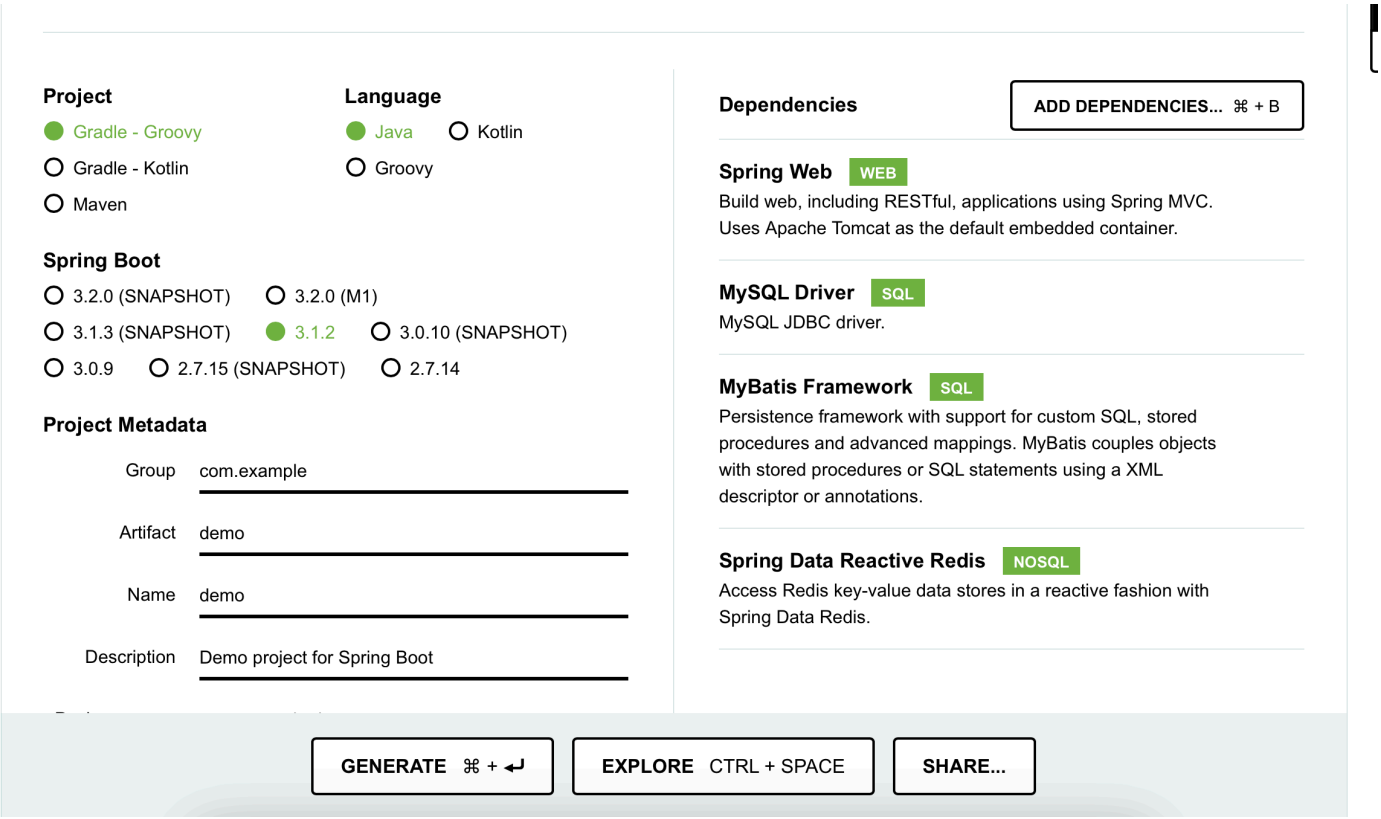
Springboot学习笔记（第一个项目）

> 相关技术：Springboot、Maven、Mysql、Mybatis、Redis、CRUD、统一异常处理、AOP、redis存储session(filter拦截登录)

