SpringCloud配置缓存

一、开启缓存技术

在程序的入口中加入@ EnableCaching开启缓存技术：

```
@SpringBootApplication
@EnableCaching
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

二、在需要缓存的地方加入@Cacheable注解

在需要缓存的地方加入@Cacheable注解，比如在getByIsbn（ ）方法上加入@Cacheable(“books”)，这个方法就开启了缓存策略，当缓存有这个数据的时候，会直接返回数据，不会等待去查询数据库。

```
@Component
public class SimpleBookRepository implements BookRepository {
    @Override
    @Cacheable("books")
    public Book getByIsbn(String isbn) {
        simulateSlowService();
        return new Book(isbn, "Some book");
    }

    // Don't do this at home
    private void simulateSlowService() {
        try {
            long time = 3000L;
            Thread.sleep(time);
        } catch (InterruptedException e) {
            throw new IllegalStateException(e);
        }
    }
}
```

三、分布式缓存更新策略

第一条原则：

先更新DB，再删除缓存（推荐）

> 我最推荐的方案：先更新DB，再删除缓存。
>
> 先更新DB，在还没有删除的这段时间内，即使有操作读取到了缓存中的脏数据，那也只是一时的，因为随后肯定会对缓存进行删除。
>
> 该方案在没有缓存过期时间的情况下，应该是最优的方案了。
>
> 其实该方案同样存在问题：
>
> 1. A线程读取到老数据，准备建立缓存。
> 2. B线程更新DB
> 3. B线程删除缓存。
> 4. A线程开始建立缓存（老数据的缓存）
>
> 因为写数据操作比缓存建立耗时，所以该情况发生的概率很小。
>
> 为了保证缓存的一致性，该方案只需要保证删除缓存成功即可：
>
> 1. 通过Retry不断尝试进行缓存删除，直到删除成功为知。
> 2. 将删除操作放到MQ中，让消费者消费，直到成功为止。

第二条原则：

给缓存设置过期时间

> 为了保证数据的一致性，不论何种方式处理缓存，都应该给缓存设置过期时间，这个是缓存必须的要素，否则数据的最终一致性很难得到保证。
>
> 下述讨论的讨论方案中，是在没有设置缓存过期时间 的情况下的极端讨论，仅仅为了理清思路，实际开发过程中，都应该给缓存加上过期时间。

第三条原则：

对于数据变化，不应该同步更新缓存

----------只有数据被读取的时候才建立缓存----------

> 先下结论：对于数据变化，不应该同步更新缓存。
>
> 因为：只有被查询的数据的数据建立缓存，才有意义。
>
> 一个数据只会被更新，长期或者永远不会被查询，建立缓存就是浪费资源。
>
> 综上：建立缓存的操作应该是在数据被读取的时候。

结论

1. 缓存必须加上过期时间，防止偶然的脏数据。
2. 数据发生变更时候，不应该是重新建立缓存，而应该是直接删除缓存。
3. 缓存删除应该在DB更新后进行。

四、重点讲解缓存注解

> @Cacheable、@CacheEvict、@CachePut使用及注解失效时间

4.1、@Cacheable

@Cacheable可以标记在一个方法上，也可以标记在一个类上。当标记在一个方法上时表示该方法是支持缓存的，当标记在一个类上时则表示该类所有的方法都是支持缓存的。对于一个支持缓存的方法，Spring会在其被调用后将其返回值缓存起来，以保证下次利用同样的参数来执行该方法时可以直接从缓存中获取结果，而不需要再次执行该方法。Spring在缓存方法的返回值时是以键值对进行缓存的，值就是方法的返回结果，至于键的话，Spring又支持两种策略，默认策略和自定义策略，这个稍后会进行说明。需要注意的是当一个支持缓存的方法在对象内部被调用时是不会触发缓存功能的。@Cacheable可以指定三个属性，value、key和condition。

| 参数 | 解释 | example | | --- | --- | --- | | value | 缓存的名称，在 spring 配置文件中定义，必须指定至少一个 | 例如: @Cacheable(value=”mycache”) @Cacheable(value={”cache1”,”cache2”} | | key | 缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合 | @Cacheable(value=”testcache”,key=”#userName”) | | condition | 缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存 | @Cacheable(value=”testcache”,condition=”#userName.length()>2”) |

1.1 value属性指定Cache名称 value属性是必须指定的，其表示当前方法的返回值是会被缓存在哪个Cache上的，对应Cache的名称。其可以是一个Cache也可以是多个Cache，当需要指定多个Cache时其是一个数组。

