导航

博客园
首 页
新随笔
联 系
订 阅 XML
管 理

< 2018年3月 >

日	一	二	三	四	五	六
25	26	27	28	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

公告

昵称：arch-system
园龄：1年1个月
粉丝：231
关注：1
+加关注

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

Metrics(1)
架构设计(1)
应用分层(1)
应用监控(1)

随笔分类

中小研发团队架构实践系列(4)

随笔档案

2018年2月 (1)
2018年1月 (3)

文章分类

中小研发团队架构实践系列

最新评论

1. Re:中小研发团队架构实践之应用监控Metrics
侵入性比较强啊
--wdwwtzy
2. Re:中小研发团队架构实践之应用监控Metrics
期待楼主能写一篇关于Metrics.NET+InfluxDB+Grafana环境搭建配置的文章~~~
--大漠孤阳
3. Re:中小研发团队架构实践之应用监控Metrics
CAT is another choice.
点评CAT
--JeffWong
4. Re:中小研发团队架构实践之应用监控Metrics
支持支持支持支持
--~雨落忧伤~

中小研发团队架构实践之开篇

中小型研发团队很多，而社区在中小型研发团队架构实践方面的探讨却很少。中小型研发团队特别是50至200人的研发团队，在早期的业务探索阶段，更多关注业务逻辑，快速迭代以验证商业模式，很少去关注技术架构。这时如果继续按照原有的架构及研发模式，会出现大量的问题，再也无法玩下去了。能不能有一套可直接落地、基于开源、成本低，可快速搭建的中间件及架构升级方案呢？我是一个有十多年经验的IT老兵，曾主导了两家公司的技术架构升级改造，现抛砖引玉，与大家一起探讨这方面的问题。整个系列有18篇文章，可分为三个部分，包括框架篇、架构篇和公共应用篇。框架篇即中间件或工具的使用，如缓存、消息队列、集中式日志、度量、微服务框架等，工欲善其事，必先利其器。架构篇主要是设计思想的提升，有企业总体架构、单个项目架构设计、统一应用分层等。公共应用篇是业务与技术的结合，有单点登录和企业支付网关，以下是具体篇章的介绍：

一、框架篇——工欲善其事，必先利其器

如果说运维是地基，那么框架就是承重墙。农村建住房是一块砖一块砖地往上垒，而城市建大House则是先打地基，再建承重墙，最后才是垒砖，所以中间件的搭建和引进是建设高可用、高性能、易扩展可伸缩的大中型系统的前提。框架篇中的每篇主要由四部分组成：它是什么、工作原理、使用场景和可直接调试的Demo。其中Demo及中间件是历经两家公司四年时间的考验，涉及几百个应用，100多个库1万多张表，日订单从几万张到十几万，年GMV从几十亿到几百亿。所有中间件及工具都是基于开源，早期我们也有部分自主研发如集中式日志和度量框架。后期在第二家公司时为了快速地搭建，降低成本，易于维护和扩展，全部改为开源。这样不仅利于个人的学习成长、知识重用和职业生涯，也利于团队的组建和人才的引进。

