

目录

第1章 编码

01 开篇词：为什么学习本专栏 [已学完](#)

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 [已学完](#)

05 分层领域模型使用解读 [已学完](#)

06 Java属性映射的正确姿势 [最近阅读](#)

07 过期类、属性、接口的正确处理姿势 [已学完](#)

08 空指针引发的血案 [已学完](#)

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

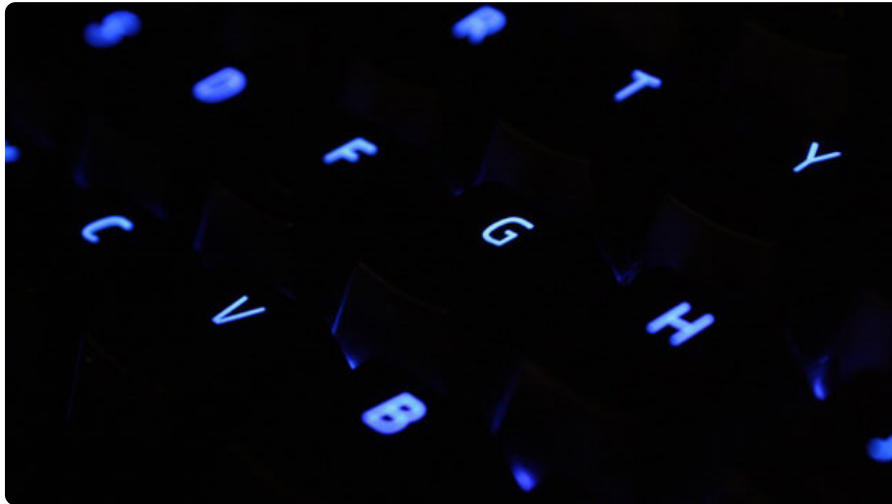
第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

06 Java属性映射的正确姿势

更新时间：2019-11-01 11:54:54



“

生活永远不像我们想像的那样好，但也不会像我们想像的那样糟。

——莫泊桑

”

如果断更，请联系QQ/微信642600657

1. 前言

前一节讲到项目为了更容易维护，易于拓展等原因会使用各种分层领域模型。在多层应用中，常常需要对各种不同的分层对象进行转换，这就会存在一个非常棘手的问题即：编写不同的模型之间相互转换的代码非常麻烦。其中最常见和最简单的方式是编写对象属性转换函数，即普通的Getter/Setter方法。除此之外各种各种属性映射工具。

- 那么常见的Java属性映射工具有哪些？
- 它们的原理以及对其性能怎样？
- 实际开发中该如何选择？

本节将给出解答。

2. 常见的Java属性映射的工具及其原理

2.1 常见的Java属性映射工具

常见的Java属性映射工具有以下几种：

1. `org.apache.commons.beanutils.BeanUtils#copyProperties`
2. `org.springframework.beans.BeanUtils#copyProperties(java.lang.Object, java.lang.Object)`
3. `org.dozer.Mapper#map(java.lang.Object, java.lang.Class<T>)`
4. `net.sf.cglib.beans.BeanCopier#copy`
5. `ma.glasnost.orika.MapperFacade#map(S, D)`
6. `mapstruct`

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 最近阅读

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势



1、Getter/Setter 方式使用原生的语法，虽然简单但是手动编写非常耗时；

2、通过 [dozer](#) 的 [maven](#) 依赖可以看出，dozer 并没有使用字节码增强技术，因为并没有引用任何字节码增强技术的 jar 包；我们再从其核心类 [org.dozer.MappingProcessor](#) 中寻找线索：

```
import java.lang.reflect.Array;
import java.lang.reflect.Modifier;
import java.lang.reflect.InvocationTargetException;
...
```

我们可以断定，dozer 使用的是反射机制。

3、同样的 commons 和 Spring 的 [BeanUtil](#) 工具类也采用的是反射方式。优点是两个是非常常用的类库，不需要引用更多复杂的包；4、cglib 的 [BeanCopier](#) 的原理是不是也是反射机制呢？我们可以通过 [cglib](#) 的 [maven](#) 库的编译依赖中找到线索：

Compile Dependencies (2)				
Category/License	Group / Artifact		Version	Updates
 Apache 2.0	org.apache.ant » ant (optional)		1.10.3	1.10.6
 BSD	org.ow2.asm » asm		7.1	✓

