2、mysql亿级大表如何优化?

一、线上事故回顾

1、事故现象

某一天，晚上凌晨12点30左右，突然收到线上某个mysql业务库实例**CPU告警(使用率飙升到70%)**，除此之外，还收到大量的**慢SQL告警（大于1s的，超过100条）**。

2、排查事故思路

研发人员看到告警后，第一时间打开电脑，通过堡垒机，开始远程访问内网，主要排查思路如下：

2.1 根据经验初步预判数据库CPU飙高的原因

晚上12点30分钟左右，一般是业务低峰期，系统的流量理论上是也是低峰期，应用负载QPS比较低，mysql应该是空闲状态比较合理，但是这个时候数据库的CPU出现突刺，从日常的20%飙高到70%，有持续不断增加的趋势，大概率是有跑批的任务在大批量处理业务数据。

2.2 通过云监控观察数据库是否有大量慢SQL

晚上12点30分钟左右，超过1s以上的SQL有100多条，部分SQL执行时间超过20s，而且慢SQL产生的时间非常集中，此外，还有部分SQL处于lock状态，mysql CPU线程非常繁忙。

2.3 通过explain分析慢SQL性能

从云监控系统导出top20的慢SQL语句，通过explain执行计划进行分析，看下SQL**区分度比较高**的查询字段是否命中索引(看两个指标：扫描记录行数、命中索引的type：结果值从好到差依次是：system > const > eq_ref > ref > range > index > ALL)

2.4 通过慢SQL分析反查代码是哪块业务处理不当引起的

从云监控系统中导出top20的慢SQL语句，分析慢SQL，反查连接这个数据库对应应用的业务代码，可以快速定位是什么业务引起的。

3、初步结论

在1个月前，业务方导入7000万客群，客群的过期时间是1个月，晚上凌晨12点30，刚好有个延迟任务在处理这7000w过期的客群明细数据，导致数据库CPU告警，由于是分批处理过期数据，有大量的慢SQL。

二、紧急处理方案

1、指导原则

尽可能保证mysql业务库实例CPU不被打挂，然后再考虑优化SQL语句或者优化程序代码。

2、处理措施

2.1 找到锁表的慢SQL线程，执行kill id

```
--步骤1: 使用root账户，查看当前正在运行的线程
SHOW FULL PROCESSLIST

--步骤2: 找到lock状态的线程ID：9874。command为waitting的就是锁住的表，info为执行某条语句的信息，id为线程Id。

--步骤3: 执行kill命令，停掉lock的线程
kill 9874;
```

2.2 找到耗时久的慢SQL线程，执行kill id

```
--步骤1：查询执行时间超过10s的线程，然后拼接成kill语句
select
   concat('kill ', id, ';') as '慢SQL'
from
   information_schema.processlist
where
    command != 'Sleep' and time > 10
order by time desc;

--步骤2：得到步骤1的返回结果
'慢SQL'
--------------------------------------------------
kill 9874;
kill 9879;
kill 9983;

--步骤3：执行步骤2返回的结果
kill 9874;
kill 9879;
kill 9983;
```

2.3 再次观察mysql cpu，看看是否有明显的下降趋势

完成1，2步骤后，再次通过云监控系统观察mysql业务库实例CPU使用率，看看是否有明显的下降。如果mysql CPU没有明显的下降，这个时候就需要通过explain执行计划分析慢SQL，看下是否缺少索引或者命中的索引字段区分度太低，导致扫描的行数太多。

2.4 通过explain执行计划分析慢SQL

2.4.1 优先考虑对大表加索引(成本低，见效快)

通过explain执行计划，分析当前慢SQL是**否缺乏索引或者命中的索引字段区分度太低**，导致扫描的行数太多(百万或千万、甚至亿级别），我们一般在业务低峰期(一般是晚上12点以后)，通过在线DDL，给亿级大表添加索引或者更换索引(对于千万甚至亿级数据量的大表变更字段或索引，一般大公司的做法是通过建新表、数据copy，rename、数据打平的方式进行平滑处理）。

