苍穹外卖day5

1. redis入门

redis是一个基于内存的key-value结构数据库。

- 基于内存存储，读写性能高
- 适合存储热点数据（热点商品，咨询，新闻）
- 企业应用广泛

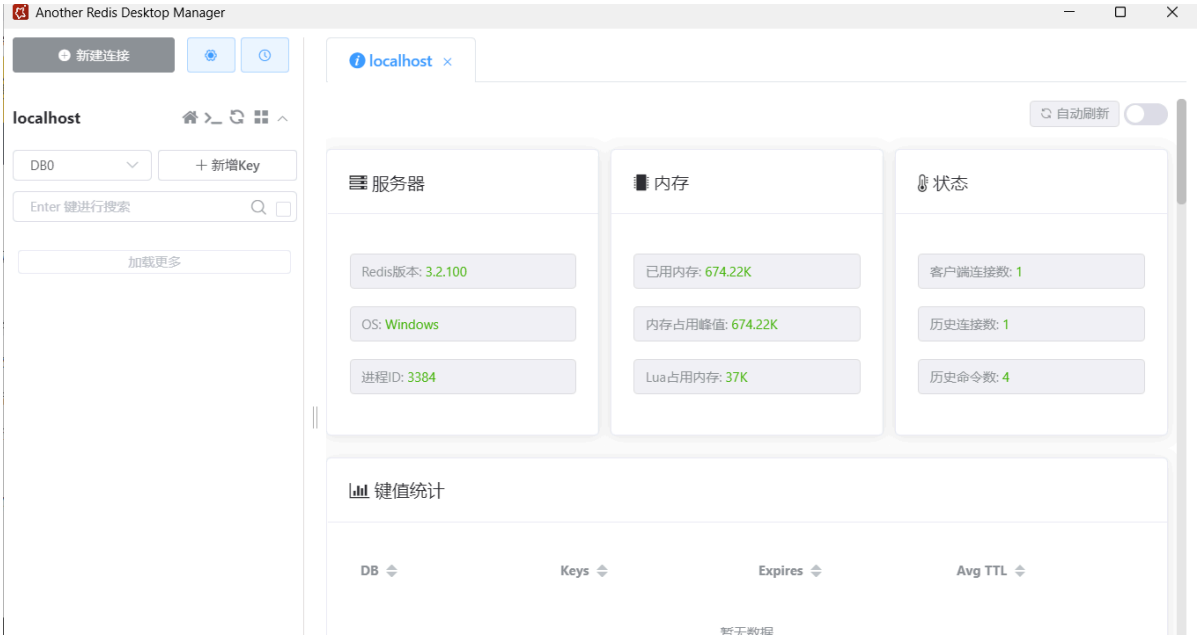
解压`\资料\day05\Redis-x64-3.2.100Redis-x64-3.2.100.zip`

修改配置文件`redis.windows.conf`，把`# requirepass foobared`改成`requirepass 123456`

服务器端启动命令：`redis-server.exe redis.windows.conf`

客户端连接命令`redis-cli.exe -h localhost -p 6379 -a 123456`

使用图形化工具`Another-Redis-Desktop-Manager.1.5.5`管理redis



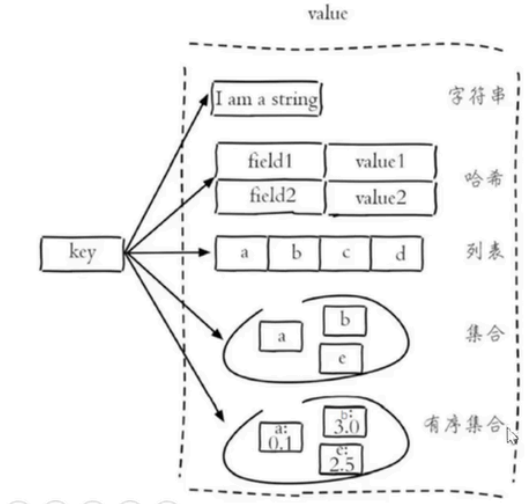
2. redis常见数据类型

Redis存储的是key-value结构的数据，其中key是字符串类型，value有s种常用的数据类型：

- 字符串string
- 哈希hash
- 列表ist
- 集合set

- 有序集合sorted set/zset

各种数据类型的特点



- 字符串(string)：普通字符串，Redis中最简单的数据类型
- 哈希(hash)：也叫散列，类似于Java中的HashMap结构
- 列表(list)：按照插入顺序排序，可以有重复元素，类似于Java中的LinkedList
- 集合(set)：无序集合，没有重复元素，类似于Java中的HashSet
- 有序集合(sorted set / zset)：集合中每个元素关联一个分数（score），根据分数升序排序，没有重复元素

