Java框架研发思考

发表于2015-05-20 14:05| 2643次阅读| 来源CSDN| 4 条评论| 作者彭晨阳

[特别策划](http://www.csdn.net/tag/%E7%89%B9%E5%88%AB%E7%AD%96%E5%88%92/news)[Java](http://www.csdn.net/tag/java/news)[Java20周年](http://www.csdn.net/tag/java20%E5%91%A8%E5%B9%B4/news)[《程序员》电子刊](http://www.csdn.net/tag/%E3%80%8A%E7%A8%8B%E5%BA%8F%E5%91%98%E3%80%8B%E7%94%B5%E5%AD%90%E5%88%8A/news)

**摘要：**回顾这些年的Java之路，特别是开发Jdon这个Java开源框架的前后经历，我想也许有必要和大家分享其中的得与失，供后来者借鉴。本文谈谈Jdon框架的开发经历和我的思想转变发展过程。

这次应CSDN之邀为Java20周年写稿，多谢大家能够记得我，本是想写一篇具体实战技术的文章，但这些技巧也许在其20岁成年之时已经代表不了什么新潮流，尽管20岁年龄的人类还是属于追求潮流的一代。回顾这些年的Java之路，特别是开发Jdon这个Java开源框架的前后经历，我想也许有必要和大家分享其中的得与失，供后来者借鉴。



彭晨阳

下面我大概谈谈Jdon框架的开发经历和我的思想转变发展过程。

最初开发Jdon框架的想法是因为Web流行导致的三层直至多层架构对开发流程的影响。现代多层架构将后端架构分解为多个层次，灵活性得到提高，因为你维护拓展任何一个层都很少影响其他层，这也是设计模式或面向对象思想的最初初衷吧。但灵活性也是有副作用的，副作用是带来了复杂性，简单的数据增删改查四个功能需要经过三层，那么就可能产生12道工序的开发。

因此，我希望能够在不牺牲多层架构基础上提高快速开发的能力，于是需要集成表现层、业务层和持久层等现有框架的集成整合能力，当时表现层框架是Struts1.x，业务层框架是Spring，Spring其实是一个业务容器，集成了依赖注入DI和AOP；而当时我对另外一个轻量反转注入微容器PicoContainer比较熟悉；在持久层选择上，当时有Hibernate等ORM选择，但考虑到这些ORM可能会绑架业务实体对象，而我设计的业务层框架容器不应该像Spring那样只包含服务，而且也应该业务实体对象。

这种想法也让我开发的Jdon框架完全不同于当时各种开发框架，同时可能走上另类的道路，就是今天大部分开发路线都是服务和实体分离，实体由数据库管理，而我的设计思路是让实体对象常驻在内存Inmemory中，虽然在性能上有很大提升，但是在数据完整性与一致性方面完全脱离了关系数据库的ACID支持。

为了体现这种业务服务和业务实体合并在一个容器中的特点，我还需要寻找理论上的支持，当时恰逢DDD领域驱动设计思想的诞生，通过简单接触，发现其分析设计理念完全符合我理想中的设计，DDD对业务需求的分析是从面向对象OO分析方法开始，与传统的基于关系数据库的E-R模型分析方法不同，DDD强调从需求中找出有界上下文与微服务，还有实体与值对象两种数据模型，DDD的实体模型对象不同于关系数据库的实体，它不但包含数据，还包含改变这些数据的方法行为，也就是将可变的状态用对象方法封装起来，将改变状态的方法行为暴露给外界，这种设计理念与后来并发模型Actor模型非常类似。

基于并发的Actor模型在Erlang和Scala中都有实现，我非常想将这种Actor模型引入到框架中，用来作为DDD实体模型的落地实现，这时发现了Disruptor框架，其内部RingBuffer模型能够实现两个线程之间无锁通信，而之前Java世界基本都是基于有锁实现，Disruptor在LMAX系统中实现每秒600万的交易。于是我将Disruptor作为框架的业务层容器核心基础组件，服务与实体之间可以通过发送异步消息通信，不再使用传统的方法直接调用，因为这会造成两者的耦合，当然引入接口，也会和接口耦合，现在实体与服务之间通过消息通道耦合，发送者无论哪个消费者接收只管发送消息，消费者也无论哪个发送者发送的消息只管接收就可以了。后来我发现，其实这个并发模型更类似Go语言的Channel并发模型。

