链滴

AutoLoadCache 使用以及规范

作者：TWanGT

原文链接：https://ld246.com/article/1591717050212

来源网站：链滴

简介

AutoLoadCache 是基于AOP+Annotation等技术实现的高效的缓存管理解决方案，实现缓存与业务辑的解耦，并增加异步刷新及“拿来主义机制”，以适应高并发环境下的使用。

使用AOP + Annotation 来解决这个问题，同时使用自动加载机制 来实现数据“常驻内存”

---> 跳转官方git

如何使用

本文的自动缓存使用的redis作为缓存技术, 请确保自己系统中的redis能够正常使用

引入jar包

```
<!-- autoload-cache 依赖 -->
<dependency>
<groupId>com.github.qiujiayu</groupId>
<artifactId>autoload-cache-spring-boot-starter</artifactId>
<version>7.0.8</version>
</dependency>
<!-- aop 依赖 -->
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

配置类

```
package  com.xxx

import com.jarvis.cache.ICacheManager;
import com.jarvis.cache.autoconfigure.AutoloadCacheProperties;
import com.jarvis.cache.clone.ICloner;
import com.jarvis.cache.map.MapCacheManager;
import com.jarvis.cache.serializer.FastjsonSerializer;
import com.jarvis.cache.serializer.ISerializer;
import org.springframework.context.annotation.Bean;

/**
* @Author: WanG
* @Date: 2020/6/9 10:32
* @version: v1.0
* @description: 自动缓存框架配置
*/
public  class  AutoloadCacheConfiguration {
// @Bean
public  ICacheManager  mapCacheManager(AutoloadCacheProperties  config, ICloner  cloner)
{
return  new  MapCacheManager(config.getConfig(), cloner);
}
```

```
@Bean
public ISerializer<Object> autoloadCacheSerializer() {
return new FastjsonSerializer();
}
}
```

注解

Annotation: 官方例子

参数说明

- #retVal 数组类型统一用这个取值, 或者用为取返回值
- #args 按坐标取传入参数
- #hash 使用 hash函数
- #empty 使用 empty判断
- expireExpression 过期时间表达式
- condition 执行条件, 满足改条件才生效(用于修改和删除数据的时候连带删除缓存)
- @ExCache 额外生成关联缓存
- @CacheDelete 删除缓存注解
- @CacheDeleteKey 生成删除缓存Key注解
- @CacheDeleteTransactional 事务环境中批量删除缓存注解
- @LocalCache 本地缓存注解

缓存数据

在需要进行数据缓存的地方加上@Cache注解即可

```
@Cache(expire = "{过期时间(这个是数字类型)}", key = "{缓存使用的key}")
```

清理缓存

在需要清理缓存的地方加上@CacheDelete 注解

```
@CacheDelete({ @CacheDeleteKey(value = "{缓存使用的key}") })
```

动态过期时间

当返回值为空时缓存有效期为20分钟, 当返回值不为空有效期为60分钟

```
@Cache(expire = EXPIRE_TIME, key = CACHE_KEY_INITUSERINFO, expireExpression = "null == #retVal ? 20: 60")
```

动态

当删除语句执行成功, 才将对应key的缓存删除

```
@CacheDelete({ @CacheDeleteKey(value = "'user-byid-' + #args[0]", condition = "#retVal > 0") })

int  deleteUserById(Long id);
```

动态取值

注解里面可以用#args[i]来动态获取入参

```
@Cache(expire = 60, expireExpression = "null == #retVal ? 30: 60", key = "'user-byid-' + args[0]")

UserDO  getUserById(Long id);
```

使用规范

1. 缓存 key用:作为分隔符
2. 缓存纯数据库字段 key要加上table关键字
3. 缓存方法数据 key要加上method关键字
4. 对数据进行 修改或者删除时务必同步将缓存删除(带上condition, 操作成功才动缓存)
5. 方法缓存 key中带上类名和方法名(用.隔开)
6. key中带上缓存的入参, 并用=连接参数名和参数值
7. 多个入参用 ,分隔

举例

```
## 缓存key值:

user-server:method:+CacheServiceImpl.initUserInfo:personId=1,homeId=1
```

代码样例

```
public static final String CACHE_KEY_INITUSERINFO = CacheConfig.KEY_METHOD + "CacheServiceImpl.initUserInfo:personId='+#args[0]+',homeId='+#args[1]";
```

~~~~
~~~~