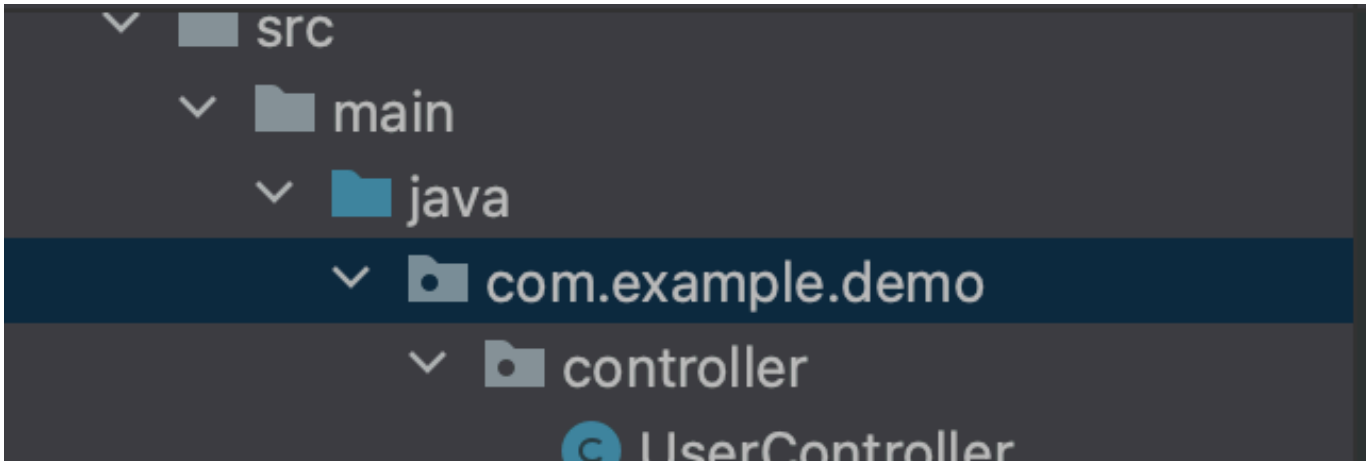
项目初始化

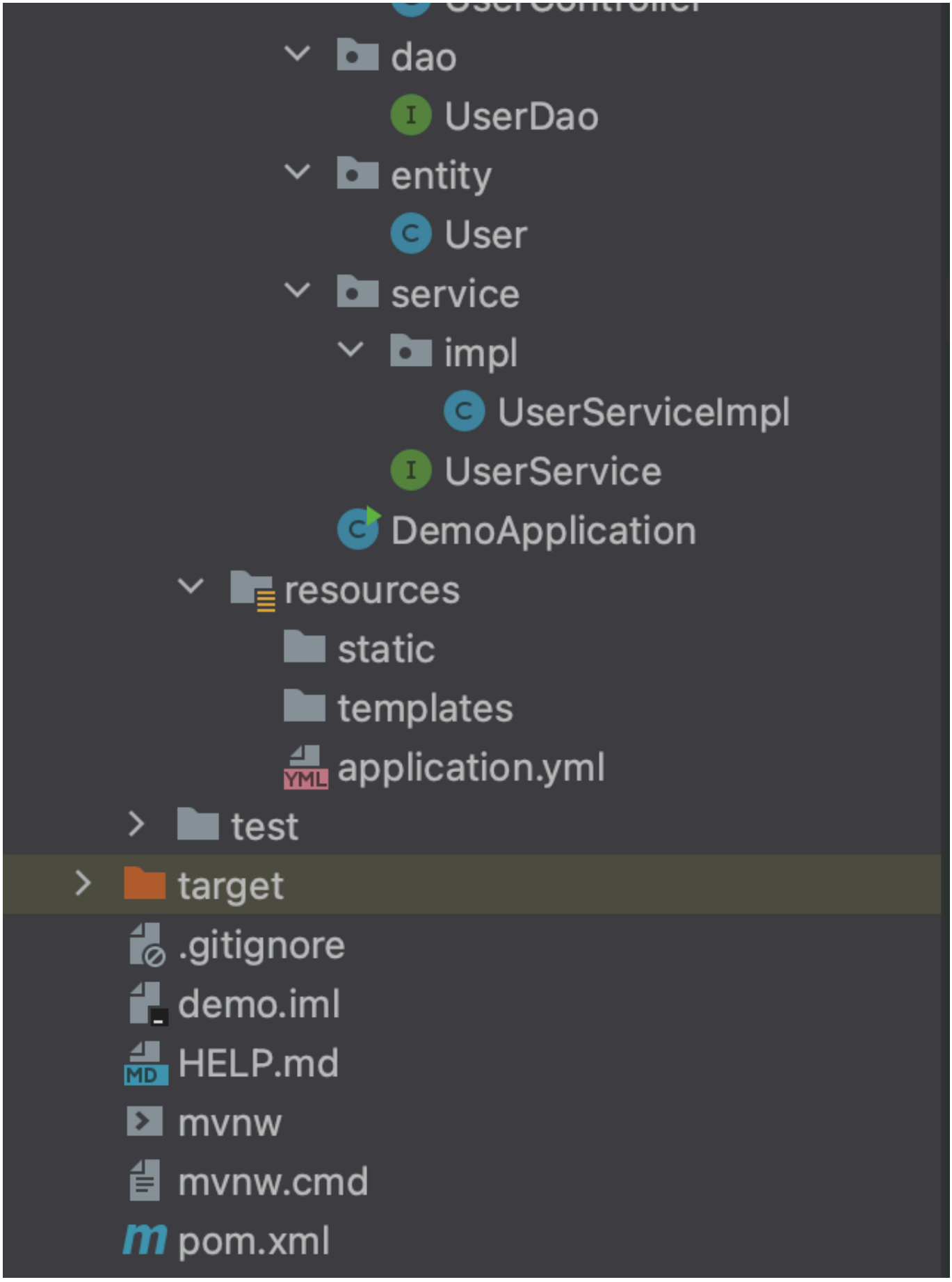
在官方网站 https://start.spring.io 中，我们可以下载到Springboot基础框架。