如果断更，请联系QQ/微信642600657

发现该库依赖了 [asm](#)，我们去 [asm](#) 官网可以看到它的介绍：

asm 库是一个 Java 字节码操作和分析框架，它可以用来修改已经存在的字节码或者直接二进制形式动态生成 class 文件。asm 的特点是小且快。

、同样的，我们可以通过 [orika](#) 的 [maven](#) 库得到其实现依赖的核心技术：

Compile Dependencies (6)				
Category/License	Group / Artifact		Version	Updates
 Apache 2.0	com.carrotsearch » java-sizeof		0.0.5	✓
 Apache 2.0	com.google.guava » guava (optional)		24.0-jre	28.0-jre
 BSD	com.thoughtworks.paranamer » paranamer		2.8	✓
 BSD	org.codehaus.janino » janino		3.0.8	3.0.14
 Apache 2.0 LGPL 2.1 MPL 1.1	org.javassist » javassist		3.24.0-GA	3.25.0-GA
 MIT	org.slf4j » slf4j-api		1.7.25	1.7.26

其中 [javassist](#) 我们知道它是一个字节码操作工具。

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 最近阅读

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

javassist 让操作字节码非常容易。javassist 允许 java 程序运行时定义一个新的类，也可以实现在 JVM 加载类文件时修改它。javassist 提供两种级别的 API，一种是源码级别；一种是字节码级别。使用源码级别的 API，无需对 java 字节码特定知识有深入的了解就可以轻松修改类文件。字节码级别的 API 则允许用户直接修改类文件。

6、通过 [MapStruct 的官网](#) 的介绍我们可以看出，mapstruct 采用原生的方法调用，因此更快速，更安全也更容易理解。根据官网的介绍我们知道，使用时只需要使用它的注解，定义好转换接口，转换函数，编译时会自动生成转换工具的实现类、调用属性赋值和取值函数实现转换。mapstruct 还支持通过注解形式定义不同属性名的映射关系等，功能很强大。

转换代码：

```
@Mapper
public interface UserMapper {
    UserMapper INSTANCE = (UserMapper)Mappers.getMapper(UserMapper.class);

    UserDTO userDo2Dto(UserDO var1);
}
```

编译后生成自动的转换接口的实现类：

```
public class UserMapperImpl implements UserMapper {
    public UserMapperImpl() {
        // ...
    }

    public UserDTO userDo2Dto(UserDO userDO) {
        if (userDO == null) {
            return null;
        } else {
            UserDTO userDTO = new UserDTO();
            userDTO.setName(userDO.getName());
            userDTO.setAge(userDO.getAge());
            userDTO.setNickName(userDO.getNickName());
            userDTO.setBirthDay(userDO.getBirthDay());
            return userDTO;
        }
    }
}
```

大大简化了代码。

官方还提供了非常详细的[参考文档](#)和使用范例，提供了很多高级用法。

2.3 性能

接下来按照惯例，我们对比一下它们的性能。

我们在 `com.imooc.basic.converter.UserConverterTest` 类中对上面的常见对象转换方式进行单测 `UserDO` 对象：

```
@Data
public class UserDO {
```

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 最近阅读

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

```
private String nickName;  
private Date birthDay;  
}
```

目标对象：

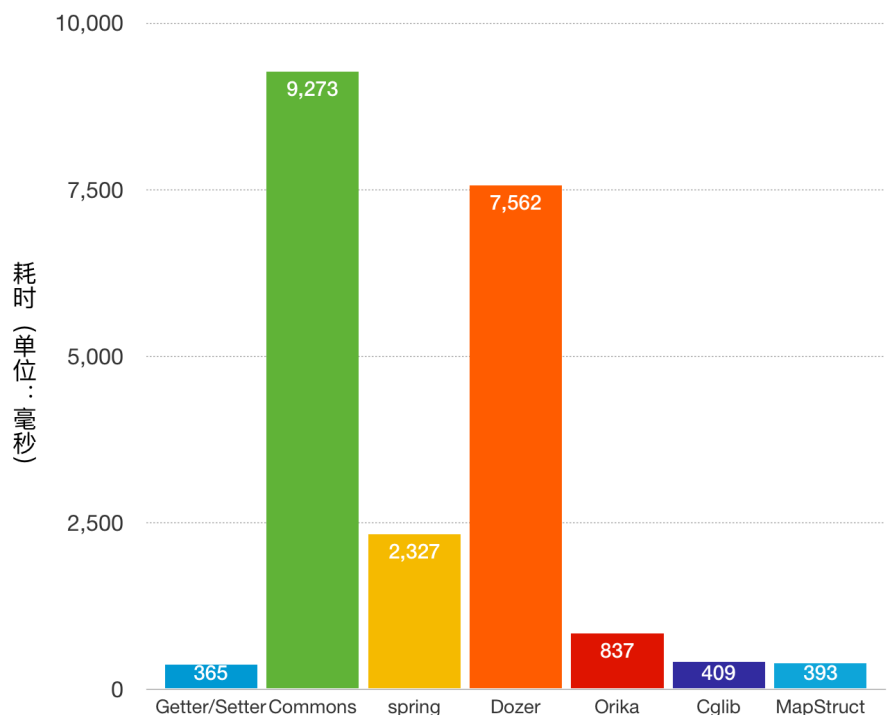
```
@Data  
public class UserDTO {  
    private String name;  
    private Integer age;  
    private String nickName;  
    private Date birthDay;  
}
```

