# RESTful

## REST是什么？

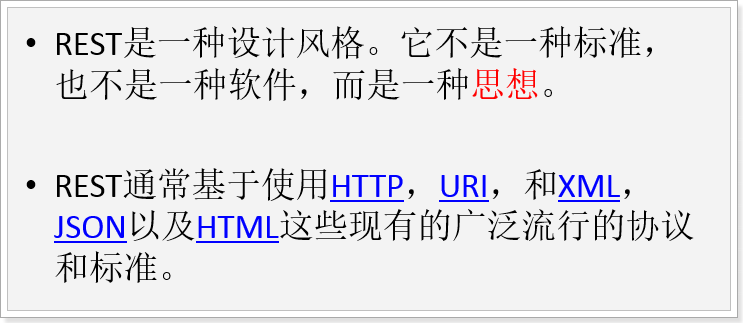
大神：



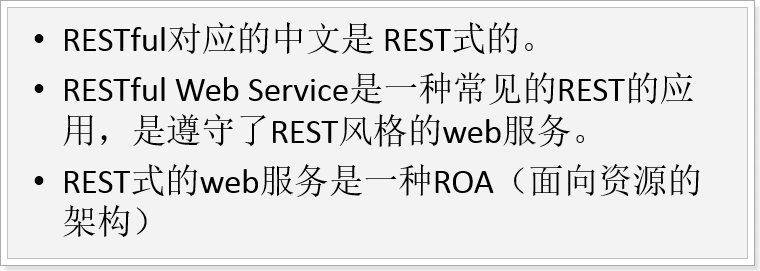
论文：Roy Thomas Fielding博士论文REST(中文版).pdf



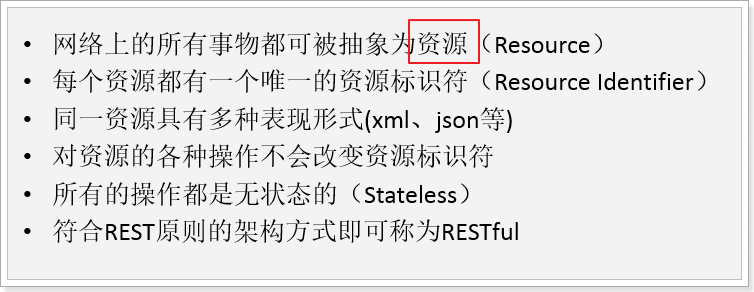
## REST到底是什么？



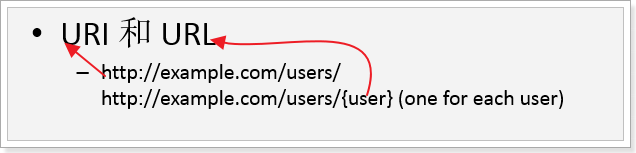
## RESTful是什么？



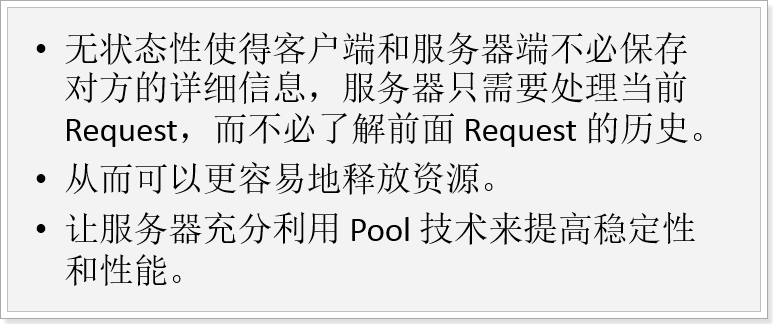
## REST 架构的主要原则



## URI和URL



## 无状态性



## 资源操作



之前的操作：

<http://127.0.0.1/user/query/1> GET 根据用户id查询用户数据

http://127.0.0.1/user/save POST 新增用户

http://127.0.0.1/user/update POST 修改用户信息

http://127.0.0.1/user/delete GET/POST 删除用户信息

RESTful用法：

<http://127.0.0.1/user/1> GET 根据用户id查询用户数据

<http://127.0.0.1/user> POST 新增用户

<http://127.0.0.1/user> PUT 修改用户信息

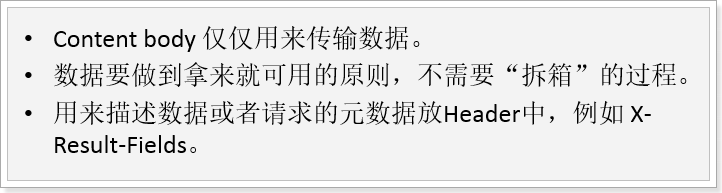
<http://127.0.0.1/user> DELETE 删除用户信息

## 最佳实践

### REST接口设计



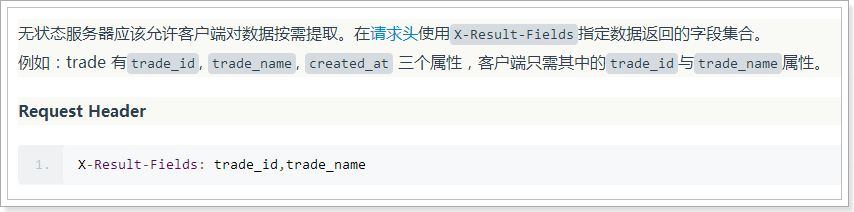
### 响应设计



### 响应示例



### 指定响应的属性字段



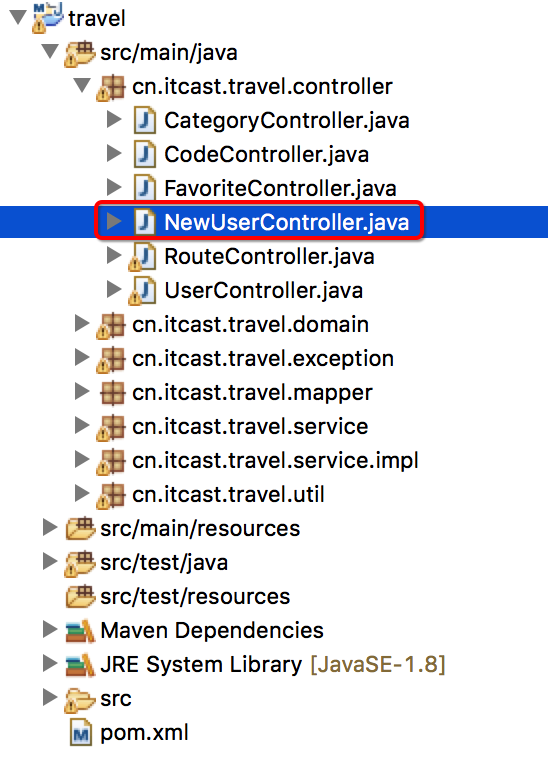
# http响应状态码



# SpringMVC实现RESTful服务



创建NewUserController,采用rest风格来设计这个Controller



## 查询资源

在NewUserController中创建根据uid查询用户的方法

@Controller

@RequestMapping("rest/user")

**public** **class** NewUserController {

@Autowired

**private** UserService userService;

/\*\*

\* 根据uid查询用户对象

\* **@param** uid

\* **@return**

\*/

@GetMapping("{uid}")

**public** ResponseEntity<User> findUserById(@PathVariable("uid")Integer uid){

**try** {

**if**(uid == **null** || uid <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).body(**null**);

}

User user = userService.findUserById(uid);

**if**(user == **null**){

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).body(**null**);

}

//查询成功：200

// return ResponseEntity.status(HttpStatus.OK).body(user);

**return** ResponseEntity.*ok*(user);

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

}

在UserService中编写findUserById方法：

**public** **interface** UserService {

/\*\*

\* 用户注册

\* **@param** user

\* **@return**

\* **@throws** Exception

\*/

**boolean** register(User user) **throws** Exception;

/\*\*

\* 用户激活

\* **@param** code

\* **@return**

\*/

**boolean** active(String code);

/\*\*

\* 用户登陆

\* **@param** username

\* **@param** password

\* **@return**

\* **@throws** Exception

\*/

User login(String username, String password) **throws** Exception;

