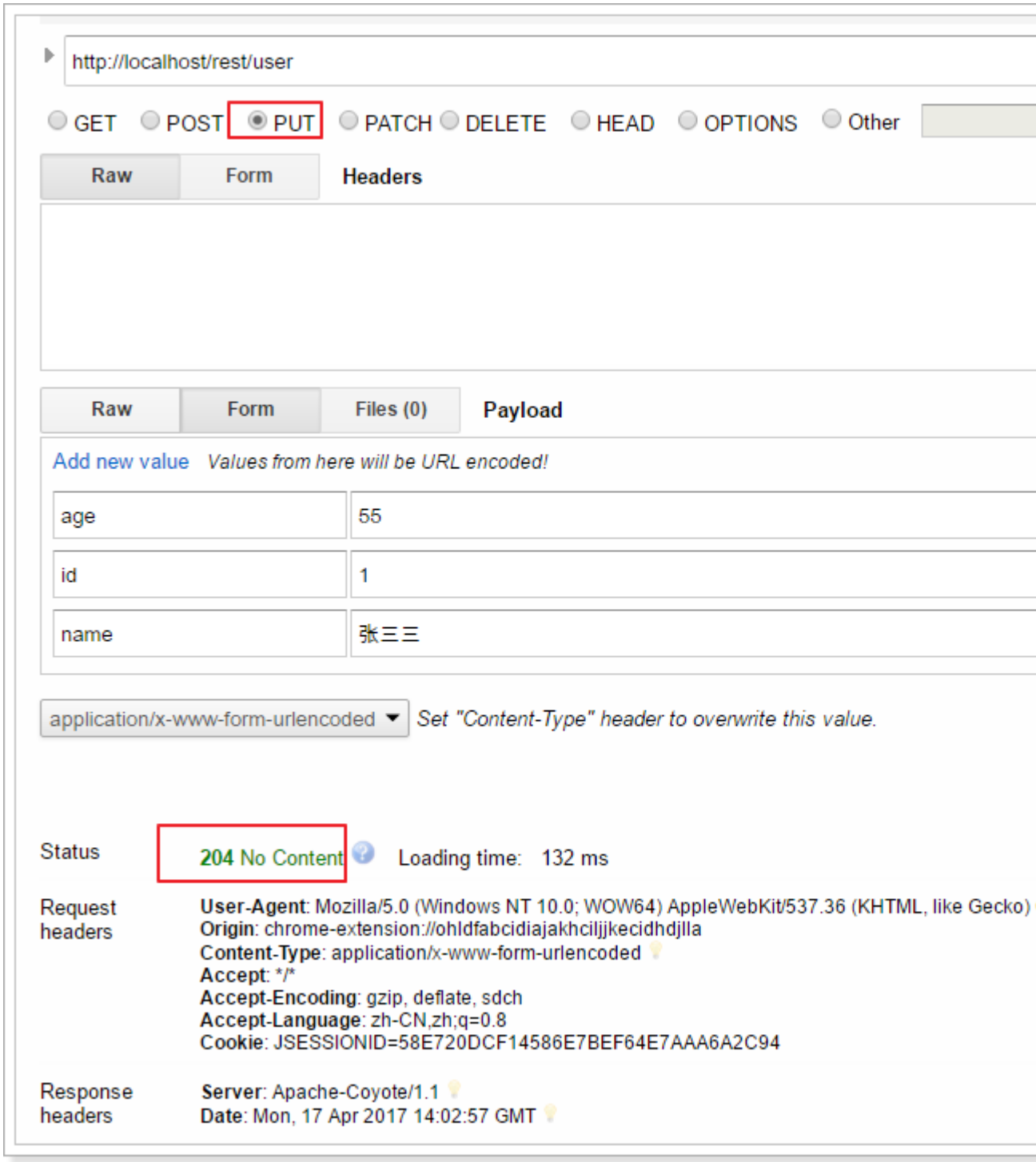
收不到！

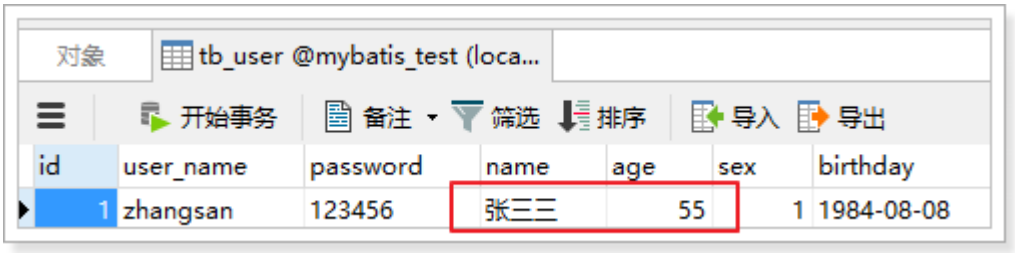
4.3.3、 解决 PUT 请求参数为 null

SpringMVC 提供了一个过滤器，可以帮我们解析 PUT 请求的参数：

```
    <!-- 解决PUT请求的参数为null问题 -->
    <filter>
        <filter-name>HttpMethodFilter</filter-name>
        <filter-
class>org.springframework.web.filter.HttpPutFormContentFilter</fi
lter-class>
    </filter>
    <filter-mapping>
        <filter-name>HttpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

再次测试：

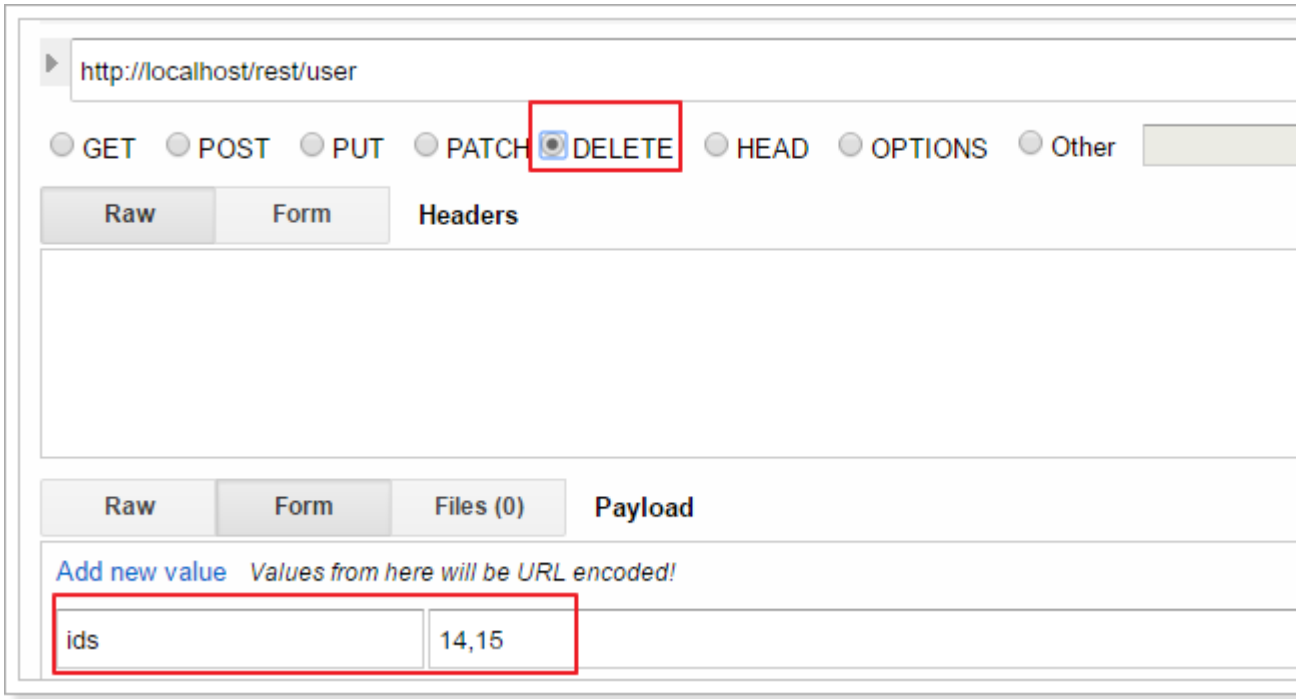

http://localhost/rest/user
GET POST PUT PATCH DELETE HEAD OPTIONS Other
Raw Form Headers
Raw Form Files (0) Payload