PS：如果表数据量在百万以下，可以不用等业务低峰期操作，直接通过在线DDL添加或者更换索(先建新索引，再删除索引)，可能会出现短暂的锁表。

2.4.2 优化慢SQL

2.4.2.1 检查索引是否失效

检查索引字段是否存在：类型转换、函数计算、全模糊查询、not in、or、索引项存在null值

2.4.2.2 检查高频查询字段是否建立联合索引

联合索引遵守最左匹配原则，(a,b,c)，ab，ac，abc字段查询走索引，bc字段查询不走索引

比如：客户表: customer_info 字段: id, user_id,mobile,real_name,level,last_login_date

前端业务或外部接口经常需要根据userId、mobile或userId、real_name查询, 这里可以创建联合索引（userId，mobile，real_name）

2.4.2.3 检查字段是否存在like全模糊前缀查询

比如：SQL中使用全模糊查询：字段 like %关键字%，需要改为左模糊查询：字段 like 关键字%

2.4.2.4 检查SQL语句是否尽量走覆盖索引(select 字段只包含索引字段，目的是减少回表查询数据行)

比如：客户表: customer_info 字段: id, user_id,mobile,real_name,level,last_login_date 索引: id,user_id,mobile

错误写法：select * from customer_info

正确写法：select id, user_id,mobile from customer_info

2.4.2.5 检查大表中是否使用类似LIMIT M,N深度分页查询，以及对非索引字段进行排序

PS；M=(start-1)*pageSize N=pageSize , start: 页码 pageSize: 页大小

这是一个典型的深度分页问题，当start越大，扫描的行数越大，如果表是百万或千万、甚至亿级别数据量，假如前端页面查询最后一页，将进行千万大表的全表扫描，妥妥的慢sql。

业界通用的解决方案：**游标法**(每次查询找到表中主键id最大的值，作为参数向下传递)

Plain Text

```
// 这里提供一个案例，后面会提供相应的实现代码
SELECT
  c.*
FROM
  t_staff c
WHERE
  c.delete_flag = 0 and id > #{maxId}
  order by id limit #{pageSize}
```

另外格外注意：对于大表，排序字段务必加上索引。

2.4.2.6 大数据量处理改为多次小批量处理

对大数据量表处理，用小批量的方式代替，可以减少对主节点的压力和主从延迟。常见的处理方式：

- 缩小查询条件中查询时间的范围

  比如：1次删除1个月(30天)的数据，改为1次删除1天的数据，分30次删除完成

- 减少单次处理数据的大小

  比如：1次删除1个月(30天)所有的数据，改为先查询最大的id，然后1次处理1000条，如下所示：

```
Plain Text
1  SELECT MAX(id) INTO @max_id FROM t_staff WHERE create_date<'2022-10-01' ;
2  DELETE FROM t_staff WHERE id<@max_id LIMIT 1000;
3  select ROW_COUNT();  #返回1000
```

三、长期解决方案

1、数据库扩容

扩容是一种简单粗暴又非常有效的方案，可以快速降低CPU使用率。不过扩容前建议考虑下未来业务增长量，不然扩容后使用率很低，是一种资源浪费。比如：数据库配置从2c 4G提升到4c 8G，支撑的QPS从1000提升到2000。

2、长期治理慢SQL

- 每天9点30定时发送整理后的慢SQL邮件(发送人：各组leader、核心骨干)
- 各组leader根据慢SQL影响面、业务优先级，制定一个长期的慢SQL治理计划，分迭代逐步偿还这些技术债务,一般的慢SQL建议不要超过1个迭代，遇到紧急情况走hotfix版本修复。
- 每个月底组织各组leader召开一次复盘会，梳理当前进展及未来的计划，各组分享慢SQL优化经验，互相借鉴，沉淀技术文档，避免以后踩同样的坑。

3、单库迁移到分片库

亿级大表单库读写能力弱，当业务量突增，系统风险很高，库容易被打挂，所以互联网公司用的比较多的方案是采用分片库代替单库应对几倍甚至几十倍的突发流量，使用分片库以后，我们需要做哪些工作呢?

