芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2018-11-21
Dubbo

精尽 Dubbo 源码分析 —— 过滤器（十）之 CacheFilter

> 本文基于 Dubbo 2.6.1 版本，望知悉。

1. 概述

本文分享 dubbo-filter-cache 项目的 CacheFilter 过滤器，用于服务消费者和提供者中，提供 结果缓存 的功能。在 《Dubbo 用户指南 —— 结果缓存》 定义如下：

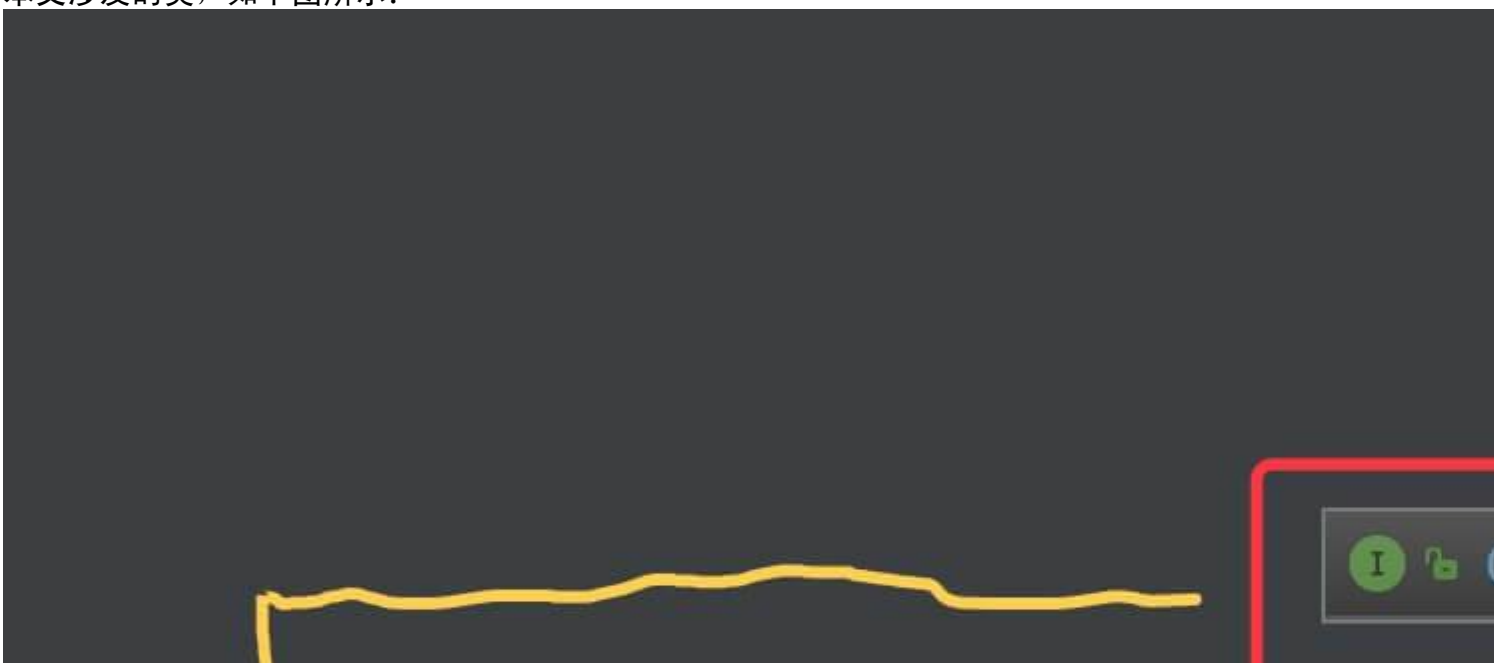
> 结果缓存 ，用于加速热门数据的访问速度，Dubbo 提供声明式缓存，以减少用户加缓存的工作量。

Dubbo 提供了三种实现：

> lru ： 基于最近最少使用原则删除多余缓存，保持最热的数据被缓存。
> threadlocal ： 当前线程缓存，比如一个页面渲染，用到很多 portal，每个 portal 都要去查用户信息，通过线程缓存，可以减少这种多余访问。
> jcache ： 与 JSR107 集成，可以桥接各种缓存实现。

