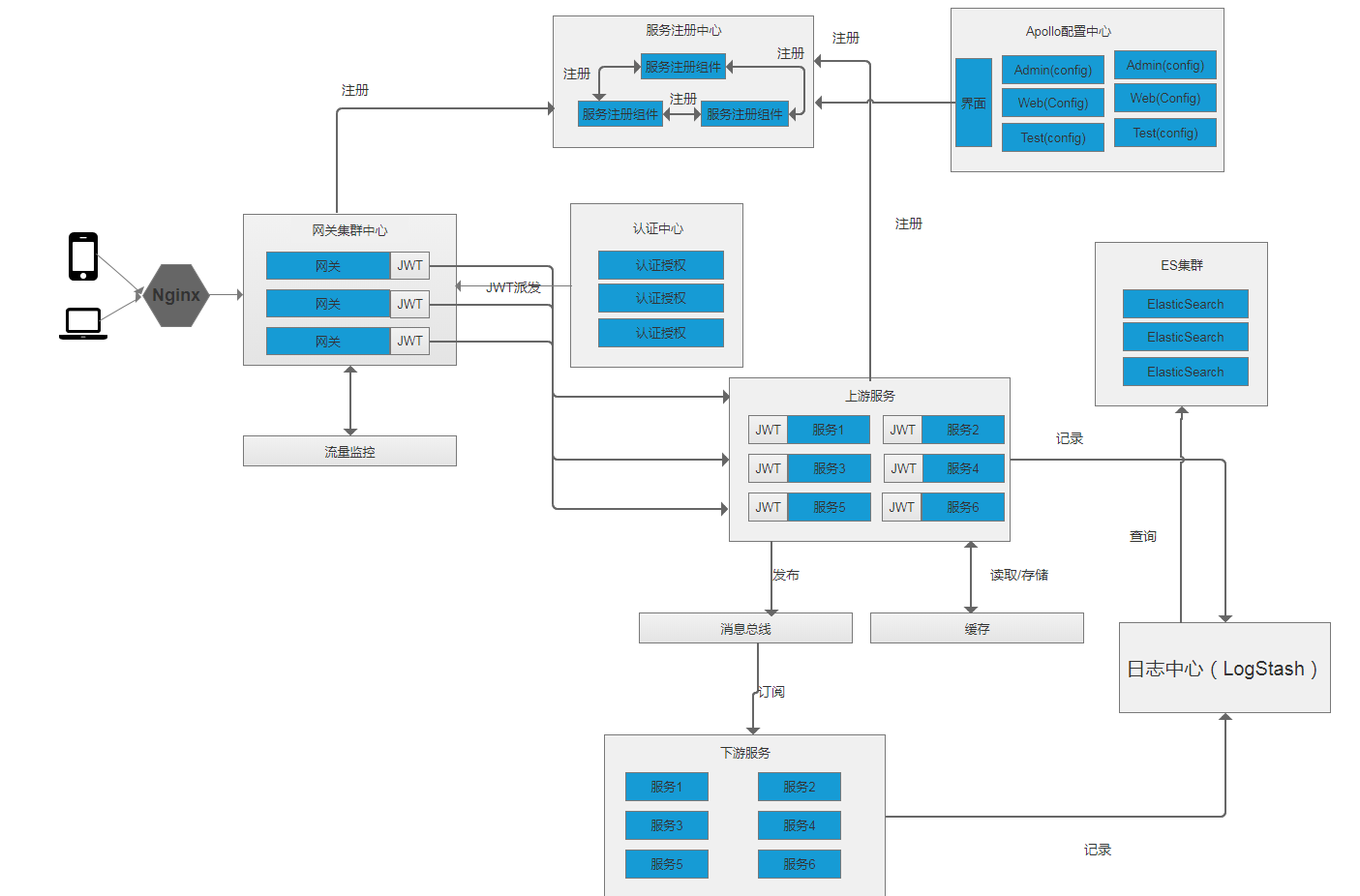
**智慧模具平台架构总览**

**智慧模具平台微服务架构说明**

相对于单体架构和SOA，它的主要特点是组件化、松耦合、自治、去中心化，体现在以下几个方面：

* 一组小的服务   
  服务粒度要小，而每个服务是针对一个单一职责的业务能力的封装，专注做好一件事情。
* 独立部署运行和扩展   
  每个服务能够独立被部署并运行在一个进程内。这种运行和部署方式能够赋予系统灵活的代码组织方式和发布节奏，使得快速交付和应对变化成为可能。
* 独立开发和演化   
  技术选型灵活，不受遗留系统技术约束。合适的业务问题选择合适的技术可以独立演化。服务与服务之间采取与语言无关的API进行集成。相对单体架构，微服务架构是更面向业务创新的一种架构模式。
* 独立团队和自治   
