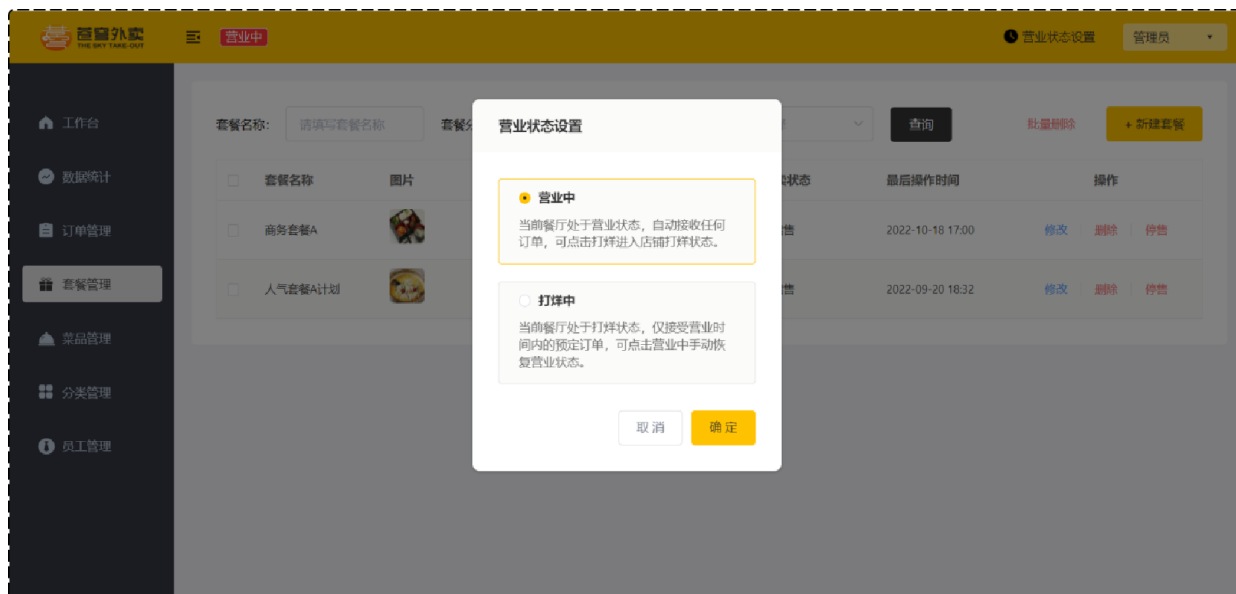


今日内容

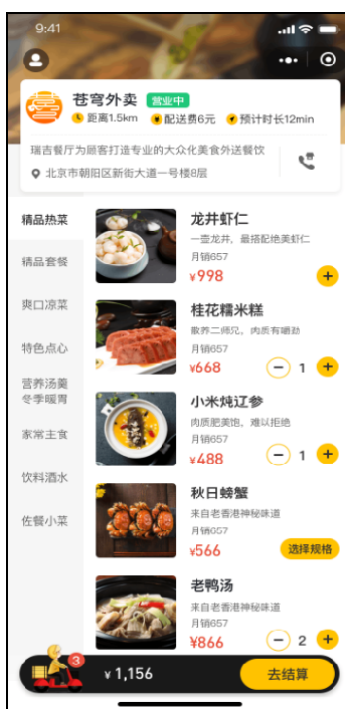
- Redis 入门
- Redis 数据类型
- Redis 常用命令
- 在 Java 中操作 Redis
- 店铺营业状态设置

功能实现：**营业状态设置**

效果图：



选择**营业中**，客户可在小程序端下单：



选择**打烊中**，客户无法在小程序端下单：



一、Redis入门

1、Redis简介

Redis 是一个基于**内存**的 key-value 结构数据库。Redis 是互联网技术领域使用最为广泛的**存储中间件**。

官网：<https://redis.io>

中文网：<https://www.redis.net.cn/>

key-value 结构存储：

key	value
id	101
name	小智
city	北京

主要特点：

- 基于内存存储，读写性能高

- 适合存储热点数据（热点商品、资讯、新闻）
- 企业应用广泛

Redis 是用 C 语言开发的一个开源的高性能键值对（key-value）数据库，官方提供的数据是可以达到 100000+ 的QPS（每秒内查询次数）。它存储的 value 类型比较丰富，也被称为结构化的 NoSql 数据库。

NoSql（Not Only SQL），不仅仅是 SQL，泛指**非关系型数据库**。NoSql 数据库并不是要取代关系型数据库，而是关系型数据库的补充。

关系型数据库（RDBMS）：

- Mysql
- Oracle
- DB2
- SQLServer

非关系型数据库（NoSql）：

- Redis
- Mongo db
- MemCached

2、Redis下载与安装

2.1、Redis下载















Redis 安装包分为 windows 版和 Linux 版：

- Windows 版下载地址：<https://github.com/microsoftarchive/redis/releases>
- Linux 版下载地址：<https://download.redis.io/releases/>

2.2、Redis安装

1) 在 Windows 中安装 Redis（项目中使用）

Redis 的 Windows 版属于绿色软件，直接解压即可使用，解压后目录结构如下：

-  EventLog.dll
-  Redis on Windows Release Notes.docx
-  Redis on Windows.docx
-  redis.windows.conf Redis配置文件
-  redis.windows-service.conf
-  redis-benchmark.exe
-  redis-benchmark.pdb
-  redis-check-aof.exe
-  redis-check-aof.pdb
-  redis-cli.exe Redis客户端
-  redis-cli.pdb
-  redis-server.exe Redis服务端
-  redis-server.pdb
-  Windows Service Documentation.docx

2) 在 Linux 中安装 Redis (简单了解)

在 Linux 系统安装 Redis 步骤:

1. 将 Redis 安装包上传到 Linux
2. 解压安装包, 命令: `tar -zxvf redis-4.0.0.tar.gz -C /usr/local`
3. 安装 Redis 的依赖环境 gcc, 命令: `yum install gcc-c++`
4. 进入 /usr/local/redis-4.0.0, 进行编译, 命令: `make`
5. 进入 redis 的 src 目录进行安装, 命令: `make install`

安装后重点文件说明:

- /usr/local/redis-4.0.0/src/redis-server: Redis 服务启动脚本
- /usr/local/redis-4.0.0/src/redis-cli: Redis 客户端脚本
- /usr/local/redis-4.0.0/redis.conf: Redis 配置文件

3、Redis 服务启动与停止

以 window 版 Redis 进行演示:

3.1、服务启动命令

```
redis-server.exe redis.windows.conf
```

```
C:\Windows\System32\cmd.exe - redis-server.exe redis.windows.conf
Microsoft Windows [版本 10.0.19043.2130]
(c) Microsoft Corporation。保留所有权利。

D:\software\Redis-x64-3.2.100>redis-server.exe redis.windows.conf

Redis 3.2.100 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 21980

http://redis.io

[21980] 19 Oct 10:22:05.904 # Server started, Redis version 3.2.100
[21980] 19 Oct 10:22:05.912 * DB loaded from disk: 0.001 seconds
[21980] 19 Oct 10:22:05.912 * The server is now ready to accept connections on port 6379
```

Redis 服务默认端口号为 6379，通过快捷键 **Ctrl + C** 即可停止 Redis 服务

当 Redis 服务启动成功后，可通过客户端进行连接。

3.2、客户端连接命令

`redis-cli.exe`

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19043.2130]
(c) Microsoft Corporation。保留所有权利。

D:\software\Redis-x64-3.2.100>redis-cli.exe
127.0.0.1:6379> keys *
1) "city"
2) "zset1"
3) "list1"
127.0.0.1:6379> exit