1、集中式缓存Redis

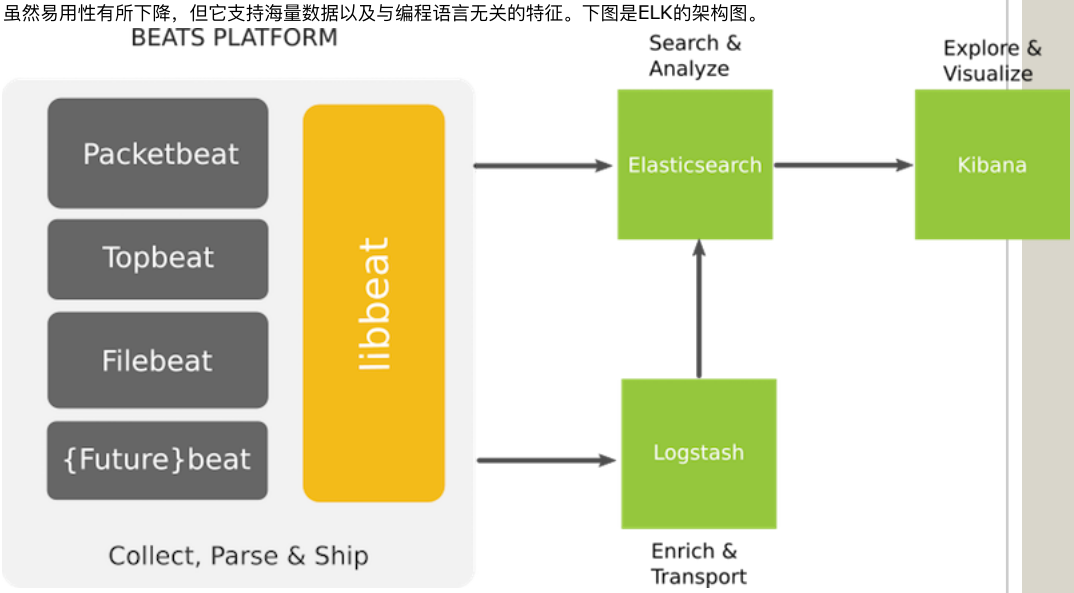
缓存是计算机的难题之一，分布式缓存亦是如此。Redis看起来非常简单，但它影响着系统的效率、性能、数据一致性。用好它不容易，具体包括：缓存时长（复杂多维度的计算）、缓存失效处理（主动更新）、缓存键（Hash和方便人工干预）、缓存内容及数据结构的选择、缓存雪崩的处理、缓存穿透的处理等。Redis除了缓存的功能，还有其它功能如Lua计算能力、Limit与Session时间窗口、分布式锁等。我们使用ServiceStack.Redis做客户端，使用方法详见Demo。

2、消息队列RabbitMQ

消息队列好比葛洲坝，有大量数据的堆积能力，然后再可靠地进行异步输出。它是EDA事件驱动架构的核心，也是CQRS同步数据的关键。为什么选择RabbitMQ而没有选择Kafka，因为业务系统有对消息的高可靠性要求，以及对复杂功能如消息确认Ack的要求。

3、集中式日志ELK

日志主要分为系统日志和应用日志两类。试想一下，你该如何在一个具有几百台服务器的集群中定位到问题？如何追踪每天产生的几G甚至几T的数据？集中式日志就是此类问题的解决方案。早期我们使用自主研发的Log4Net+MongoDB来收集和检索日志信息，但随着数据量的增加，查询速度却变得越来越慢。后期改为开源的ELK，虽然易用性有所下降，但它支持海量数据以及与编程语言无关的特征。下图是ELK的架构图。



4、任务调度Job

任务调度Job如同数据库作业或Windows计划任务，是分布式系统中异步和批处理的关键。我们的Job分为WinJob和HttpJob：WinJob是操作系统级别的定时任务，使用开源的框架Quartz.NET实现；而HttpJob则是自主研发实现，采

5. Re:中小研发团队架构实践之统一应用分层
整体还是贫血模型，在面对复杂业务时会力不从心，推荐看看领域驱动设计：。
--清香白莲素还真

阅读排行榜

1. 中小研发团队架构实践之开篇(8323)
2. 中小研发团队架构实践之总体架构(3587)
3. 中小研发团队架构实践之统一应用分层(2080)
4. 中小研发团队架构实践之应用监控Metrics(1910)

评论排行榜

1. 中小研发团队架构实践之开篇(77)
2. 中小研发团队架构实践之统一应用分层(23)
3. 中小研发团队架构实践之总体架构(17)
4. 中小研发团队架构实践之应用监控Metrics(4)