3. redis常用命令

Redis字符串类型常用命令：

```
SET key value 设置指定key的值
GET key 获取指定key的值
SETEX key seconds value 设置指定key的值，并将key的过期时间设为seconds秒
SETNX key value 只有在key不存在时设置key的值
```

哈希操作命令

Redis hash是一个string类型的field和value的映射表，hash特别适合用于存储对象，常用命令：

```
HSET key field value
 将哈希表key中的字段field的值设为value，比如hset 100 name xiaoming
 hset 100 age 20
HGET key field
 获取存储在哈希表中指定字段的值
HDEL key field
 删除存储在哈希表中的指定字段
HKEYS key
 获取哈希表中所有字段
HVALS key
 获取哈希表中所有值
```

Redis列表是简单的字符串列表，按照插入顺序排序，常用命令：

```
LPUSH key value1 [value2]
 将一个或多个值插入到列表头部,比如LPUSH mylist a b c 666 hello
LRANGE key start stop
 获取列表指定范围内的元素,比如 LRANGE mylist  0 -1输出
    hello
    666
    c
    b
    a

RPOP key
 移除并获取列表最后一个元素
LLEN key
 获取列表长度
```

Redis set是string类型的无序集合。集合成员是唯一的，集合中不能出现重复的数据，常用命令：

```
SADD key member1 [member2]
 向集合添加一个或多个成员
SMEMBERS key
 返回集合中的所有成员
SCARD key
 获取集合的成员数
SINTER key1 [key2]
 返回给定所有集合的交集
SUNION key1 [key2]
 返回所有给定集合的并集
SREM key member1 [member2]
 删除集合中一个或多个成员
```

Redis有序集合是string类型元素的集合，且不允许有重复成员。每个元素都会关联一个double类型的分数。常用命令：

```
ZADD key score1 member1 [score2 member2]
 向有序集合添加一个或多个成员
ZRANGE key start stop [WITHSCORES]
 通过索引区间返回有序集合中指定区间内的成员
ZINCRBY key increment member
 有序集合中对指定成员的分数加上增量increment
ZREM key member [member...]
 移除有序集合中的一个或多个成员
```

```
> zadd zset1 10.0 a 10.5 b
2
> zadd zset1 10.2 c
1
> zrange zset1 0 -1
a
c
b
> zrange zset1 0 -1 withscores
a
10
c
10.199999999999999
b
10.5
```

Redis的通用命令是不分数据类型的，都可以使用的命令：

```
KEYS pattern
 查找所有符合给定模式(pattern)的key
EXISTS key
 检查给定key是否存在
TYPE key
 返回key所储存的值的类型
DEL key
 该命令用于在key存在时删除key
```

4. java中操作redis

Redis的常见客户端：

- Jedis
- Lettuce
- Spring Data Redis

下面以Spring Data Redis为重点。

操作步骤：

1. 导入Spring Data Redis的maven坐标
2. 配置Redis数据源
3. 编写配置类，创建RedisTemplate对象
4. 通过RedisTemplate对象操作Redis

sky-server/pom.xml已导入

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
```

application.yml

```
spring:
  profiles:
    active: dev #指定当前激活的配置文件为dev环境
  main:
    allow-circular-references: true #允许循环依赖（特殊场景使用）
  datasource:
    druid: #使用Druid连接池，通过占位符${}动态注入数据库参数
      driver-class-name: ${sky.datasource.driver-class-name}
      url:
jdbc:mysql://${sky.datasource.host}:${sky.datasource.port}/${sky.datasource.
database}?
serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf-
8&zeroDateTimeBehavior=CONVERT_TO_NULL&useSSL=false&allowPublicKeyRetrieval=
true
      username: ${sky.datasource.username}
      password: ${sky.datasource.password}
  redis:
    host: ${sky.redis.host}
    port: ${sky.redis.port}
    password: ${sky.redis.password}
    database: ${sky.redis.database} #redis有16个数据库
```

application-dev.yml添加

