二级缓存框架

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **修订日期** | **修订人** | **修订内容** |
| 1 |  |  |  |

参考：https://www.cnblogs.com/wchxj/p/8159609.html

## 设计初衷

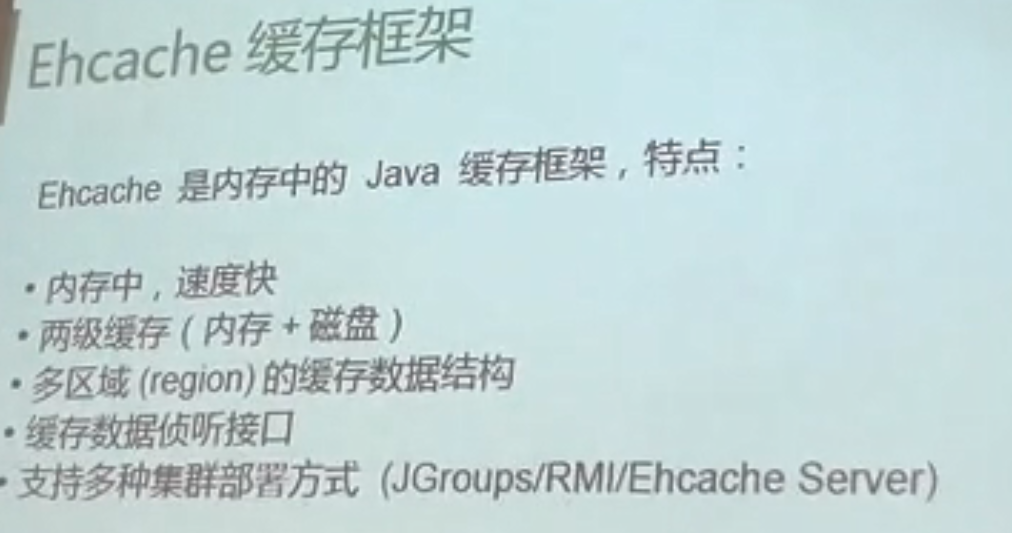
原系统（送达系统）类属性缓存缺陷：

1. 不灵活，占用大量内存空间：系统初始化时将缓存数据放入内存
2. 只能缓存小数据量（如果是全国法院的话数据量可能非常大，用户，部门等）--目前可能会遇到的。
3. 一旦服务器环境出现问题，导致内存丢失，只能通过重启服务器还原缓存。
4. 重启服务器后缓存消失，需要重新访问数据库进行缓存，如果是高访问量的情况下，数据库负担严重，无法快速读取到数据。
5. 只能永久缓存，无法灵活配置(无法设置失效时间)。

想要的缓存组件：

1. 可配置占用内存大小。
2. 只使用Ehcache缺陷：（集群节点越多，开销越大）

ehcache会随着java项目的消失而消失。

集群开销大：ehcache集群节点之间数据同步通过组播的方式，节点间大量的数据复制带来额外的开销，在节点多的情况下此问题越发严重。  
  


1. 只使用redis缺陷：（访问量越大，开销越大）

通过网络传输向其他节点提供数据。

访问量增大，大量的缓存数据访问使得应用服务器和缓存服务器之间的网络I/O消耗越大。

## 思路：

J2Cache 的两级缓存结构

L1： 进程内缓存(caffeine\ehcache)

L2： Redis/Memcached 集中式缓存

在项目启动时

先在一级缓存ehcache中获取，如果没有缓存，则在访问redis获取。

使用第三方缓存组件：

<https://gitee.com/ld/J2Cache>

## 应用场景：

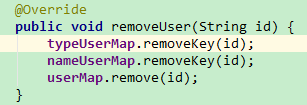
1、根据type获取 Map<String,SysUser>



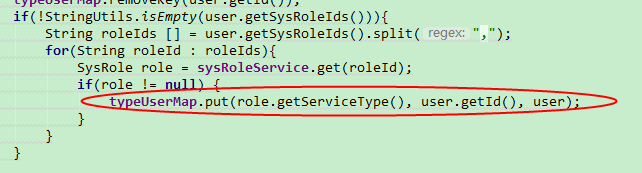
2、获取所有type，不能有重复 （外部实现 ---- 直接缓存所有type ）