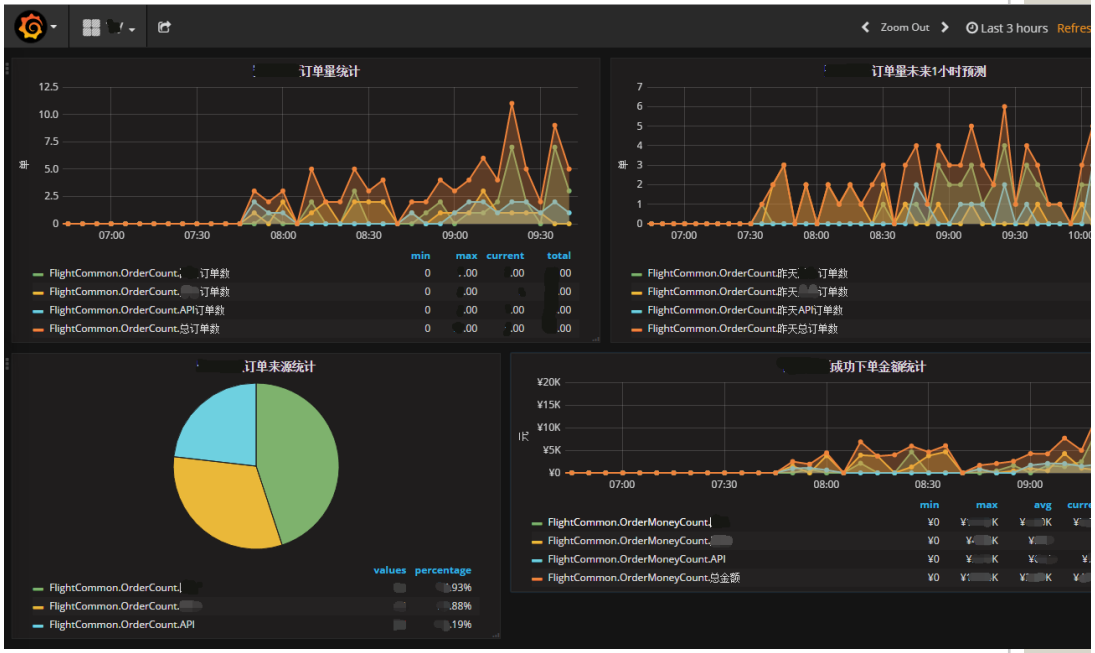
推荐排行榜

1. 中小研发团队架构实践之开篇(177)
2. 中小研发团队架构实践之总体架构(48)
3. 中小研发团队架构实践之统一应用分层(45)
4. 中小研发团队架构实践之应用监控Metrics(17)

用URL方式可定时调用微服务。HttpJob借助集群巧妙地解决了WinJob的单点和发布问题，并集中管理所有的调度规则，调度规则有简单规则和Cron表达式。HttpJob它简单易用，但间隔时间不能低于1分钟，毕竟通过URL方式来调度并不高效。下图是HttpJob的管理后台。