在**Project**中我们可以选择构建工具，在右边的**Dependencies**中添加需要的依赖，最后点击**GENERATE**生成压缩包。

项目结构





一般来说，一个项目的结构设计到如下文件：

- pom.xml： **maven**的配置文件，用于说明继承自什么框架，添加需要的依赖等等
- controller层：用于处理前端的请求，通过调用Service层的后端接口进行处理

- service层：真正用于处理的层，一般分为接口和实现两部分
- dao层：用于访问数据库的层
- entity：数据库表对应的实体对象
- application.yml：用于配置数据库、mybatis、redis等信息的配置文件，放在resources文件夹下。也可以用.properties格式的配置文件。两者都存在时，优先使用.properties。

entity

设计一个简单的实体类即可，如果需要用redis进行缓存，那么需要实现序列化接口。

```
public class User implements Serializable { //对象进行redis缓存时需要实现序列化接口
    private int ID;
    private String password;
    private String name;
    private int degree;

    public void setID(int ID){
        this.ID = ID;
    }
    ...
}
```

controller

在类前加上`@RestController`注解用于标明这个可以处理控制HTTP请求。在方法前加上`@GetMapping("/xx")`注解，会在用户访问localhost:8080/xx时调用这个方法处理。controller主要调用Service层来处理。在定义其他层的成员时，需要加@Autowired注解。

```
@RestController
public class UserController {

    @Autowired
    private UserServiceImpl userService;

    @GetMapping("/hello")
    public String hello(){
        return "hello track";
    }

    @GetMapping("/findAll")
    public List<User> findAll(){
        return userService.findAll();
    }

    @GetMapping("/findUser")
```

```
    public String findUser(@RequestParam int ID){
        User user = userService.findUser(ID);
        if(user == null){
            return "can't find";
        }
        else {
            String result = "ID: " + user.getID();
            result += " password: " + user.getPassword();
            result += " name: " + user.getName();
            result += " degree: " + user.getDegree();
            return result;
        }
    }
    ...
}
```

service

Service定义为接口，里面只放方法的定义。

```
public interface UserService {

    public List<User> findAll();

    public User findUser(int ID);

    public int deleteUser(int ID);

    public int updateUser(User user);

}
```

在ServiceImpl中，实现Service定义的方法，类最前面需要加@Service注解。redis缓存是以<key, value>的键值对形式存储的。删除和判断是否存在使用redisTemplate即可，而取出和放入都需要opsForValue。

如果使用 @Autowired 注解完成自动装配，那么RedisTemplate要么不指定泛型，要么泛型为<Stirng,String> 或者<Object,Object>如果你使用其他类型的比如RedisTemplate<String,Object>那么请使用 @Resource 注解

```
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDao userDao;
    @Autowired
    private RedisTemplate redisTemplate;

    public List<User> findAll(){
        return userDao.findAll();
```

```
    }

    public User findUser(int ID){
        String key = Integer.toString(ID);
        boolean hasKey = redisTemplate.hasKey(key);
        ValueOperations<String, User> operations = redisTemplate.opsForValue();
        User user;
        if(hasKey){
            user = operations.get(key);
            System.out.println("从缓存中获取数据：ID = "+ user.getID());
            System.out.println("----------------------------------------");
        }
        else{
            user = userDao.findUser(ID);
            System.out.println("从数据库中获取数据：ID = "+ user.getID());
            operations.set(key,user,5, TimeUnit.HOURS);
            System.out.println("写入缓存");
            System.out.println("----------------------------------------");
        }
        return user;
    }
    ...
}
```

