链滴

实现忽略大小写的 copyProperties

作者：keepgoingxjw
原文链接：https://ld246.com/article/1564226077416
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

实现的原因

Spring中的BeanUtils.copyProperties(source,target),不能忽略大小写，类型不同不能相互赋值。速并不是特别理想。自己根据反射实现了工具类。

其中ReflectASM需要引入下面jar包

```
        <!---RelectAsm-->
        <dependency>
            <groupId>com.esotericsoftware</groupId>
            <artifactId>reflectasm</artifactId>
            <version>1.11.7</version>
        </dependency>
```

#具体代码

```
import com.esotericsoftware.reflectasm.MethodAccess;

import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.util.*;

/**
 *
 * @Auther: xjw
 * @Description: 此类主要用于反射，对反射的一些操作进行缓存起来。
 */
public class BeanUtils {

    /**
     * 集合 copy
     *
     * @param sources
     * @param target
     * @param <T>
     * @return
     */
    public static <T> List<T> listCopyPropertiesASM(List sources, Class<T> target) {
        List list = new ArrayList<>(sources.size());
        sources.forEach(it -> {
            list.add(copyPropertiesASM(it, target));
        });
        return list;
    }

    /**
     * 通过 target的Class 复制
     * target 必须有一个无参构造器
     *
     * @param source
     * @param target
     * @param <T>
     * @return
     */
```

```
    public static <T> T copyPropertiesASM(Object source, Class<T> target) {
        T newT = null;
        try {
            newT = target.newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
        return copyPropertiesASM(source, newT);
    }

    /**
     * 通过 ASM反射 速度比 Spring BeanUtils.copyProperties(source,target) 快一倍
     * 大小写可以忽略
     * 下划线 _ 被忽略
     *
     * @param source 源对象
     * @param target 目标实例化对象
     * @param <T>
     * @return
     */
    public static <T> T copyPropertiesASM(Object source, Object target) {
        MethodAccess sourceMethodAccess = CacheMethodAccess.getMethodAccess(source.ge
Class());
        MethodAccess targetMethodAccess = CacheMethodAccess.getMethodAccess(target.get
lass());
        Map<String, String> sourceGet = CacheAsmFiledMethod.getMethod("get", source.getCl
ss());
        Map<String, String> targetSet = CacheAsmFiledMethod.getMethod("set", target.getClas
());
        CacheFieldMap.getFieldMap(target.getClass()).keySet().forEach((it) -> {
            String sourceIndex = sourceGet.get(it);
            if (sourceIndex != null) {
                Object value = sourceMethodAccess.invoke(source, sourceIndex);
                String setIndex = targetSet.get(it);
                targetMethodAccess.invoke(target, setIndex, value);
            }
        });
        return (T) target;
    }

    /**
     * 模仿Spring中 BeanUtils.copyProperties(source,target)
     * 大小写可以忽略
     * 下划线 _ 被忽略
     *
     * @param source
     * @param target
     * @param <T>
     * @return
     */
    public static <T> T copyProperties(Object source, Object target) {
        Map<String, Field> sourceMap = CacheFieldMap.getFieldMap(source.getClass());
```

```
    CacheFieldMap.getFieldMap(target.getClass()).values().forEach((it) -> {
        Field field = sourceMap.get(it.getName().toLowerCase().replace("_", ""));
        if (field != null) {
            it.setAccessible(true);
            field.setAccessible(true);
            try {
                it.set(target, field.get(source));
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
    });
    return (T) target;
}

private static class CacheAsmFiledMethod {
    private static Map<String, Map<String, String>> cacheGetMethod = new HashMap<>();
    private static Map<String, Map<String, String>> cacheSetMethod = new HashMap<>();

    private static Map<String, String> getMethod(String type, Class clazz) {
        MethodAccess methodAccess = CacheMethodAccess.getMethodAccess(clazz);
        Map<String, Field> allFields = CacheFieldMap.getFieldMap(clazz);
        Map<String, String> result = null;
        if (type.equals("get")) {
            result = cacheGetMethod.get(clazz.getName());
        } else if (type.equals("set")) {
            result = cacheSetMethod.get(clazz.getName());
        }
        if (result == null) {
            synchronized (CacheAsmFiledMethod.class) {
                if (result == null) {
                    Map<String, String> set = new HashMap<>();
                    Map<String, String> get = new HashMap<>();
                    allFields.values().forEach((it) -> {
                        //判断是否是静态
                        if (!Modifier.isStatic(it.getModifiers())) {
                            //首字母大写
                            char[] f = it.getName().toCharArray();
                            f[0] -= 32;
                            String fieldName = new String(f);
                            get.put(fieldName.toLowerCase().replace("_", ""), "get" + fieldName);
                            set.put(fieldName.toLowerCase().replace("_", ""), "set" + fieldName);
                        }
                    });
                    cacheGetMethod.put(clazz.getName(), get);
                    cacheSetMethod.put(clazz.getName(), set);
                    if (type.equals("get")) {
                        result = cacheGetMethod.get(clazz.getName());
                    } else if (type.equals("set")) {
                        result = cacheSetMethod.get(clazz.getName());
                    }
                }
            }
        }
```

```
        return result;
    }
}

private static class SingelClass {
    private static Map<String, Object> cacheObject = new HashMap<>();

    private SingelClass() {
    }

    private static <T> T getObject(Class<T> clazz) {
        T result = (T) cacheObject.get(clazz.getName());
        if (result == null) {
            synchronized (SingelClass.class) {
                if (result == null) {
                    try {
                        cacheObject.put(clazz.getName(), clazz.newInstance());
                        result = (T) cacheObject.get(clazz.getName());
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }
        return result;
    }
}

private static class CacheMethodAccess {

    private CacheMethodAccess() {
    }

    private static Map<String, MethodAccess> cache = new HashMap<>();

    private static MethodAccess getMethodAccess(Class clazz) {
        MethodAccess result = cache.get(clazz.getName());
        if (result == null) {
            synchronized (CacheMethodAccess.class) {
                if (result == null) {
                    cache.put(clazz.getName(), MethodAccess.get(clazz));
                    result = cache.get(clazz.getName());
                }
            }
        }
        return result;
    }
}

private static class CacheFieldMap {
    private static Map<String, Map<String, Field>> cacheMap = new HashMap<>();

    private static Map<String, Field> getFieldMap(Class clazz) {
        Map<String, Field> result = cacheMap.get(clazz.getName());
```

```
            if (result == null) {
                synchronized (CacheFieldMap.class) {
                    if (result == null) {
                        Map<String, Field> fieldMap = new HashMap<>();
                        for (Field field : clazz.getDeclaredFields()) {
                            fieldMap.put(field.getName().toLowerCase().replace("_", ""), field);
                        }
                        cacheMap.put(clazz.getName(), fieldMap);
                        result = cacheMap.get(clazz.getName());
                    }
                }
            }
            return result;
        }
    }
}
```

测试

```
    public static void main(String[] args) {
        Man man = new Man("name", 20);
        Woman woman = BeanUtils.copyPropertiesASM(man, Woman.class);
        System.out.println(woman);
        List<Man> manList = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            manList.add(new Man("name" + i, i));
        }
        List<Woman> womens = BeanUtils.listCopyPropertiesASM(manList,Woman.class);
        System.out.println(womens);
    }
```