第2章-查询与缓存

学习目标：

1 基础微服务-条件查询

1.1 标签-条件查询

POST /label/search 根据条件查询城市列表

（1）修改LabelService ,增加方法

```
/**
 * 构建查询条件
 * @param searchMap
 * @return
 */
private Specification<Label> createSpecification(Map searchMap){
    return new Specification<Label>() {
        @Override
        public Predicate toPredicate(Root<Label> root, CriteriaQuery<?>
criteriaQuery, CriteriaBuilder cb) {
            List<Predicate> predicateList=new ArrayList<>();
            if(searchMap.get("labelname")!=null &&
!"".equals(searchMap.get("labelname"))){
                predicateList.add(cb.like(
root.get("labelname").as(String.class), "%"+
(String)searchMap.get("labelname")+"%"  ) );
            }
            if(searchMap.get("state")!=null &&
!"".equals(searchMap.get("state"))){
                predicateList.add(cb.equal(
root.get("state").as(String.class), (String)searchMap.get("state") ) );
            }
            if(searchMap.get("recommend")!=null &&
!"".equals(searchMap.get("recommend"))){
                predicateList.add(cb.equal(
root.get("recommend").as(String.class),
(String)searchMap.get("recommend") ) );
            }
            return cb.and( predicateList.toArray( new
Predicate[predicateList.size()]) );
        }
    };
}

/**
 * 条件查询
 * @param searchMap
 * @return
 */

public List<Label> findSearch(Map searchMap){
```

```
    Specification specification= createSpecification(searchMap);
    return labelDao.findAll( specification);
}
```

（2）修改LabelController,增加方法

```
    /**
     * 根据条件查询
     * @param searchMap
     * @return
     */
    @RequestMapping(value="/search",method = RequestMethod.POST)
    public Result<List> findSearch( @RequestBody Map searchMap){
        return new Result<>(true,StatusCode.OK,"查询成
功",labelService.findSearch(searchMap));
    }
```

1.2 带分页的条件查询

（1）修改LabelService，增加方法

```
    /**
     * 分页条件查询
     * @param searchMap
     * @param page
     * @param size
     * @return
     */
    public Page<Label> findSearch(Map searchMap,int page,int size){
        Specification specification= createSpecification(searchMap);
        PageRequest pageRequest=PageRequest.of(page-1,size);
        return labelDao.findAll( specification ,pageRequest);
    }
```

（2）修改LabelController，增加方法

```
    /**
     *  条件+分页查询
     * @param searchMap
     * @param page
     * @param size
     * @return
     */
    @RequestMapping(value="/search/{page}/{size}",method =
RequestMethod.POST)
    public Result<List> findSearch( @RequestBody Map searchMap
,@PathVariable int page,@PathVariable int size ){
        Page pageList= labelService.findSearch(searchMap,page,size);
        return new Result(true,StatusCode.OK,"查询成功",new PageResult<>
(pageList.getTotalElements(),pageList.getContent()  ));
    }
```

2 招聘微服务开发

2.1 表结构分析

招聘微服务主要有两块：企业信息和招聘信息

企业表	**tb_enterprise**		
字段名称	字段含义	字段类型	备注
id	ID	文本	
name	企业名称	文本	
summary	企业简介	文本	
address	企业地址	文本	
labels	标签列表	文本	用逗号分隔
coordinate	企业位置坐标	文本	经度，纬度
ishot	是否热门	文本	0：非热门 1：热门
logo	LOGO	文本	
jobcount	职位数	数字	
url	URL	文本	

招聘信息表	**tb_recruit**		
字段名称	字段含义	字段类型	备注
id	ID	文本	
jobname	招聘职位	文本	
salary	薪资范围	文本	
condition	经验要求	文本	
education	学历要求	文本	
type	任职方式	文本	
address	办公地址	文本	
eid	企业ID	文本	
createtime	发布日期	日期	
state	状态	文本	0：关闭 1:开启 2：推荐
url	原网址	文本	
label	标签	文本	
content1	职位描述	文本	
content2	职位要求	文本	