> 具体的配置方式，在 《Dubbo 用户指南 —— 结果缓存》 文档中，已经详细分享。

本文涉及的类，如下图所示：



2. CacheFilter

com.alibaba.dubbo.cache.filter.CacheFilter ，实现 Filter 接口，缓存过滤器实现类。代码如下：

```
 1: @Activate(group = {Constants.CONSUMER, Constants.PROVIDER}, value = Constants.CACHE_KEY)
 2: public class CacheFilter implements Filter {
 3:
 4:     /**
 5:      * CacheFactory$Adaptive 对象。
 6:      *
 7:      * 通过 Dubbo SPI 机制，调用 {@link #setCacheFactory(CacheFactory)} 方法，进行注入
 8:      */
 9:     private CacheFactory cacheFactory;
10:
11:     public void setCacheFactory(CacheFactory cacheFactory) {
12:         this.cacheFactory = cacheFactory;
13:     }
14:
15:     @Override
16:     public Result invoke(Invoker<?> invoker, Invocation invocation) throws RpcException {
17:         // 方法开启 Cache 功能
18:         if (cacheFactory != null && ConfigUtils.isNotEmpty(invoker.getUrl().getMethodParameter(invocation.getMeth
19:             // 基于 URL + Method 为维度，获得 Cache 对象。
20:             Cache cache = cacheFactory.getCache(invoker.getUrl().addParameter(Constants.METHOD_KEY, invocation.ge
21:             if (cache != null) {
22:                 // 获得 Cache Key
23:                 String key = StringUtils.toArgumentString(invocation.getArguments());
24:                 // 从缓存中获得结果。若存在，创建 RpcResult 对象。
25:                 Object value = cache.get(key);
26:                 if (value != null) {
27:                     return new RpcResult(value);
28:                 }
29:                 // 服务调用
30:                 Result result = invoker.invoke(invocation);
31:                 // 若非异常结果，缓存结果
32:                 if (!result.hasException()) {
33:                     cache.put(key, result.getValue());
34:                 }
35:                 return result;
36:             }
37:         }
38:         // 服务调用
39:         return invoker.invoke(invocation);
40:     }
41:
42: }
```

第 18 行：判断方法开启 Cache 功能。因为，一个服务里，可能只有部分方法开启了 Cache 功能。
第 20 行：调用 CacheFactory$Adaptive#getCache(url) 方法，基于 URL + Method 为维度，获得 Cache 对象。

第 23 行：调用 StringUtils#toArgumentString(Object[] args) 方法，获得 Cache Key 。代码如下：

```
    /**
```

```
     * 将参数数组，拼接成字符串。
     *
     * 1. 使用逗号分隔
     * 2. 使用 JSON 格式化对象
     *
     * @param args 参数数组
     * @return 字符串
     */
    public static String toArgumentString(Object[] args) {
        StringBuilder buf = new StringBuilder();
        for (Object arg : args) {
            if (buf.length() > 0) {
                buf.append(Constants.COMMA_SEPARATOR); // 分隔
            }
            // 拼接参数
            if (arg == null || ReflectUtils.isPrimitives(arg.getClass())) {
                buf.append(arg);
            } else {
                try {
                    buf.append(JSON.toJSONString(arg)); // 使用 JSON 格式化对象
                } catch (Exception e) {
                    logger.warn(e.getMessage(), e);
                    buf.append(arg);
                }
            }
        }
        return buf.toString();
    }
```

第 24 至 28 行：调用 `Cache#get(key)` 方法，从缓存中获得结果。若存在，创建 RpcResult 对象并返回。

第 30 行：调用 `Invoker#invoke(invocation)` 方法，服务调用。
第 31 至 34 行：若非异常，调用 `Cache#put(key, value)` 方法，缓存正常的结果。
第 35 行：返回调用结果。
第 39 行：若不使用 Cache 功能，直接调用 `Invoker#invoke(invocation)` 方法，服务调用。

3. API 定义

3.1 Cache

`com.alibaba.dubbo.cache.Cache` ，缓存容器接口。方法如下：

```
public interface Cache {

    /**
     * 添加键值
     *
     * @param key 键
     * @param value 值
     */
    void put(Object key, Object value);

    /**
```

```
     * 获得值
     *
     * @param key 键
     * @return 值
     */
    Object get(Object key);

}
```