/\*\*

\* 根据uid查询用户

\* **@param** id

\* **@return**

\*/

User findUserById(Integer uid);

}

在UserServiceImpl中实现findUserById方法：

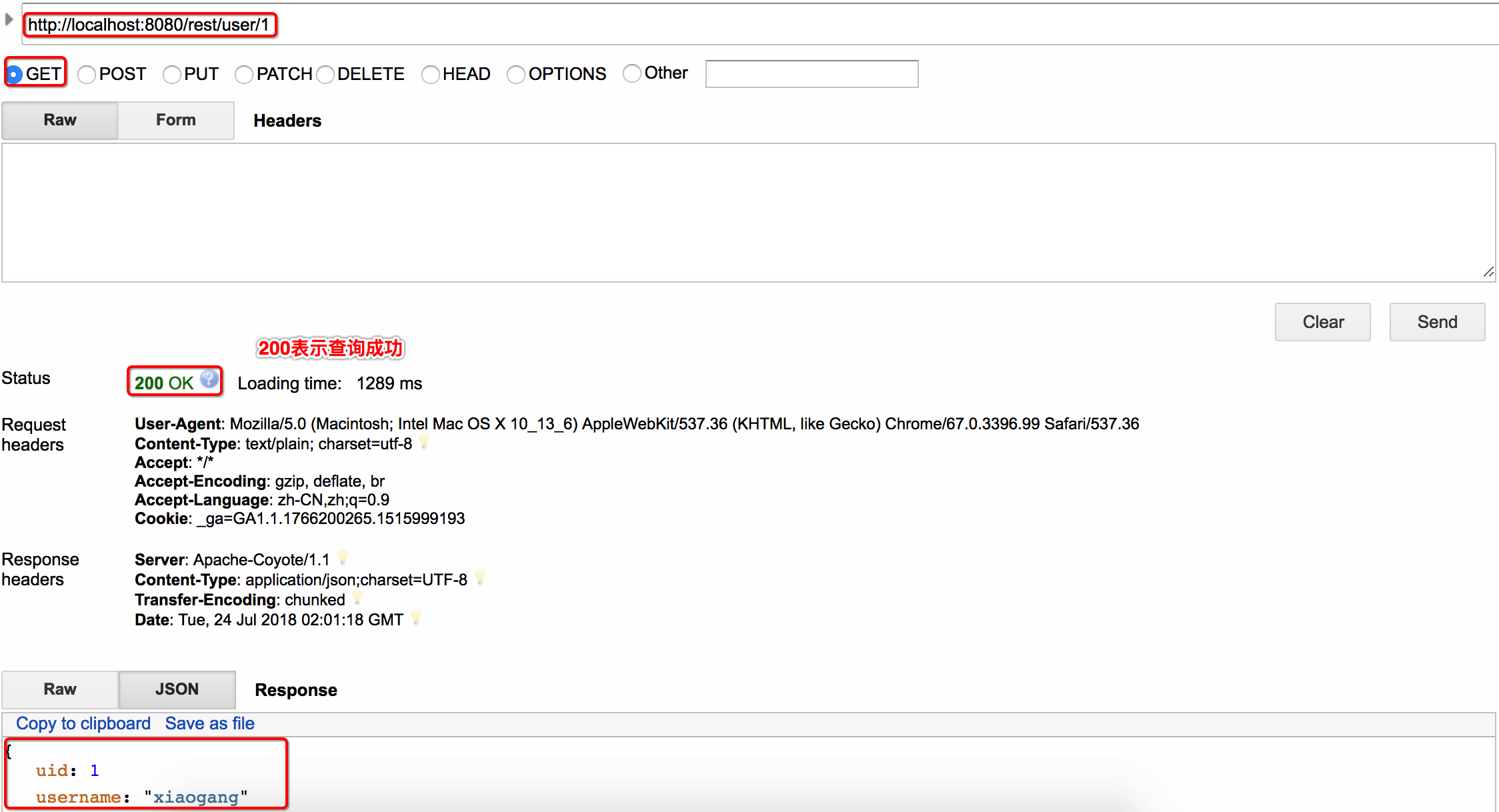
@Override

**public** User findUserById(Integer uid) {

**return** userMapper.selectByPrimaryKey(uid);

}

测试：



## 新增资源

在NewUserController中编写保存用户的方法：

@Controller

@RequestMapping("rest/user")