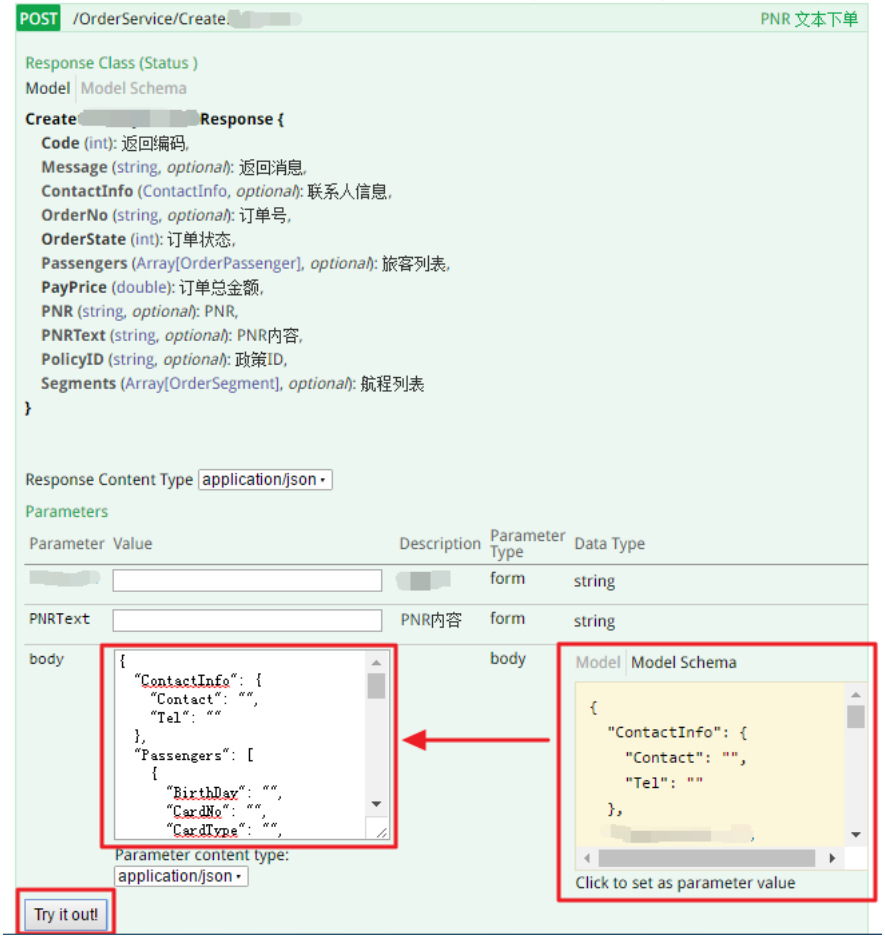
5、应用监控Metrics

“没有度量就没有提升”，度量是改进优化的基础，是做好一个系统的前置条件。Zabbix一般用于系统级别的监控，Metrics则用于业务应用级别的监控。业务应用是个黑盒子，通过数据埋点来收集应用的实时状态，然后展示在大屏或看板上。它是报警系统和数字化管理的基础，还可以结合集中式日志来快速定位和查找问题。我们的业务监控系统使用Metrics.NET+InfluxDB+Grafana。



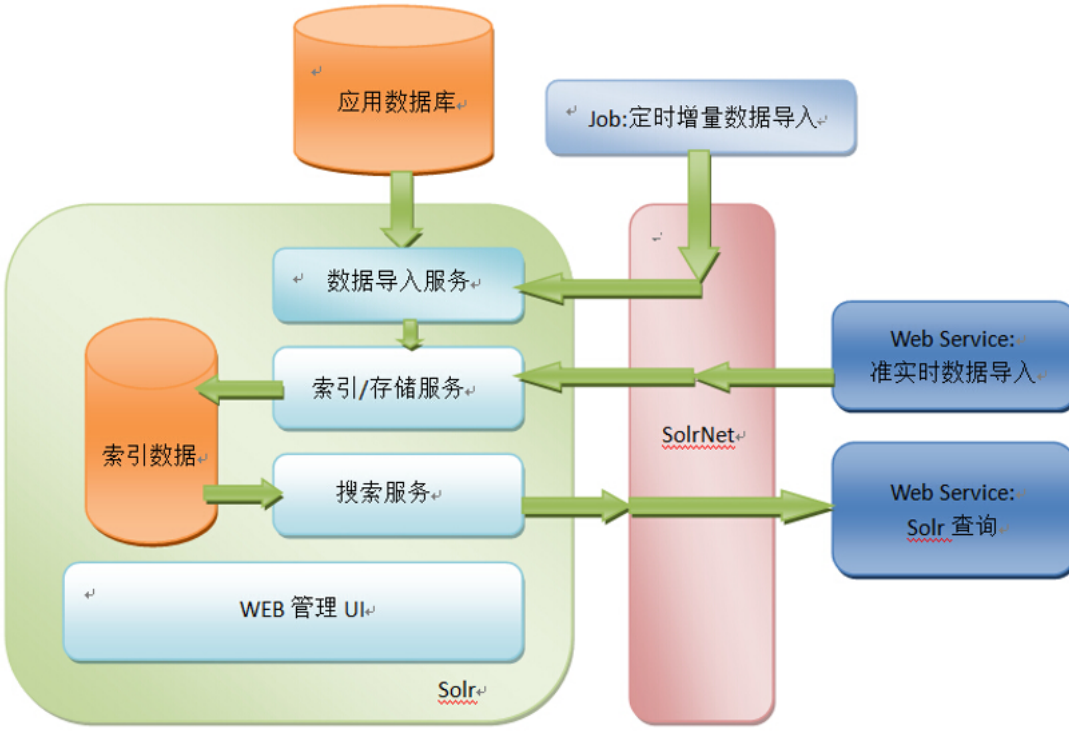
6、微服务框架MSA

微服务是细粒度业务行为的重用，需要与业务能力及业务阶段相匹配。微服务框架是实现微服务及分布式架构的关键组件，我们的微服务框架是基于开源ServiceStack来实现。它简单易用、性能好，文档自动生成、方便调试测试，调试工具Swagger UI、自动化接口测试工具SoapUI。微服务的接口开放采用我们自主研发的微服务网关，通过治理后台简单的配置即可。网关以NIO、IOCP的方式实现高并发，主要功能有鉴权、超时、限流、熔断、监控等，下图是Swagger UI调试工具。



