Software Engineer Sandbox

воскресенье, 12 мая 2013 г.

Dozer vs Orika vs Manual

Development of enterprise software often require creation of APIs for each application components. On implementation phase this involves mapping between API models on different layers of components (e.g. mapping persistence model to domain model to DTO). Such mapping often prone to boiler plate code, consuming development efforts and time.

There are multiple libraries which aim to solve this problem:

- some of them automate mapping only partially. For example Spring Framework's BeanUtils capabilities are limited with only primitive types and fields of only one class. It can not convert object hierarchy to another object hierarchy.
- others provide full automation of mapping process (e.g. Dozer), they allow to handle complicated cases, like mapping lists of object to maps of primitives. Automation usually comes at a price of performance degradation. The more complicated model is converted the more time it will take to map objects. This is usually caused by the fact that mapping libraries often use reflection and dynamic type resolution to support generic logic of conversion.

There is new player in the market of object mapping, called Orika. This library provide solution in automating mapping using code generation. Code is generated in the manner like it would be written by developers to map one bean to another. With Orika it takes just few minutes to add a library to the project and write couple lines of code to enable mapping between types. Everything else would be done by Orika it self. Though developers of Orika have prepare exhaustive functional documentation they did not provide any prove of performance efficiency. Lets compare all alternatives enumerated above:

- Dozer — each bean is created using reflection, mapping is customized using XML and Java code.
- Orika — beans are created and initialized using code generated by library, mapping can be customized using Java code and own expression language.
- Manual conversion — each bean is created and initialized with code written manually. All conversion is performed in Java code.

Data model that has been used for test include combinations which usually appear in Java Beans, such as:

- primitives
- object types
- collections
- enums

Source code of tests is available at GitHub.
Tests has been performed on:

- OS: Linux 2.6.32 x86_64
- CPU: 2.0GHz 4MB cache × 8 cores
- RAM: 16GB
- JVM: -Xmx1g -Xms1g -XX:MaxPermSize=128m -XX:NewSize=512m -XX:MaxNewSize=512m -XX:SurvivorRatio=6 -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -

Related Links

Software Engineer Magazine

Подписка

Сообщения

Комментарии

Ярлыки

Agile bug concurrency database dozer epson exceptions flex git grails groovy gwt hardware HTTP java jaxb ldap maven nosql off topic openam opensso orika Outsourcing performance Product owner prototype REST Scrum sitemesh spring sql templates unit testing vcs мысли в слух ООП оптимизация пользовательский интерфейс рефакторинг функциональное программирование

Twitter

История

- ▼ 2013 (1)
  - ▼ Май (1)
    - Dozer vs Orika vs Manual
- ► 2012 (1)
- ► 2011 (11)
- ► 2010 (13)

XX:+UseParNewGC -XX:+CMSIncrementalMode
- Number of iterations 1 000 000

Major GC was not happened during test.
Following test results were received:

Indicator	Dozer	Orika	Manual
Min, nano seconds	290 000	21 000	3 000
Mid, nano seconds	318 000	32 000	4 000
90%, nano seconds	390 000	41 000	5 000
95%, nano seconds	436 000	45 000	5 000
99%, nano seconds	534 000	54 000	6 000
99.9%, nano seconds	730 000	114 000	33 000
99.99%, nano seconds	1 607 000	374 000	65 000
99.999%, nano seconds	7 574 000	1 032 000	580 000
99.9999%, nano seconds	21 171 000	7 952 000	1 387 000
Max, nano seconds	21 171 000	222 581 000	9 810 000

Orika has maximum latency which differs from general distribution. This latency appears only once at first conversion, when Orika actually perform code generation for mappers.
Orika's latency has considerable deviation form Manual mapping, this caused by unnecessary object recreations performed by Orika. For example, when string is shifted from on variable to another, new string is created using concatenation with empty string.
In general test proved that Orika is is almost as efficient as manually written code and much more performant than Dozer.

на 1:28 g+1 +4 Рекомендовать в Google

Ярлыки: dozer, java, orika, performance

3 комментария:

Raja Nagendra Kumar 7 сентября 2013 г., 15:18

Would it not be good idea to generate code at build time, so that, it would be as fast as manual but without one writing the code..

Not sure all the dto frameworks operate at runtime.. I wish them to be working at compile time.

Ответить

Jon Smith 5 октября 2013 г., 15:18

A Business Phone Service answering service is more personal than a voice mail or pre-recorded voice because the needs of the customers are addressed directly and immediately. A business phone call can be redirected to a business phone answering service line if no one picks up your office line.

Ответить

Gunnar Morling 8 мая 2014 г., 16:43

Raja, you might be interested in MapStruct (http://www.mapstruct.org/) which does exactly what you describe: it generates type-safe mapping code from Java interfaces at compile time, in your IDE as well as via Maven, Ant etc. (Disclaimer: I'm working on MapStruct).

--Gunnar

Ответить

Введите комментарий...

Подпись комментария: Аккаунт Google

Публикация Просмотр

Главная страница Предыдущее

Подписаться на: Комментарии к сообщению (Atom)