2.2 代码生成

我们使用开源代码生成器codeutil 来完成代码的生成

开源网址： https://gitee.com/chuanzhiliubei/codeutil

（1）使用代码生成器生成招聘微服务代码 tensquare_recruit

（2）拷贝到当前工程，并在父工程引入。

（3）修改Application类名称为RecruitApplication

（4）修改application.yml 中的端口为9002 ,url 为

```
jdbc:mysql://192.168.184.134:3306/tensquare_recruit?characterEncoding=UTF8
```

（5）进行浏览器测试

2.3 代码编写

2.3.1 热门企业列表

需求：查询企业表ishot字段为1的记录

（1）EnterpriseDao新增方法定义

```
    /**
     * 根据热门状态获取企业列表
     * @param ishot
     * @return
     */
    public List<Enterprise> findByIshot(String ishot);
```

（2）EnterpriseService新增方法

```
    /**
     * 热门企业列表
     * @return
     */
    public List<Enterprise> hotlist(){
        return enterpriseDao.findByIshot("1");
    }
```

（3）EnterpriseController新增方法

```
    /**
     * 查询热门企业
     * @return
     */
    @RequestMapping(value="/search/hotlist",method=RequestMethod.GET)
    public Result hotlist(){
        return new Result(true, StatusCode.OK, "查询成功",
enterpriseService.hotlist());
    }
```

（4）测试 http://localhost:9002/enterprise/search/hotlist

2.3.2 推荐职位列表

需求分析：查询状态为2并以创建日期降序排序，查询前4条记录

（1）在RecruitDao新增方法定义

```
/**
* 查询最新职位列表(按创建日期降序排序)
* @return
*/
public List<Recruit> findTop4ByStateOrderByCreatetimeDesc(String state);
```

（2）RecruitService新增方法

```
    /**
     * 根据状态查询
     * @param state
     * @return
     */
    public List<Recruit> findTop4ByStateOrderByCreatetimeDesc(String
state){
        return recruitDao.findTop4ByStateOrderByCreatetimeDesc(state);
    }
```

（3）RecruitController新增方法

```
    @RequestMapping(value="/search/recommend",method= RequestMethod.GET)
    public Result recommend(){
        List<Recruit> list =
recruitService.findTop4ByStateOrderByCreatetimeDesc("2");
        return new Result(true,StatusCode.OK,"查询成功",list);
    }
```

（4）测试： http://localhost:9002/recruit/search/recommend

2.3.3 最新职位列表

需求分析：查询状态不为0并以创建日期降序排序，查询前12条记录

（1）在RecruitDao新增方法定义

```
    /**
     * 最新职位列表
     * @param state
     * @return
     */
    public List<Recruit> findTop12ByStateNotOrderByCreatetimeDesc(String
state);
```

（2）RecruitService新增方法

```
    /**
     * 最新职位列表
     * @return
     */
    public List<Recruit> newlist(){
        return recruitDao.findTop12ByStateNotOrderByCreatetimeDesc("0");
    }
```

（3）RecruitController新增方法

```
    /**
     * 最新职位列表
     * @return
     */
    @RequestMapping(value="/search/newlist",method= RequestMethod.GET)
    public Result newlist(){
        return new Result(true,StatusCode.OK,"查询成
功",recruitService.newlist());
    }
```

（4）测试：http://localhost:9002/recruit/search/newlist

3 问答微服务开发

3.1 表结构分析

问题表	**tb_problem**		
字段名称	字段含义	字段类型	备注
id	ID	文本	
title	问题标题	文本	
content	问题内容	文本	
createtime	发布日期	日期	
updatetime	更新日期	日期	
userid	发布人ID	文本	
nickname	发布人昵称	文本	
visits	浏览量	整型	
thumbup	点赞数	整型	
reply	回复数	整型	
solve	是否解决	文本	
replyname	最新回复人	文本	
replytime	最新回复时间	日期	

回答表	**tb_reply**		
字段名称	字段含义	字段类型	备注
id	ID	文本	
problemid	问题ID	文本	
content	回答内容	文本	
createtime	回答日期	日期	
updatetime	更新日期	日期	
userid	回答人ID	文本	
nickname	回答人昵称	文本	