7、搜索利器Solr

分库分表后的关联查询，大段文本的模糊查询，这些要如何实现呢？显然传统的数据库没有很好的解决办法，这时可以借助专业的检索工具。全文检索工具Solr不仅简单易用性能好，而且支持海量数据高并发，只需实现系统两边数据的准实时或定时同步即可。下图是Solr的工作原理。



8、更多工具

- 分布式协调器ZooKeeper：ZK工作原理、配置中心、Master选举、Demo，一篇足以；
- ORM框架：Dapper.NET语法简单、运行速度快，与数据库无关，SQL自主编写可控，是一款适合于互联网系统的数据库访问工具；

- 对象映射工具EmitMapper和AutoMapper：EmitMapper性能较高，AutoMapper易用性较好；
- IoC框架：控制反转IoC轻量级框架Autofac；
- DLL包管理：公司内部DLL包管理工具NuGet，可解决DLL集中存储、更新、引用、依赖问题；
- 发布工具Jenkins：一键编译、发布、自动化测试、一键回滚，高效便捷故障低。

二、架构篇——思想提升

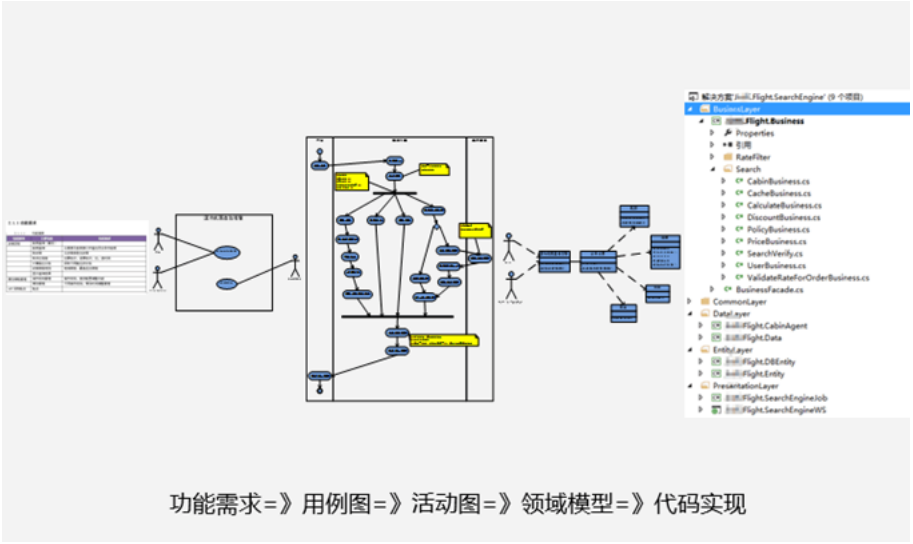
会使用以上框架并不一定能成为优秀的架构师，但一位优秀架构师一定会使用框架。架构师除了会使用工具外，还需要设计思想的提升和性能调优技能。此篇以真实项目为背景，思想方法追求简单有效，主要内容包括企业总体架构、单个项目架构设计、统一应用分层、调试工具WinDbg。

1、企业总体架构

当我们有了几百个上千个应用后，不仅仅需要单个项目的架构设计，还需要企业总体架构做顶层思考和指导。大公司与小商贩的商业思维是一样的，但大公司比较难看到商业全貌和本质。而小公司又缺乏客户流量和中间件的应用场景，中型公司则兼而有之，所以企业总体架构也相对好落地。企业总体架构需要在技术、业务、管理之间游刃有余地切换，它包括业务架构、应用架构、数据架构和技术架构。附档是一份脱敏感信息后的真实案例，有参考TOGAF标准。但内容以解决公司系统的架构问题为导向、以时间为主线，包括企业商务模型、架构现状、架构规划和架构实施。