```
sky:
  redis:
    host: localhost
    port: 6379
    password: 123456
    database: 0 #redis有16个数据库
```

sky-server/src/main/java/com/sky/config/RedisConfiguration.java

```java
package com.sky.config;

import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
@Slf4j
public class RedisConfiguration {

    @Bean //将redisTemplate()方法返回的对象注册为Spring管理的Bean，使其他组件能通过@Autowired注入
    public RedisTemplate redisTemplate(RedisConnectionFactory redisConnectionFactory) {
        //redis连接工厂对象,RedisConnectionFactory表示依赖Spring容器中已存在的连接工厂实例，由容器自动注入

        log.info("开始创建redis模板对象...");
        RedisTemplate redisTemplate = new RedisTemplate();//定义如何构建RedisTemplate对象，并显式关联连接工厂和序列化器
        //设置redis连接工厂对象
        redisTemplate.setConnectionFactory(redisConnectionFactory);//建立模板与工厂的关联
        //设置redis key的序列化器
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        //解决Redis的键值存储特性：Redis本身只接受二进制数据
        //序列化器负责：
        //将Java对象转换为字节数组（序列化）
        //将字节数组还原为Java对象（反序列化）
        return redisTemplate;
    }
}
```

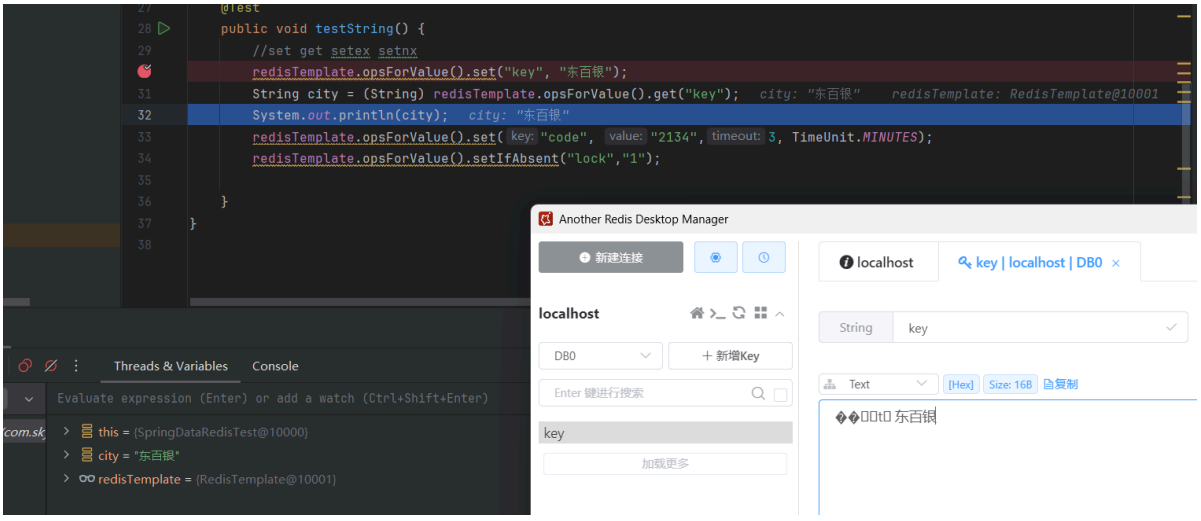
sky-server/src/test/java/com/sky/test/SpringDataRedisTest.java

```java
package com.sky.test;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.redis.connection.DataType;
import org.springframework.data.redis.core.*;

import java.util.List;
import java.util.Set;
import java.util.concurrent.TimeUnit;
```

```java

@SpringBootTest //Spring Boot提供的集成测试注解，用于启动完整的Spring应用上下文（包括所有自动配置的Bean）
public class SpringDataRedisTest {

    @Autowired
    private RedisTemplate redisTemplate;