  团队对服务的整个生命周期负责，工作在独立的上下文中，自己决策自己治理，而不需要统一的指挥中心。团队和团队之间通过松散的社区部落进行衔接。

## 智慧模具平台系统底座

要包含以下功能：

* 日志和审计，主要是日志的汇总，分类和查询
* 监控和告警，主要是监控每个服务的状态，必要时产生告警
* 消息总线，轻量级的MQ或HTTP
* 注册发现
* 负载均衡
* 部署和升级
* 事件调度机制
* 资源管理，如：底层的虚拟机，物理机和网络管理

以下功能不是最小集的一部分，但也属于底座功能：

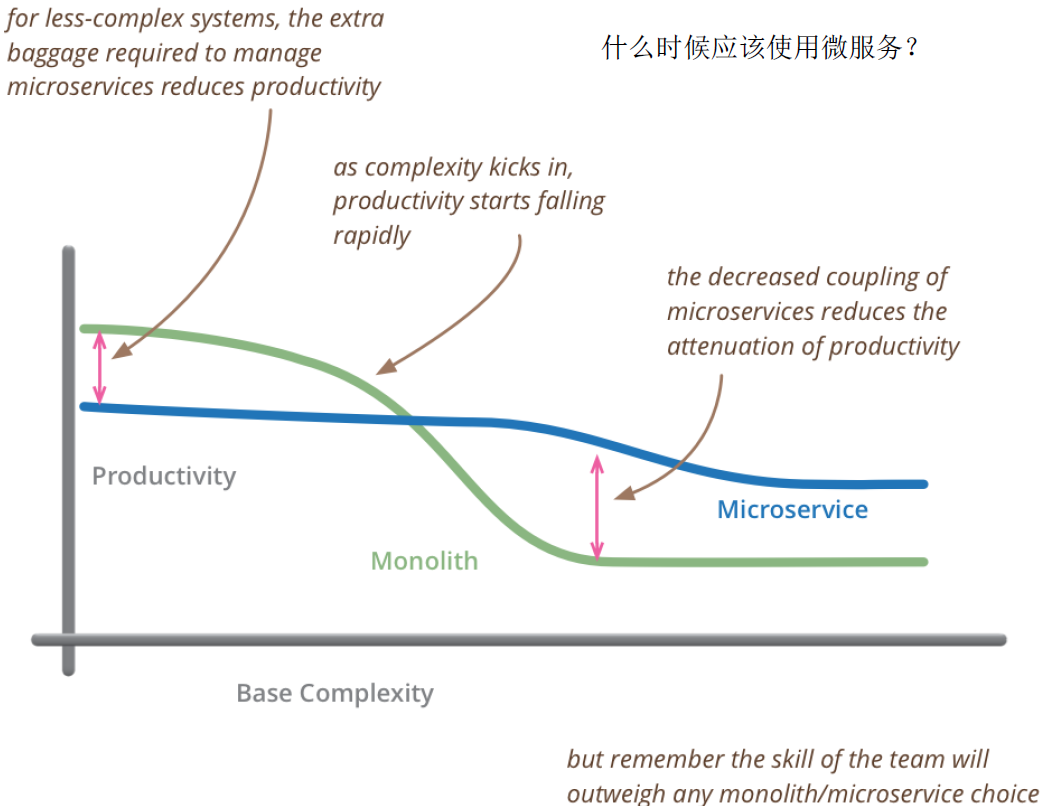
* 认证和鉴权
* 微服务统一代码框架，支持多种编程语言
* 统一服务构建和打包
* 统一服务测试
* 微服务CI/CD流水线
* 服务依赖关系管理
* 统一问题跟踪调试框架，俗称调用链
* 灰度发布
* 蓝绿部署

