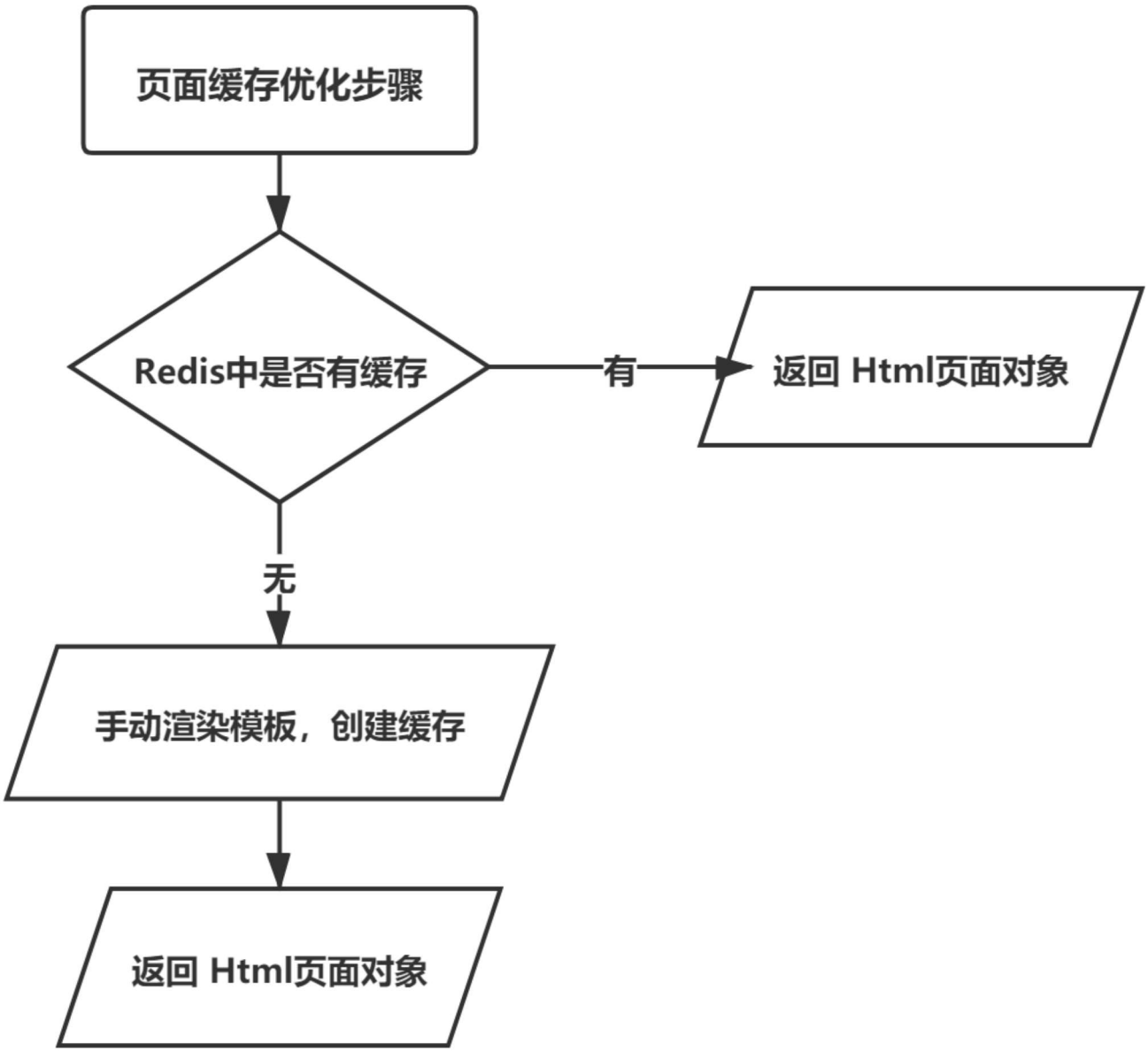
目录

1. 页面缓存优化

1.1 未经优化之前的代码

```
    @RequestMapping("/to_list")
    public String toList(Model model,MiaoShaUser user){
        model.addAttribute("user",user);
        List<GoodsVo> goodsVos = goodsService.listGoodsVo();
        model.addAttribute("goodsList",goodsVos);
        return "goods_list";
    }
```

1.2 优化产生的改变




```
    @RequestMapping(value = "/to_list",produces = "text/html")
    @ResponseBody
    public String toList(HttpServletRequest request, HttpServletResponse response, Model
        model.addAttribute("user",user);
        //在有缓存的情况下，取出缓存
        String html = redisService.get(GoodsKey.goodsKeyPrefix, "", String.class);
        if(! StringUtils.isEmpty(html)) return html;

        //在没有缓存的时候，手动渲染，添加缓存
        List<GoodsVo> goodsVos = goodsService.listGoodsVo();
        model.addAttribute("goodsList",goodsVos);
        IWebContext ctx = new WebContext(request,response,request.getServletContext(),req
        html = thymeleafViewResolver.getTemplateEngine().process("goods_list",ctx);//这里
        if(!StringUtils.isEmpty(html)){
            redisService.set(GoodsKey.goodsKeyPrefix,"",html);
        }
```

```
        return html;
        //return "goods_list";
    }
```