问答标签中间表	**tb_pl**		
字段名称	字段含义	字段类型	备注
problemid	问题ID	文本	
labelid	标签ID	文本	

3.2 代码生成

（1）使用代码生成器生成招聘微服务代码 tensquare_qa

（2）拷贝到当前工程，并在父工程引入。

（3）修改Application类名称为QaApplication

（4）修改application.yml 中的端口为9003 ,url 为

```
jdbc:mysql://192.168.184.134:3306/tensquare_qa?characterEncoding=UTF8
```

（5）进行浏览器测试

3.3 代码编写

3.3.1 最新回答列表

需求分析：最新回复的问题显示在上方， 按回复时间降序排序。

（1）创建中间表pl的实体类

```
@Entity
@Table(name="tb_pl")
public class Pl implements Serializable{

    @Id
    private String problemid;

    @Id
    private String lableid;

    public String getLableid() {
        return lableid;
    }
    public void setLableid(String lableid) {
        this.lableid = lableid;
    }
    public String getProblemid() {
        return problemid;
    }
    public void setProblemid(String problemid) {
        this.problemid = problemid;
    }
}
```

（2）ProblemDao新增方法定义

```
    /**
     * 根据标签ID查询最新问题列表
     * @param labelId
     * @param pageable
     * @return
     */
    @Query("select p from Problem p where id in( select problemid from Pl
where labelid=?1 ) order by replytime desc")
    public Page<Problem> findNewListByLabelId(String labelId, Pageable
pageable);
```

（3）ProblemService新增方法

```
    /**
     * 根据标签ID查询问题列表
     * @param lableId 标签ID
     * @param page 页码
     * @param size 页大小
     * @return
     */
    public Page<Problem> findNewListByLabelId(String lableId,int page,
int size) {
        PageRequest pageRequest = PageRequest.of(page-1, size);
        return  problemDao.findNewListByLabelId(lableId,pageRequest);
    }
```

（4）ProblemController新增方法

```
    /**
     * 根据标签ID查询最新问题列表
     * @param labelid
     * @return
     */

@RequestMapping(value="/newlist/{labelid}/{page}/{size}",method=RequestMe
thod.GET)
    public Result findNewListByLabelId(@PathVariable String
labelid,@PathVariable int page,@PathVariable int size ){
        Page<Problem> pageList =
problemService.findNewListByLabelId(labelid, page, size);
        PageResult<Problem> pageResult = new PageResult<>
(pageList.getTotalElements(), pageList.getContent());
        return  new Result(true, StatusCode.OK, "查询成功",pageResult);
    }
```

3.3.2 热门问答列表

需求分析：按回复数降序排序

（1）ProblemDao新增方法定义

```
    /**
     * 根据标签ID查询热门问题列表
     * @param labelId
     * @param pageable
     * @return
     */
    @Query("select p from Problem p where id in( select problemid from Pl
where labelid=?1 ) order by reply desc")
    public Page<Problem> findHotListByLabelId(String labelId, Pageable
pageable);
```

（2）ProblemService新增方法

```
    /**
     * 根据标签ID查询热门问题列表
     * @param lableId 标签ID
     * @param page 页码
     * @param size 页大小
     * @return
     */
    public Page<Problem> findHotListByLabelId(String lableId,int page,
int size) {
        PageRequest pageRequest = PageRequest.of(page-1, size);
        return  problemDao.findHotListByLabelId(lableId,pageRequest);
    }
```

（3）ProblemController新增方法

```
    /**
     * 根据标签ID查询热门问题列表
     * @param labelid
     * @return
     */

@RequestMapping(value="/hotlist/{labelid}/{page}/{size}",method=RequestMe
thod.GET)
    public Result findHotListByLabelId(@PathVariable String
labelid,@PathVariable int page,@PathVariable int size ){
        Page<Problem> pageList =
problemService.findHotListByLabelId(labelid, page, size);
        PageResult<Problem> pageResult = new PageResult<>
(pageList.getTotalElements(), pageList.getContent());
        return  new Result(true, StatusCode.OK, "查询成功",pageResult);
    }
```

3.3.3 等待回答列表

（1）ProblemDao新增方法定义

```
    /**
     * 根据标签ID查询等待回答列表
     * @param labelId
     * @param pageable
     * @return
     */
    @Query("select p from Problem p where id in( select problemid from Pl
where labelid=?1 ) and reply=0 order by createtime desc")
    public Page<Problem> findWaitListByLabelId(String labelId, Pageable
pageable);
```