**public** **class** NewUserController {

@Autowired

**private** UserService userService;

/\*\*

\* 根据uid查询用户对象

\* **@param** uid

\* **@return**

\*/

@GetMapping("{uid}")

**public** ResponseEntity<User> findUserById(@PathVariable("uid")Integer uid){

**try** {

**if**(uid == **null** || uid <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).body(**null**);

}

User user = userService.findUserById(uid);

**if**(user == **null**){

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).body(**null**);

}

//查询成功：200

// return ResponseEntity.status(HttpStatus.OK).body(user);

**return** ResponseEntity.*ok*(user);

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

/\*\*

\* 保存用户

\* **@param** user

\* **@return**

\*/

@PostMapping

**public** ResponseEntity<Void> saveUser(User user){

**try** {

**if**(StringUtils.*isEmpty*(user.getUsername()) || StringUtils.*isEmpty*(user.getPassword())){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

Boolean flag = userService.saveUser(user);

**if**(flag){

//保存成功：201

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).build();

}

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

}

在UserService中编写保存用户的方法：

**public** **interface** UserService {

/\*\*

\* 用户注册

\* **@param** user

\* **@return**

\* **@throws** Exception

\*/

**boolean** register(User user) **throws** Exception;

/\*\*

\* 用户激活

\* **@param** code

\* **@return**

\*/

**boolean** active(String code);

/\*\*

\* 用户登陆

\* **@param** username

\* **@param** password

\* **@return**

\* **@throws** Exception

\*/

User login(String username, String password) **throws** Exception;

/\*\*

\* 根据uid查询用户

\* **@param** id

\* **@return**

\*/

User findUserById(Integer uid);

/\*\*

\* 保存用户

\* **@param** user

\* **@return**

\*/

Boolean saveUser(User user);

}

在UserServiceImpl中实现saveUser方法：

@Override

**public** Boolean saveUser(User user) {

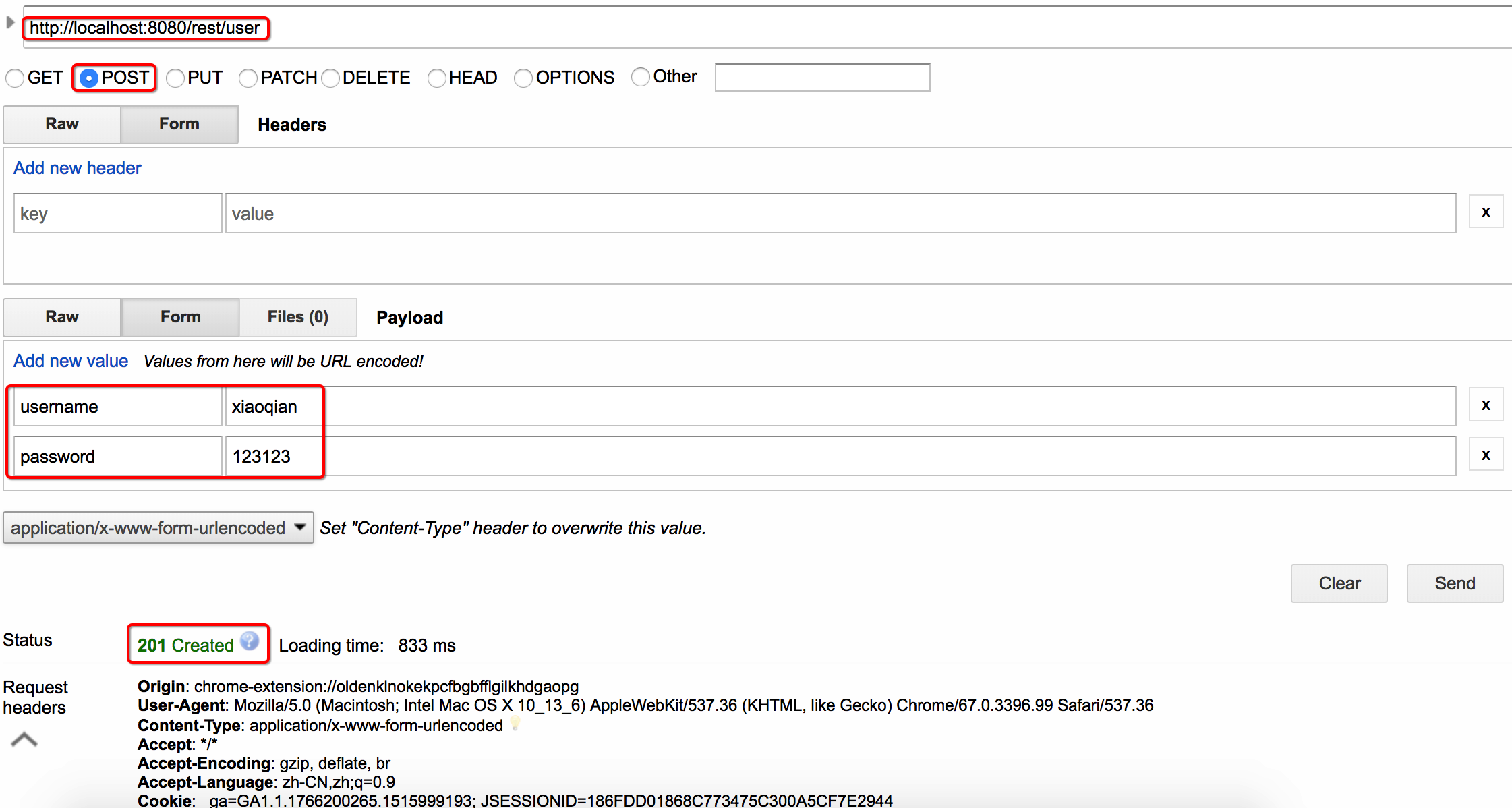
user.setStatus("N");