3、移除每个type组中的key



4、将user添加进每个type组中



## 初始设计：

#### 第一阶段：

1. 数据读取
2. 数据更新

实现频繁查询缓存（与数据库查询相结合） --需要考虑的：频繁更新导致的缓存更新。

#### 第二阶段：（已实现）

1. 树形结构的缓存（最后存储的是字符串或者实体类—属性为基本数据类型 ） -- 有没有必要？？

--难点：某一层级，成员的获取。（要通过遍历key）

###### 实现方式：

TODO 通过构造key （group:group….:key ） 来实现层级关系

###### 缺陷：

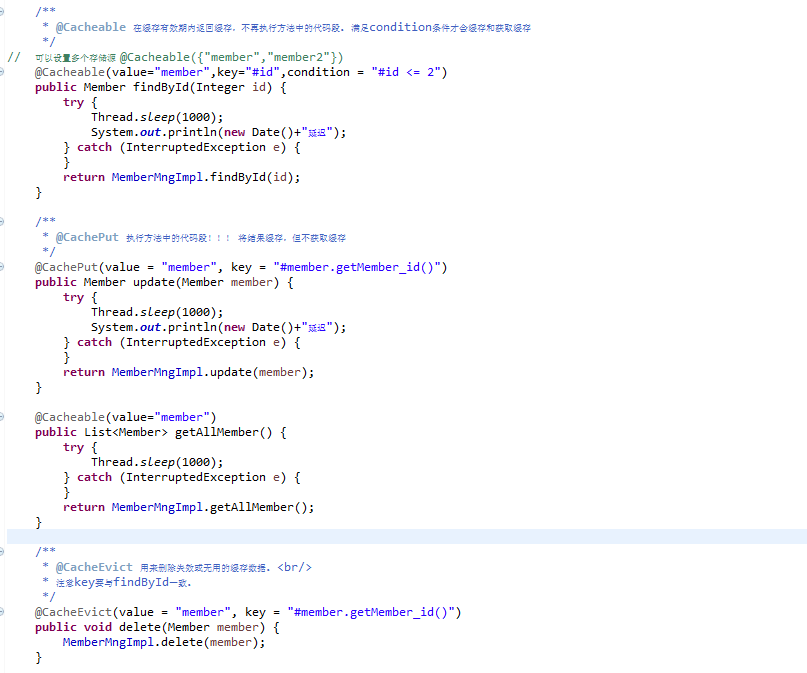
1、获取整个组时需要先获取所有的key，再遍历判断是否是改组成员。

Redis获取keys时性能低下

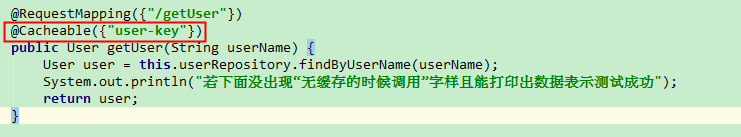