Cache 是个缓存容器，内部可以管理缓存的键值。

3.2 CacheFactory

com.alibaba.dubbo.cache.CacheFactory ，Cache 工厂接口。方法如下：

```
@SPI("lru")
public interface CacheFactory {

    /**
     * 获得缓存对象
     *
     * @param url URL 对象
     * @return 缓存对象
     */
    @Adaptive("cache")
    Cache getCache(URL url);

}
```

@SPI("lru") 注解，Dubbo SPI 拓展点，默认为 "lru" 。
@Adaptive("cache") 注解，基于 Dubbo SPI Adaptive 机制，加载对应的 Cache 实现，使用 URL.cache 属性。

3.2.1 AbstractCacheFactory

com.alibaba.dubbo.cache.support.AbstractCacheFactory ，Cache 工厂抽象类。代码如下：

```
public abstract class AbstractCacheFactory implements CacheFactory {

    /**
     * Cache 集合
     *
     * key: URL
     */
    private final ConcurrentMap<String, Cache> caches = new ConcurrentHashMap<String, Cache>();

    @Override
    public Cache getCache(URL url) {
        // 获得 Cache 对象
        String key = url.toFullString();
        Cache cache = caches.get(key);
        // 不存在，创建 Cache 对象，并缓存
        if (cache == null) {
```

```
                caches.put(key, createCache(url));
                cache = caches.get(key);
            }
            return cache;
        }

        /**
         * 创建 Cache 对象
         *
         * @param url URL
         * @return Cache 对象
         */
        protected abstract Cache createCache(URL url);

    }
```

4. LRU 实现

lru ，基于最近最少使用原则删除多余缓存，保持最热的数据被缓存。

4.1 LruCache

com.alibaba.dubbo.cache.support.lru.LruCache ，实现 Cache 接口，代码如下：

```
    public class LruCache implements Cache {

        /**
         * 缓存集合
         */
        private final Map<Object, Object> store;

        public LruCache(URL url) {
            // `"cache.size"` 配置项，设置缓存大小
            final int max = url.getParameter("cache.size", 1000);
            // 创建 LRUCache 对象
            this.store = new LRUCache<Object, Object>(max);
        }

        @Override
        public void put(Object key, Object value) {
            store.put(key, value);
        }

        @Override
        public Object get(Object key) {
            return store.get(key);
        }

    }
```

"cache.size" 配置项，设置缓存大小。
基于 com.alibaba.dubbo.common.utils.LRUCache 实现。

4.1.1 LRUCache

com.alibaba.dubbo.common.utils.LRUCache ，实现 LinkedHashMap 类，LRU 缓存实现类。代码比较多，胖友点击链接查看。笔者说几个关键点：

构造方法，设置 LRUCache 为按访问顺序(调用get方法)的链表。代码如下：

```
public LRUCache(int maxCapacity) {
    super(16, DEFAULT_LOAD_FACTOR, true); // 最后一个参数，按访问顺序(调用get方法)的链表
    this.maxCapacity = maxCapacity;
}
```

重写 removeEldestEntry 方法返回 true 值，指定插入元素时移除最老的元素。代码如下：

```
@Override
protected boolean removeEldestEntry(java.util.Map.Entry<K, V> eldest) {
    return size() > maxCapacity;
}
```

- 根据链表中元素的顺序可以分为：按插入顺序的链表，和按访问顺序(调用get方法)的链表。默认是按插入顺序排序，如果指定按访问顺序排序，那么调用get方法后，会将这次访问的元素移至链表尾部，不断访问可以形成按访问顺序排序的链表。

lock 属性，锁。避免并发读写，导致死锁。参见 《疫苗：JAVA HashMap 的死循环》 。涉及该属性的方法示例：

```
@Override
public V put(K key, V value) {
    try {
        lock.lock();
        return super.put(key, value);
    } finally {
        lock.unlock();
    }
}

@Override
public V remove(Object key) {
    try {
        lock.lock();
        return super.remove(key);
    } finally {
        lock.unlock();
    }
}
```

4.2 LruCacheFactory

com.alibaba.dubbo.cache.support.lru.LruCacheFactory ，实现 AbstractCacheFactory 抽象类，代码如下：

```
public class LruCacheFactory extends AbstractCacheFactory {
```

```
    @Override
    protected Cache createCache(URL url) {
        return new LruCache(url);
    }

}
```