**优点**

* 每个微服务都很小，这样能聚焦一个指定的业务功能或业务需求。
* 微服务能够被小团队单独开发，这个小团队是2到5人的开发人员组成。
* 微服务是松耦合的，是有功能意义的服务，无论是在开发阶段或部署阶段都是独立的。
* 微服务能使用不同的语言开发。
* 微服务允许容易且灵活的方式集成自动部署，通过持续集成工具，如Jenkins, bamboo 。
* 一个团队的新成员能够更快投入生产。
* 微服务易于被一个开发人员理解，修改和维护，这样小团队能够更关注自己的工作成果。无需通过合作才能体现价值。
* 微服务允许你利用融合最新技术。
* 微服务只是业务逻辑的代码，不会和HTML,CSS 或其他界面组件混合。
* 微服务能够即时被要求扩展。
* 微服务能部署中低端配置的服务器上。
* 易于和第三方集成。
* 每个微服务都有自己的存储能力，可以有自己的数据库。也可以有统一数据库。

**架构的缺点**

* 微服务架构可能带来过多的操作。
* 需要DevOps技巧
* 可能双倍的努力。
* 分布式系统可能复杂难以管理。
* 因为分布部署跟踪问题难。
* 当服务数量增加，管理复杂性增加。



通过图我们可以得到在系统复杂的不高时我们选择单体架构更有利于我们系统开发 当系统复杂的到达一定程度后微服务的开发的开发复杂的和维护成本要远远低于单体架构的这就是我们为什么选择微服务来开发我们的项目

**负载均衡**

nginx的代理过程，就是将请求发送给nginx，然后将请求转发给后端服务器，后端服务器处理完毕之后将结果再发给nginx，nginx再把结果发送给客户端。后端服务器可在远程也可在本地，也可以是nginx服务器内部定义的其他虚拟主机。这些接收nginx转发的服务器被称为上游(upstream)使用nginx做代理的目的之一是扩展基础架构的规模。nginx可以处理大量并发连接，请求到来后，nginx可将其转发给任意数量的后台服务器进行处理，这等于将负载均衡分散到整个集群

**Ocelot作API网关**

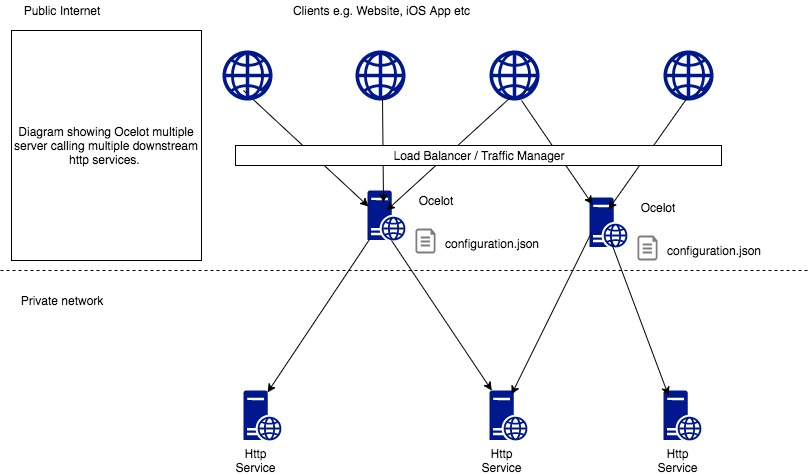
API网关是系统暴露在外部的一个访问入口。就像一个公司的门卫承担着寻址、限制进入、安全检查、位置引导、等等功能。从面向对象设计的角度看，它与外观模式类似。API网关封装了系统内部架构，为每个客户端提供一个定制的API。它可能还具有其它职责，如身份验证、监控、负载均衡、缓存、请求分片与管理、静态响应处理等等。

API网关方式的核心要点是，所有的客户端和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。通常，网关也是提供REST/HTTP的访问API。服务端通过API-GW注册和管理服务。

Ocelot在API网关实现上有什么优点呢？

Ocelot是一个用.NET Core技术实现并且开源的API网关技术。除此之外还有什么优点呢？那就是它强大的功能以及使用上的简单了。它的功能包括了：路由、请求聚合、服务发现、认证、鉴权、限流熔断、并内置了负载均衡器、Service Fabric、Skywalking等的集成。而且这些功能都只需要简单的配置即可完成。