D:\software\Redis-x64-3.2.100>
```

通过 `redis-cli.exe` 命令默认连接的是本地的 redis 服务，并且使用默认 6379 端口。也可以通过指定如下参数连接：

- `-h` ip地址
- `-p` 端口号
- `-a` 密码（如果需要）

3.3、修改 Redis 配置文件

设置 Redis 服务密码，修改 `redis.windows.conf`

```
1 requirepass 123456
```

注意：

- 修改密码后需要重启 Redis 服务才能生效
- Redis 配置文件中 `#` 表示注释

重启 Redis 后，再次连接 Redis 时，需加上密码，否则连接失败。

```
1 redis-cli.exe -h localhost -p 6379 -a 123456
```

```
D:\software\Redis-x64-3.2.100>redis-cli.exe -h localhost -p 6379
localhost:6379> keys *
(error) NOAUTH Authentication required.
localhost:6379> exit

D:\software\Redis-x64-3.2.100>redis-cli.exe -h localhost -p 6379 -a 123456
localhost:6379> keys *
(empty list or set)
localhost:6379> _
```

此时，-h 和 -p 参数可省略不写。

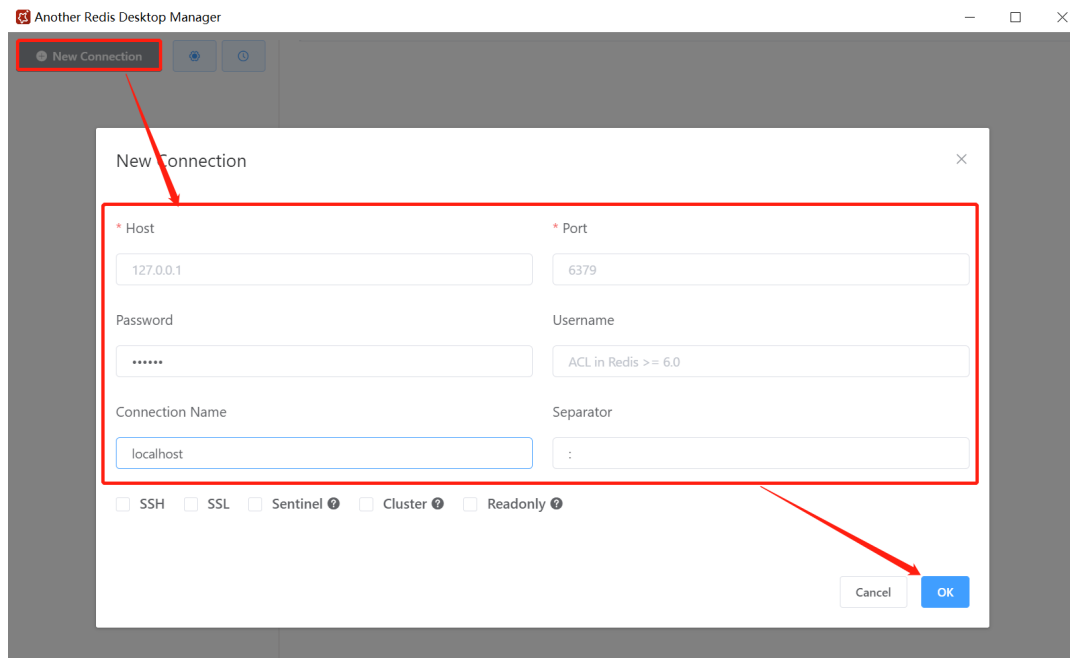
3.4、Redis 客户端图形工具

默认提供的客户端连接工具界面不太友好，同时操作也较为麻烦，接下来，引入一个 Redis 客户端图形工具。

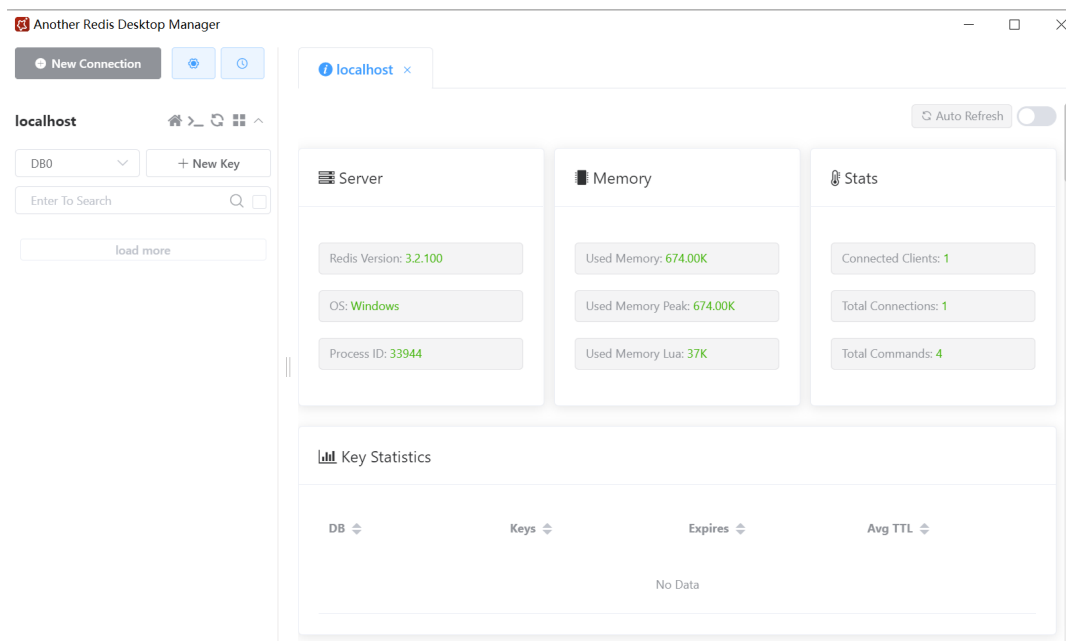
推荐使用 Another-Redis-Desktop-Manager，下载地址：<https://github.com/qishibo/AnotherRedisDesktopManager/releases>

安装完毕后，直接双击启动

新建连接



连接成功



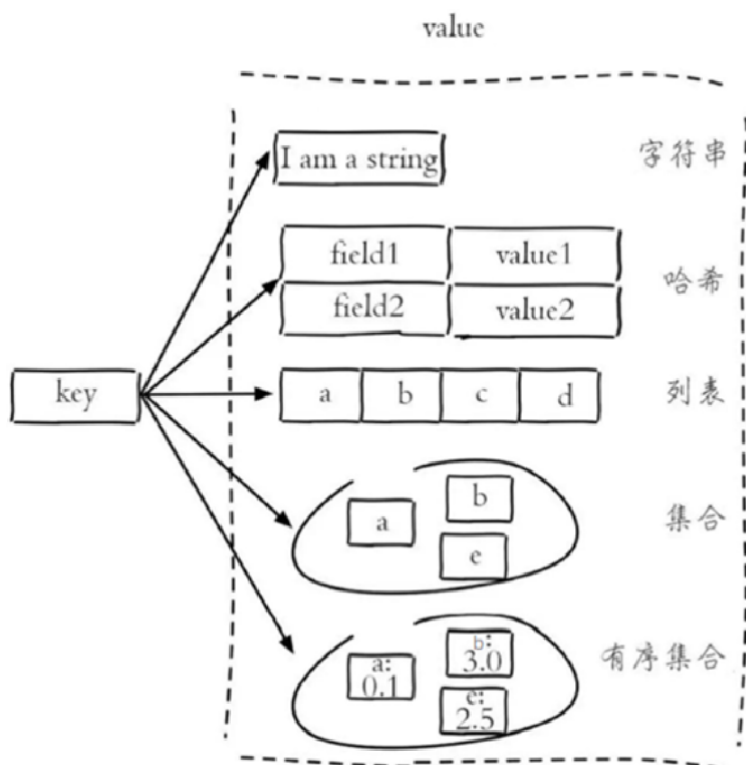
二、Redis 数据类型

1、五种常用数据类型介绍

Redis 存储的是 key-value 结构的数据，其中 key 是字符串类型，value 有 5 种常用的数据类型：

- 字符串 string
- 哈希 hash
- 列表 list
- 集合 set
- 有序集合 sorted set / zset

2、各种数据类型特点



解释说明：

- 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

三、Redis常用命令

1、字符串操作命令

Redis 中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

更多命令可以参考 Redis 中文网：<https://www.redis.net.cn>

