目录

第一章　前言

笔者在写这款框架以前，也用了很长时间的开源框架，像 spring，springMvc，hibernate，mybatis 等等，正是在用这些开源的时候，有了一些自己对开源框架的感受，这些感受促使我自己写了这款框架。

第一个感受，就是这些开源框架帮助我们做的事情太多，如果我们过度的依赖于这些开源框架，会很不利于个人的提升，甚至有可能不进则退。就像一个人经常坐电梯，突然有一天停电了，电梯不能用了，让他爬到二楼都会气喘吁吁。所以我写这个框架的一个原因就是让自己得到提升，那个时候并没有想太多，没有想这个框架最终能否写成，只是想让自己在这个过程中，获得一些提升。

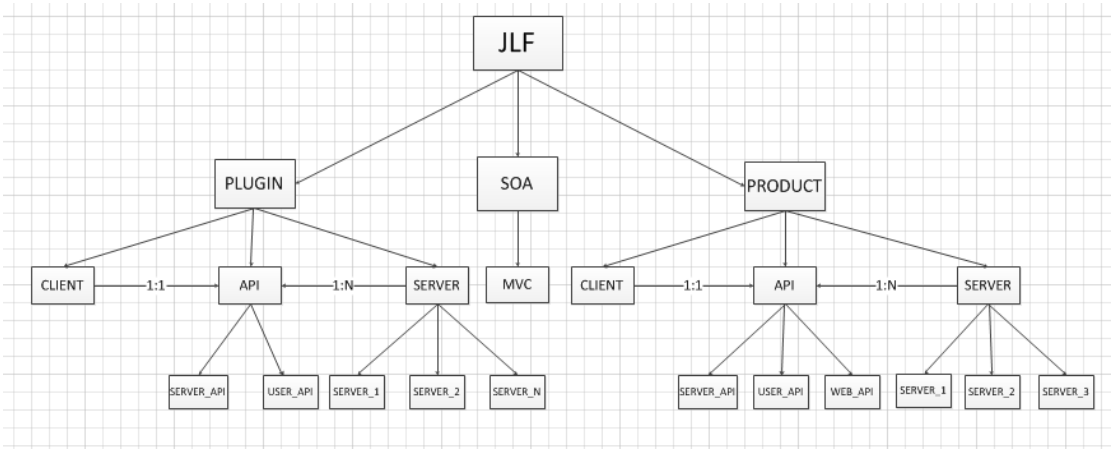
第二个感受，就是这个开源框架太重量级，它们都提供了很多很多功能，但是有很多功能，在工作中是不太适用的，或者完全用不到的。像 hibernate 提供的一级缓存和延迟加载，这两个功能只是听起来根强大，但是在工作中完全不适用。或者像 spring 提供的几种事务的传播方式和实现依赖注入的几种方式，笔者觉得，如果有一种方式可以适用于所有的场景，那么你提供那么多的其它的方式有什么用？完全是画蛇添足，不仅增加了用户对框架的学习成本，还增加了框架的复杂性。其实感觉现在很多人，在写代码时，都会把一些很简单的事情，写的很复杂，用到了各种各样的设计模式，我不喜欢这样，我喜欢把复杂的事情简单化，越简单越好。对于这些设计模式，如果你在适合用它的场景用它，那它真的会提升代码的扩展性、健壮性。但是如果在不适合用的场景非要用，那样反而会适得其反。

基于这个感受，所以我写的这个框架有两个特点，第一个特点就是没有提供太多的功能，但是提供的每一个功能都是可以在工作中真实的用的到的。第二个特点就是简单，无论是用户的上手，还是功能实现，都会特别简单，没有什么复杂的逻辑，这样可以从一定程度上减少 bug 的发生，而且框架一旦出现 bug，由于框架实现逻辑很简单，所以 bug 也很容易去跟踪，很容易去定位。

另一个促使我写这个框架的一个原因，就是笔者觉得，不管从事哪个行业，最终都要做出真正属于自己的东西，也许我这个框架很差很烂，但是至少我做出来了，我的第一步已经迈出来了，我以后可以不断的在这个基础上做完善，做优化。其实万事开头难，只要第一步勇敢的迈出来了，即使你踏进来泥潭，也比原地踏步强很多。

第二章　框架概要

首先来看一下这个框架的一个架构图：



这个框架分为三个模块：PLUGIN、SOA、PRODUCT。

PLUGIN 模块，主要是对项目中用到的第三方工具包做了一层封装，使得在项目中用到第三方工具包时，实现对代码的零侵入。对于两个实现了同样功能的工具包，用户可以在项目中自由切换，而无需修改代码。

SOA 模块，目前主要做了一个 MVC 框架，这个 MVC 框架在实现了 MVC 基本功能的基础上，又添加了以下功能：

（1）基于约定大于配置原则，基本实现零配置。

（2）提供多数据库之间的事物管理。

（3）集成缓存框架，并提供对数据库 id 黑名单管理，防止缓存穿透。

（4）封装 ORM 关系映射，比使用 hibernate 和 mybatis 更加简单。

（5）实现了 bean 容器的管理以及依赖注入。

（6）可以通过实体类，导出相应的 sql 脚本和数据字典。

（7）可以导出每个请求的请求参数、验证规则、响应参数等信息，方便前后端的协作开发。

PRODUCT 模块，是基于 PLUGIN 模块，做了一下升级，因为 PLUGIN 模块主要是面向于功能，提供给程序开发者去使用。但是 PRODUCT 模块是面向于业务的，它除了具备 PLUGIN 的功能以外，还可以接收用户从 web 端提交过来的请求。

