链滴

为什么又要造一个叫 Latke 的轮子

作者：88250

原文链接：https://ld246.com/article/1403847528022

来源网站：链滴

框架

使用框架的好处很多，它规范了我们的开发方式，减少了出错的可能性，让我们可以更快地完成开发标，后续维护也可以有章可循；使用框架的弊端也很明显，它束缚了我们，离开熟悉的框架进行开发们可能会手足无措，它让我们身陷其中。但无论如何，我们是离不开框架的，多认识几种框架是没错，Java Web 领域更是如此（选择很多，同时也很少）。

到目前为止，我所认识的框架无一不例外都是以 class 作为实体类型的，为什么会这样？为什么不能其他形式（例如 map）作为实体载体？我觉得这些问题很值得讨论（虽然以前可能已经讨论过无数）、很值得进行实践。

为了把这个“想法”表达清楚，我们先看看一直以来在应用开发领域热议的一些话题，最后再看看“法”结晶——Latke 这个轮子是否能跑。

类型

在讨论编程语言的时候，我们经常会听到如“XXX 语言不是类型安全的”，“XXX 是动态语言，编时检查不了类型错误”等等此类。在过去（10 多年以前吧）这可能真的是一个“缺陷”，但在今天看，弱类型动态语言在应用开发领域（不是系统开发领域）越来越受欢迎，并且其他一些编程语言也做类似的语法糖。

我想最大的原因就是弱类型语言在代码修改时更快捷、成本更低，尽管我们现在使用的 IDE 重构辅助力很强，可一旦实体模型发生字段变化，相关的修改也是够头疼的（特别是应用间交互的 DTO，修成本瞬间飙升）。

当然，弱类型的缺点也是显而易见的，就是除了开发者本人，其他人很难搞清楚（只看代码片段）这变量到底是什么，到底包含了什么字段，要彻底搞清楚只能通过通读相关程序代码 （如果有准确的文就方便多了）。无论如何，现如今很多应用开发都是选择弱类型语言，并且已经得到了广泛运维验证PHP、Node.js）。

JSON

近些年，JSON 无疑已经成为了数据载体的一个标准，几乎所有编程语言、开发框架、存储系统都支持JSON 格式，最重要的是 JSON 是在浏览器端默认支持的结构化数据的方式。

在服务器端，使用 JSON 的地方（或者说和 JSON 相关的开发）也越来越多，POJO（实体对象 /Enti y）和 JSON 相互转换无时不在发生：前端提交请求，参数是 JSON 格式，控制器接到请求后将 JSON 实参转为 Java POJO，操作这个对象、生成响应（可能也是一个 JSON），最终返回前端，完成这次求处理。在这个过程中，至少包含了两次 JSON 和 POJO 的相互转换，虽然有很多工具（例如 Jacks n 能够帮助我们完成 JSON-POJO 映射，但是这样做的副作用也很明显：需要再学习一个工具（要能正确使用它）；额外的性能开销（有时很小，但有时很巨大）；代码看上去怪怪的（不自然，工程化“模型变换”）。

JSON 的确是好（简单有效，没有过度设计），但为什么不能从前到后的使用 JSON 呢?

ORM

将 POJO 持久化到关系型数据库的过程就是 ORM。但因为存在阻抗不匹配的问题，所以再优秀的 O M 方案也是存在问题的（性能问题、复杂查询问题），在解决这类问题的时候，通常做法都是直接写QL。从 ORM 实际实现上看，xBatis 的思路比 JPA 系更正确一些，但同时也略显繁琐了一些（需要义 mapper.xml）。

数据库表是二维的，数据总是可以转为键值对集合 /map 的（JDBC 结果集接口就是这样干的），反亦然。一个查询 SQL 返回的结果集可以很容易就转换为 map，复杂的是将这个 map 转换为 POJO 嵌套的实体必须根据嵌套元信息才能完成映射）。

言至此，我们肯定会问一个问题：为什么不能直接使用 map 呢?