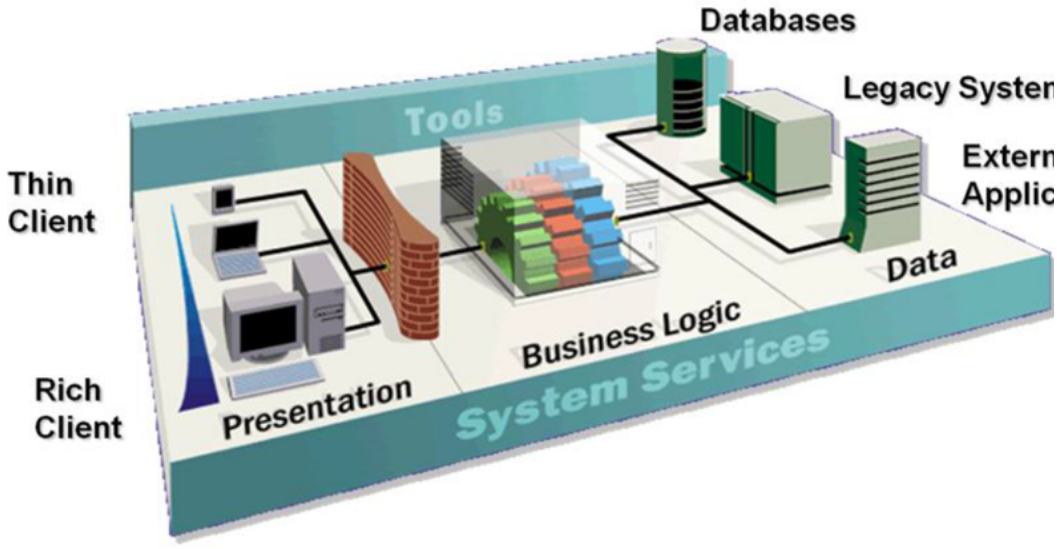
2、单个项目架构设计

单个项目的架构设计如同施工图纸，能直接指导工程代码的实施。上一环是功能需求，下一环是代码实施，这是架构设计的价值所在。从功能需求到用例，到用例活动图，到领域图、架构分层，到核心代码，它们之间环环相扣。做不好领域图可能源自没有做好用例活动图，因为用例活动图是领域图的上一环。关注职责、边界、应用关系、存储、部署是架构设计的核心，下图是具体案例参考。



3、统一应用分层

给应用分层这件事情很简单，但是让一家公司的几百个应用采用统一的分层结构，这可不是件简单的事情。它要做到可大可小、简单易用、支持多种场景，我们使用IPO方式：I表示Input、O表示Output、P表示Process，一进一出一处理。应用系统的本质就是机器，是处理设备，也是一进一出一处理，IPO方式相对于DDD而言更为简单实用。