user.setCode(UuidUtil.*getUuid*());

**return** userMapper.insert(user) > 0;

}

测试：



## 更新资源

在NewUserController中编写更新用户的方法：

@Controller

@RequestMapping("rest/user")

**public** **class** NewUserController {

@Autowired

**private** UserService userService;

/\*\*

\* 根据uid查询用户对象

\* **@param** uid

\* **@return**

\*/

@GetMapping("{uid}")

**public** ResponseEntity<User> findUserById(@PathVariable("uid")Integer uid){

**try** {

**if**(uid == **null** || uid <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).body(**null**);

}

User user = userService.findUserById(uid);

**if**(user == **null**){

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).body(**null**);

}

//查询成功：200

// return ResponseEntity.status(HttpStatus.OK).body(user);

**return** ResponseEntity.*ok*(user);

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

/\*\*

\* 保存用户

\* **@param** user

\* **@return**

\*/

@PostMapping

**public** ResponseEntity<Void> saveUser(User user){

**try** {

**if**(StringUtils.*isEmpty*(user.getUsername()) || StringUtils.*isEmpty*(user.getPassword())){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

Boolean flag = userService.saveUser(user);

**if**(flag){

//保存成功：201

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).build();

}

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

/\*\*

\* 更新用户

\* **@param** user

\* **@return**

\*/

@PutMapping

**public** ResponseEntity<Void> updateUser(User user){

**try** {

**if**(user.getUid() == **null** || user.getUid() <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

Boolean flag = userService.updateUser(user);

**if**(flag){

//更新成功：204

**return** ResponseEntity.*status*(HttpStatus.***NO\_CONTENT***).build();

}**else**{

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).build();

}

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

}

在UserService中编写更新用户的方法：

**public** **interface** UserService {

/\*\*

\* 用户注册

\* **@param** user

\* **@return**

\* **@throws** Exception

\*/

**boolean** register(User user) **throws** Exception;

/\*\*

\* 用户激活

\* **@param** code

\* **@return**

\*/

**boolean** active(String code);

/\*\*

\* 用户登陆

\* **@param** username

\* **@param** password

\* **@return**

\* **@throws** Exception

\*/

User login(String username, String password) **throws** Exception;

/\*\*

\* 根据uid查询用户

\* **@param** id

\* **@return**

\*/

User findUserById(Integer uid);