使用 easyrandom（后面的单元测试环节会重点介绍）构造 10 万个 UserDO 随机对象进行性能对比。spring 版本为 5.1.8.RELEASE，dozer 版本为 5.5.1，orika-core 版本为 1.5.4，cglib 版本为 3.2.12，commons-lang3 包版本为 3.9，10 次运行取平均值，最终结果如下：

1. 普通 Getter/Setter 耗时 365ms；
2. org.apache.commons.beanutils.BeanUtils#copyProperties 耗时 9s273ms；
3. org.springframework.beans.BeanUtils#copyProperties(java.lang.Object, java.lang.Object) 耗时 2s327ms；
4. org.dozer.Mapper#map(java.lang.Object, java.lang.Class<T>) 耗时 9s271ms；
5. ma.glasnost.orika.MapperFacade#map(S, D) 耗时 837ms；
6. net.sf.cglib.beans.BeanCopier#copy 耗时 409ms；
7. MapStruct 393ms。

如果断更，请联系QQ/微信642600657

Java 常见属性映射工具转换耗时对比



由于机器的性能不同结果会有偏差，本实验并没有将转换框架的功能发挥到极致，也没有使用更复杂的对象进行对比，因此本实验的结果仅作为一个大致的参考。

目录

第1章 编码

01 开篇词：为什么学习本专栏 已学完

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 已学完

05 分层领域模型使用解读 已学完

06 Java属性映射的正确姿势 最近阅读

07 过期类、属性、接口的正确处理姿势 已学完

08 空指针引发的血案 已学完

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

因此从性能来讲首推 Getter/Setter 方式（含 MapStruct），其次是 cglib。

3. 用哪个？为什么？怎么用？

3.1 用什么？为什么？

通过以上的分析，我们对 Java 属性转换有了一个基本的了解。

选择太多往往会比较纠结，实际开发中我们用哪种更好呢？

我在业务代码中见到同事用的转换工具主要有 Getter/Setter 方式、orika 和 commons/spring 的属性拷贝工具。

属性转换工具的优势：用起来方便，往往一行行代码就实现多属性的转换，而且属性不对应可以通过注解或者修改配置方式自动适配，功能非常强大。

属性转换工具的缺点：

1. 多次对象映射（从 A 映射到 B，再从 B 映射到 C）如果属性不完全一致容易出错；
2. 有些转换工具，属性类型不一致自动转换容易出现意想不到的 BUG；
3. 基于反射和字节码增强技术的映射工具实现的映射，对一个类属性的修改不容易感知到对它转换类的影响。

我们可以想想这样一个场景：如果刚入职，请联系我 / 微信642600657

一个 **UserDO** 如果属性超多，转换到 **UserDTO** 再被转换成 **UserVO**。如果你修改 **UserDTO** 的一个属性命名，其它类待映射的类新增的对应属性有一个字母写错了，编译期间不容易发现问题，造成 BUG。

如果使用原始的 Getter/Setter 方式转换，修改了 **UserDO** 的属性，那么转换代码就会报错，编译都不通过，这样就可以逆向提醒我们注意到属性的变动的的影响。

因此强烈建议使用定义转换类和转换函数，使用插件实现转换，不需要引入其它库，降低了复杂性，可以支持更灵活的映射。

大家可以想想这种场景：

如果一个 A 映射到 B，B 有两个属性来自 C，一个属性来自于传参或者计算等。

此时自定义转换函数就更方便。

如果使用属性映射工具推荐使用 MapStruct，更安全一些，转换效率也很高。

3.2 怎么用？

目录

第1章 编码

01 开篇词：为什么学习本专栏 [已学完](#)

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 [已学完](#)

05 分层领域模型使用解读 [已学完](#)

06 Java属性映射的正确姿势 [最近阅读](#)

07 过期类、属性、接口的正确处理姿势 [已学完](#)