Add new value Values from here will be URL encoded!
age 55
id 1
name 张三三
application/x-www-form-urlencoded Set "Content-Type" header to overwrite this value.
Status 204 No Content Loading time: 132 ms
Request headers
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Origin: chrome-extension://ohldfabcidiajakhciljjkecidhdjlla
Content-Type: application/x-www-form-urlencoded
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Cookie: JSESSIONID=58E720DCF14586E7BEF64E7AAA6A2C94
Response headers
Server: Apache-Coyote/1.1
Date: Mon, 17 Apr 2017 14:02:57 GMT


对象
tb_user @mybatis_test (loca...
开始事务 备注 筛选 排序 导入 导出
id user_name password name age sex birthday
1 zhangsan 123456 张三三 55 1 1984-08-08

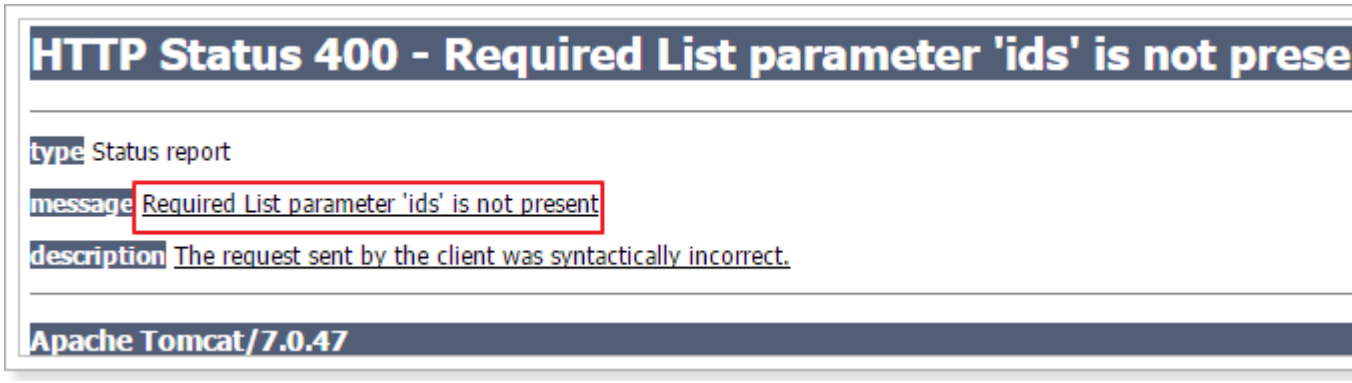
4.3.4、 删除用户

```
    /**
     * 删除用户
     * @param user
     * @return
     */
    @RequestMapping(method=RequestMethod.DELETE)
    public ResponseEntity<Void>
deleteUser(@RequestParam("ids")Long[] ids){
        try {
            // 删除用户
            this.userService.deleteUserByIds(ids);
            // 成功，返回204 ,操作成功，但是不返回数据
            return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
        } catch (Exception e) {
            e.printStackTrace();
            // 出现异常，返回500
            return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