```
@Cacheable("cache1")//Cache是发生在cache1上的
public User find(Integer id) {
 return null;
}

```

```
@Cacheable({"cache1", "cache2"})//Cache是发生在cache1和cache2上的
public User find(Integer id) {
   return null;
}
```

1.2 使用key属性自定义key key属性是用来指定Spring缓存方法的返回结果时对应的key的。该属性支持SpringEL表达式。当我们没有指定该属性时，Spring将使用默认策略生成key。我们这里先来看看自定义策略，至于默认策略会在后文单独介绍。

```
自定义策略是指我们可以通过Spring的EL表达式来指定我们的key。这里的EL表达式可以使用方法参数及它们对应的属性。使用方法参数时我们可以直接使用“#参数名”或者“#p参数index”。下面是几个使用参数作为key的示例。
```

```
@Cacheable(value="users", key="#id")
public User find(Integer id) {
    return null;
}
```

```java
//取第一个参数id--paramter
@Cacheable(value="users", key="#p0")
public User find(Integer id) {
    return null;
}

//key取第二个参数code
@Cacheable(value="users", key="#p1")
public User find(String name, String code) {
    return null;
}
```

```java
@Cacheable(value="users", key="#user.id")
public User find(User user) {
    return null;
}
```

```java
//取第一个参数user.id
@Cacheable(value="users", key="#p0.id")
public User find(User user) {
    return null;
}
```

- **@Cacheable 拼接key**
- 分页查询用户信息【value=page_user，key=pageNumber_pageSize】

```java
@Cacheable(value = "page_user",key ="T(String).valueOf(#pageNumber).concat('-
').concat(#pageSize)",unless = "#result=null")
//由于pageNumber是int型,concat要求变量必须为String,所以强转一下
@Override
public List<SysUserEntity> page(int pageNumber, int pageSize) {
    return userMapper.page(pageNumber,pageSize);
}
```

- 如果不指定key值的情况，可以使用**keyGenerator：**

```
 * The class Redis configuration.
 *
 * @author hucais.com@gmail.com
 */
@Configuration
@EnableCaching
public class RedisConfiguration {
    /**
     * Wisely key generator key generator.
     *
     * @return the key generator
     */
    @Bean
    public KeyGenerator keyGenerator() {
        return (target, method, params) -> {
            StringBuilder sb = new StringBuilder();
            sb.append(target.getClass().getName());
            sb.append(method.getName());
            for (Object obj : params) {
                sb.append(obj.toString());
            }
            return sb.toString();
        };

    }

```

```
@Bean
public KeyGenerator keyGenerator() {
    return (target, method, params) -> {
        StringBuilder sb = new StringBuilder();
        sb.append(target.getClass().getName());
        sb.append(method.getName());
        for (Object obj : params) {
            sb.append(obj.toString());
        }
        return sb.toString();
    };

}

```

- 使用默认keyGenerator生成策略

```
@Cacheable(value = "usercache", keyGenerator="keyGenerator")
public User getUser(String no,String name){
    LogCore.BASE.debug("invoke persistent:{},{}", no, name);
```

```
    return new User(no, name);
}
/**
 * allEntries=true 是立即清除所有缓存
 */
@CacheEvict(value = "usercache", keyGenerator="keyGenerator")
public boolean clearUser(String no,String name){
    LogCore.BASE.debug("invoke clear:{},{}", no, name);
    return true;
}
@CachePut(value = "usercache", keyGenerator="keyGenerator")
public User putUser(String no, String name){
    User usr = new User(no, name);
    return usr;
}
```

1.3 conditions和unless

- SPEL表达式：

要想控制加入缓存的条件，可以使用SPEL表达式，可以用conditions和unless前者是对传入的参数进行筛选，后者可以对返回值进行筛选

例如：