**网关集群配置**

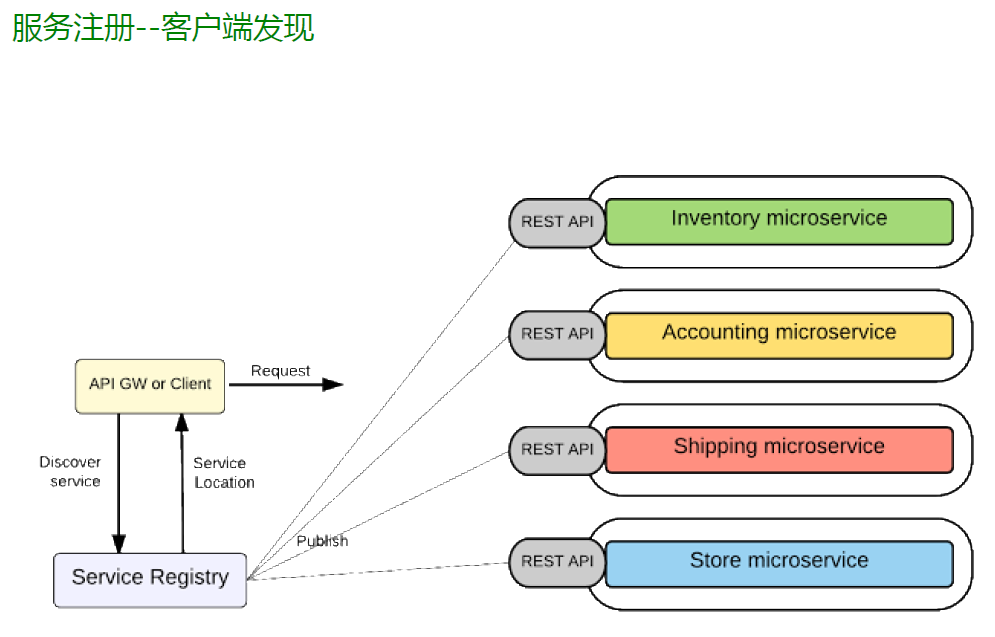


**Consul服务发现，服务治理**

**微服务治理**

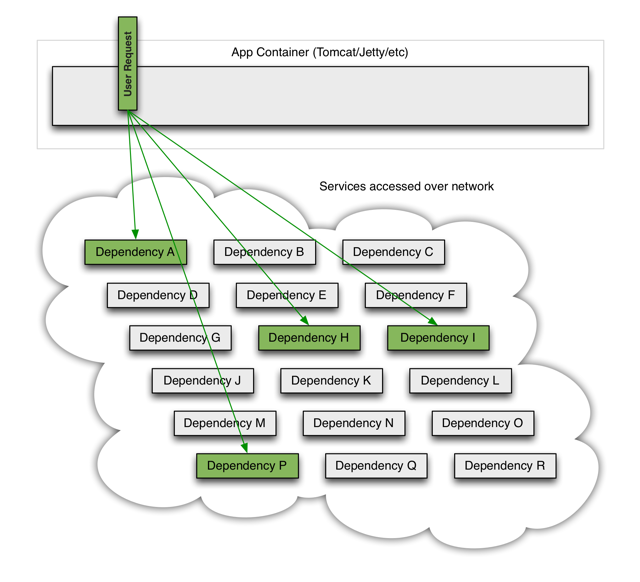
•按需伸缩   
–部署与监控运维成本   
•独立部署   
–机器数量与部署成本   
•业务独立   
–服务依赖、治理，版本管理、事务处理   
•技术多样性   
–环境部署成本、约定成本

•运行状态治理   
–监控、限流、SLA、LB、日志分析   
•服务注册与发现   
•部署   
–快速、复制、扩容   
–单机开发   
•调用   
–安全、容错、服务降级、调用延时



## 服务容错

     当企业微服务化以后，服务之间会有错综复杂的依赖关系，例如，一个前端请求一般会依赖于多个后端服务，技术上称为1 -> N扇出. 在实际生产环境中，服务往往不是百分百可靠，服务可能会出错或者产生延迟，如果一个应用不能对其依赖的故障进行容错和隔离，那么该应用本身就处在被拖垮的风险中。在一个高流量的网站中，某个单一后端一旦发生延迟，可能在数秒内导致所有应用资源(线程，队列等)被耗尽，造成所谓的雪崩效应(Cascading Failure)，严重时可致整个网站瘫痪。