```
1 > set name jack
2 OK
3 > get name
```

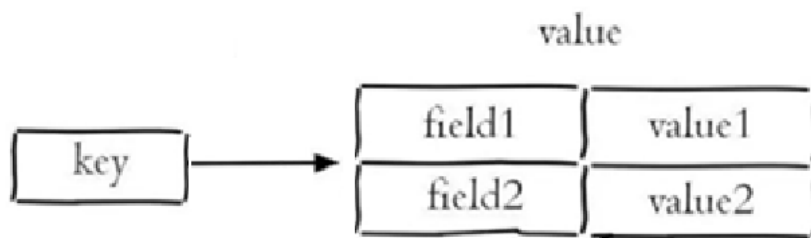


```
4 jack
5 > get abc
6 null
7 > setex code 60 1234
8 OK
9 > setnx key1 itcast
10 1
11 > setnx key1 itheima
12 0
13 > get key1
14 itcast
```

2、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
- **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

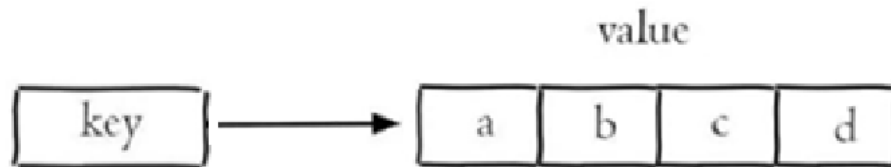


```
1 > hset 100 name xiaoming
2 1
3 > hset 100 age 22
4 1
5 > hget 100 name
6 xiaoming
7 > hget 100 age
8 22
9 > hdel 100 name
10 1
11 > hset 100 name itcast
12 1
13 > hkeys 100
14 age
15 name
16 > hvals 100
17 22
18 itcast
```

3、列表操作命令

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
- `LRANGE key start stop`：获取列表指定范围内的元素
- `RPOP key`：移除并获取列表最后一个元素
- `LLEN key`：获取列表长度
- `BRPOP key1 [key2] timeout`：移出并获取列表的最后一个元素，如果列表没有元素会阻塞列表直到等待超时或发现可弹出元素为止

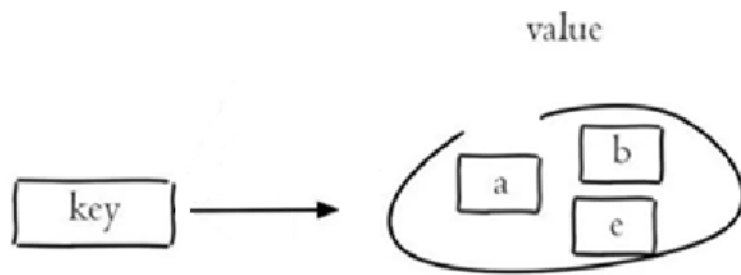


```
1 > lpush mylist a b c
2 3
3 > lpush mylist d
4 4
5 > lrange mylist 0 -1
6 d
7 c
8 b
9 a
10 > rpop mylist
11 a
12 > rpop mylist
13 b
14 > llen mylist
15 2
```

4、集合操作命令

Redis set 是 string 类型的无序集合。集合成员是唯一的，这意味着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]`：移除集合中一个或多个成员



```
1 > sadd set1 a b c d
2 4
3 > sadd set1 a
4 0
5 > smembers set1
6 b
7 a
8 d
9 c
10 > scard set1
11 4
12 > sadd set2 a b x y
13 4
14 > sinter set1 set2
15 b
16 a
17 > sunion set1 set2
18 x
19 a
20 b
21 d
22 y
23 c
24 > srem set1 a
25 1
26 > smembers set1
27 b
28 d
29 c
```

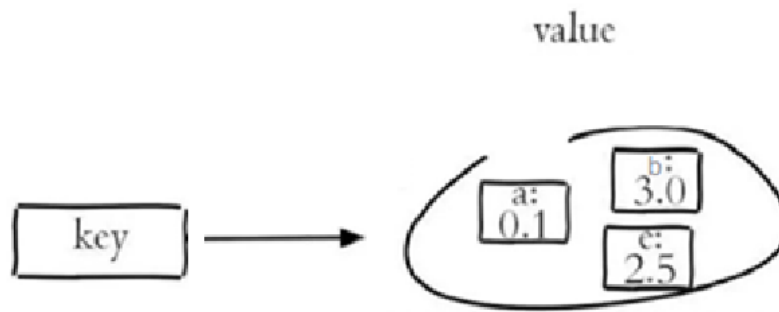
5、有序集合操作命令

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



```

1  > zadd zset1 10.0 a 10.5 b
2  2
3  > zadd zset1 10.2 c
4  1
5  > zrange zset1 0 -1
6  a
7  c
8  b
9  > zrange zset1 0 -1 withscores
10 a
11 10
12 c
13 10.199999999999999
14 b
15 10.5
16 > zincrby zset1 5.0 a
17 15
18 > zrange zset1 0 -1 withscores
19 c
20 10.199999999999999
21 b
22 10.5
23 a
24 15
25 > zrem zset1 b
26 1

```

6、通用命令

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