/\*\*

\* 保存用户

\* **@param** user

\* **@return**

\*/

Boolean saveUser(User user);

/\*\*

\* 更新用户

\* **@param** user

\* **@return**

\*/

Boolean updateUser(User user);

Boolean deleteUser(Integer uid);

}

在UserServiceImpl中实现updateUser方法：

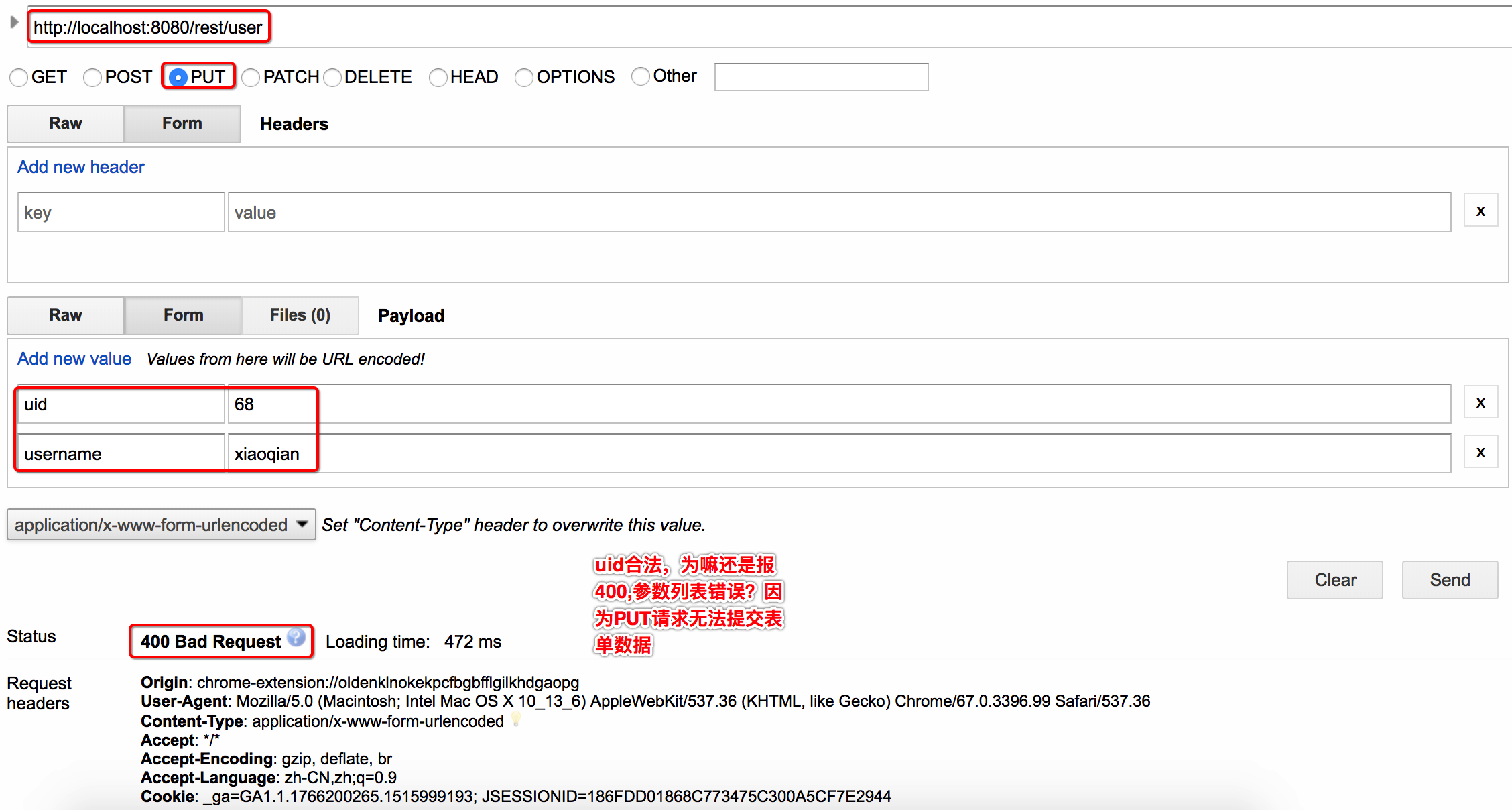
@Override

**public** Boolean updateUser(User user) {

**return** userMapper.updateByPrimaryKeySelective(user) > 0;

}

测试：



默认情况下，PUT请求是无法提交表单数据的，需要在web.xml中添加过滤器解决：

<!-- 解决PUT请求无法提交表单数据的问题 -->

<filter>

<filter-name>HttpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HttpPutFormContentFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>HttpMethodFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## 删除资源