08 空指针引发的血案 [已学完](#)

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

为了避免转换函数散落到多个业务类中，不容易复用，我们可以在工具包或者对象包下定义一个专门的转换包（converter 或者 mapper 包），在转换的包下编写转换工具类。

第一种方式：可以实现 `org.springframework.core.convert.converter.Converter` 接口。

代码如下：

```
import org.springframework.core.convert.converter.Converter;

public class UserDO2DTOConverter implements Converter<UserDO, UserDTO> {

    @Override
    public UserDTO convert(UserDO source) {
        UserDTO userDTO = new UserDTO();
        userDTO.setName(source.getName());
        userDTO.setAge(source.getAge());
        userDTO.setNickName(source.getNickName());
        userDTO.setBirthDay(source.getBirthDay());
        return userDTO;
    }
}
```

上述只能实现单向转换，我们如果想双向转换该怎么做呢？

这时候我们可以采用**第二种方式**，可以继承 `com.google.common.base.Converter` 接口实现双向转换。如果断更，请联系QQ/微信642600657

```
import com.imooc.basic.converter.entity.UserDO;
import com.imooc.basic.converter.entity.UserDTO;
import com.google.common.base.Converter;

public class UserDO2DTOConverter extends Converter<UserDO, UserDTO> {

    @Override
    protected UserDTO doForward(UserDO userDO) {
        UserDTO userDTO = new UserDTO();
        userDTO.setName(userDO.getName());
        userDTO.setAge(userDO.getAge());
        userDTO.setNickName(userDO.getNickName());
        userDTO.setBirthDay(userDO.getBirthDay());
        return userDTO;
    }

    @Override
    protected UserDO doBackward(UserDTO userDTO) {
        UserDO userDO = new UserDO();
        userDO.setName(userDTO.getName());
        userDO.setAge(userDTO.getAge());
        userDO.setNickName(userDTO.getNickName());
        userDO.setBirthDay(userDTO.getBirthDay());
        return userDO;
    }
}
```

目录

第1章 编码

01 开篇词：为什么学习本专栏 [已学完](#)

02 Integer缓存问题分析

03 Java序列化引发的血案

04 学习浅拷贝和深拷贝的正确方式 [已学完](#)

05 分层领域模型使用解读 [已学完](#)

06 Java属性映射的正确姿势 [最近阅读](#)

07 过期类、属性、接口的正确处理姿势 [已学完](#)

08 空指针引发的血案 [已学完](#)

09 当switch遇到空指针

10 枚举类的正确学习方式

11 ArrayList的subList和Arrays的asList学习

12 添加注释的正确姿势

13 你真得了解可变参数吗？

14 集合去重的正确姿势

15 学习线程池的正确姿势

16 虚拟机退出时机问题研究

17 如何解决条件语句的多层嵌套问题？

加餐1：工欲善其事必先利其器

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势

此时可以自定义接口或者抽象类，支持更多种对象的转换。

更推荐大家直接定义某个对象的转换器类，在其内部编写该对象各层对象的转换函数：

```
public class UserConverter {

    public static UserDTO convertToDTO(UserDO source) {
        UserDTO userDTO = new UserDTO();
        userDTO.setName(source.getName());
        userDTO.setAge(source.getAge());
        userDTO.setNickName(source.getNickName());
        userDTO.setBirthDay(source.getBirthDay());
        return userDTO;
    }

    public static UserDO convertToDO(UserDO source) {
        UserDO userDO = new UserDO();
        userDO.setId(source.getId());
        userDO.setName(source.getName());
        userDO.setAge(source.getAge());
        userDO.setNickName(source.getNickName());
        userDO.setBirthDay(source.getBirthDay());
        return userDO;
    }

    // 转换成UserVO等
}
```

如果断更，请联系QQ/微信642600657

有些同学可能会抱怨，Getter/Setter 方式转换函数编写非常耗时而且容易漏，怎么办？

这里推荐一个 IDEA 插件：**GenerateAllSetter** 或者 **GenerateO2O**。

定义好转换函数之后，鼠标放在 `convertToDTO` 上使用快捷键，选择“generate setter getter converter”即可实现根据目标对象的属性名适配同名源对象自动填充，注意如果有个别属性不对应，需手动转换。

另外推荐使用 **mapstruct** 实现对象属性映射：

```
@Mapper
public interface UserMapper {
    UserMapper INSTANCE = Mappers.getMapper(UserMapper.class);

    UserDTO userDo2Dto(UserDO userDO);
}
```