（2）ProblemService新增方法

```
    /**
     * 根据标签ID查询等待回答列表
     * @param lableId 标签ID
     * @param page 页码
     * @param size 页大小
     * @return
     */
    public Page<Problem> findWaitListByLabelId(String lableId,int page,
int size) {
        PageRequest pageRequest = PageRequest.of(page-1, size);
        return  problemDao.findWaitListByLabelId(lableId,pageRequest);
    }
```

（3）ProblemController新增方法

```
    /**
     * 根据标签ID查询等待回答列表
     * @param labelid
     * @return
     */

@RequestMapping(value="/waitlist/{labelid}/{page}/{size}",method=RequestM
ethod.GET)
    public Result findWaitListByLabelId(@PathVariable String
labelid,@PathVariable int page,@PathVariable int size ){
        Page<Problem> pageList =
problemService.findWaitListByLabelId(labelid, page, size);
        PageResult<Problem> pageResult = new PageResult<>
(pageList.getTotalElements(), pageList.getContent());
        return  new Result(true, StatusCode.OK, "查询成功",pageResult);
    }
```

4 文章微服务开发

4.1 表结构分析

文章表	**tb_article**		
字段名称	字段含义	字段类型	备注
id	ID	文本	
columnid	专栏ID	文本	
userid	用户ID	文本	
title	文章标题	文本	
content	文章内容	文本	
image	文章封面	文本	
createtime	发表日期	日期	
updatetime	修改日期	日期	
ispublic	是否公开	文本	0：不公开 1：公开
istop	是否置顶	文本	0：不置顶 1：置顶
visits	浏览量	整型	
thumbup	点赞数	整型	
comment	评论数	整型	
state	审核状态	文本	0：未审核 1：已审核
channelid	所属频道	整型	关联频道表ID
url	URL地址	文本	
type	文章类型	文本	0：分享 1：专栏

4.2 代码生成

（1）使用代码生成器生成招聘微服务代码 tensquare_article

（2）拷贝到当前工程，并在父工程引入。

（3）修改Application类名称为ArticleApplication

（4）修改application.yml 中的端口为9004 ,url 为

```
jdbc:mysql://192.168.184.134:3306/tensquare_article?characterEncoding=UTF8
```

（5）进行浏览器测试

4.3 代码编写

4.3.1 文章审核

（1）ArticleDao新增方法

```
    /**
     * 审核
     * @param id
     */
    @Modifying
    @Query("update Article set state='1' where id=?1")
    public void examine(String id);
```

（2）ArticleService新增方法

```
    /**
     * 文章审核
     * @param id
     */
    @Transactional
    public void examine(String id){
        articleDao.examine(id);
    }
```

（3）ArticleController新增方法

```
/**
 * 审核
 * @param id
 * @return
 */
@RequestMapping(value="/examine/{id}",method=RequestMethod.PUT)
public Result examine(@PathVariable String id){
    articleService.examine(id);
    return new Result(true, StatusCode.OK, "审核成功！");
}
```

4.3.2 文章点赞

（1）ArticleDao新增方法定义

```
/**
 * 点赞
 * @param id
 * @return
 */
@Modifying
@Query("update Article a set thumbup=thumbup+1 where id=?1")
public int updateThumbup(String id);
```

（2）ArticleService新增方法

```
/**
 * 点赞
 * @param id 文章ID
 * @return
 */
@Transactional
public int updateThumbup(String id){
    return articleDao.updateThumbup(id);
}
```

（3）ArticleController新增方法

```
    /**
     * 点赞
     * @param id
     * @return
     */
    @RequestMapping(value="/thumbup/{id}",method=RequestMethod.PUT)
    public Result updateThumbup(@PathVariable String id){
        articleService.updateThumbup(id);
        return new Result(true, StatusCode.OK,"点赞成功");
    }
```

5 缓存处理

为了提高查询的性能，我们通常采用Redis缓存解决。

5.1 Redis环境搭建

我们以docker的形式搭建Redis 服务

```
docker run -di --name=tensquare_redis -p 6379:6379 redis
```

5.2 SpringDataRedis

Spring-data-redis是spring大家族的一部分，提供了在srping应用中通过简单的配置访问redis服务，对reids底层开发包(Jedis, JRedis, and RJC)进行了高度封装，RedisTemplate提供了redis各种操作。