```

1  > keys *

```

```
2    key1
3    set1
4    zet1
5    100
6    name
7    set2
8    mylist
9    > keys set*
10   set1
11   set2
12   > exists name
13   1
14   > exists abc
15   0
16   > type name
17   string
18   > type set1
19   set
20   > type mylist
21   list
22   > del name
23   1
24   > del 100
25   1
26   > del set1 set2 zset1
27   3
```

四、在 Java 中操作 Redis

1、Redis的Java客户端

前面我们讲解了 Redis 的常用命令，这些命令是我们操作 Redis 的基础，那么我们在 Java 程序中应该如何操作Redis呢？这就需要使用 Redis 的 Java 客户端，就如同我们使用 JDBC 操作 MySQL 数据库一样。

Redis 的 Java 客户端很多，常用的几种：

- Jedis
- Lettuce
- Spring Data Redis

Spring 对 Redis 客户端进行了整合，提供了 Spring Data Redis，在 SpringBoot 项目中还提供了对应的 Starter，即 spring-boot-starter-data-redis。

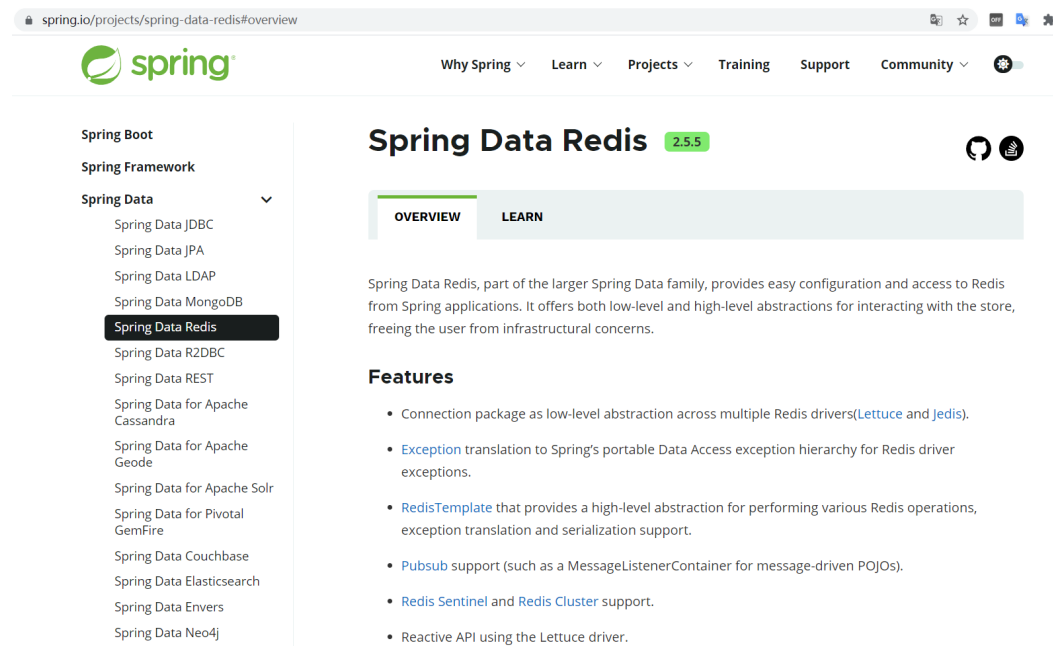
接下来我们重点学习 **Spring Data Redis**。

2、Spring Data Redis 使用方式

2.1、介绍

Spring Data Redis 是 Spring 的一部分，提供了在 Spring 应用中通过简单的配置就可以访问 Redis 服务，对 Redis 底层开发包进行了高度封装。在 Spring 项目中，可以使用 Spring Data Redis 来简化 Redis 操作。

网址：<https://spring.io/projects/spring-data-redis>



Spring Boot 提供了对应的 Starter，maven坐标：

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

Spring Data Redis 中提供了一个高度封装的类：**RedisTemplate**，对相关 api 进行了归类封装，将同一类型操作封装为 operation 接口，具体分类如下：

- ValueOperations：string 数据操作
- SetOperations：set 类型数据操作
- ZSetOperations：zset 类型数据操作
- HashOperations：hash 类型的数据操作
- ListOperations：list 类型的数据操作

2.2、环境搭建

进入到 sky-server 模块

1). 导入 Spring Data Redis 的 maven 坐标(已完成)

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

2). 配置 Redis 数据源

在 application-dev.yml 中添加

```
1  sky:
2    redis:
3      host: localhost
4      port: 6379
5      password: 123456
6      database: 10
```

解释说明：

database：指定使用 Redis 的哪个数据库，Redis 服务启动后默认有 16 个数据库，编号分别是 0 到 15。

可以通过修改 Redis 配置文件来指定数据库的数量。

在 application.yml 中添加读取 application-dev.yml 中的相关 Redis 配置

```
1  spring:
2    profiles:
3      active: dev
4    redis:
5      host: ${sky.redis.host}
6      port: ${sky.redis.port}
7      password: ${sky.redis.password}
8      database: ${sky.redis.database}
```

3). 编写配置类，创建 RedisTemplate 对象

```
1  package com.sky.config;
2
3  import lombok.extern.slf4j.Slf4j;
4  import org.springframework.context.annotation.Bean;
5  import org.springframework.context.annotation.Configuration;
6  import org.springframework.data.redis.connection.RedisConnectionFactory;
7  import org.springframework.data.redis.core.RedisTemplate;
8  import org.springframework.data.redis.serializer.StringRedisSerializer;
9
10 @Configuration
11 @Slf4j
12 public class RedisConfiguration {
13     @Bean
14     public RedisTemplate redisTemplate(RedisConnectionFactory redisConnectionFactory)
15     {
16         log.info("开始创建redis模板对象...");
17         RedisTemplate redisTemplate = new RedisTemplate();
18         //设置redis的连接工厂对象
19         redisTemplate.setConnectionFactory(redisConnectionFactory);
20         //设置redis key的序列化器
21         redisTemplate.setKeySerializer(new StringRedisSerializer());
22         return redisTemplate;
23     }
24 }
```

解释说明：

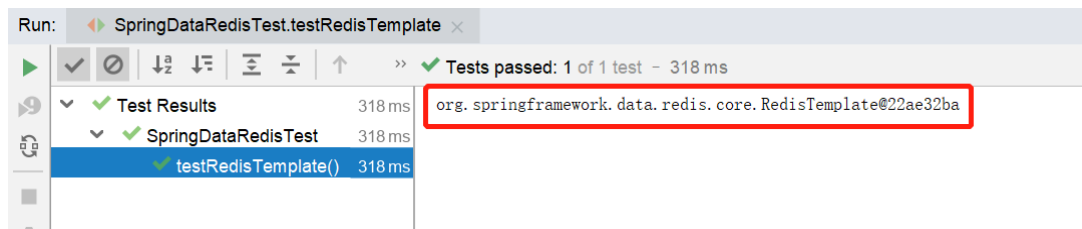
当前配置类不是必须的，因为 SpringBoot 框架会自动装配 RedisTemplate 对象，但是默认的 key 序列化器为 JdkSerializationRedisSerializer，导致我们存到 Redis 中后的数据和原始数据有差别，故设置为 StringRedisSerializer 序列化器。

4). 通过 RedisTemplate 对象操作 Redis

在 test 下新建测试类

```
1 package com.sky.test;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.boot.test.context.SpringBootTest;
6 import org.springframework.data.redis.core.*;
7
8 @SpringBootTest
9 public class SpringDataRedisTest {
10     @Autowired
11     private RedisTemplate redisTemplate;
12
13     @Test
14     public void testRedisTemplate(){
15         System.out.println(redisTemplate);
16         //string数据操作
17         ValueOperations valueOperations = redisTemplate.opsForValue();
18         //hash类型的数据操作
19         HashOperations hashOperations = redisTemplate.opsForHash();
20         //list类型的数据操作
21         ListOperations listOperations = redisTemplate.opsForList();
22         //set类型数据操作
23         SetOperations setOperations = redisTemplate.opsForSet();
24         //zset类型数据操作
25         ZSetOperations zSetOperations = redisTemplate.opsForZSet();
26     }
27 }
```

测试：



说明 RedisTemplate 对象注入成功，并且通过该 RedisTemplate 对象获取操作 5 种数据类型相关对象。

上述环境搭建完毕后，接下来，我们就来具体对常见 5 种数据类型进行操作。

2.3、操作常见类型数据

1). 操作字符串类型数据

```
1  /**
2   * 操作字符串类型的数据
3   */
4  @Test
5  public void testString(){
6      // set get setex setnx
7      ValueOperations valueOperations = redisTemplate.opsForValue();
8      valueOperations.set("city", "北京");
9      String city = (String) valueOperations.get("city");
10     System.out.println(city);
11     valueOperations.set("code", "1234", 3, TimeUnit.MINUTES);
12     valueOperations.setIfAbsent("lock", "1");
13     valueOperations.setIfAbsent("lock", "2");
14 }
```

2). 操作哈希类型数据

```
1  /**
2   * 操作哈希类型的数据
3   */
4  @Test
5  public void testHash(){
6      //hset hget hdel hkeys hvals
7      HashOperations hashOperations = redisTemplate.opsForHash();
8
9      hashOperations.put("100", "name", "tom");
10     hashOperations.put("100", "age", "20");
11
12     String name = (String) hashOperations.get("100", "name");
13     System.out.println(name);
14
15     Set keys = hashOperations.keys("100");
16     System.out.println(keys);
17
18     List values = hashOperations.values("100");
19     System.out.println(values);
20
21     hashOperations.delete("100", "age");
22 }
```

3). 操作列表类型数据

```
1  /**
2   * 操作列表类型的数据
3   */
4  @Test
5  public void testList(){
6      //lpush lrange rpop llen
7      ListOperations listOperations = redisTemplate.opsForList();
```