dao

最前面需要加@Mapper注解，可以使用@Select这样的形式将方法和mysql语句进行关联，delete和update的返回值会是int（代表修改的数据个数）。

```
@Repository
@Mapper
public interface UserDao {
    @Select("select * from user")
    List<User> findAll();
    @Select("select * from user where ID = #{ID}")
    User findUser(int ID);
    @Delete("delete from user where ID = #{ID}")
    int deleteUser(int ID);
    @Update("update user set password = #{password}, name =#{name}, degree = #{degree}
where ID = #{ID}")
    int updateUser(User user);
}
```

yml配置

```
spring:
  # 数据库配置
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource #暂不确定要不要
    url: jdbc:mysql://localhost:3306/db?
serverTimezone=UTC&characterEncoding=utf8&useUnicode=true&useSSL=false
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: 1277709017zyh_


  #mybatis的相关配置
  mybatis:
    type-aliases-package: com.example.demo.entity
    #开启驼峰命名
    configuration:
      map-underscore-to-camel-case: true

  # Redis配置
  redis:
    database: 0 #Redis数据库索引
    host: 127.0.0.1
    port: 6379 #Redis连接端口
    password:
    jedis:
      pool:
        max-active: 8 #最大连接数
        max-wait: -1 #最大阻塞等待时间
        max-idle: 8 #最大空闲连接
        min-idle: 0 #最小空闲连接
    timeout: 1200 #连接超时时间
```

统一异常处理

定义一个enum类型，绑定一些状态码和错误类型

```
public enum ExceptionEnum {
    SUCCESS("1000", "success"),
    FAIL("1001", "fail"),
    VALIDATE_ERROR("1002","parameter error"),
    UNKNOWN("1003", "unknown error");

    private String code; //错误码
    private String msg; //错误信息
}
```

自定义异常

```
public class BusinessException extends RuntimeException{
    private ExceptionEnum exceptionEnum;
    private String code;
    private String msg;
}
```

定义处理异常的handler，`@ExceptionHandler`后参数决定遇到哪种异常时调用

```
@ControllerAdvice //全局异常处理
public class GlobalExceptionHandler {
    /* 业务异常 */
    @ExceptionHandler(value = BusinessException.class)
    @ResponseBody
    public ResultResponse exceptionHandler(BusinessException e){
        return ResultResponse.error(e.getExceptionEnum());
    }
    /* 系统异常 */
    @ExceptionHandler(value = Exception.class)
    @ResponseBody
    public ResultResponse exceptionHandler(Exception e){
        return ResultResponse.error(ExceptionEnum.UNKNOWN);
    }
}
```

自定义response用于返回信息

```
public class ResultResponse {
    //状态码，状态信息以及具体描述
    private String code;
    private String msg;
    private Object data;

  public static ResultResponse success(Object data){
    ResultResponse resultResponse = new ResultResponse();
    resultResponse.setCode(ExceptionEnum.SUCCESS.getCode());
```

```
    resultResponse.setMsg(ExceptionEnum.SUCCESS.getMsg());
    resultResponse.setData(data);
    return resultResponse;
  }
}
```

在controller中，可以抛出我们自定义的异常来返回错误信息。

AOP

```
@Aspect
@Component
public class LogAspect {

    @Pointcut("execution (* com.example.demo.controller.*.*(..))")
    //                  返回值类型                          类 方法 参数
    public void cut(){
    }

    @Before("cut()")
    public void beforeRun(){
        Date date = new Date();
        DateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
        String dateStr = sdf.format(date);
        System.out.println("request time:  "+ dateStr);
    }

    @After("cut()")
    public void afterRun(){
        Date date = new Date();
        DateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
        String dateStr = sdf.format(date);
        System.out.println("complete time:  " + dateStr);
    }
}
```

