**学习文档**

指向目标：鉴权服务

目标版本：2.6.1

Description：鉴于数据鉴权关口服务auth性能问题，定向对整个鉴权服务做了优化，改动点包括鉴权服务的所有类所有方法，以及redis中存储的数据模式类型。结构设计，代码开发历经三周，中间的版本也有经历了四个版本：

key-list  -->  key-hash  -->  缓存分片(横向切分) + 分页  --> 数据结构切分(纵向切分) + 扩展字段配置化。

1. key-list 严格来说是key-set 的存储方式，缺点比较明显：
   * + 1. tps比较低，记录数据为：15~25
       2. cpu, 缓存, io压力都特别大，初始化缓存数据时速度慢，将近40min
       3. 各服务模块的压力也比较大，因为每次都是大批量数据流在进行交互，而且许多都是无用数据，

结果表现：1、性能差，压力大

1. 数据初始化压力大、速度慢
2. key-hash 鉴权服务的压力稍微缓解，数据量过多也是给个服务很大压力，特别是业务基础，而且同样是大批量数据在io流中，表现为：
   * + 1. tps依旧比较低，记录数据为：30~40
       2. cpu，缓存，io压力依旧比较大，初始化速度慢
       3. 各服务间大数据流交互

结果表现：1、io及缓存压力大

1. 数据初始化压力大、速度慢也需要半个小时
2. redis缓存分片(横向切分) + 分页

经历了两个阶段：

1. 根据数据类型划分缓存，outKey为数据类型，当前数据类型有11种，对redis稍微缓解，但是性能要求依旧不够
2. 根据时间以及数据类型分片，outKey为时间以及数据类型 ，并且针对当前业务 区别 划分缓存数据：最新是获取一个月内的数据，最热是获取15天的数据，全部、活动以及其他基本是获取两年内的数据集(由于数据清理还没有上，所以暴露出的问题有：有些过期数据不显示--针对此bug已将鉴权数据提到获取五年内数据，然后如果数据清理不上，数据越来越多会降低服务性能，需要数据清理也尽快上线)
3. tps 表现明显 最新接口 200，最热接口 190 全部 125
4. cpu 压力稍大，刷数据时候表现特别明显

结果表现：1、数据更新流程复杂，需要改动多，且需要频繁的大批量改动缓存，容易引起服务阻塞或者鉴权数据取不到

1. 耦合性高，业务需求如果变化，所有涉及到的接口调用都需要同步更改，历史鉴权数据也需要重新更新，业务需求变化困难，不符合本方案低耦合易修改的设计初衷
2. 初始化刷数据auth服务压力较大

结果表现：1、auth服务以及cpu压力稍大

2、数据初始化速度明显，时间将近十分钟，使用游标后降低到五分钟

1. 数据结构切分(纵向切分) + 扩展字段配置化

在横向切分的基础上再配合localCache 做纵向切分，将数据的部分属性存储到本地

1. tps 表现明显 最新接口250~350,最热接口220，全部113
2. 非全部请求的压力都较小，全部请求时候auth 以及cpu压力较大

表现在数据量上：切割的数据一部分存在本地，扩容性不好，数据量过大时对速率有一定影响（强依赖于数据清理）

结果表现：1、auth服务以及cpu压力稍大

2、数据初始化速度明显，使用游标后降低到两分钟，初始化本地缓存时间需要三分钟

**当前版本截止到上述过程，统计数据都是在本地环境，服务器上面的表现已经很明显了，接口响应tps也远远达到需求。**

**基于V4 以后可做的优化方式如下**：

1. 缓存数据结构切分(根据企业id划分缓存) 以及数据id的压缩(鉴于当前的数据id不统一：17位或者32位，包含字母等等各种类型的复杂度，后续版本建议将id统一，以及业务字段统一)，由于本地缓存考虑到服务能分到的内存，所以存储的数据量有限，随着日后数据过度膨胀，服务压力以及接口速度也会越发明显。