服务注册、发现、负载均衡和健康检查，假定采用进程内LB方案，那么服务自注册一般统一做在服务器端框架中，健康检查逻辑由具体业务服务定制，框架层提供调用健康检查逻辑的机制，服务发现和负载均衡则集成在服务客户端框架中。

监控日志，框架一方面要记录重要的框架层日志、metrics和调用链数据，还要将日志、metrics等接口暴露出来，让业务层能根据需要记录业务日志数据。在运行环境中，所有日志数据一般集中落地到企业后台日志系统，做进一步分析和处理。

REST/RPC和序列化，框架层要支持将业务逻辑以HTTP/REST或者RPC方式暴露出来，HTTP/REST是当前主流API暴露方式，在性能要求高的场合则可采用Binary/RPC方式。针对当前多样化的设备类型(浏览器、普通PC、无线设备等)，框架层要支持可定制的序列化机制，例如，对浏览器，框架支持输出Ajax友好的JSON消息格式，而对无线设备上的Native App，框架支持输出性能高的Binary消息格式。

配置，除了支持普通配置文件方式的配置，框架层还可集成动态运行时配置，能够在运行时针对不同环境动态调整服务的参数和配置。

限流和容错，框架集成限流容错组件，能够在运行时自动限流和容错，保护服务，如果进一步和动态配置相结合，还可以实现动态限流和熔断。

管理接口，框架集成管理接口，一方面可以在线查看框架和服务内部状态，同时还可以动态调整内部状态，对调试、监控和管理能提供快速反馈。Spring Boot微框架的Actuator模块就是一个强大的管理接口。

统一错误处理，对于框架层和服务的内部异常，如果框架层能够统一处理并记录日志，对服务监控和快速问题定位有很大帮助。

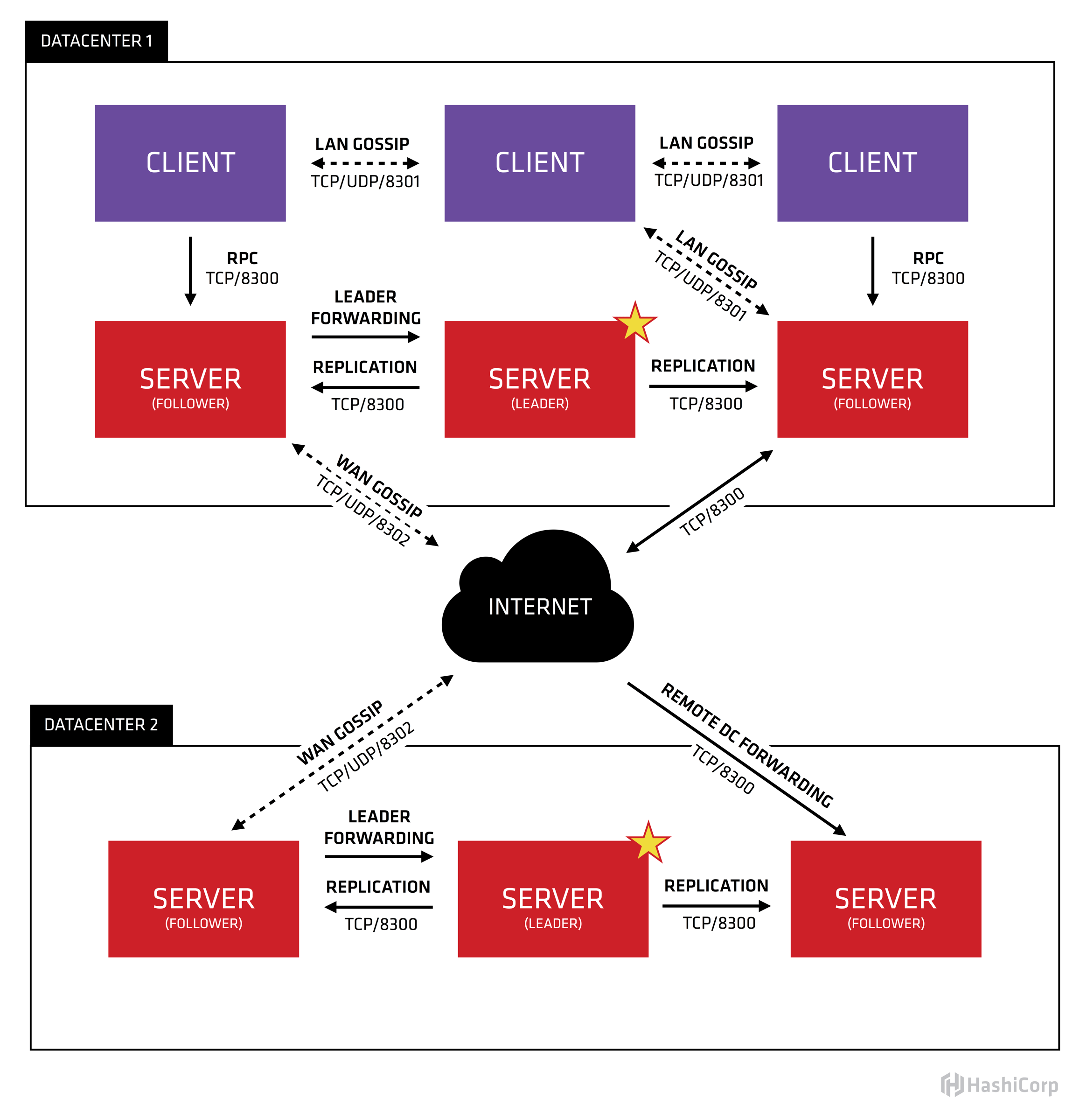
安全，安全和访问控制逻辑可以在框架层统一进行封装，可做成插件形式，具体业务服务根据需要加载相关安全插件。