测试：



再次出错：



参数依然没有传递过来。也就是说，上面解决 PUT 请求的过滤器，对 DELETE 请求没用。

需要引入一个新的过滤器：

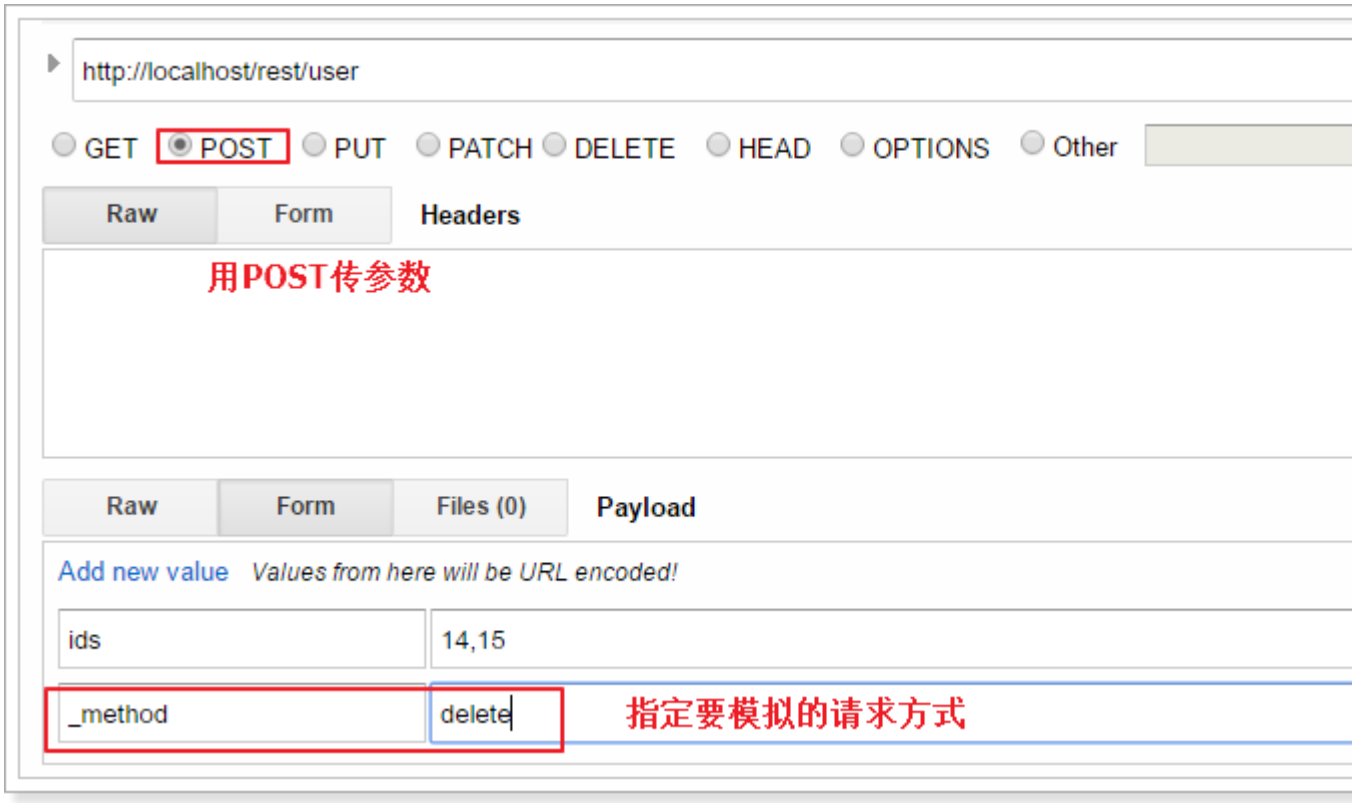
```
    <!-- 通过POST请求传递参数，再 用_method指定要转化的请求方式（DELETE或
PUT），最后转为DELETE或PUT请求 -->
    <filter>
        <filter-name>HiddenHttpMethodFilter</filter-name>
        <filter-
class>org.springframework.web.filter.HiddenHttpMethodFilter</filt
er-class>
    </filter>
    <filter-mapping>
        <filter-name>HiddenHttpMethodFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

这个过滤器并不能真正的去解决 DELETE 请求问题，而是需要在页面发请求时，满足两点：

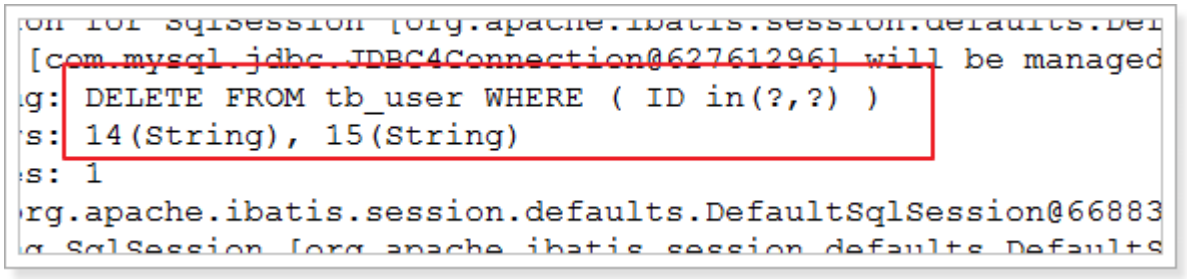
1） 依然发 POST 请求，

2） 同时传递_method 参数，指定要模拟的请求方式

然后过滤器通过 POST 接收参数，然后再把请求方式改为 DELETE



成功接收：



5、 改造前端 JS

5.1、 查询用户列表首页：

5.1.1、 定义分页查询 controller