```

8
9     listOperations.leftPushAll("mylist", "a", "b", "c");
10    listOperations.leftPush("mylist", "d");
11
12    List mylist = listOperations.range("mylist", 0, -1);
13    System.out.println(mylist);
14
15    listOperations.rightPop("mylist");
16
17    Long size = listOperations.size("mylist");
18    System.out.println(size);
19 }

```

4). 操作集合类型数据

```

1    /**
2     * 操作集合类型的数据
3     */
4    @Test
5    public void testSet(){
6        //sadd smembers scard sinter sunion srem
7        SetOperations setOperations = redisTemplate.opsForSet();
8
9        setOperations.add("set1", "a", "b", "c", "d");
10       setOperations.add("set2", "a", "b", "x", "y");
11
12       Set members = setOperations.members("set1");
13       System.out.println(members);
14
15       Long size = setOperations.size("set1");
16       System.out.println(size);
17
18       Set intersect = setOperations.intersect("set1", "set2");
19       System.out.println(intersect);
20
21       Set union = setOperations.union("set1", "set2");
22       System.out.println(union);
23
24       setOperations.remove("set1", "a", "b");
25   }

```

5). 操作有序集合类型数据

```

1    /**
2     * 操作有序集合类型的数据
3     */
4    @Test
5    public void testZset(){
6        //zadd zrange zincrby zrem
7        ZSetOperations zSetOperations = redisTemplate.opsForZSet();
8
9        zSetOperations.add("zset1", "a", 10);
10       zSetOperations.add("zset1", "b", 12);

```

```

11     zSetOperations.add("zset1", "c", 9);
12
13     Set zset1 = zSetOperations.range("zset1", 0, -1);
14     System.out.println(zset1);
15
16     zSetOperations.incrementScore("zset1", "c", 10);
17
18     zSetOperations.remove("zset1", "a", "b");
19 }

```

6). 通用命令操作

```

1  /**
2   * 通用命令操作
3   */
4  @Test
5  public void testCommon(){
6      //keys exists type del
7      Set keys = redisTemplate.keys("*");
8      System.out.println(keys);
9
10     Boolean name = redisTemplate.hasKey("name");
11     System.out.println(name);
12     Boolean set1 = redisTemplate.hasKey("set1");
13     System.out.println(set1);
14
15     for (Object key : keys) {
16         DataType type = redisTemplate.type(key);
17         System.out.println(type.name());
18     }
19
20     redisTemplate.delete("mylist");
21 }

```

五、店铺营业状态设置

1、需求分析和设计

1.1、产品原型

进到苍穹外卖后台，显示餐厅的营业状态，营业状态分为**营业中**和**打烊中**，若当前餐厅处于营业状态，自动接收任何订单，客户可在小程序进行下单操作；若当前餐厅处于打烊状态，不接受任何订单，客户便无法在小程序进行下单操作。



点击**营业状态**按钮时，弹出更改营业状态

更改营业状态

默认

营业

当前餐厅处于营业状态，自动接收任何订单，可点击打烊进入店铺打烊状态

打烊

当前餐厅处于打烊状态，不接受任何订单，可点击营业手动恢复营业状态

选择**营业**，设置餐厅为**营业中**状态

选择**打烊**，设置餐厅为**打烊中**状态

状态说明：

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

1.2、接口设计

根据上述原型图设计接口，共包含 3 个接口。

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

注：从技术层面分析，其实管理端和用户端查询营业状态时，可通过一个接口去实现即可。因为营业状态是一致的。但是，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。

因为访问路径不一致，故分为两个接口实现。

1). 设置营业状态

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			

2). 管理端营业状态

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		

3). 用户端营业状态

基本信息

Path: /user/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		店铺状态: 1为营业, 0为打烊
msg	string	非必须		

1.3、营业状态存储方式

虽然，可以通过一张表来存储营业状态数据，但整个表中只有一个字段，所以意义不大。

营业状态数据存储方式：基于 Redis 的字符串来进行存储

key	value
SHOP_STATUS	1

约定：1 表示营业，0 表示打烊

2、代码开发

2.1、设置营业状态

在 sky-server 模块中，创建 ShopController.java

根据接口定义创建 ShopController 的 setStatus 设置营业状态方法：

```
1 package com.sky.controller.admin;
2
3 import com.sky.result.Result;
4 import io.swagger.annotations.Api;
5 import io.swagger.annotations.ApiOperation;
6 import lombok.extern.slf4j.Slf4j;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.data.redis.core.RedisTemplate;
9 import org.springframework.web.bind.annotation.PathVariable;
10 import org.springframework.web.bind.annotation.PutMapping;
11 import org.springframework.web.bind.annotation.RequestMapping;
```

```

12 import org.springframework.web.bind.annotation.RestController;
13
14 @RestController("adminShopController")
15 @RequestMapping("/admin/shop")
16 @Api(tags = "店铺相关接口")
17 @Slf4j
18 public class ShopController {
19     public static final String KEY = "SHOP_STATUS";
20
21     @Autowired
22     private RedisTemplate redisTemplate;
23
24     /**
25      * 设置店铺的营业状态
26      * @param status
27      * @return
28      */
29     @PutMapping("/{status}")
30     @ApiOperation("设置店铺的营业状态")
31     public Result setStatus(@PathVariable Integer status){
32         log.info("设置店铺的营业状态为: {}", status == 1 ? "营业中" : "打烊中");
33         redisTemplate.opsForValue().set(KEY, status);
34         return Result.success();
35     }
36 }

```

2.2、管理端查询营业状态

根据接口定义创建ShopController的getStatus查询营业状态方法：

```

1  /**
2   * 获取店铺的营业状态
3   * @return
4   */
5  @GetMapping("/status")
6  @ApiOperation("获取店铺的营业状态")
7  public Result<Integer> getStatus(){
8      Integer status = (Integer) redisTemplate.opsForValue().get(KEY);
9      log.info("获取到店铺的营业状态为: {}", status == 1 ? "营业中" : "打烊中");
10     return Result.success(status);
11 }

```

2.3、用户端查询营业状态

创建 com.sky.controller.user 包，在该包下创建 ShopController.java

根据接口定义创建 ShopController 的 getStatus 查询营业状态方法：

```

1 package com.sky.controller.user;
2
3 import com.sky.result.Result;
4 import io.swagger.annotations.Api;

```