- 数据迁移

基于binlog，通过canal+kafka+数据处理应用，完成亿级大表单库到分片库的平滑迁移。

- 分库后的数据查询、插入、更新、删除

  分库中间件架构选型：市面上用的比较多的分库分表中间件：mycat和shardingsphere，从成本、性能、稳定性、可维护性这几个方面综合考虑，我们优先选择shardingsphere。

  应用接入：目前Java语言开发的应用大部分都是基于springboot开发，集成shardingsphere很方便，网上有很多集成示例，这里不过多介绍。

4、夯实底盘监控与异常实时告警

- 云数据库所在的主机监控告警

  网络IO、磁盘

- 云数据库自身各项指标监控告警

  cpu、内存使用率、磁盘、连接数、平均RT

- 慢SQL监控
- 分片库热点数据倾斜监控告警

比如：大客户下单，某个大客户1次推送2w+订单，以客户custId为分片键，2w+QPS，瞬间将单库写爆，

所以一定要做好热点数据写入或更新的底盘监控，第一时间发现。

四、扩展高频面试题与分析

问题1：假设白天业务高峰期出现mysql cpu告警，但是没有慢SQL，那我们的解决方案是?

答案：通过监控看到数据库的连接数较平时大量增加，但是没有明显的慢SQL，这个时候大概率是我们的业务流量突增，遇到这种情况，我们第一时间要找运维对业务的数据库实例进行在线平滑扩容（目前大公司的数据库或中间件基础都是部署在云上，基于k8s进行扩容几乎秒级生效），例如：数据库的硬件配置从2c 4G扩容到4c 8G。

问题2: 假设上了分片库以后，有大客户单次写入TPS几w+订单，那我们如何保证系统不被打垮呢?

