# 数据应用门户接口优化方案

**文档修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **日期** | **修改人** | **摘要** |
| V1.0 | 2023-03-02 | TOPS终端项目组 | 新建文档 |

## 问题背景

鉴于当前门户主要是通过调用三方API进行简单的查询操作，并且代码不涉及到复杂的业务逻辑处理，针对接口响应时间过长的问题，项目组进行了充分的分析与论证（参考附件，接口响应时间统计）。由于数据来源于数仓，其本身的优化空间小，那么项目组就考虑通过在调用方添加缓存来解决响应慢的问题。

## 目标

在较少的工作量下，实现页面数据加载用时在3秒以内，进而满足日常使用。

## 整体方案

* 前端

针对前台静态资源(js、css、font、img)设置浏览器缓存，可通过nginx代理配置实现，可提升页面20%提升，领需注意前端每次发布发布文件需注意文件版本问题；缓存高频交互接口（如字典数据、最大交易日查询），登录后请求后缓存，登出失效删除缓存，进而减少单页后端请求数，降低队列请求数。开启nginx传输gzip压缩，可提升2-5倍传输性能。

* 后端

首先针对调用第三方的API的查询场景梳理出性能慢的接口清单,针对重点接口添加查询缓存。目前优先提升一二级菜单页面单接口初次加载的请求场景添加缓存。如何使用缓存目前有两个方案：

A)新建定时任务，主动缓存页面接口数据，定时过期缓存，优点是体验好，缺点是改动大不易落地，第二次自定义野蛮查询无缓存。

B) 请求的接口的时候缓存接口数据，通过定时任务定时过期数据。优点是代码改动量小，灵活，支持更多数据缓存，缺点是占用内存空间多，更倾向此方案。

C) 采用缓存框架缓存数据。数据主要分为永久性数据和临时数据。永久数据：结合项目特点，我们将通过DB或接口接收大量数据、特别影响效率的数据认为是永久数据，通过策略在不同时段缓存数据；临时数据；通过接口查询且效率较低的数据，相同查询第一次后使用缓存。优点是与springboot结合、缓存策略成熟、分布式部署方案等均较为成熟、数据注入缓存及清理缓存等功能使用配置文件结合注解，使得代码的侵入较少。

缓存技术选型：

目前市面上比较好的分布式缓存组件ehcache和redis，结合目前场景，我们可选择redis作为缓存的底层缓存库，优点是支持多种部署方式，丰富的数据结构，缓存过期策略、灾备机制等，技术比较成熟。

另一种方案，使用基于springboot的缓存框架（如：J2Cache、Jetcache），框架本身不是缓存技术，而是把主流缓存技术有效的结合在一起，更方便、更高效、更安全的使用缓存方案。目前主要框架如J2cache、jetcache，都非常成熟。优点是：对微服务的支持更好，可以方便的多节点分布式集群部署。可以有效解决重启导致的内存雪崩问题。有效解决多节点部署数据不同步问题。有效解决集中式（如Ehcache、Redis）缓存的不足、集群方案的不足。

## 缓存设计

**后端设计：**

缓存key设计，举例**dap:{biz}:{[paramv1:paramv2:...]}**，biz代表业务接口编号，{[paramv1:paramv2:...]}代表接口参数值拼接，其他参数不可为空，例如：dap:bondMap:20230201:20230501。

缓存添加在访问接口时，根据条件查询是否存在缓存如果不存在则直接查询并缓存结果，并设置过期策略。

提供统一过期缓存接口，使用场景针对缓存数据过时与业务实际数据有差异场景。实现可通过命令SCAN cursor [MATCH pattern] [COUNT count] 批量获取Key并删除。