- 首先，我们应用缓存，一定要引入RedisService

1. @RequestMapping(value = “/to_list”, `produces = "text/html"`)produces标注了返回值的类型，必须与@ResponseBody搭配使用
2. 手动渲染过程中，我们要注入 `ThymeleafViewResolver` ，这个是框架给我们准备好的Bean，利用它来渲染页面，其中第二个参数，需要注入 `IContext`
3. 在 `Spring5` 版本中， `SpringWebContext` 已经没有了，我们需要使用 `WebContext` 来代替。它剔除了之前对ApplicationContext 过多的依赖，现在thymeleaf渲染不再过多依赖spring容器
4. 再者，我们对Redis缓存的时间设置了 `60秒` 的限制，超过60秒过期，这个时间不宜过长。在60秒内我们看到的网页一直一样是暂且可以接受的

---

2. 对象缓存与缓存更新

2.1 对象缓存

对象缓存，我们之前已经做过了一个，就是在MiaoshaService中的 `getByToken` 方法，通过token值，从Redis中获取对象信息。
这次，我们实现一个getById()方法，即通过Id值，从Redis中获取user对象。（对象缓存 没有设置过期时间 ，而且对象缓存是 粒度最小 的缓存）

```
    public MiaoShaUser getById(long id){
        //先从缓存中取
        MiaoShaUser user = redisService.get(MiaoShaUserKey.idPrefix, "" + id, MiaoShaUser
        if(user != null) return user;

        //缓存中没有，从数据库中取，并且把它添加到缓存中
        user = miaoShaUserDao.getById(id);
        if(user != null) redisService.set(MiaoShaUserKey.idPrefix,"" + id,user);

        return user;
    }
```

2.2 缓存更新

我们模拟一个场景，我们要对密码进行修改，那么缓存也需要修改，现在先列出视频中给的方法，通过Id值取出用户，修改数据库，之后，对token-user缓存进行修改，id-user缓存进行删除

```
    public boolean updatePassword(long id,String formPass,String token){
        //取出user
        MiaoShaUser user = getById(id);
        //没有这个用户
        if(user == null) throw new GlobalException(CodeMsg.MOBILE_NOT_EXIST);

        //修改密码，更新数据库
        user.setPassword(MD5Util.formPassToDBPass(formPass,user.getSalt()));
        miaoShaUserDao.update(user);
        //更新缓存,token-user缓存（登陆用的）这个不能删除，id-user缓存删除
        redisService.set(MiaoShaUserKey.getTokenPrefix,token,user);
        redisService.delete(MiaoShaUserKey.idPrefix,id);

        return true;
    }
```

- **个人理解**：我们上网时的多数场景，修改完密码之后都要我们进行重新登录，而且在我们这个项目中，登录的过程中会对token-user缓存进行重新添加，那么我们在修改密码的时候，可以直接将token-user和id-user全部都删除，而不需要对其中的缓存进行值的修改。

---

3. 页面静态化

3.1 将商品详情页进行静态化处理（订单详情也做了静态化）

通常情况下，页面不采用第一种缓存的方式实现优化，而是通过静态化处理，比较常用的技术有Vue。通过静态化处理，我们将页面缓存在客户端浏览器中，不需要与服务器交互就能访问到页面。

以下，我们用JQuery实现。

3.1.1 对后端代码进行处理

```
    @RequestMapping(value = "/detail/{goodsId}")
    @ResponseBody
    public Result<GoodsDetailVo> toDetail(HttpServletRequest request, HttpServletResponse

        GoodsVo goodsVo = goodsService.getGoodsVoByGoodsId(goodsId);

        //秒杀开始、结束时间，当前时间
        long startDate = goodsVo.getStartDate().getTime();
        long endDate = goodsVo.getEndDate().getTime();
```

```
        long now = System.currentTimeMillis();

        //秒杀状态，0为没开始，1为正在进行，2为秒杀已经结束
        int miaoshaStatus = 0;
        //距离秒杀剩余的时间
        int remainSeconds = 0;

        if(now < startDate){
            //秒杀没开始，进行倒计时
            remainSeconds = (int) (startDate - now) / 1000;
        }else if(now > endDate){
            //秒杀已经结束
            miaoshaStatus = 2;
            remainSeconds = -1;
        }else {
            //秒杀进行时
            remainSeconds = 0;
            miaoshaStatus = 1;
        }
        GoodsDetailVo goodsDetailVo = new GoodsDetailVo();
        goodsDetailVo.setGoods(goodsVo);
        goodsDetailVo.setUser(user);
        goodsDetailVo.setMiaoshaStatus(miaoshaStatus);
        goodsDetailVo.setRemainSeconds(remainSeconds);

        return Result.success(goodsDetailVo);
    }
```

