背景

在降本提效过程中, 发现较多的慢sql其大部分场景是在列表查询中 过多的展示关联表的信息 导致(**慢sql中join数量过多**)。

而业务上又难以去除这些内容, 为了改善这种场景的性能问题, 我们需要在数据源返回后, 对关联表信息进行二次查询并返回的统一解决方案。

本质上就是去除多余的慢sql中的join采用二次查询的方案展示, 从而提升查询效率, 优化慢sql, 降低数据库压力。

简介

- 名称: 二次查询缓存系统
- 方案: **关联表信息不再实时查询; 在接口返回时从缓存中提取;**
- 工具: cache2
- 原理: 采用 二次查询并缓存 的方式, **去除join操作**
- 适用: 需要二次查询的所有场景(如: 字段提取、一对一、一对多)
- 愿景: 愿天下所有的查询都没有JOIN

案例

接口现状

- 案例中订单列表需要将订单中的状态名称进行展示, 但是订单没有冗余订单状态名称字段, 而是需要从t_dict表中进行获取;

  > 现在接口中会每次都关联一次或多次t_dict表进行查询, 在慢sql中影响查询性能大约: 200ms。

- 一些需要展示但是有不想使用join关键字关联查询的操作, 都是在代码中进行二次查询

  > 这样的话相对难维护且每次都需要编码。

预期效果

- 仅在接口返回中第一次进行全量查询, 后续每次都从缓存中提取, 预期降低99.5%(约199ms)的性能损耗。
- 不需要写逻辑只需要给实体类加上注解即可完成二次查询操作。

添加依赖

> pom.xml

```xml
<dependency>
    <groupId>${project.groupId}</groupId>
    <artifactId>tools-cache2</artifactId>
    <version>${last.version}</version>
</dependency>
```

添加注解

> @Cache

```
@Data
public class OrderVo extends Order {
    /**
     * 案例: 字段提取
     * ----------------------------------
     * target*: 缓存主题（表名/索引名/主题等）
     * key*: 外键名称
     * put*: 上传已知字段
     * ----------------------------------
     * get: 下载缓存字段(默认与字段名一致)
     * where: 源数据的查询条件(默认查表中所有数据)
     * dataType: 数据源类型(默认是mysql数据库: 可以自定义)
     * expire: 过期时间(默认-1永不过期; 为0则不缓存直接从源数据取)
     * enable: 是否启用(默认true)
     */
    @Cache(
            target = "t_dict",
            key = "dict_code",
            dataType = DataType.MYSQL,
            put = "status",
            get = "dict_name",
            where = "dict_type_code = 'OD_STATUS' and delete_tag = false",
            expire = 24 * 3600,
            enable = true
    )
    private String statusName;

    /**
     * 案例: 一对一
     */
    @Cache(target = "t_dict", key = "dict_code", put = "status", where =
"dict_type_code = 'OD_STATUS' and delete_tag = false")
    private Dict dict;

    /**
     * 案例: 一对多
     */
    @Cache(target = "t_dict", key = "dict_code", put = "status", where =
"dict_type_code = 'OD_STATUS' and delete_tag = false", expire = 0)
    private List<Dict> dictList;

}
```

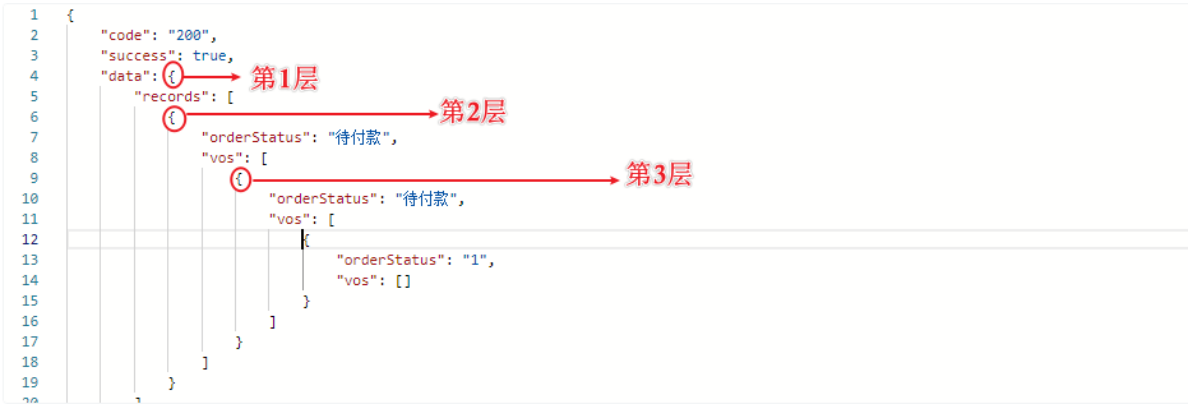
启用缓存

> @EnableCache

```
@EnableCache2
@SpringBootApplication
@ComponentScan(basePackages = {"cn.gmlee.demo.css.cache", "cn.gmlee.tools"})
public class CacheApp {
    public static void main(String[] args) {
        SpringApplication.run(CacheApp.class, args);
    }
}
```

缓存深度

> 默认深度: 3 (从0开始, 一层{}算1级, 数组不算)



运行日志

```
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :56
- ----------------------
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :59
- 缓存表名：t_dict
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :60
- 填充属性：statusName
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :61
- 填充内容：还款中
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :62
- 当前耗时：0(ms)
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :63
- 平均耗时：0(ms)
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :64
- 总访问次：85
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :65
- 命中次数：85
2022-04-28 16:03:59.682  [nio-8888-exec-9] INFO  StatKit                    :66
- 命中概率：100.0%
2022-04-28 16:03:59.683  [nio-8888-exec-9] INFO  StatKit                    :56
- ----------------------
```

拓展

数据源

> cache2从实际情况出发默认就是采用mysql作为数据源, 如果你的缓存数据是在es、redis、oralce等环境可以自定义

Mysql

- dataType = DataType.MYSQL

Oracle

- dataType = DataType.ORACLE

API

- dataType = DataType.API

缓存类型

> at-cache内置了redis和disk两种缓存实现, 并且支持自定义缓存

Redis

- 在项目中引入spring-redis的jar包就会自动选用Redis作为缓存系统;
- Redis缓存默认采用TOOLS:CACHE2:KEY_作为前缀, 可通过tools.cache2.key注入其他值

Memory

- 当项目中没有spring-redis或手动排除了Redis缓存机制(不会影响其他业务使用)则自动启用Disk实现

```
@SpringBootApplication(exclude =
CacheAutoConfiguration.RedisCacheServerConfig.class)
```

自定义实现

- 当以上两种缓存系统无法满足时

> 在项目中注入自定义实现即可: CacheServer