```
@Cacheable(value = "usercache", keyGenerator="keyGenerator",condition="#name!=null", unless="#result==null")
public User getUser(String no, String name) {
    LogCore.BASE.debug("invoke getUser:{},{}", no, name);
    if (Util.anyNonEmpty(no, name)) {
        return new User(no, name);
    }
    return null;
}
```

4.2、 @CacheEvict

CacheEvict:指明某个方法或者某个类的所有方法触发清除缓存。注意缓存中加上allEntries=true使其立即清除所有缓存。

- @CacheEvict 作用和配置方法

参数	解释	example
value	缓存的名称，在 spring 配置文件中定义，必须指定至少一个	@CacheEvict(value="my cache")
key	缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合	@CacheEvict(value="testcache",key="#userName")
condition	缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存	@CacheEvict(value="testcache",condition="#userName.length()>2")
allEntries	是否清空所有缓存内容，缺省为 false，如果指定为 true，则方法调用后将立即清空所有缓存	@CachEvict(value="testcache",allEntries=true)
beforeInvocation	是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存	@CachEvict(value="testcache"，beforeInvocation=true)

```
@CacheEvict(value="accountCache",key="#account.getName()")// 清空accountCache 缓存
public void updateAccount(Account account) {
     updateDB(account);
}

@CacheEvict(value="accountCache",allEntries=true)// 清空accountCache 缓存
public void reload() {
     reloadAll()
}
```

- **allEntries属性** allEntries是boolean类型，表示是否需要清除缓存中的所有元素。默认为false，表示不需要。当指定了allEntries为true时，Spring Cache将忽略指定的key。有的时候我们需要Cache一下清除所有的元素，这比一个一个清除元素更有效率。

```
@CacheEvict(value="users", allEntries=true)
public void delete(Integer id) {
    System.out.println("delete user by id: " + id);
}
```

- **beforeInvocation属性** 清除操作默认是在对应方法成功执行之后触发的，即方法如果因为抛出异常而未能成功返回时也不会触发清除操作。使用beforeInvocation可以改变触发清除操作的时间，当我们指定该属性值为true时，Spring会在调用该方法之前清除缓存中的指定元素。

```
//不推荐这种策略，推荐策略：先删除数据库，再删除缓存
@CacheEvict(value="users", beforeInvocation=true)
public void delete(Integer id) {
 System.out.println("delete user by id: " + id);
}
```

4.3、@CachePut【静止使用】

CachePut: 指明某个方法或者某个类的所有方法触发添加缓存。与Cacheable相反，此注解总是触发目标方法，并添加到缓存

```
    除了上述使用方法参数作为key之外，Spring还为我们提供了一个root对象可以用来生成key。通过该root对象我们可以获取到以下信息。
```

4.4、@CacheConfig

所有的@Cacheable（ ）里面都有一个value = “xxx”的属性，这显然如果方法多了，写起来也是挺累的，如果可以一次性声明完 那就省事了， 所以，有了@CacheConfig这个配置，@CacheConfig is a class-level annotation that allows to share the cache names，如果你在你的方法写别的名字，那么依然以方法的名字为准。

```
@CacheConfig("books")
public class BookRepositoryImpl implements BookRepository {
    @Cacheable
    public Book findBook(ISBN isbn) {...}
}
```

4.4、注解失效时间