```
/**
 * 查询全部用户
 * @return
 */
@RequestMapping(method=RequestMethod.GET)
```

```
    @ResponseBody
    public ResponseEntity<DataGridResult<User>> queryUserByPage(
            @RequestParam("page")Integer page,
            @RequestParam("rows")Integer rows){
        try {
            // 查询用户
            DataGridResult<User> result =
this.userService.queryUserListByPage(page, rows);
            if(result != null){
                // 资源存在，返回200
                return new ResponseEntity<>(result, HttpStatus.OK);
            }
            // 资源不存在，返回404
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        } catch (Exception e) {
            e.printStackTrace();
            // 出现异常，返回500
            return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

5.1.2、 查询数据页面 JS

```
<div>
<table class="easyui-datagrid" id="userList" title="会员列表"
        data-options="singleSelect:false,collapsible:true,paginatio
    <thead>
        <tr>
```

5.2、 新增用户 JS

```
function submitForm(){
    if(!$('#content').form('validate')){
        $.messager.alert('提示','表单还未填写完成！');
        return ;
    }
    $.ajax({
            type: "POST",
            url: "/rest/user",
            data: $("#content").serialize(),
            statusCode:{
                201:function(){
                    $.messager.alert('提示','新增会员成功！');
                     $('#userAdd').window('close');
                     $("#userList").datagrid("reload");
                     clearForm();
                },
                500:function(){
                    $.messager.alert('提示','新增会员失败！');
                }
            }
        });
```

作业 1：配置 mybatis 插件时，通用 Mapper 和分页助手的顺序是否可以调整？为什么？
作业 2：实现修改和删除的 JS 改造为 REST 风格

5.3、 删除用户 JS

```
text:'删除',
iconCls:'icon-cancel',
handler:function(){
    var ids = getSelectionsIds();
    if(ids.length == 0){
        $.messager.alert('提示','未选中用户!');
        return ;
    }
    $.messager.confirm('确认','确定删除ID为 '+ids+' 的会员吗? ',function(r)
        if (r){
            $.ajax({
               type: "POST",
               url: "/rest/user",
               data: {"ids":ids,"_method":"delete"},
               statusCode:{
                    204:function(){
                       $.messager.alert('提示','删除会员成功!',undefined
                         $("#userList").datagrid("reload");
                       });
                    },
                    500:function(){
                        $.messager.alert('提示','新增会员失败!');
                    }
               }
            });
        }
    });
```