5.3 实现文章的缓存处理

5.3.1 查询文章操作缓存

（1）在tensquare_article 的pom.xml引入依赖

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

（2）修改application.yml ,在spring节点下添加配置

```
  redis:
    host: 192.168.184.134
```

（3）修改ArticleService 引入RedisTemplate，并修改findById方法

```
    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 根据ID查询实体
     * @param id
     * @return
     */
    public Article findById(String id) {
        //从缓存中提取
        Article article=
(Article)redisTemplate.opsForValue().get("article_"+id);
        // 如果缓存没有则到数据库查询并放入缓存
        if(article==null) {
            article = articleDao.findById(id).get();
            redisTemplate.opsForValue().set("article_" + id, article);
        }
        return article;
    }
```

这样在查询的时候，就会自动将文章放入缓存

5.3.2 修改或删除后清除缓存

```
    /**
     * 修改
     * @param article
     */
    public void update(Article article) {
        redisTemplate.delete( "article_" + article.getId() );//删除缓存
        articleDao.save(article);
    }

    /**
     * 删除
     * @param id
     */
    public void deleteById(String id) {
        redisTemplate.delete( "article_" + id );//删除缓存
        articleDao.deleteById(id);
    }
```

5.3.3 缓存过期处理

修改findById方法 ， 设置1天的过期时间

```
redisTemplate.opsForValue().set("article_" + id, article,1,
TimeUnit.DAYS);
```

为了方便测试，我们可以把过期时间改为10秒，然后观察控制台输出

```
redisTemplate.opsForValue().set("article_" + id, article,10,
TimeUnit.SECONDS);
```

5.4 Spring Cache

Spring Cache使用方法与Spring对事务管理的配置相似。Spring Cache的核心就是对某个方法进行缓存，其实质就是缓存该方法的返回结果，并把方法参数和结果用键值对的方式存放到缓存中，当再次调用该方法使用相应的参数时，就会直接从缓存里面取出指定的结果进行返回。

常用注解：

@Cacheable-------使用这个注解的方法在执行后会缓存其返回结果。

@CacheEvict--------使用这个注解的方法在其执行前或执行后移除Spring Cache中的某些元素。

5.5 活动信息的缓存

5.5.1 活动微服务代码生成

（1）使用代码生成器生成招聘微服务代码 tensquare_gathering

（2）拷贝到当前工程，并在父工程引入。

（3）修改Application类名称为GatheringApplication

（4）修改application.yml 中的端口为9005 ,url 为

```
jdbc:mysql://192.168.184.134:3306/tensquare_gathering?
characterEncoding=UTF8
```

（5）进行浏览器测试

5.5.2 活动详情的缓存实现

步骤：

（1）我们在tensquare_gathering的pom.xml中引入SpringDataRedis

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

（2）修改application.yml , 在spring节点下添加redis 配置

```
  redis:
    host: 192.168.184.134
```

（3）为GatheringApplication添加@EnableCaching开启缓存支持

```
@EnableCaching
```

（4）在GatheringService的findById方法添加缓存注解，这样当此方法第一次运行，在缓存中没有找到对应的value和key，则将查询结果放入缓存。

```
    /**
     * 根据ID查询实体
     * @param id
     * @return
     */
    @Cacheable(value="gathering",key="#id")
    public Gathering findById(String id) {
        return gatheringDao.findById(id).get();
    }
```

（5）当我们对数据进行删改的时候，需要更新缓存。其实更新缓存也就是清除缓存，因为清除缓存后，用户再次调用查询方法无法提取缓存会重新查找数据库中的记录并放入缓存。

在GatheringService的update、deleteById方法上添加清除缓存的注解

```
    /**
     * 修改
     * @param gathering
     */
    @CacheEvict(value="gathering",key="#gathering.id")
    public void update(Gathering gathering) {
        gatheringDao.save(gathering);
    }

    /**
     * 删除
     * @param id
     */
    @CacheEvict(value="gathering",key="#id")
    public void deleteById(String id) {
        gatheringDao.deleteById(id);
    }
```

面试问题总结

在项目中哪部分业务用到缓存

你说一下项目中是如何使用缓存的

说一下如何设置缓存过期时间