文档自动生成，文档的书写和同步一直是一个痛点，框架层如果能支持文档的自动生成和同步，会给使用API的开发和测试人员带来极大便利。Swagger是一种流行Restful API的文档方案。

consul的三个主要应用场景：[服务发现](https://www.consul.io/discovery.html" \o "consul usage: service discovery)、[服务隔离](https://www.consul.io/segmentation.html)、[服务配置](https://www.consul.io/configuration.html)。

[服务发现](https://www.consul.io/discovery.html)场景中，consul作为注册中心，服务地址被注册到consul中以后，可以使用consul提供的dns、http接口查询，consul支持health check。

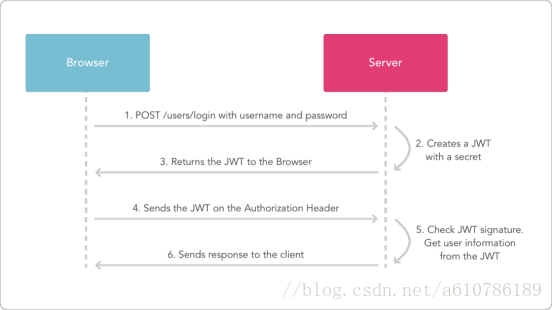
[服务隔离](https://www.consul.io/segmentation.html)场景中，consul支持以服务为单位设置访问策略，能同时支持经典的平台和新兴的平台，支持tls证书分发，service-to-service加密



* 每个被注册到consul中的node上，都部署一个consul agent，这个agent负责对本地的服务进行监控检查，以及将查询请求转发给consul server。
* consul server负责存放、备份数据(使用raft协议保证一致性)，通常要有多台形成集群，选举出一个leader。
* 查询服务地址的时候，可以直接向consul server发起查询，也可以通过consul agent查询，后者将转发给consul server。
* 如果是多数据中心，每个数据中心部署一组consul server。跨数据中心查询通过本数据中心的consul server进行。
* 注意：多数据中心的时候，不同数据中心的consul server之间不会同步key-value数据。

**Jwt登录授权**

JWT (JSON Web Token) 是一套特别流行于分布式系统采用的授权标准 ，在采用加密等手段保证安全高效的同时，其基于JSON做为权限认证令牌的特性可以使其携带诸多非敏感数据，其所携带的数据可以确保授权的灵活高效；



JWT的过程，由客户端登录系统时登录成功，系统将生成相应的json数据编码并加密生成签名，形成一个 头部json Base64编码 + 荷载(playload) json Base64 + 对(头部json+ playload json)HA256加密生成的签名 的三段编码，各段编码以 “ . ” 隔开；注意第一二段(header & playload)采用Base64 编码 并非不可逆的加密算法，其仅仅起到编码作用，任何第三方都可以用Base64解码器对其数据进行解析还原，这是很多人存在的一个误区，第三段才是真实的不可逆加密采用HA256或MD5等都可以后面讲具体讲解加密流程及为何通过该段内容可以获知jwt真实性；