在NewUserController中编写根据uid删除用户的方法：

@Controller

@RequestMapping("rest/user")

**public** **class** NewUserController {

@Autowired

**private** UserService userService;

/\*\*

\* 根据uid查询用户对象

\* **@param** uid

\* **@return**

\*/

@GetMapping("{uid}")

**public** ResponseEntity<User> findUserById(@PathVariable("uid")Integer uid){

**try** {

**if**(uid == **null** || uid <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).body(**null**);

}

User user = userService.findUserById(uid);

**if**(user == **null**){

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).body(**null**);

}

//查询成功：200

// return ResponseEntity.status(HttpStatus.OK).body(user);

**return** ResponseEntity.*ok*(user);

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

/\*\*

\* 保存用户

\* **@param** user

\* **@return**

\*/

@PostMapping

**public** ResponseEntity<Void> saveUser(User user){

**try** {

**if**(StringUtils.*isEmpty*(user.getUsername()) || StringUtils.*isEmpty*(user.getPassword())){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

Boolean flag = userService.saveUser(user);

**if**(flag){

//保存成功：201

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).build();

}

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

/\*\*

\* 更新用户

\* **@param** user

\* **@return**

\*/

@PutMapping

**public** ResponseEntity<Void> updateUser(User user){

**try** {

**if**(user.getUid() == **null** || user.getUid() <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

Boolean flag = userService.updateUser(user);

**if**(flag){

//更新成功：204

**return** ResponseEntity.*status*(HttpStatus.***NO\_CONTENT***).build();

}**else**{

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).build();

}

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

/\*\*

\* 根据uid删除用户

\* **@param** uid

\* **@return**

\*/

@DeleteMapping

**public** ResponseEntity<Void> deleteUser(@RequestParam("uid")Integer uid){

**try** {

**if**(uid == **null** || uid <= 0){

//参数列表错误：400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

Boolean flag = userService.deleteUser(uid);

**if**(flag){

//删除成功：204

**return** ResponseEntity.*status*(HttpStatus.***NO\_CONTENT***).build();

}**else**{

//资源找不到：404

**return** ResponseEntity.*status*(HttpStatus.***NOT\_FOUND***).build();

}

} **catch** (Exception e) {

e.printStackTrace();

}

//服务器错误：500

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

}

在UserService中编写根据uid删除用户的方法：

**public** **interface** UserService {

/\*\*

\* 用户注册

\* **@param** user

\* **@return**

\* **@throws** Exception

\*/

**boolean** register(User user) **throws** Exception;

/\*\*

\* 用户激活

\* **@param** code

\* **@return**

\*/

**boolean** active(String code);

/\*\*

\* 用户登陆

\* **@param** username

\* **@param** password

\* **@return**

\* **@throws** Exception

\*/

User login(String username, String password) **throws** Exception;

/\*\*

\* 根据uid查询用户

\* **@param** id

\* **@return**

\*/

User findUserById(Integer uid);

/\*\*

\* 保存用户

\* **@param** user

\* **@return**

\*/

Boolean saveUser(User user);

/\*\*

\* 更新用户

\* **@param** user

\* **@return**

\*/

Boolean updateUser(User user);

/\*\*

\* 删除用户

\* **@param** uid

\* **@return**

\*/

Boolean deleteUser(Integer uid);