Redis存储session

定义一个filter控制登录（需要在主类加入注解注册filter）

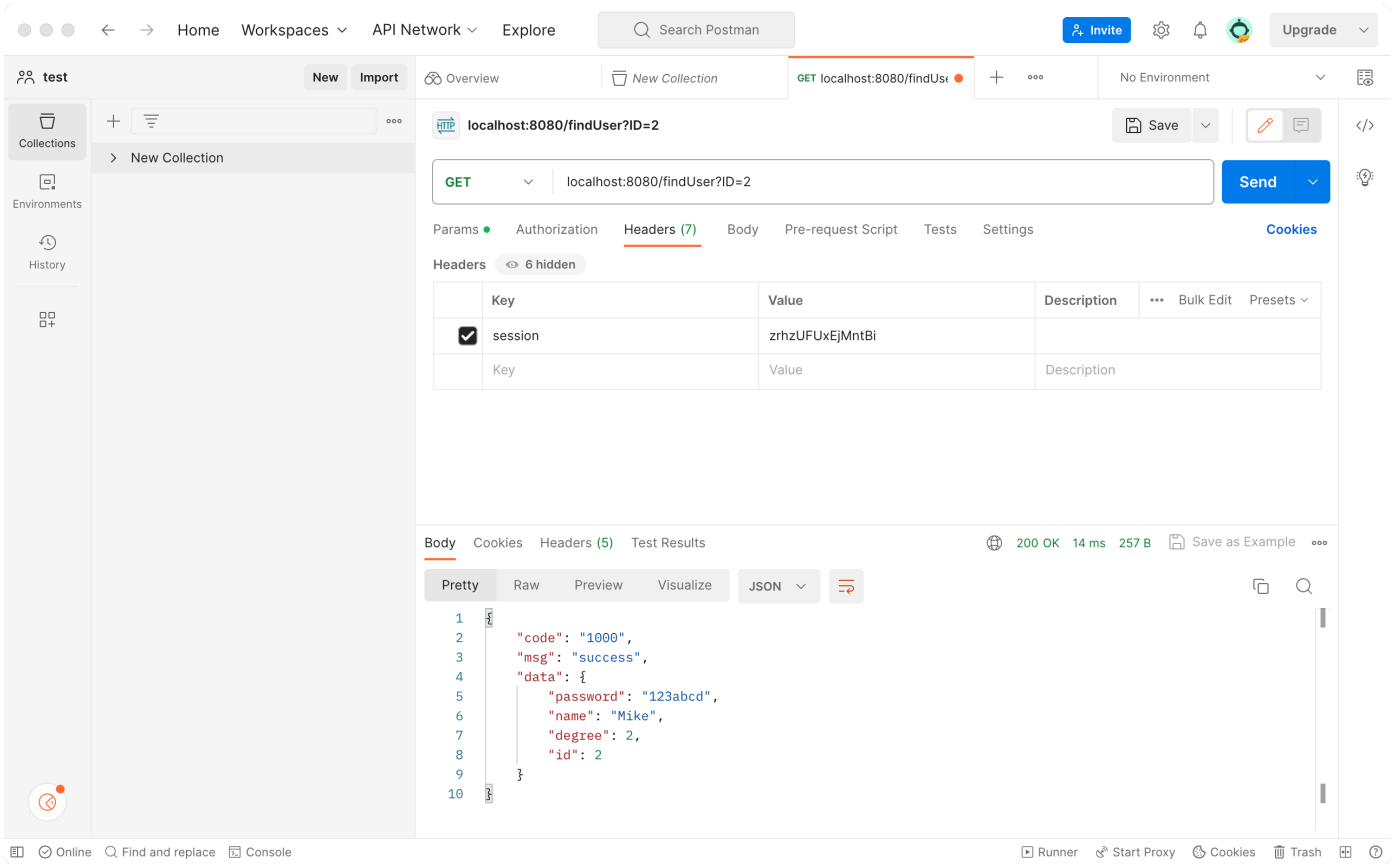
```
 @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest)request;
        String url = httpRequest.getRequestURL().toString();
```

```
        if(url.contains("login") || url.contains("logout")){
            chain.doFilter(request, response);
            System.out.println("登录或登出页面，直接放行");
        }
        else{
            String session = ((HttpServletRequest) request).getHeader("session");
            System.out.println("session is " + session);
            if(userService.getRedisTemplate().hasKey(session)){
                ValueOperations<String, Integer> operations =
userService.getRedisTemplate().opsForValue();
                Integer ID = operations.get(session);
                System.out.println("已登录 ID: " + ID);
                chain.doFilter(request,response);
            }
            else{
                System.out.println("未登录! ");
            }
        }
```

在登录时将session储存

```
    public User login(int ID){
        User user = findUser(ID);
        if(user != null){
            ValueOperations<String, Integer> operations = redisTemplate.opsForValue();
            String session = RandomStringUtils.randomAlphabetic(15);
            System.out.println("设置的session是: " + session);
            operations.set(session, user.getID(),10, TimeUnit.MINUTES);
        }
        return user;
    }
```

postman测试


Home
Workspaces
API Network
Explore
Search Postman
Invite
Upgrade
test
New
Import
Overview
New Collection
GET localhost:8080/findUs
No Environment
Collections
Environments
History
New Collection
localhost:8080/findUser?ID=2
Save
GET
localhost:8080/findUser?ID=2
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Headers
6 hidden
Key
Value
Description
Bulk Edit
Presets
session
zrhzUFUxEjMntBi
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
200 OK
14 ms
257 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
{
"code": "1000",
"msg": "success",
"data": {
"password": "123abcd",
"name": "Mike",
"degree": 2,
"id": 2
}
}
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Trash