暂时没有更好的解决方案。谨慎使用。

#### 第三阶段：

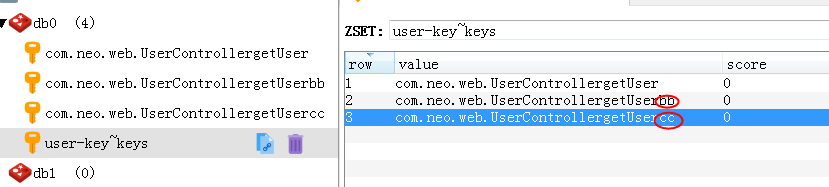
切点式查询和缓存（或者和数据库框架的整合）



**可参考spring boot的redis集成**



Redis中的存储：

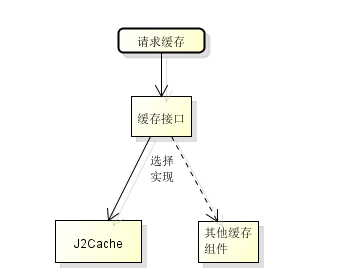


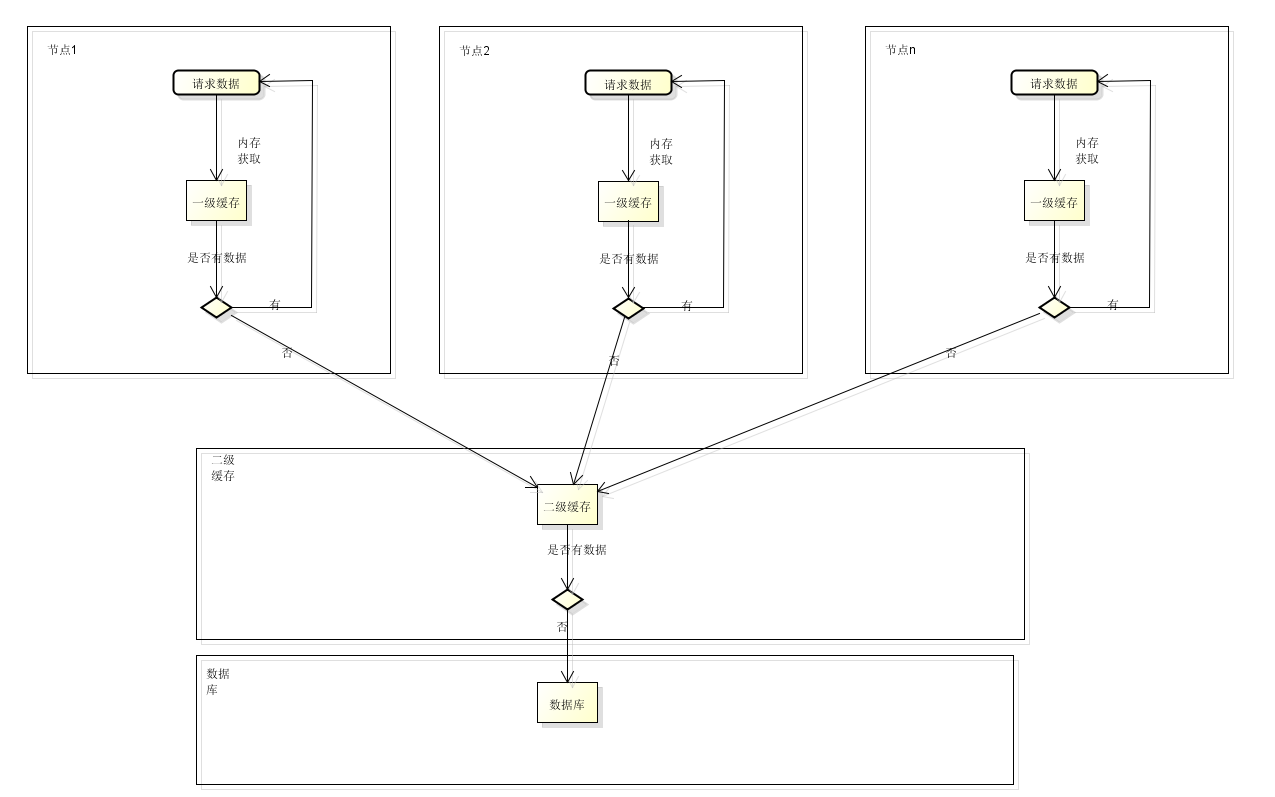
#### 第四阶段：

5、缓存规则定义（提高命中率） --难点：使用场景的把握

### 缓存组件层级设计：

初衷：可以更换缓存实现，而不用更改业务代码。





## 遇到问题：

#### redis挂掉了保证项目正常运行

## 技术点

1. ehcache
2. redis

## 功能点

## 关键类