5. ThreadLocal 实现

基于 ThreadLocal ，当前线程缓存，比如一个页面渲染，用到很多 portal，每个 portal 都要去查用户信息，通过线程缓存，可以减少这种多余访问。

5.1 ThreadLocalCache

com.alibaba.dubbo.cache.support.threadlocal.ThreadLocalCache ，实现 Cache 接口，代码如下：

```
public class ThreadLocalCache implements Cache {

    private final ThreadLocal<Map<Object, Object>> store; // 线程变量

    public ThreadLocalCache(URL url) {
        this.store = new ThreadLocal<Map<Object, Object>>() {

            @Override
            protected Map<Object, Object> initialValue() {
                return new HashMap<Object, Object>();
            }

        };
    }

    @Override
    public void put(Object key, Object value) {
        store.get().put(key, value);
    }

    @Override
    public Object get(Object key) {
        return store.get().get(key);
    }

}
```

基于 ThreadLocal 实现，相当于一个线程，一个 ThreadLocalCache 对象。
ThreadLocalCache 目前没有过期或清理机制，所以需要注意。

5.2 ThreadLocalCacheFactory

com.alibaba.dubbo.cache.support.threadlocal.ThreadLocalCacheFactory ，实现 AbstractCacheFactory 抽象类，代码如下：

```
public class ThreadLocalCacheFactory extends AbstractCacheFactory {

    @Override
    protected Cache createCache(URL url) {
        return new ThreadLocalCache(url);
    }

}
```

6. JCache 实现

与 JSR107 集成，可以桥接各种缓存实现。

6.1 JCache

com.alibaba.dubbo.cache.support.jcache.JCache ，实现 Cache 接口，代码如下：

```
public class JCache implements com.alibaba.dubbo.cache.Cache {

    private final Cache<Object, Object> store;

    public JCache(URL url) {
        // 获得 Cache Key
        String method = url.getParameter(Constants.METHOD_KEY, "");
        String key = url.getAddress() + "." + url.getServiceKey() + "." + method;
        // `"jcache"` 配置项，为 Java SPI 实现的全限定类名
        // jcache parameter is the full-qualified class name of SPI implementation
        String type = url.getParameter("jcache");

        // 基于类型，获得 javax.cache.CachingProvider 对象，
        CachingProvider provider = type == null || type.length() == 0 ? Caching.getCachingProvider() : Caching.getCac
        // 获得 javax.cache.CacheManager 对象
        CacheManager cacheManager = provider.getCacheManager();
        // 获得 javax.cache.Cache 对象
        Cache<Object, Object> cache = cacheManager.getCache(key);
        // 不存在，则进行创建
        if (cache == null) {
            try {
                // 设置 Cache 配置项
                // configure the cache
                MutableConfiguration config =
                        new MutableConfiguration<Object, Object>()
                                // 类型
                                .setTypes(Object.class, Object.class)
                                // 过期策略，按照写入时间过期。通过 `"cache.write.expire"` 配置项设置过期时间，默认为
                                .setExpiryPolicyFactory(CreatedExpiryPolicy.factoryOf(new Duration(TimeUnit.MILLISECO
                                .setStoreByValue(false)
                                // 设置 MBean
                                .setManagementEnabled(true)
                                .setStatisticsEnabled(true);
                // 创建 javax.cache.Cache 对象
                cache = cacheManager.createCache(key, config);
            } catch (CacheException e) {
                // 初始化 cache 的并发情况
                // concurrent cache initialization
```

```
                cache = cacheManager.getCache(key);
            }
        }

        this.store = cache;
    }

    @Override
    public void put(Object key, Object value) {
        store.put(key, value);
    }

    @Override
    public Object get(Object key) {
        return store.get(key);
    }

}
```

已经添加详细中文注释，胖友自己查看。
笔者对 JCache 了解不多，推荐阅读 [《 Java Caching(缓存)-策略和JCache API》]

6.2 JCacheFactory

com.alibaba.dubbo.cache.support.jcache.JCacheFactory ， 实现 AbstractCacheFactory 抽象类，代码如下：

```
public class JCacheFactory extends AbstractCacheFactory {

    @Override
    protected Cache createCache(URL url) {
        return new JCache(url);
    }

}
```

666. 彩蛋

周末的疯狂输出，写了好多篇呀！

文章目录

回到首页