缓存切面设计在后端服务层统一以注解的形式可结合spring cache实现。

缓存框架方案：

在需要缓存的业务代码使用注解，结合缓存策略的配置文件，即可完成对数据的缓存注入及清理，对代码侵入较少。

可采用多级缓存策略，计划使用二级缓存策略：

L1：Ehcache

L2：Redis

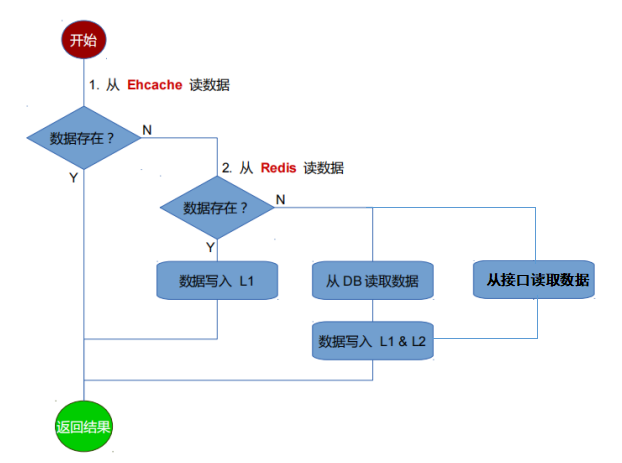
**前端设计：**

1. 静态资源（css、img、font、js）添加nginx过期策略

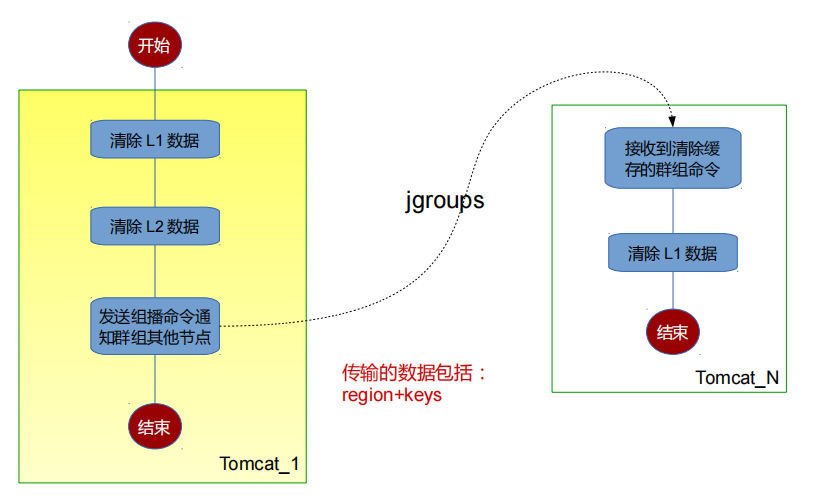
2、高频接口前端缓存（如字典维度数据），登录后请求后数据缓存，登出失效删除缓存,优化相关场景的代码逻辑。

## 接口设计（demo）

* 采用spring cache 封装注解机制，添加缓存@cacheable，
* 新增定期清理缓存接口，每天定时8点清除缓存。
* 采用springboot的主流缓存框架（如：J2Cache、Jetcache）
* 读取流程



* 更新流程



## 数据一致性问题（解决方案）

考虑数据一致性问题，主要考虑缓存与实际接口平台存在差异问题，原因可能为平台接口数据未跑完情况或应用调整了逻辑。第一步可减少数据缓存过期时间，提升时效。针对应用逻辑调整的情况可手工触发一次清空缓存，兜底方案是每天定时清理清理缓存一次。

不同的业务或接口获得数据注入缓存及清理缓存的策略可能存在差异，因此对应不同的需求，通过配置文件添加不同缓存的缓存策略，并在不同场景的代码中使用注解，完成数据个性化缓存的需求

## 关联系统的影响

应用发布上线或者数据平台重新跑任务需要，根据实际情况是否手工触发一次缓存清理操作。

## 是否有利于后期技术拆分

应用缓存以注解的形式添加在service 实现层业务方法上， 对代码浸入性少，如果考虑不适用缓存，则在全局配置文件禁用即可。

## 资源分配情况（人力、硬件以及具体配置）

**工作量预估：**

初步统计需要添加缓存的接口总数是80个上下，经过初步评估工作量在40人天左右，其他如缓存结构设计、定时清理缓存、手工清理单接口缓存等额外功能工作量10人天。总计50人天

**硬件及配置：**

Redis 服务器需要2~3台做集群，硬件配置如下：

CPU：2个或更多的CPU核心，建议使用高性能的CPU，例如Intel Xeon。

内存：至少16GB RAM

存储：使用高速磁盘，如SSD，至少50GB的存储空间，用于持久化数据快照和日志文件。

## 预估工期与时间节点