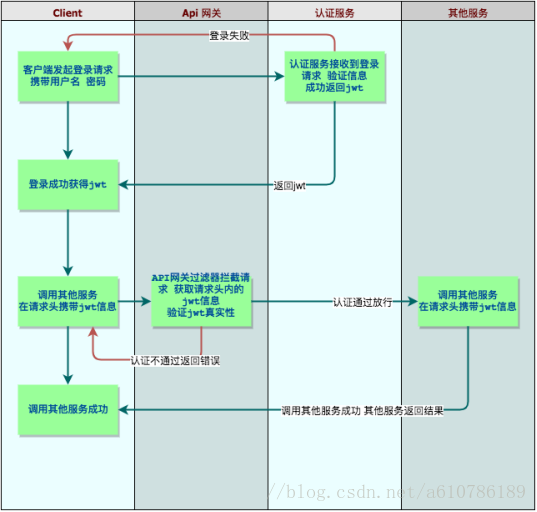
1.client 发起登录请求 携带 username password 等信息

2.server 接收用户登录请求，验证账户正确，正确返回一条jwt编码

3.客户端接收到jwt编码，之后每次发起请求在请求头中加入一条k/v值

4.server接收到client请求，获取到请求头中的jwt信息，对jwt进行校验若签名核对成功，则为server自身发出并未有第三方修改过真实可靠的信息，若为正确信息，对jwt中的header与playload进行Base64解析获取原json，读取json 中的信息获取用户的UserId、Role、jwt有效期等信息，在此时可以对用户访问的资源(路径)进行鉴权是否匹配该用户权限（该部分将在下一篇文章中提到）；

5.若jwt合法，且符合server的鉴权规则那么返回给客户端相应请求；



**ElasticSearch 日志中心**

ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于[云计算](https://baike.baidu.com/item/%E4%BA%91%E8%AE%A1%E7%AE%97/9969353)中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

我们建立一个网站或应用程序，并要添加搜索功能，但是想要完成搜索工作的创建是非常困难的。我们希望搜索解决方案要运行速度快，我们希望能有一个零配置和一个完全免费的搜索模式，我们希望能够简单地使用JSON通过HTTP来索引数据，我们希望我们的搜索服务器始终可用，我们希望能够从一台开始并扩展到数百台，我们要实时搜索，我们要简单的多租户，我们希望建立一个云的解决方案。因此我们利用Elasticsearch来解决所有这些问题及可能出现的更多其它问题

分布式系统会产生大量日志数据用传统的写日志方式已不适合现有程序，传统日志方式查询缓慢，系统追踪日志麻烦而ElasticSearch能很好解决现有问题

Elasticsearch是基于倒排索引的 能在TB级数据在毫秒级就能返回

当用户使用Elasticsearch时进行搜索时，Elasticsearch就自动帮他分好词了

Elasticsearch支持全文搜索和相关度评分。这样在返回结果就会根据分数由高到低排列。分数越高，意味着和查询语句越相关。

Kibana可视化界面完美支持Elasticsearch。研发人员能够在上面快速地进行概念验证，分析结果，提高开发效率。

**Apollo 配置中心**

Apollo（阿波罗）是携程框架部门研发的分布式配置中心，能够集中化管理应用不同环境、不同集群的配置，配置修改后能够实时推送到应用端，并且具备规范的权限、流程治理等特性，适用于微服务配置管理场景。