    @Test //标记方法为测试用例，测试框架会自动执行这些方法
    public void testRedisTemplate() {
        System.out.println(redisTemplate);
        ValueOperations valueOperations = redisTemplate.opsForValue();
        HashOperations hashOperations = redisTemplate.opsForHash();
        ListOperations listOperations = redisTemplate.opsForList();
        SetOperations setOperations = redisTemplate.opsForSet();
        ZSetOperations zSetOperations = redisTemplate.opsForZSet();

    }

    /**
     * 操作字符串类型的数据
     */
    @Test
    public void testString() {
        //set get setex setnx
        redisTemplate.opsForValue().set("key", "东百银");
        String city = (String) redisTemplate.opsForValue().get("key");
        System.out.println(city);
        redisTemplate.opsForValue().set("code", "2134", 3,
TimeUnit.MINUTES);
        redisTemplate.opsForValue().setIfAbsent("lock", "1");

    }

    /**
     * 操作哈希类型的数据
     */
    @Test
    public void testHash() {
        //hset hget hdel hkeys hvals
        HashOperations hashOperations = redisTemplate.opsForHash();
        hashOperations.put("100", "name", "tom");//key,filed,value
        hashOperations.put("100", "age", 18);

        String name = (String) hashOperations.get("100", "name");
        System.out.println(name);

        Set keys = hashOperations.keys("100");
        System.out.println(keys);

        List values = hashOperations.values("100");
        System.out.println(values);

        hashOperations.delete("100", "age");

    }
```

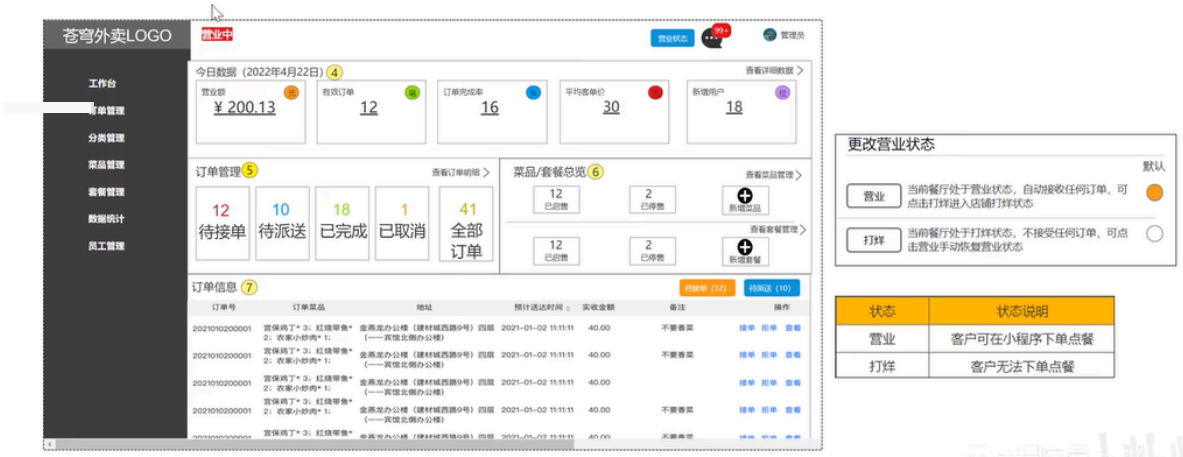
```
66
67      /**
68       * 操作列表类型的数据
69       */
70      @Test
71      public void testList() {
72          //Lpush Lrange rpop LLen
73          ListOperations listOperations = redisTemplate.opsForList();
74          listOperations.leftPushAll("mylist", "a", "b", "c");
75          listOperations.leftPush("mylist", "d");
76
77          List mylist = listOperations.range("mylist", 0, -1);
78          System.out.println(mylist);
79
80          listOperations.rightPop("mylist");
81          Long size = listOperations.size("mylist");
82          System.out.println(size);
83
84      }
85
86      /**
87       * 操作集合类型的数据
88       */
89      @Test
90      public void testset() {
91          //sadd smembers scard sinter sunion srem
92          SetOperations setOperations = redisTemplate.opsForSet();
93          setOperations.add("set1", "a", "b", "c", "d");
94          setOperations.add("set2", "a", "b", "x", "y");
95
96          Set members = setOperations.members("set1");
97          System.out.println(members);
98
99          Long size = setOperations.size("set1");
100         System.out.println(size);
101
102         Set intersect = setOperations.intersect("set1", "set2");
103         System.out.println(intersect);
104
105         Set union = setOperations.union("set1", "set2");
106         System.out.println(union);
107
108         setOperations.remove("set1", "a", "b");
109
110     }
111
112     /**
113      *  操作有序集合类型的数据
114      */
115     @Test
116     public void testzset() {
117         //zadd zrange zincrby zrem
118         ZSetOperations zSetOperations=redisTemplate.opsForZSet();
119
120         zSetOperations.add("zset1", "a", 10);
121         zSetOperations.add("zset1", "b", 12);
```