```

5     import io.swagger.annotations.ApiOperation;
6     import lombok.extern.slf4j.Slf4j;
7     import org.springframework.beans.factory.annotation.Autowired;
8     import org.springframework.data.redis.core.RedisTemplate;
9     import org.springframework.web.bind.annotation.*;
10
11     @RestController("userShopController")
12     @RequestMapping("/user/shop")
13     @Api(tags = "店铺相关接口")
14     @Slf4j
15     public class ShopController {
16         public static final String KEY = "SHOP_STATUS";
17
18         @Autowired
19         private RedisTemplate redisTemplate;
20
21         /**
22          * 获取店铺的营业状态
23          * @return
24          */
25         @GetMapping("/status")
26         @ApiOperation("获取店铺的营业状态")
27         public Result<Integer> getStatus(){
28             Integer status = (Integer) redisTemplate.opsForValue().get(KEY);
29             log.info("获取到店铺的营业状态为: {}", status == 1 ? "营业中" : "打烊中");
30             return Result.success(status);
31         }
32     }

```

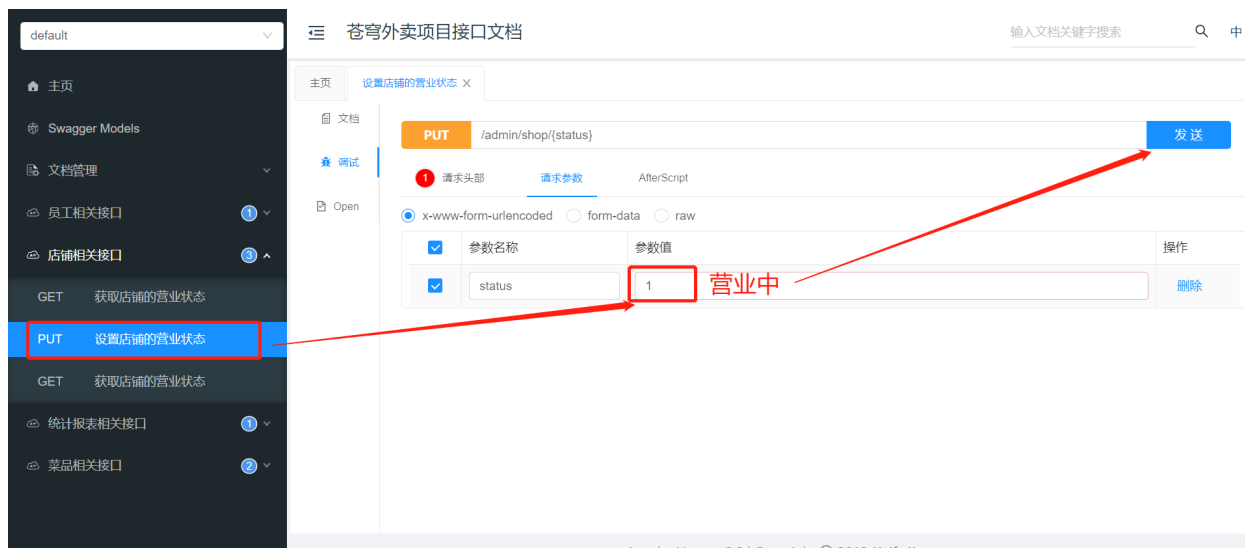
3、功能测试

3.1、接口文档测试

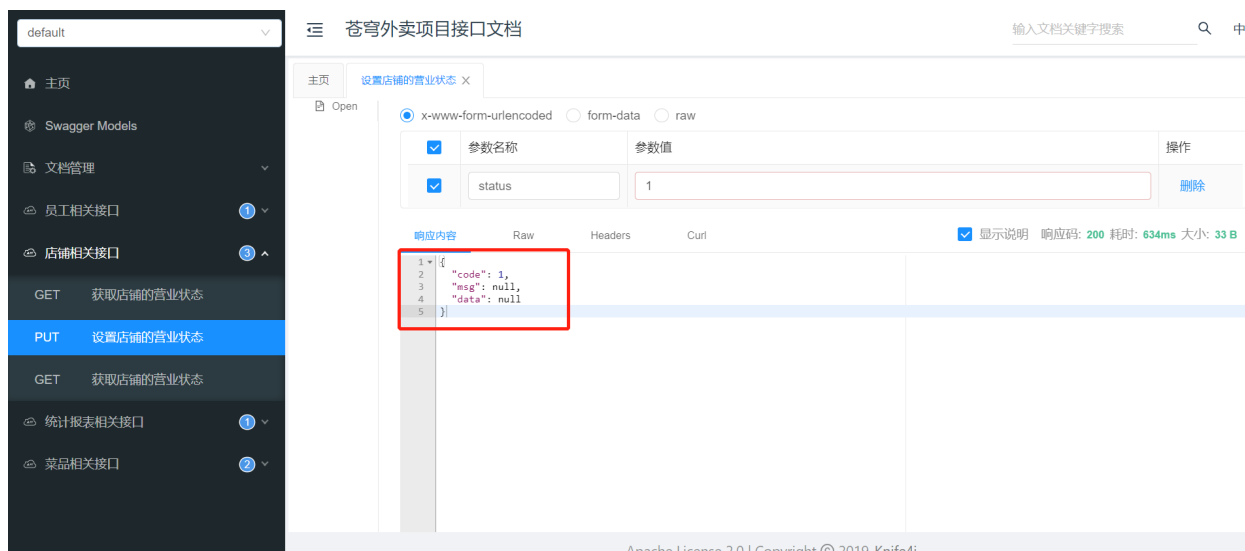
启动服务：访问 <http://localhost:8080/doc.html>，打开店铺相关接口

注意：使用 admin 用户登录重新获取 token，防止 token 失效。

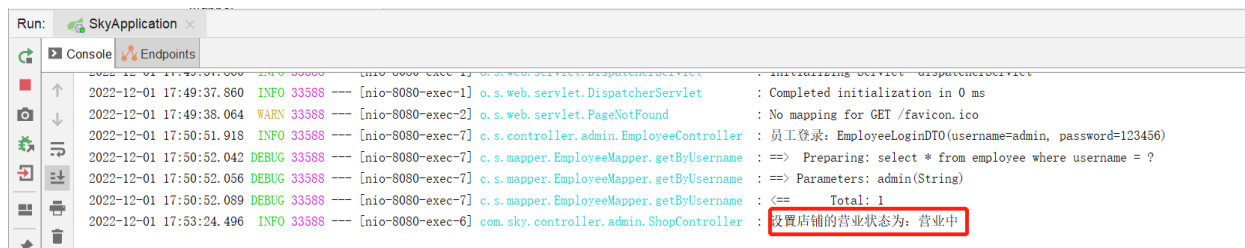
设置营业状态：



点击发送



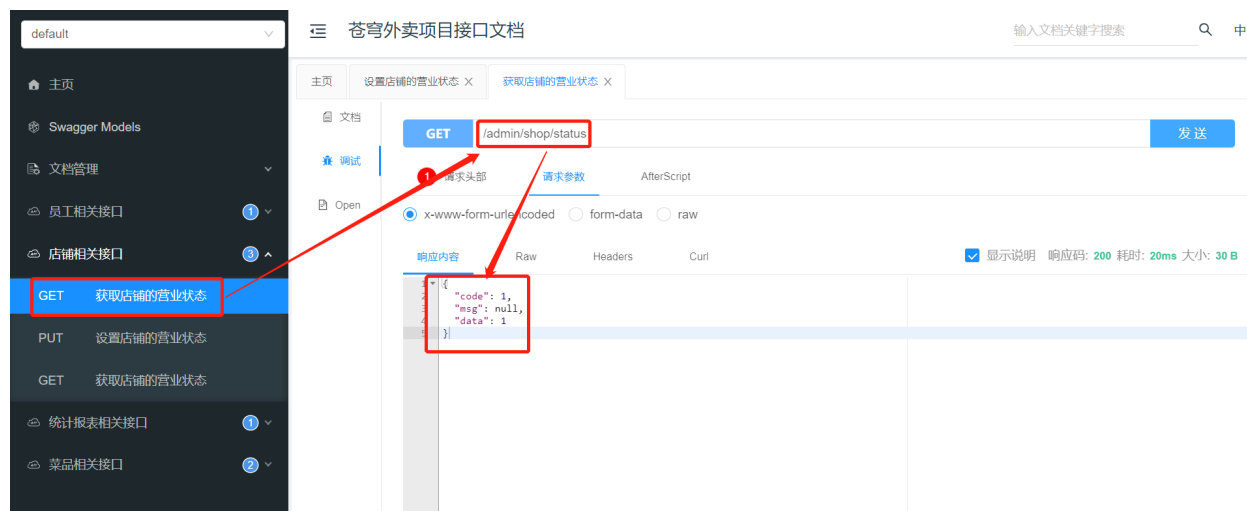
查看 Idea 控制台日志



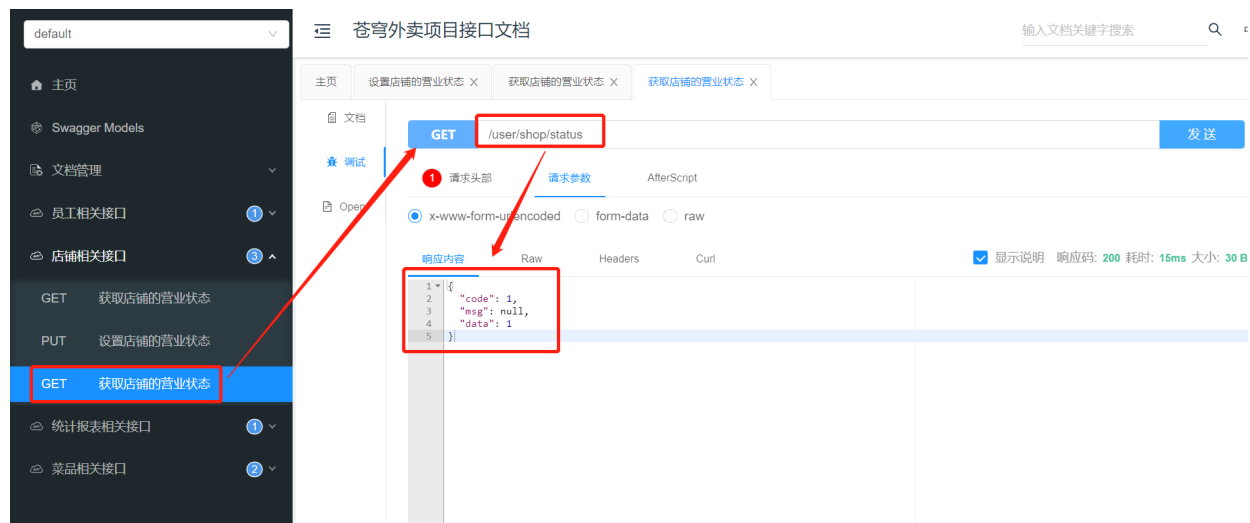
查看 Redis 中数据



管理端查询营业状态：



用户端查询营业状态：



3.2、接口分组展示

在上述接口文档测试中，管理端和用户端的接口放在一起，不方便区分。



接下来，我们要实现管理端和用户端接口进行区分。

在 WebMvcConfiguration.java 中，分别扫描 "com.sky.controller.admin" 和 "com.sky.controller.user" 这两个包。