使用时一行代码即可搞定：

```
UserDTO userDTO = UserMapper.INSTANCE.userDo2Dto(userDO);
```

相当于把 IDE 插件自动生成的这部分任务改为了使用注解，通过插件编译时自动生成。

4. 总结

<div><div>←慕课专栏</div><div>☰码出规范：《阿里巴巴Java开发手册》详解 / 06 Java属性映射的正确姿势</div></div>	
目录	除了考虑性能外，兼顾考虑安全性和可维护性。
第1章 编码	下节将介绍过期代码的正确处理方式。
01 开篇词：为什么学习本专栏 <div>已学完</div>	5. 课后练习
02 Integer缓存问题分析	
03 Java序列化引发的血案	自定义一个 OrderDO 和 OrderDTO 两个类，自定义属性，使用 StructMap 实现属性映射。
04 学习浅拷贝和深拷贝的正确方式 <div>已学完</div>	
05 分层领域模型使用解读 <div>已学完</div>	← 05 分层领域模型使用解读 07 过期类、属性、接口的正确处理姿势 →
06 Java属性映射的正确姿势 <div>最近阅读</div>	
07 过期类、属性、接口的正确处理姿势 <div>已学完</div>	精选留言 6
08 空指针引发的血案 <div>已学完</div>	
09 当switch遇到空指针	欢迎在这里发表留言，作者筛选后可公开显示
10 枚举类的正确学习方式	
11 ArrayList的subList和Arrays的asList学习	<div><div>hoofungson</div><div>嗯，优秀！！得多用多总结。</div><div><div>👍 1</div><div>回复</div><div>2019-12-03</div></div><div><div>明明如月</div><div>回复 hoofungson</div><div>如果断更，请联系QQ/微信642600657。主要是用之前先想清楚，从性能、可维护性等角度去选择。不要用了之后出 BUG 或者故障再去总结。</div><div><div>回复</div><div>2019-12-04 13:04:54</div></div></div></div>
12 添加注释的正确姿势	
13 你真得了解可变参数吗？	伊莉莎白菜
14 集合去重的正确姿势	
15 学习线程池的正确姿势	感觉业务上一些小对象的转换并不是性能瓶颈，代码行数少就完了，spring BeanUtils最好使。有时对象转换包含空值统一处理等逻辑，干脆手写反射
16 虚拟机退出时机问题研究	
17 如何解决条件语句的多层嵌套问题？	<div><div>👍 0</div><div>回复</div><div>2019-12-02</div></div> <div><div>明明如月</div><div>回复 伊莉莎白菜</div><div>的确属性不多性能不是最大的瓶颈。不推荐使用一些基于反射或者字节码的工具类并不是因为性能问题。而是可维护性，这点要特别注意。尤其项目大了以后，很容易出错，尤其多次转换。同属性名，类型不一致，使用工具类可能不报错，修改了属性名影响面也无法编译期间感知等等，这才是最大的问题。</div><div><div>回复</div><div>2019-12-03 10:57:26</div></div></div>
加餐1：工欲善其事必先利其器	
第2章 异常日志	<div><div>明明如月</div><div>回复 伊莉莎白菜</div><div>对于大型系统，高并发场景，性能不可忽略，积少成多，小型项目用反射也无妨。另外更重要的是可维护性。因为无用转换工具造成的故障，见过三四起了。有些是工具本身的BUG，有些是转换类型原因，有些是修改感知的原因等。</div><div><div>回复</div><div>2019-12-03 11:20:18</div></div></div>
18 一些异常处理建议	
19 日志学习和使用的正确姿势	后居上
	一直有这个疑惑，原来还有这种操作，读上一章的时候还在发愁呢。学到了，谢谢老是

明明如月 回复 后居上

工作中这种问题很常见，最好要养成好的习惯，否则很容易趟坑

回复

2019-11-25 11:57:26

目科将

再推荐一个:<http://modelmapper.org/>

👍 0

回复

2019-11-19

明明如月 回复 目科将

多谢补充。能转换的框架很多，更重要的是大家要思考如何选择。

回复

2019-11-19 20:38:37

PB310

不错哦，长知识了，用到项目开发中去

👍 0

回复

2019-11-18

慕粉3543028

很棒棒

回复

2019-10-31

明明如月 回复 慕粉3543028

大家可以参考一下这些建议，在学习和工作中尽量采用更稳妥和高效的转换方式。

回复

2019-10-31 22:20:49

千学不如一看，千看不如一练。

第2章 异常日志

18 一些异常处理建议

19 日志学习和使用的正确姿势