```
122         zSetOperations.add("zset1", "c", 9);
123
124         Set zset1= zSetOperations.range("zset1", 0, -1);
125         System.out.println(zset1);
126
127         zSetOperations.incrementScore("zset1", "c", 10);
128         zSetOperations.remove("zset1","a", "b");
129     }
130
131
132     /**
133      * 通用命令操作
134      */
135     @Test
136     public void testCommon() {
137         //keys exists type del
138         Set keys= redisTemplate.keys("*");
139         System.out.println(keys);
140
141         Boolean name = redisTemplate.hasKey("name");
142         Boolean set1 = redisTemplate.hasKey("set1");
143
144         for (Object key : keys) {
145             DataType type = redisTemplate.type(key);
146             System.out.println(type.name());
147         }
148         redisTemplate.delete("mylist");
149     }
150 }
```



5. 店铺营业状态设置

产品原型：



状态	状态说明
营业	客户可在小程序下单点餐
打烊	客户无法下单点餐

接口设计：

- 设置营业状态
- 管理端查询营业状态
- 用户端查询营业状态

本项目约定：

- 管理端 发出的请求，统一使用 /admin 作为前缀
- 用户端 发出的请求，统一使用 /user 作为前缀

需求分析和设计

接口设计：

- 设置营业状态
- 管理端查询营业状态
- 用户端查询营业状态

基本信息

Path: /admin/shop/{status}

Method: PUT

接口描述:

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

路径参数

参数名称	示例	备注
status	1	店铺营业状态：1为营业，0为打烊

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	string	非必须			
msg	string	非必须			

基本信息

Path: /admin/shop/status

Method: GET

接口描述:

请求参数

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	integer	必须		店铺营业状态：1为营业，0为打烊
msg	string	非必须		

为了一个`status`营业状态使用redis，挺幽默的。

`SpringDataRedisTest.java`注释掉所有代码，防止每次启动都做一遍测试。

sky-server/src/main/java/com/sky/controller/admin/ShopController.java

```java
package com.sky.controller.admin;

import com.sky.result.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.web.bind.annotation.*;

@RestController("adminShopController")
@RequestMapping("/admin/shop")
@Api(tags = "店铺相关接口")
@Slf4j
public class ShopController {

    public static final String KEY = "SHOP_STATUS";

    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 设置店铺的营业状态
     * @param status
     * @return
     */
    @PutMapping("/{status}")
    @ApiOperation("设置店铺营业状态")
    public Result setStatus(@PathVariable Integer status){
        log.info("设置店铺的营业状态为：{}",status == 1 ? "营业中":"打烊中" );
        redisTemplate.opsForValue().set(KEY,status);
        return Result.success();
```

```
    }

    /**
     * 获取店铺的营业状态
     * @return
     */
    @GetMapping("/status")
    @ApiOperation("获取店铺的营业状态")
    public Result<Integer> getStatus(){
        Integer status=(Integer) redisTemplate.opsForValue().get(KEY);
        log.info("获取店铺的营业状态：{}",status==1?"营业中":"打烊中");
        return Result.success(status);
    }