```
1  @Bean
2  public Docket docket1(){
3      log.info("准备生成接口文档...");
4      ApiInfo apiInfo = new ApiInfoBuilder()
5          .title("苍穹外卖项目接口文档")
6          .version("2.0")
7          .description("苍穹外卖项目接口文档")
8          .build();
9
10     Docket docket = new Docket(DocumentationType.SWAGGER_2)
11         .groupName("管理端接口")
12         .apiInfo(apiInfo)
13         .select()
14         //指定生成接口需要扫描的包
15         .apis(RequestHandlerSelectors.basePackage("com.sky.controller.admin"))
16         .paths(PathSelectors.any())
17         .build();
18
19     return docket;
20 }
21
22 @Bean
23 public Docket docket2(){
24     log.info("准备生成接口文档...");
```

```

25     ApiInfo apiInfo = new ApiInfoBuilder()
26         .title("苍穹外卖项目接口文档")
27         .version("2.0")
28         .description("苍穹外卖项目接口文档")
29         .build();
30
31     Docket docket = new Docket(DocumentationType.SWAGGER_2)
32         .groupName("用户端接口")
33         .apiInfo(apiInfo)
34         .select()
35         //指定生成接口需要扫描的包
36         .apis(RequestHandlerSelectors.basePackage("com.sky.controller.user"))
37         .paths(PathSelectors.any())
38         .build();
39
40     return docket;
41 }

```

重启服务器，再次访问接口文档，可进行选择**用户端接口**或者**管理端接口**



苍穹外卖项目接口文档

输入文档关键字搜索

主页

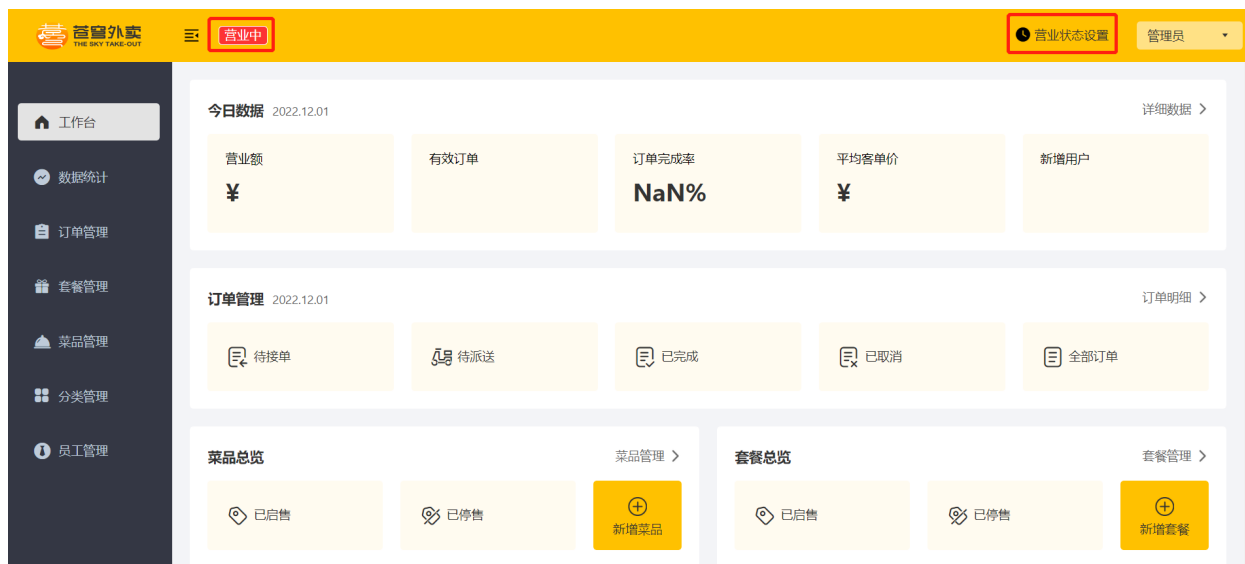
苍穹外卖项目接口文档

简介	苍穹外卖项目接口文档
作者	
版本	2.0
host	localhost:8080
basePath	/
服务Url	
分组名称	用户端接口
分组Url	/v2/api-docs?group=用户端接口
分组location	/v2/api-docs?group=用户端接口
接口统计信息	GET 1

3.3、前后端联调测试

启动 nginx，访问 <http://localhost>

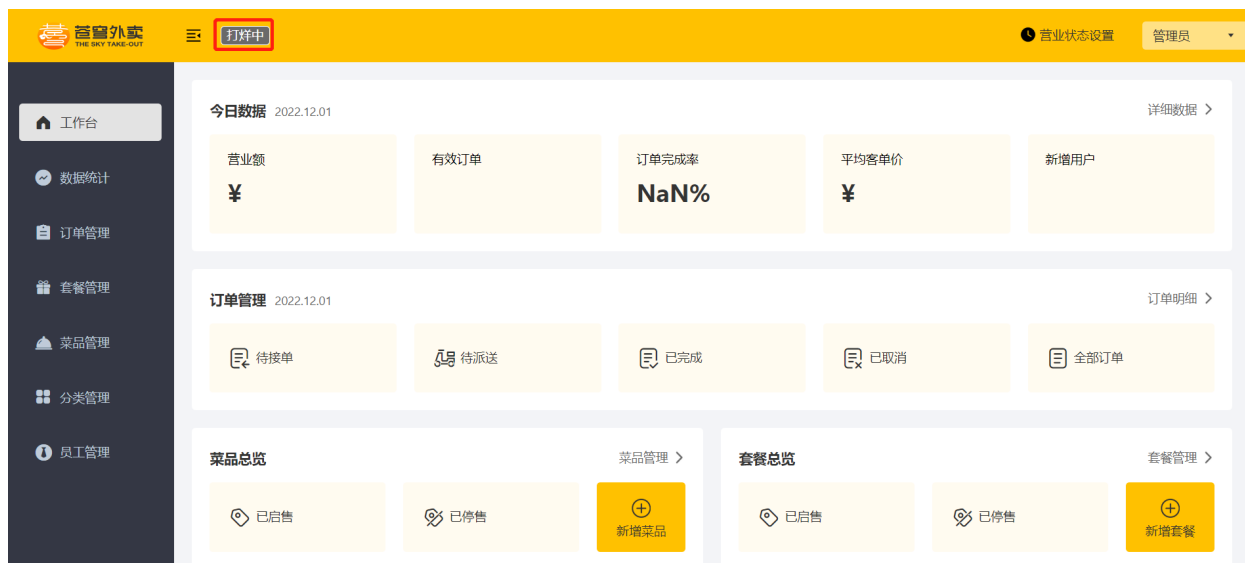
进入后台，状态为**营业中**



点击营业状态设置，修改状态为打烊中

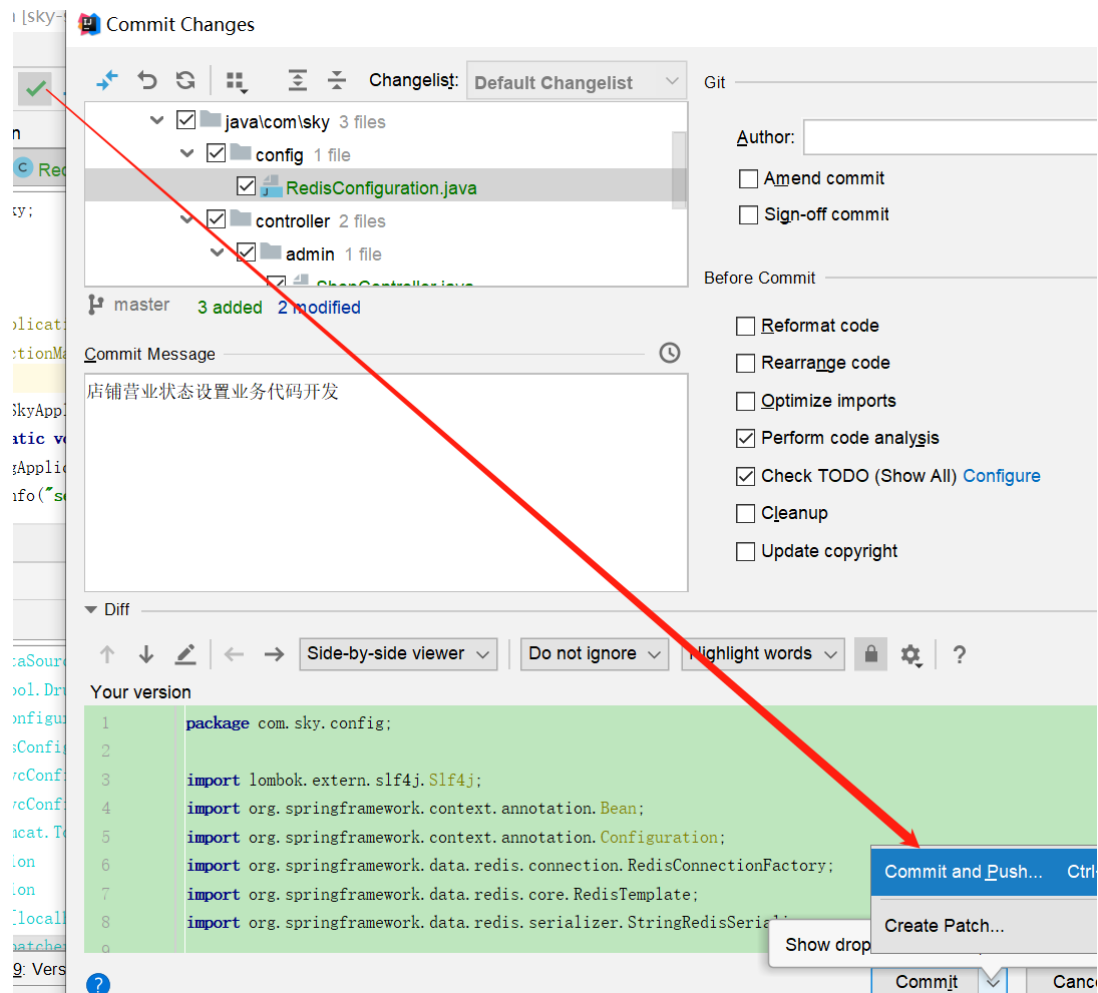


再次查看状态，状态已为打烊中

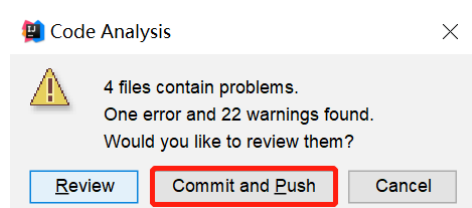


4、代码提交

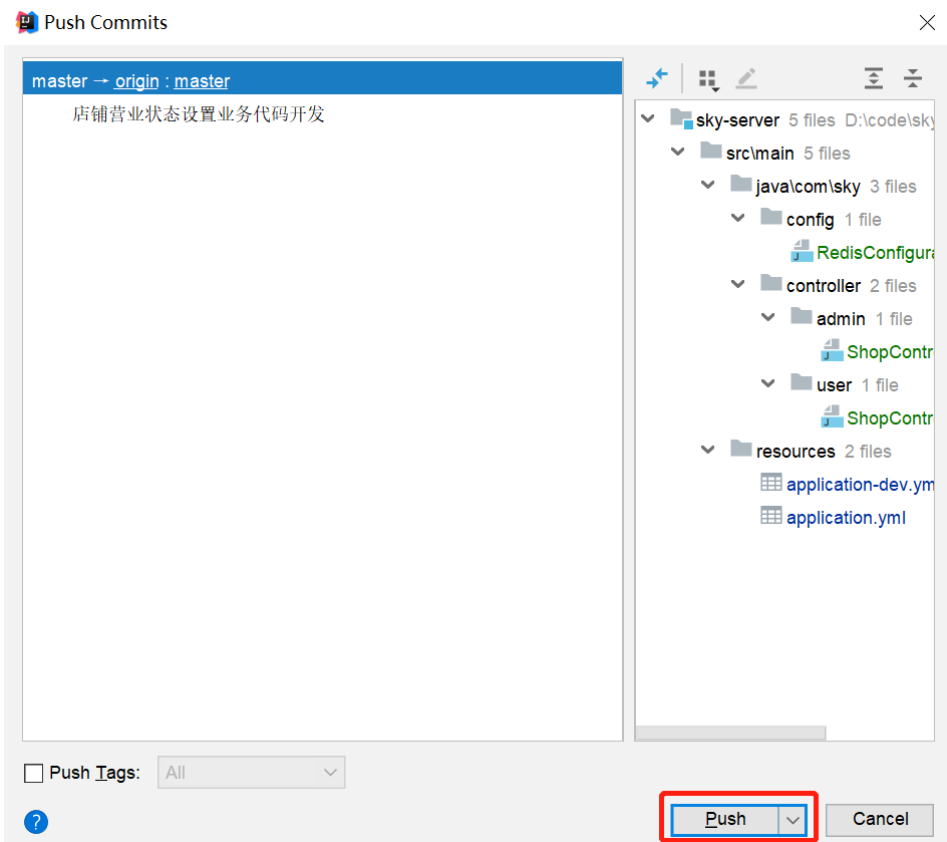
点击提交：



提交过程中，出现提示：



继续 push：



推送成功：