```
//指定cacheNameKey的缓存45秒后过期
@Cacheable(cacheNames = "cacheNameKey#45", key = "#key")
public String testRedisCache(String key) {
    System.out.println(key);
    return "12345";
}
```

4.5、SpEL上下文数据

Spring Cache提供了一些供我们使用的SpEL上下文数据，下表直接摘自Spring官方文档：

名称	位置	描述	示例
methodName	root对象	当前被调用的方法名	root.methodName
method	root对象	当前被调用的方法	root.method.name
target	root对象	当前被调用的目标对象	root.target
targetClass	root对象	当前被调用的目标对象类	root.targetClass
args	root对象	当前被调用的方法的参数列表	root.args[0]
caches	root对象	当前方法调用使用的缓存列表（如@Cacheable(value={"cache1", "cache2"})），则有两个cache	root.caches[0].name
argument name	执行上下文	当前被调用的方法的参数，如findById(Long id)，我们可以通过#id拿到参数	user.id
result	执行上下文	方法执行后的返回值（仅当方法执行之后的判断有效，如'unless'，'cache evict'的beforeInvocation=false）	result

```
@CacheEvict(value = "user", key = "#user.id", condition = "#root.target.canCache() and #root.caches[0].get(#user.id).get().username ne #user.username", beforeInvocation = true)
public void conditionUpdate(User user)
```

- **Service层应用缓存（注解方式）**

```
@Service
public class PersonService {

    @Autowired
    private PersonRepo personRepo;

   /**
     * @Cacheable 应用到读取数据的方法上，先从缓存中读取，如果没有再从DB获取数据，然后把数据添加到缓存中
    * unless 表示条件表达式成立的话不放入缓存
     * @param username
     * @return
     */
    @Cacheable(value = "user", key = "#root.targetClass + #username", unless = "#result eq null")
    public Person getPersonByName(String username) {
        Person person = personRepo.getPersonByName(username);
        return person;
    }

   /**
    * @CachePut 应用到写数据的方法上，如新增/修改方法，调用方法时会自动把相应的数据放入缓存
     * @param person
     * @return
```

```
     */
    @CachePut(value = "user", key = "#root.targetClass + #result.username", unless = "#person eq null")
    public Person savePerson(Person person) {
        return personRepo.savePerson(person);
    }

   /**
    * @CacheEvict 应用到删除数据的方法上，调用方法时会从缓存中删除对应key的数据
     * @param username
     * @return
     */
    @CacheEvict(value = "user", key = "#root.targetClass + #username", condition = "#result eq true")
    public boolean removePersonByName(String username) {
        return personRepo.removePersonByName(username) > 0;
    }

    public boolean isExistPersonName(Person person) {
        return personRepo.existPersonName(person) > 0;
    }
}
```

五、springCloud controller自定义调用redis缓存，并设置过期时间

在其他要使用redis的类中，加入

```
@Autowired
protected RedisTemplate redisTemplate;
```

使用示例：

```
package com.ppl.server.controller;

import java.util.concurrent.TimeUnit;

import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;


@RestController
public class ComputeController {
    private final Logger logger = Logger.getLogger(getClass());

    @Autowired
```

```
28      protected RedisTemplate redisTemplate;
29
30      @RequestMapping(value = "/add" ,method = RequestMethod.GET)
31      public Integer add(@RequestParam Integer a, @RequestParam Integer b) {
32
33          Integer r = a + b;
34          logger.info("/add, result:" + r);
35          return r;
36      }
37
38      @GetMapping("/hello")
39      public String sayHi(@RequestParam String name){
40
41          //从缓存中获取城市信息
42          String key = name;
43          ValueOperations<String, String> operations = redisTemplate.opsForValue();
44          //缓存存在
45          boolean hasKey = redisTemplate.hasKey(key);
46          if (hasKey) {
47              String userName = operations.get(key);
48              logger.info(" 从缓存中获取了姓名 >> " + userName);
49              return userName;
50          }
51          //缓存不存在，将数据存入缓存
52          operations.set(key, name, 10, TimeUnit.SECONDS);
53          logger.info("CityServiceImpl.findCityById() : 城市插入缓存 >>  " +name);
54
55          return " Welcome: "+name;
56      }
57  }
```

六、实例案例：门店分布统计报表

- 全部门店数量
- 今日门店新增门店数量
- 新增门店
- 修改门店
-