链滴

Spring-Simple-memcached 增加支持只取键值

作者：ixiaozhi

原文链接：https://ld246.com/article/1501949478941

来源网站：链滴

这里使用的 Spring-Simple-Memcached 的版本为：

```
compile 'com.google.code.simple-spring-memcached:spymemcached:2.8.4'
compile 'com.google.code.simple-spring-memcached:spymemcached-provider:3.1.0'
compile 'com.google.code.simple-spring-memcached:simple-spring-memcached:3.1.0'
```

ssm 支持的读取相关的方法有：

@ReadThroughAssignCache: 读取指定key缓存

@ReadThroughSingleCache: 读取单个缓存

@ReadThroughMultiCache: 读取多个缓存

在 get(key) key 值不存在时，该方法会默认把返回的 value 值添加至该 key 值的缓存。

假设，我们有需求要只取某 key 的缓存值，不存在时也不需要增加缓存，以下是对 ssm 包的扩展。

google.code.ssm.ReadOnlyThroughSingleCacheAdvice.java

```
package com.google.code.ssm.aop;

import com.google.code.ssm.aop.support.AnnotationData;
import com.google.code.ssm.api.ReadOnlyThroughSingleCache;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Created by ixiaozhi on 16/7/27.
 */
@Aspect
public class ReadOnlyThroughSingleCacheAdvice extends SingleReadCacheAdviceExtend<Re
dOnlyThroughSingleCache> {
    private static final Logger LOG = LoggerFactory.getLogger(ReadThroughSingleCacheAdvice
class);

    public ReadOnlyThroughSingleCacheAdvice() {
        super(ReadOnlyThroughSingleCache.class);
    }

    @Pointcut("@annotation(com.google.code.ssm.api.ReadOnlyThroughSingleCache)")
    public void getSingle() {
    }

    @Around("getSingle()")
    public Object cacheGetSingle(final ProceedingJoinPoint pjp) throws Throwable {
        return cache(pjp);
    }

    @Override
    protected String getCacheKey(final AnnotationData data, final Object[] args, final String me
```

```
hodDesc) throws Exception {
        return getCacheBase().getCacheKeyBuilder().getCacheKey(data, args, methodDesc);
    }

    @Override
    protected Logger getLogger() {
        return LOG;
    }
}
```

com.google.code.ssm.aop.SingleReadCacheAdviceExtend.java

```
package com.google.code.ssm.aop;

import com.google.code.ssm.aop.support.AnnotationData;
import com.google.code.ssm.aop.support.AnnotationDataBuilder;
import com.google.code.ssm.api.format.SerializationType;
import org.aspectj.lang.ProceedingJoinPoint;

import java.lang.annotation.Annotation;
import java.lang.reflect.Method;

/**
 * Created by ixiaozhi on 16/7/27.
 */
abstract class SingleReadCacheAdviceExtend<T extends Annotation> extends CacheAdvice {
    private final Class<T> annotationClass;

    protected SingleReadCacheAdviceExtend(final Class<T> annotationClass) {
        this.annotationClass = annotationClass;
    }

    protected Object cache(final ProceedingJoinPoint pjp) throws Throwable {
        if (isDisabled()) {
            getLogger().info("Cache disabled");
            return pjp.proceed();
        }
        // This is injected caching. If anything goes wrong in the caching, LOG
        // the crap outta it, but do not let it surface up past the AOP injection itself.
        final T annotation;
        final AnnotationData data;
        final SerializationType serializationType;
        String cacheKey = null;
        try {
            final Method methodToCache = getCacheBase().getMethodToCache(pjp);
            getCacheBase().verifyReturnTypeIsNoVoid(methodToCache, annotationClass);
            verifyNoUseJsonAnnotation(methodToCache);
            annotation = methodToCache.getAnnotation(annotationClass);
            serializationType = getCacheBase().getSerializationType(methodToCache);
            data = AnnotationDataBuilder.buildAnnotationData(annotation, annotationClass, met
odToCache);

            cacheKey = getCacheKey(data, pjp.getArgs(), methodToCache.toString());
```

```
            final Object result = getCacheBase().getCache(data).get(cacheKey, serializationType);
            if (result != null) {
                getLogger().debug("Cache hit.");
                return getCacheBase().getResult(result);
            } else {
                return null;
            }
        } catch (Throwable ex) {
            warn(ex, "Caching on method %s and key [%s] aborted due to an error.", pjp.toShortSt
ing(), cacheKey);
            return pjp.proceed();
        }
    }

    protected abstract String getCacheKey(final AnnotationData data, final Object[] args, final S
ring methodDesc) throws Exception;
}
```

com.google.code.ssm.api.ReadOnlyThroughSingleCache.java

```
package com.google.code.ssm.api;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Created by ixiaozhi on 16/7/27.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ReadOnlyThroughSingleCache {
    String namespace() default AnnotationConstants.DEFAULT_STRING;

    int expiration() default 0;
}
```

配置文件中的 simplesm-context.xml 添加以下实例化配置：（如果有的话；如果不存在，在自己的 pring 配置中添加也是一样的）

```
<bean id="readOnlyThroughSingleCache" class="com.google.code.ssm.aop.ReadOnlyThroug
SingleCacheAdvice">
        <property name="cacheBase" ref="cacheBase" />
</bean>
```

使用时，使用自定义的注解： @ReadOnlyThroughSingleCache，参数与 ReadThroughSingleCac e 相同，有 namespace 与 expiration。