}

在UserServiceImpl中实现deleteUser方法：

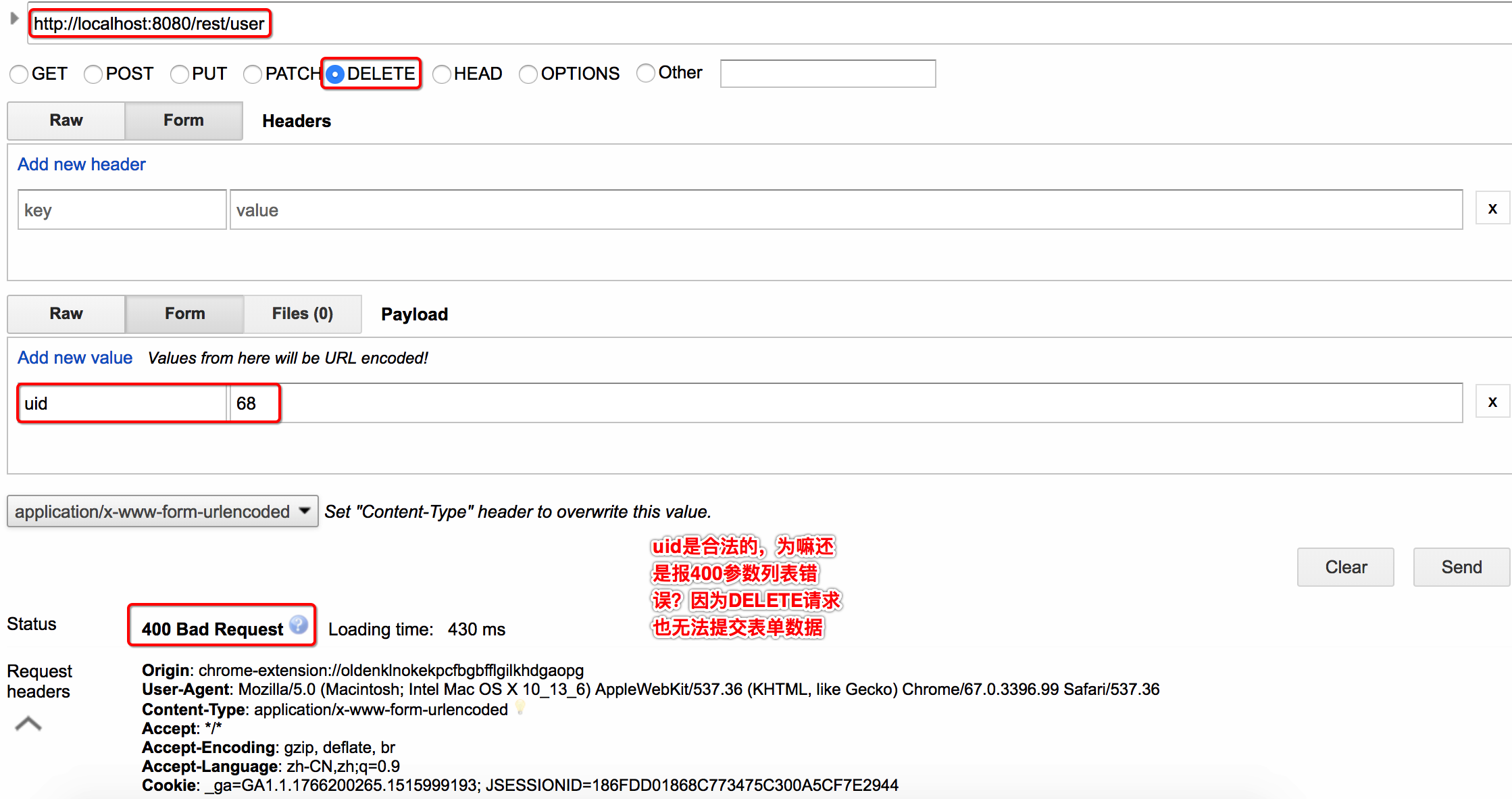
@Override

**public** Boolean deleteUser(Integer uid) {

**return** userMapper.deleteByPrimaryKey(uid) > 0;

}

测试：



需要在web.xml中添加过滤器解决DELETE请求无法提交表单数据的问题：

<!-- 将POST请求转化为PUT或DELETE请求，要用\_method参数来指定真正的请求方式-->

<filter>

<filter-name>HiddenHttpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>HiddenHttpMethodFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

再次测试，需要在请求中添加\_method参数，指定请求方式为DELETE请求：