答案： 1）上层入口做好限流 2）数据层增加拦截器，单次写入或更新数据量超过1000条，改为分批次提交事务

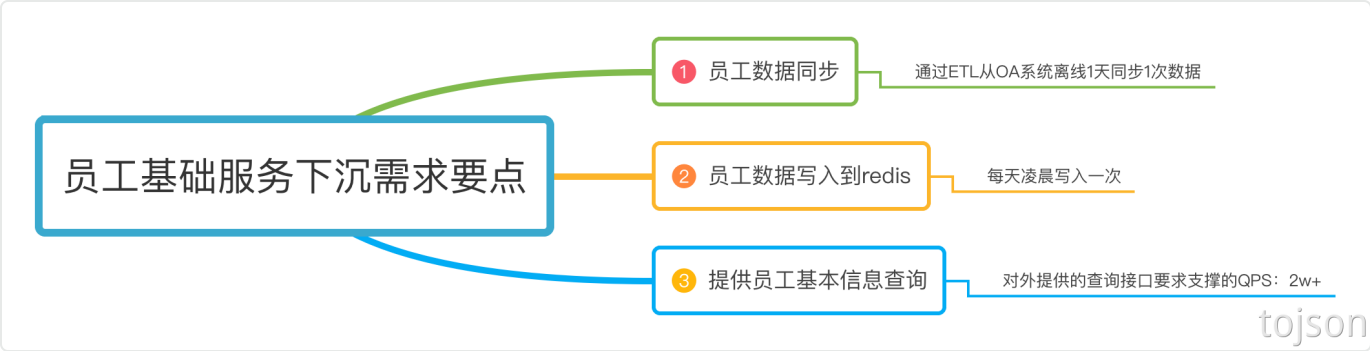
3）底盘提供热点数据写入或更新的监控告警。

五、真实案例代码落地实现

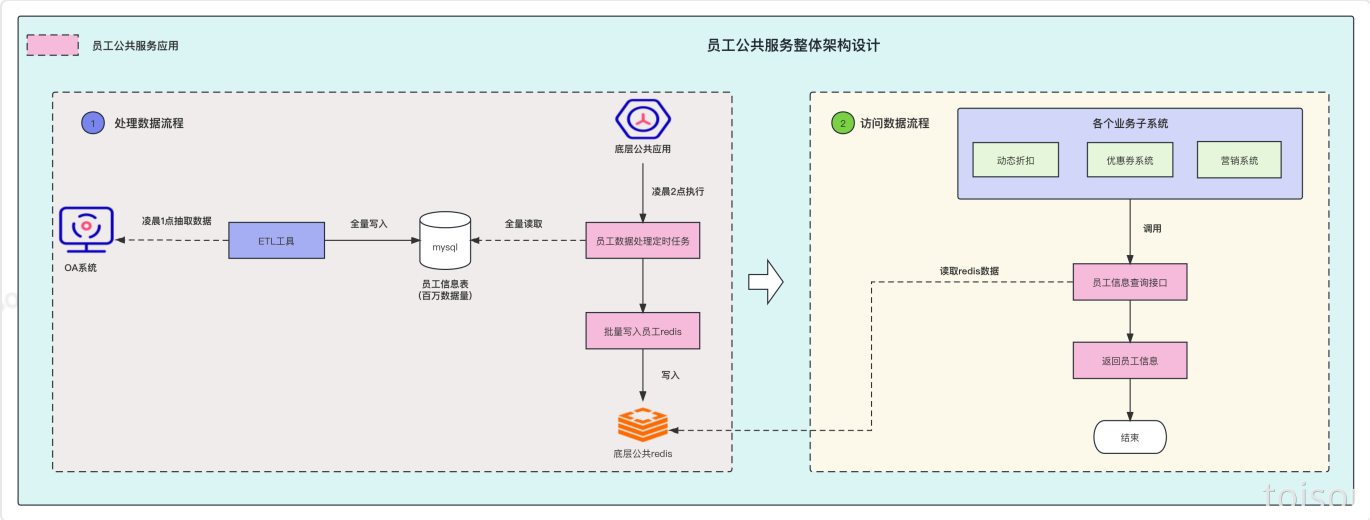
1、业务背景

目前我们公司的线上业务，比如：动态折扣、优惠券、营销活动等，经常需要查询员工的基础信息，早期为了业务需求快速上线，各个业务线都是单独对接我们公司的OA系统，重复造轮子，浪费研发资源。在大环境不好的情况下，各个公司都在主推降本增效，所以员工查询服务下沉到底盘公共服务，是一件非常有意义的事项。