- @RequestMapping中，去掉produces属性
- 去掉Model向前端传值的逻辑，只留下业务处理过程，并将所需要的的值封装在 GoodsDetailVo 对象中
- 注意事项 在GoodsDetailVo中的属性字段要与前端所需要的字段名保持一致，如下所示，这样才能获取

```
@Data
public class GoodsDetailVo {
    private long miaoshaStatus;
    private long remainSeconds;
    private GoodsVo goods;
    private MiaoShaUser user;
}
```

对应前端

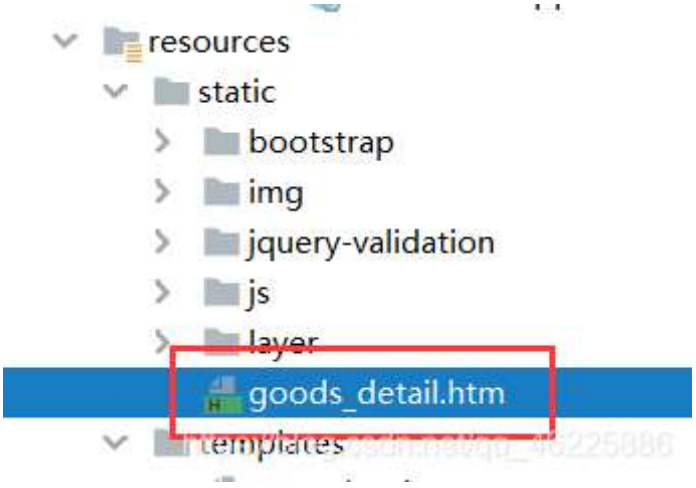
```
function render(detail){
    var miaoshaStatus = detail.miaoshaStatus;
    var remainSeconds = detail.remainSeconds;
    var goods = detail.goods;
    var user = detail.user;
    if(user){
        $("#userTip").hide();
    }
```

3.1.2 对前端跳转的修改

我们从商品列表页面跳转到商品详情页，修改为如下

```
<td th:text="${goods.stockCount}"></td>
<td><a th:href="'/goods_detail.htm?goodsId=' +${goods.id}">详情</a></td>
```

注意其中 `/goods_detail.htm` ，它是放在static目录下的静态资源，为了防止视图解析器的跳转，将 `html写为htm`



3.1.3 在application.properties中配置

```
# static
spring.resources.add-mappings=true
spring.resources.cache.period= 3600 #缓存时间
spring.resources.chain.cache=true
spring.resources.chain.enabled=true
#spring.resources.chain.gzipped=true
spring.resources.chain.html-application-cache=true
spring.resources.static-locations=classpath:/static/
```

4. POST请求和GET请求的区别

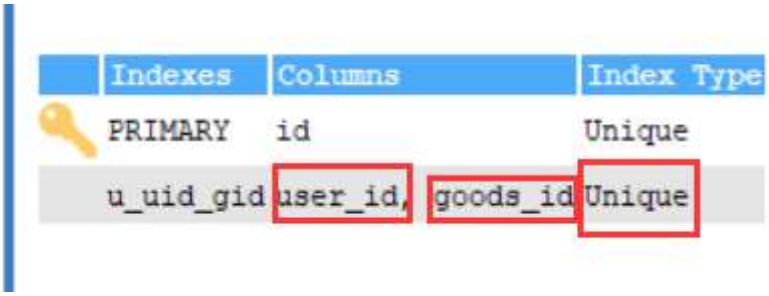
- GET：这个请求是幂等的，从服务端获取数据，反复获取不会对数据有影响。因为GET因为是读取，就可以对GET请求的数据做缓存。这个缓存可以做到浏览器本身上（彻底避免浏览器发请求），也可以做到代理上（如nginx），或者做到server端（用Etag，至少可以减少带宽消耗）
- POST：该请求是不幂等的，它会在页面表单上提交数据，请求服务器的响应，往往会对数据进行修改

5. 解决超卖问题

1. 当多个线程同时读取到同一个库存数量时，防止超卖，修改SQL语句

```
#添加stock_count > 0的条件
update miaosha_goods set stock_count = stock_count - 1 where goods_id = #{goodsId} and st
```

2. 防止同一个用户秒杀多个，添加唯一索引，绑定user_id和goods_id，这样同一个用户对同一个商品的秒杀订单是唯一的



	Indexes	Columns		Index Type
🔑	PRIMARY	id		Unique
	u_uid_gid	user_id,	goods_id	Unique