领域建模

前些年，“领域建模”这个词非常流行，任何设计方案都要带上这顶帽子才好意思和别人打招呼。那年，要解决“用户登录”都要精心建模：

> “User 类必须有。”
>
> “嗯，最好抽象出个 IUser 接口。”
>
> “既然都有接口了，再整个抽象基类吧。”
>
> “嗯，很专业，成体系。”
>
> “呃，等等，login 接口放 IUser 里吧？还是放 UserService 里？”
>
> “放 service 里，大家都这么干的，放 user 里 Spring 好像不支持吧。”

最终，一个完美的 Java 航母应用开发完了，它确实能够航行，但是要换个螺丝或者加个床位的时候“等我找下设计图，嗯，换螺丝要拆掉 A~Z 这几个地方就能看见要换的螺丝了，加床位可能不行，，婴儿床应该可以”。

这一切大部分都归咎于一开始我们讨论的：类型。类型一旦固定，就真的固定了，无论我们设计的抽体系有多圆，最终都无法做到无损扩展（不可能真正达到开闭原则），因为所有的精密抽象都是存在漏的，面向类类型的编程范式在解决问题时不够直接，并且很难修改。

进一步成 Latke

把上面讨论的内容揉在一起，揉着揉着，居然变成了一个轮子——Latke。

前后端分离

类似 Tapestry、Wicket、JSF、GWT 的思路都是反前端的，前端该是什么样就是什么样（HTML/JS/SS），当然，服务器端的模板引擎还是需要的（比如 FreeMarker）。最终前端选择什么框架、工具对是前端开发决定，和后端没什么关系。

只有 JSON

请求实参 JSON 对象（很少情况是其他格式）传到控制器后，不用转为 POJO（因为我们压根没这个，直接操作这个 JSON（修改字段值、增减字段），并且可以很容易就将它持久化到数据库中了。大数时候都不用写 SQL，少数时候就获取数据库连接，JDBC 吧。

有 Schema

虽然从前到后都是使用 JSON，但也不用担心数据结构混乱，因为表结构和 JSON 的映射是有配置文定义的，可以通过这个结构定义生成建表 SQL，也可以通过已有的数据库表生成这个结构定义。

插件

可以在不改动任何一行现有代码的前提下添加新功能，而且这个新功能是完整的（前端后端都有），以很容易就集成到现有界面中的任何地方。

性能

从实现上看是 Netty HTTP 的薄封装，理论上和直接使用 Netty 性能差距不会太大，实际上我们也是行过压测的，结果显示没有性能问题，不求心服口服。

“恰到好处”的框架

在 HTTP 层，我们希望 Latke 是一个对 Netty 的轻薄封装，这需要开发者对 Netty 有一定了解，这能够通过自定义 Handler 来改变 HTTP 处理流程。

在实体模型/持久化层，Latke 做了一个较创新的设计尝试，即使用 JSON 进行贯穿、无实体类。较想的应用实现场景是前端提交的请求数据以 JSON 作为格式，该 JSON 在 Controller、Service 中做理（校验、增减字段等）后就能直接保存到关系型数据库中，免去很多无谓的类型转换和复杂的 OR 。

除了实体模型，其他的类对象（控制器、服务、切面、事件、插件、定时任务、缓存等）都是使用 IoC 容器进行管理的，这一点和 Spring 核心原理一致。因为 IoC / AOP 确实有用，也是业界主流技术。

Latke 整体的设计理念是**以最简化的技术手段来实现业务目标**，框架只是辅助，业务才是根本。

再进一步

“用上 Latke 轮子后开发效率提升一大截，但还是嫌前进太慢，怎么破？”

这已经不是轮子圆不圆的问题了，这是造轮子时使用的材料问题（风火轮和我们汽车轮子一样圆吧，实际上速度不是一个量级的，还可以当作武器，另外，这个还和使用的人有关吧）。

Java 就这样了，既然用了就说明我们已经接受它了，既然接受了就得忍让着点，实在不行就换个编程言，种种迹象表明，golang/Node.js 在应用开发领域已经风生水起。

最后

前面说了一大推来证明 Latke 是圆的，要是你不相信，请看下《Latke 快速上手指南》，里面有个 d mo；要是你还不相信，就亲自试用吧 ;-p

参考

- 强 / 弱类型、动 / 静态语言
- 抽象泄漏法则
- 《黑客与画家》
- 综述：编程语言的发展趋势及未来方向