第三章　PLUGIN

3.1 实现功能

在很多人协同开发一个项目中，如果在项目中需要依赖第三方工具包，常常会遇到一下两个问题：

1、 例如用到 JSON 工具包，可能有人用的时候会导入阿里的 fastjson，有人用的时候会导入谷歌的 gson，这样就会导致在对代码管理时带来很多麻烦。

2、 这个工具包对代码的侵入性很强，如果想把一款工具包换成另外一款实现同样功能的工具包，需要修改大量代码。

此模块就是常用的工具包做统一管理，解决上面两个问题。

3.2 实现原理

PLUGIN 模块的实现原理，就是将一类实现了共通功能的工具包，抽象出来一个接口，然后让每一个工具包都各自作为自己的一个服务端，去实现这个接口，然后在用一个客户端，这个客户端中存放了接口的一个引用，用户可以利用这个客户端中的引用，去调用这个接口中提供的方法。

虽然有客户端和服务端的概念，但是并不需要做远程调用，每个客户端和服务端都只是一个普通的 jar 包，用户在编译代码时，只需要依赖客户端即可以完成编译，无需关注服务端基于什么实现。在启动项目前，将对应的服务端包导入到项目中即可。

在项目启动的过程中，会自动扫描项目中依赖的所有客户端，然后针对于每个客户端，在去扫描这个客户端对应的服务端，在服务端中找到接口的实现类，然后启动服务，并把实现类对象赋值给客户端中的引用。

如果想把一款工具包换成另外一款实现同样功能的工具包，也只需要替换对应的服务端包即可，无需对代码进行修改。

如果某个服务端的配置文件进行修改，无需重启整个系统，可单独对这个服务端进行重启，在集群模式下，无需对每一台都重启，在一台服务器重启成功以后，会以广播的形式通知到其它服务器，其它服务器接收到通知后，会自动进行重启。

3.3 代码分析

3.4 已实现的插件

目前主要实现了如下插件：

1、 AOP 插件：实现了面向切面编程。

2、 DBPOOL 插件：实现了对数据库连接池的管理。

3、 CACHE 插件：实现了对缓存的管理。

4、 CHECK 插件：通 bean 与注解的结合使用，实现了良好的数据校验机制。

5、 SESSION 插件：实现了对分布式的 SESSION 的良好管理。
6、 THREADPOOL 插件：实现了对线程池的管理。
7、 JSON 插件：实现了对 JSON 的管理
8、 MQ 插件：实现了消息队列的管理
9、 PUSH 插件：实现了由本地向远程服务器发送数据的功能。
10、EXCEL 插件：实现了对 EXCEL 读、写的操作。
11、TEMPLATE 插件：实现了根据模板文件生成字符串或生成文件的功能。

第四章 SOA

4.1 MVC

4.1.1 实现功能

此 MVC 框架目前主要实现了一下功能

1、路由转发及参数校验。
2、基于约定大于配置原则，基本实现零配置。
3、提供多数据库之间的事物管理。
4、集成缓存框架，并提供对数据库 id 黑名单管理，防止缓存穿透。
5、封装 ORM 关系映射，比使用 hibernate 和 mybatis 更加简单。
6、实现了 bean 容器的管理以及依赖注入。
7、可以通过实体类，导出相应的 sql 脚本和数据字典。
8、可以导出每个请求的请求参数、验证规则、响应参数等信息，方便前后端的协作开发。

下面将对每一个功能展开说明他的实现原理。

4.1.2 路由转发及参数校验

4.1.3 配置文件解析

4.1.4 多数据库之间事务管理

4.1.5 缓存集成及如何防止缓存穿透

4.1.6 ORM 关系映射

4.1.7 bean 容器的管理以及依赖注入

4.1.8 sql 脚本和数据字典的导出

4.1.9 请求和响应信息的导出

第五章 PRODUCT

5.1 实现功能

PRODUCT 模块，是基于 PLUGIN 模块，做了一下升级，因为 PLUGIN 模块主要是面向于功能，提供给程序开发者去使用。但是 PRODUCT 模块是面向于业务的，它除了具备 PLUGIN 的功能以外，还可以接收用户从 web 端提交过来的请求。

5.2 实现原理

PRODUCT 模块的实现原理，是基于 PLUGIN 模块基础上，又新增了 WEB-API 这样一个接口，并在接口中声明了访问规则，在服务端包会依赖于 SOA 模块的 MVC 框架，实现 WEB-API 接口，系统在启动时，会自动将接口的实现类注册到路由集合中，这样就可以接收用户在 WEB 端提交过来的请求。

5.3 代码分析

5.4 已实现的产品

目前主要实现了如下插件：

1、 运维管理（OPS）

运维人员可以使用此产品，通过界面，对此框架的服务端进行维护。例如某个 SERVER 端的配置文件发生变化，运维人员只需在界面上调用重新启动接口即可，无需重启服务。在集群环境下，运维人员无需对每一台服务器都调用此接口，如果其中一台服务器调用此接口成功，会以消息队列形式通知到其它服务器，其它服务器接收到通知后，即会重新加载配置文件，加载完成后，会将加载结果记录到数据库，运维人员可以在界面上查看每台服务器的执行结果。

2、 定时任务管理（QUARTZ）

与传统的根据配置文件来管理定时任务不同的是，这款产品是基于数据库来管理定时任务，用户可以在系统运行时，通过页面来控制定时任务的运行，包括对定时任务的新增、修改、删除、启动、停止、以及查看执行日志等等。并且在集群模式下，可以保证一个条任务只能在一台服务器执行，用户还可以手工切换执行的服务器。

3、 日志管理（LOG）

用户每一次对系统的操作，记录日志，包括用户 id、操作时间、发送参数、执行结果、异常信息等等。方便运维人员对系统故障的排查，以及对数据修改历史的追踪。在大数量情况下，此产品提供了按月分表的机制，提高查询效率。