框架的并发性能通过引入常驻内存和异步消息机制得到了最大化提升，下一步需要关心数据的完整性和一致性。因为摆脱了关系数据库的ACID支持。这方面参考了分布式系统的CAP定理。有人可能会说，你只是一个开发框架，又不是分布式框架，怎么会使用到CAP呢？其实因为我们的实体模型数据常驻内存，而存储层如关系数据库或NoSQL也有一份同样的数据，很显然，数据已经被分布了，那么这两个地方的同一数据如何保证一致性？这个原理类似两台服务器之间同一种数据如何保证一致性一样，这就需要CAP定理了。

在这种数据分区分布的情况下，只能在高一致性和可用性之间作平衡，如果你使用在高可用性场合，比如追求高性能，低延迟，快速响应，大的吞吐量，那么可以牺牲一点高一致性，通过最终一致性来实现；当然如果你很注重数据一致性完整性，那么可以牺牲一点可用性的性能问题。

**那么具体开发中如何平衡呢？**

CQRS架构提供了这种动态平衡，CQRS是一种读写分离的架构，数据的读取查询需要快速高性能，因此可用性为第一；而数据的保存修改等写操作，数据安全为第一；数据完整性很重要，那么高一致性为第一。

上面讨论了Jdon框架的并发性能和数据一致性问题，那么，Jdon框架的扩展性如何？所谓扩展性Scalable或称为可伸缩性是指水平分布扩展的能力，通过增加后端服务器就能线性提高系统的处理能力，这时引入了EventSourcing概念（简称ES）。

ES是一种不同于传统数据存储的思路。通常，我们是将实体对象中可变的状态直接保存到关系数据库对应的数据表中，比如订单实体模型的状态有未付款已付款和已出货三种状态，这些可变的状态保存在数据表字段中。

前面我们提到，实体模型对象与由数据表翻译过来的实体数据对象是有区别的，关键是前者包含改变状态的方向行为，也就是说，订单对象包含付款与出货两个方法，如果这两个方法被调用，那么订单的状态就会改变，比如付款方法一旦被调用完成，那么订单状态从未付款变为了已付款。这时问题来了，因为传统数据保存的是订单的当前状态，那么数据表中订单表的状态用已付款覆盖了未付款，曾经存在过的未付款这个状态就被覆盖了，如果订单状态一旦出错，我们无法追踪当前状态是由之前哪个状态切换过来，那么这时如果我们将订单对象的付款方法调用行为作为数据保存起来，就会能够追溯状态变化的历史。

前面说过，订单实体对象的方法调用是通过异步消息触发的，而不是使用普通代码直接调用的，这点Jdon框架已经通过内置的Disruptor实现了，因此，在ES架构中，我们只要将异步消息事件作为数据保存到数据库中即可。当然，ES架构不只是为了方便调试追溯历史，更重要的是易于分布扩展，如果我们将异步消息事件保存到一个消息系统中，比如ApacheKafka中，那么通过Kafak能够在分布式服务器之间执行原始节点相同的操作，从而保证整个分布式系统中相同数据类型的数据的高一致性。LinkedIn成功案例见：http://[www.jdon.com/47156。](http://www.jdon.com/47156%E3%80%82)

通过以上我个人对Jdon框架开发设计规划经历，我们发现Java世界已经形成了一个巨大生态系统，从“量子世界”（CPU并发）统一到整个“宇宙世界”（分布式架构），我们能够通过一个小小的框架直接贯穿。

**作者简介：**彭晨阳Jdon.com创办者和版主。软件开发设计咨询从业20年，10余年Java开发经验，拥有ERP、大型游戏、互动电视三网合一等架构经验。独立咨询顾问，个人擅长复杂系统的软件架构和领域建模。流行新技术思想的传道者，主持解道网站跟踪国际最新软件架构思想和设计技术。首个国内Java开源框架项目Jdon框架的设计者。