}
```

sky-server/src/main/java/com/sky/controller/user/ShopController.java

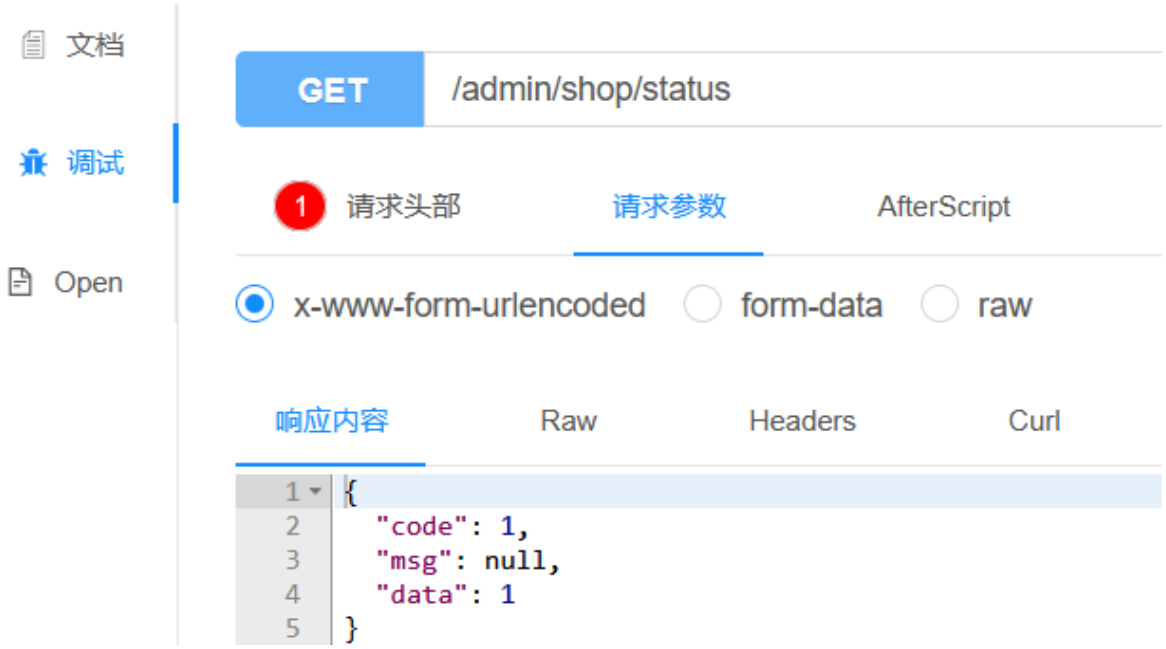
```
package com.sky.controller.user;

import com.sky.result.Result;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.web.bind.annotation.*;

@RestController("userShopController") //防止控制类名冲突
@RequestMapping("/user/shop")
@Api(tags = "店铺相关接口")
@Slf4j
public class ShopController {

    public static final String KEY = "SHOP_STATUS";

    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 获取店铺的营业状态
     * @return
     */
    @GetMapping("/status")
    @ApiOperation("获取店铺的营业状态")
    public Result<Integer> getStatus(){
        Integer status=(Integer) redisTemplate.opsForValue().get(KEY);
        log.info("获取店铺的营业状态：{}",status==1?"营业中":"打烊中");
        return Result.success(status);
    }

}
```

mvn clean

mvn compile

启动后端

http://localhost:8080/doc.html



WebMvcConfiguration.java

```
    /**
     * 通过knife4j生成接口文档
     * @return
     */
    @Bean
    public Docket docket() {
        ApiInfo apiInfo = new ApiInfoBuilder()
                .title("苍穹外卖项目接口文档")
                .version("2.0")
                .description("苍穹外卖项目接口文档")
                .build();
        Docket docket = new Docket(DocumentationType.SWAGGER_2)
                .groupName("管理端接口")
                .apiInfo(apiInfo)
                .select()
                //指定生成接口需要扫描的包

.apis(RequestHandlerSelectors.basePackage("com.sky.controller.admin"))
                .paths(PathSelectors.any())
                .build();
        return docket;
    }

    /**
     * 通过knife4j生成接口文档
```

```
     * @return
     */
    @Bean
    public Docket docket2() {
        ApiInfo apiInfo = new ApiInfoBuilder()
                .title("苍穹外卖项目接口文档")
                .version("2.0")
                .description("苍穹外卖项目接口文档")
                .build();
        Docket docket = new Docket(DocumentationType.SWAGGER_2)
                .groupName("用户端接口")
                .apiInfo(apiInfo)
                .select()
                //指定生成接口需要扫描的包
.apis(RequestHandlerSelectors.basePackage("com.sky.controller.user"))
                .paths(PathSelectors.any())
                .build();
        return docket;
    }
```

redis补充

击穿，穿透，雪崩等问题

redis持久化机制和分布式锁