2、需求分析



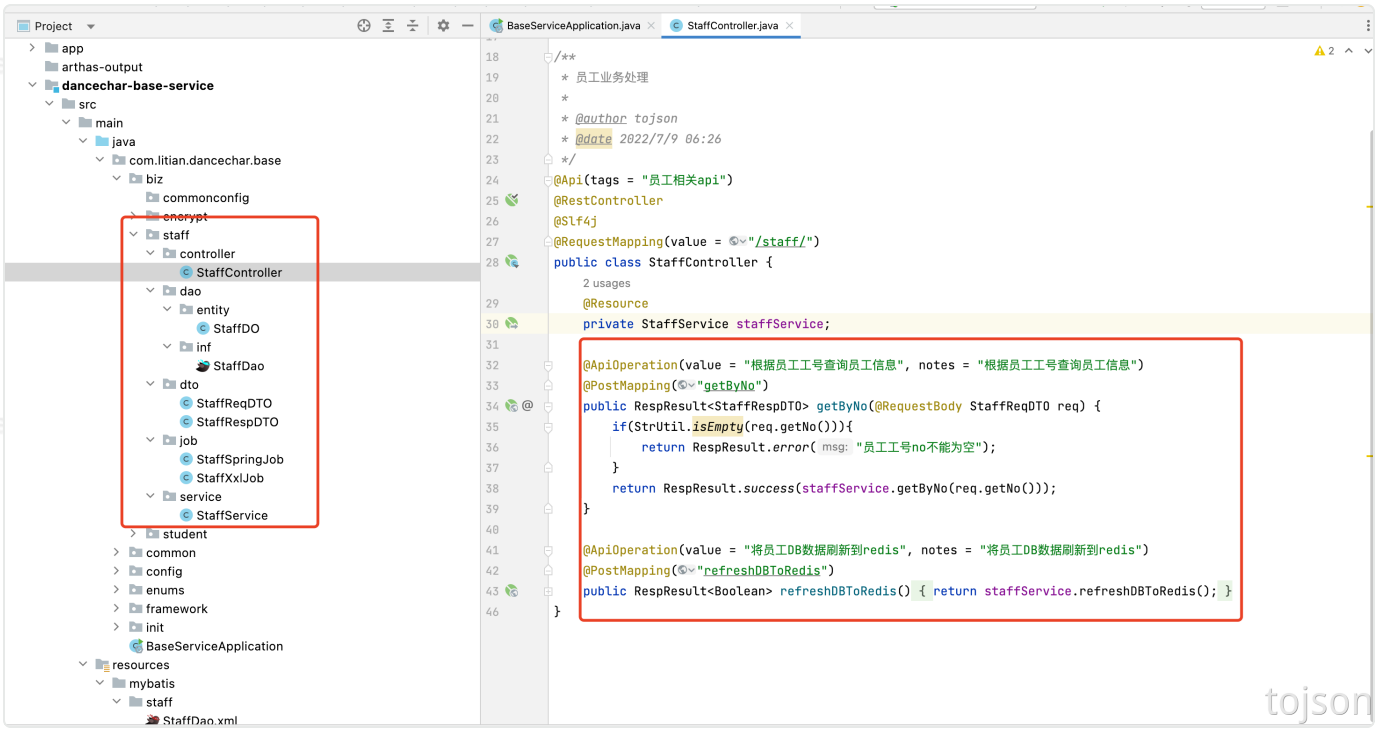
3、架构设计



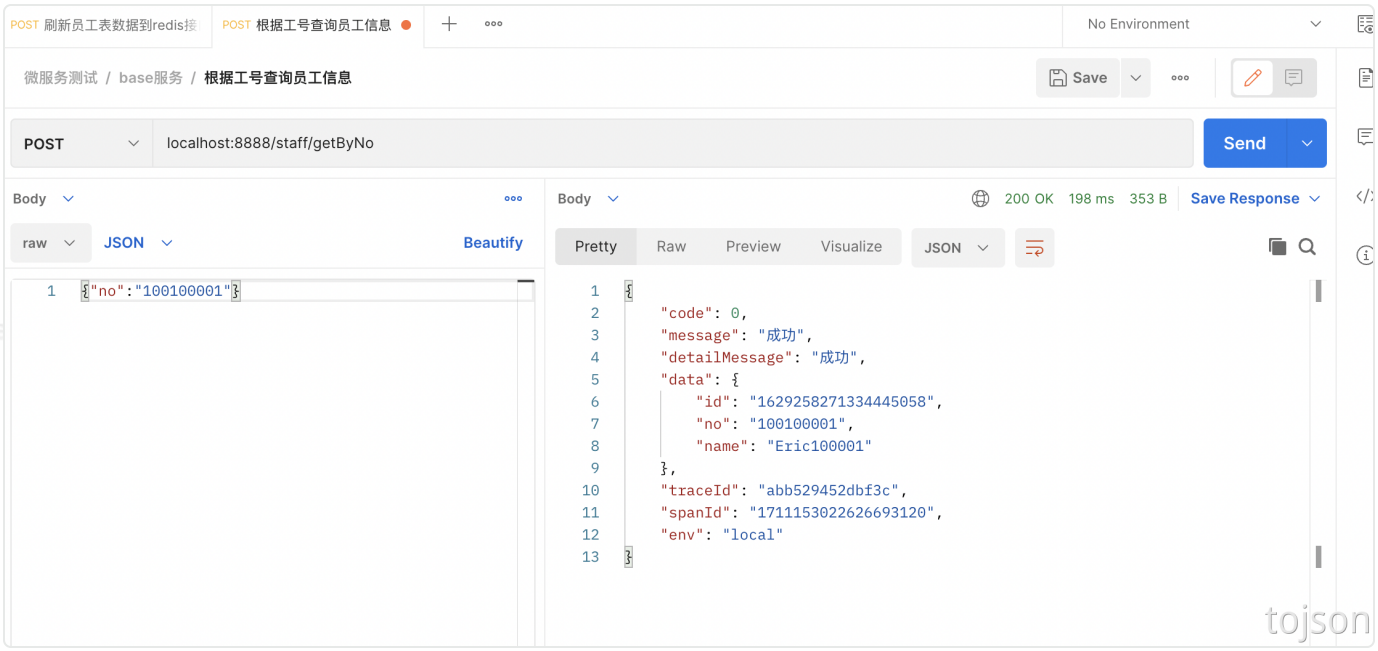
4、代码实现

应用：dancechar-base-service

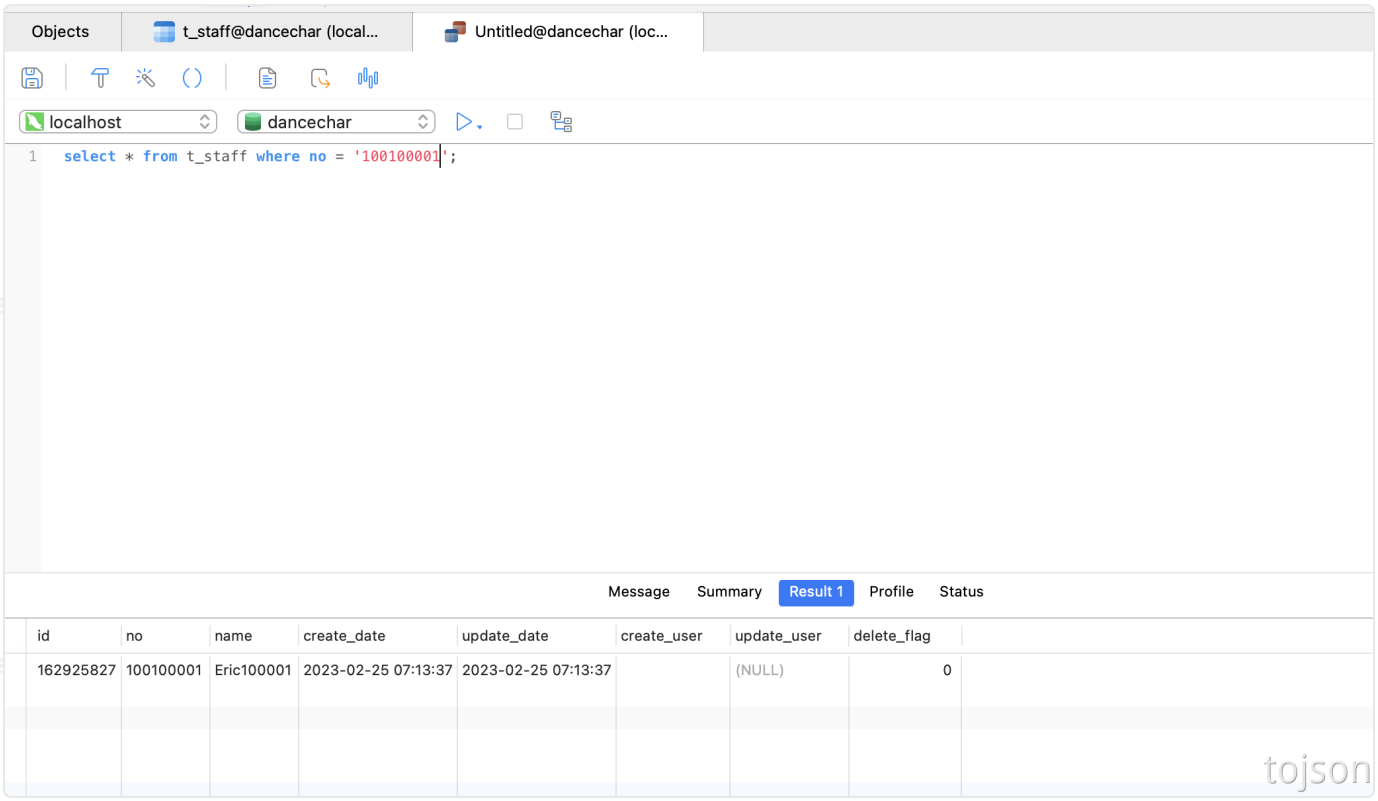
代码目录：



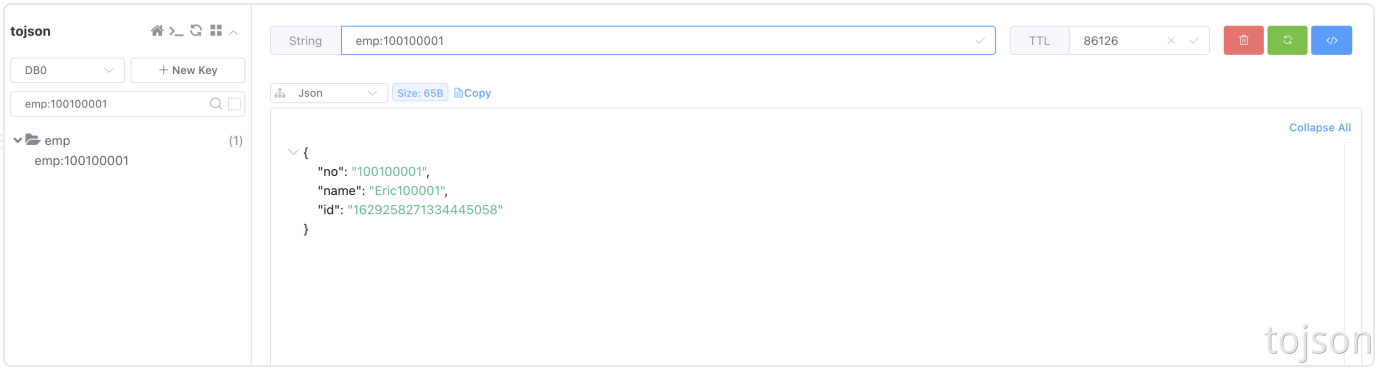
postman请求接口运行结果：



数据库数据：


Objects
t_staff@dancechar (local...
Untitled@dancechar (loc...
localhost
dancechar
1 select * from t_staff where no = '100100001';
Message
Summary
Result 1
Profile
Status
id no name create_date update_date create_user update_user delete_flag
162925827 100100001 Eric100001 2023-02-25 07:13:37 2023-02-25 07:13:37 (NULL) 0
tojson

redis数据：


tojson
DB0
New Key
emp:100100001
emp (1)
emp:100100001
String
emp:100100001
TTL
86126
Json
Size: 65B
Copy
Collapse All
{
"no": "100100001",
"name": "Eric100001",
"id": "1629258271334445058"
}
tojson