4、调试工具WinDbg

生产环境偶尔会出现一些异常问题，而WinDbg或GDB就是解决此类问题的利器。调试工具WinDbg如同医生的听诊器，是系统生病时做问题诊断的逆向分析工具，Dump文件类似于飞机的黑匣子，记录着生产环境程序运行的状态。本文主要介绍了调试工具WinDbg和抓包工具ProcDump的使用，并分享一个真实的案例。N年前不知谁写的代码，导致每一两个月偶尔出现CPU飙高的现象。我们先使用ProcDump在生产环境中抓取异常进程的Dump文件，然后在不了解代码的情况下通过WinDbg命令进行分析，最终定位到有问题的那行代码。

```
0fe8e80c 3c2ccd38 System.Diagnostics.StackFrameHelper
0fe8e818 05f5aae8 System.String    in {0}:line {1}
0fe8e81c 05f5aad0 System.String    at
0fe8e820 3c2ccd24 System.Diagnostics.StackTrace
0fe8e824 3c2cd1b8 System.Diagnostics.StackFrame
0fe8e828 3c2ccd24 System.Diagnostics.StackTrace
0fe8e830 3c2ccd24 System.Diagnostics.StackTrace
0fe8e850 3c2ccd24 System.Diagnostics.StackTrace
0fe8e854 3c2c6410 System.InvalidOperationException
0fe8e864 3c2c6410 System.InvalidOperationException
0fe8e874 01e0d9f4                 .Service.Access.ProcessLogDataAccess
0fe8e880 3c2cccfc System.String    System.Data
0fe8e884 01e0d838 System.String    SLQ Server拒 绝
0fe8e888 01d901d0 System.String
0fe8e88c 01e0d7c4 System.String                 .Service.Access.DatabaseServiceBase
0fe8e890 01e0d93c System.String    0.0.0.0
0fe8e90c 01d901d0 System.String
0fe8e910 01e0d788 System.String    sql_server_exception
0fe8e914 01e0d864 System.String    {0}, {1}, {2}
0fe8e918 05f5aa54 System.String    ConnectionString 属 性 尚 未 初 始 化 。
0fe8e91c 3c2c6410 System.InvalidOperationException
0fe8e924 3c2c63d0 System.Data.SqlClient.SqlConnection
0fe8e928 01e0d9f4                 .Service.Access.ProcessLogDataAccess
0fe8e948 01e0d9f4                 .Service.Access.ProcessLogDataAccess
0fe8e958 3c2bebf0 System.Collections.Generic.List`1[[System.Data.IDbDataParameter, System.Data]]
0fe8e95c 01e0da00 System.String    INSERT INTO [dbo].[tbl_Interface_ProcessLog] (IKey,Username,ClientIP,Module,OrderNo,LogTyp
0fe8e964 3c2bebdc                 .Service.Access.ProcessLogDataAccess+<>c__DisplayClass3
0fe8e970 3c2bebf0 System.Collections.Generic.List`1[[System.Data.IDbDataParameter, System.Data]]
0fe8e974 01e0da00 System.String    INSERT INTO [dbo].[tbl_Interface_ProcessLog] (IKey,Username,ClientIP,Module,OrderNo,LogTyp
0fe8e98c 3c2bebdc                 .Service.Access.ProcessLogDataAccess+<>c__DisplayClass3
0fe8e9b0 3c2bef88 System.Threading.ThreadPoolWaitCallback
0fe8e9c0 3c2c61c8 System.Threading.ExecutionContext+ExecutionContextRunData
```

三、公共应用篇——业务与技术的结合

先工具再框架，然后架构设计，最后深入公共应用。这不仅是架构升级改造的正确路径，也是微服务架构实施的正确路径。公共应用因为与业务系统结合紧密，但又具有一定的独立性，所以一般自主开发，不使用开源也不方便开源。公共应用主要包括单点登录、企业支付网关、CTI通讯网关（短信邮件微信），此次分享单点登录和企业支付网关。

1、单点登录

应用拆分后总要合在一起，拆分是应用实施层面的拆分，合成是用户层面的合成，而合成必须解决认证和导航问题。单点登录SSO即只需要登录一次，便可到处访问，它是建立在用户系统、权限系统、认证系统和企业门户的基础上。我们的凭证数据Token使用JWT标准，以解决不同语言、不同客户端、跨WebAPI的安全问题。

2、企业支付网关

企业支付网关集中和封装了公司的各大支付，例如支付宝、财付通、微信、预付款等。它统一了业务系统调用各支付接口的方式，简化了业务系统与支付系统的交互。它将各种支付接口统一为支付、代扣、分润、退款、退分润、补差、转账、冻结、解冻、预付款等，调用时只需选择支付类型即可。企业支付网关将各大支付系统进行集中的设计、研发、部署、监控、维护，提供统一的加解密、序列化、日志记录，安全隔离。

在接下来的一段时间里，我会陆续推出此系列文章。因个人原因，发布顺序会根据本人的准备情况而作调整，敬请谅解。根据我们以往的经验，分享者主讲一个小时左右，业务研发就可以快速地进入项目实战。对于后面新加入的团队成员，也可通过WIKI自主快速学习。这是我们之前对自己的要求，尽量降低工具对人员的要求，简单实用、降低成本。文章中部分Demo采用C#语言，但到了框架或架构层面，与语言本身没有太多直接的关系。如RabbitMQ、Job、Redis和集中式日志ELK，它们服务端的部署是一样的，只是客户端语言版本稍有不同。所有Demo都可直接运行，服务地址及管理后台也可直接访问。因为部署在公有云，牵涉到成本费用的问题，我计划持续到明年3月底。以上这些小小的基础工