服务端基于Spring Boot和Spring Cloud开发，打包后可以直接运行，不需要额外安装Tomcat等应用容器。

Java客户端不依赖任何框架，能够运行于所有Java运行时环境，同时对Spring/Spring Boot环境也有较好的支持。

.Net客户端不依赖任何框架，能够运行于所有.Net运行时环境。

Apollo特性

统一管理不同环境、不同集群的配置

Apollo提供了一个统一界面集中式管理不同环境（environment）、不同集群（cluster）、不同命名空间（namespace）的配置。

同一份代码部署在不同的集群，可以有不同的配置，比如zk的地址等

通过命名空间（namespace）可以很方便的支持多个不同应用共享同一份配置，同时还允许应用对共享的配置进行覆盖

配置修改实时生效（热发布）

用户在Apollo修改完配置并发布后，客户端能实时（1秒）接收到最新的配置，并通知到应用程序。

版本发布管理

所有的配置发布都有版本概念，从而可以方便的支持配置的回滚。

灰度发布

支持配置的灰度发布，比如点了发布后，只对部分应用实例生效，等观察一段时间没问题后再推给所有应用实例。

权限管理、发布审核、操作审计

应用和配置的管理都有完善的权限管理机制，对配置的管理还分为了编辑和发布两个环节，从而减少人为的错误。

所有的操作都有审计日志，可以方便的追踪问题。

客户端配置信息监控

可以方便的看到配置在被哪些实例使用

提供Java和.Net原生客户端

提供了Java和.Net的原生客户端，方便应用集成

支持Spring Placeholder, Annotation和Spring Boot的ConfigurationProperties，方便应用使用（需要Spring 3.1.1+）

同时提供了Http接口，非Java和.Net应用也可以方便的使用

提供开放平台API

Apollo自身提供了比较完善的统一配置管理界面，支持多环境、多数据中心配置管理、权限、流程治理等特性。

不过Apollo出于通用性考虑，对配置的修改不会做过多限制，只要符合基本的格式就能够保存。

在我们的调研中发现，对于有些使用方，它们的配置可能会有比较复杂的格式，如xml, json，需要对格式做校验。

还有一些使用方如DAL，不仅有特定的格式，而且对输入的值也需要进行校验后方可保存，如检查数据库、用户名和密码是否匹配。

对于这类应用，Apollo支持应用方通过开放接口在Apollo进行配置的修改和发布，并且具备完善的授权和权限控制

部署简单

配置中心作为基础服务，可用性要求非常高，这就要求Apollo对外部依赖尽可能地少

目前唯一的外部依赖是MySQL，所以部署非常简单，只要安装好Java和MySQL就可以让Apollo跑起来

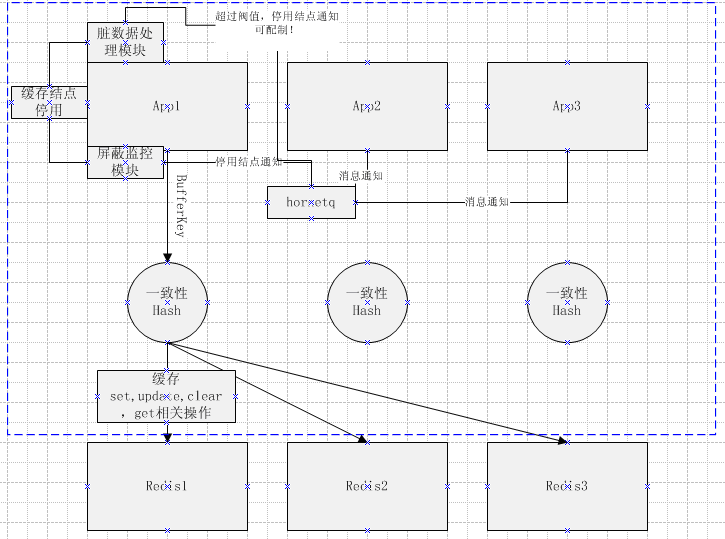
Apollo还提供了打包脚本，一键就可以生成所有需要的安装包，并且支持自定义运行时参数

**Redis缓存**

1.redis丰富的数据结构,其hash,list,set以及功能丰富的String的支持，对于实际项目中的使用有很大的帮忙。（可参考官网redis.io）

2.redis单点的性能也非常高效（利用项目中的数据测试优于memcache）.

1架构设计



2.分布式实现

通过key做一致性哈希，实现key对应redis结点的分布。

一致性哈希的实现：

   hash值计算：通过支持MD5与MurmurHash两种计算方式，默认是采用MurmurHash，高效的hash计算。

    一致性的实现：通过Java的TreeMap来模拟环状结构，实现均匀分布

3.client的选择

对于jedis修改的主要是分区模块的修改，使其支持了跟据BufferKey进行分 区，跟据不同的redis结点信息，可以初始化不同的ShardInfo，同时也修改了JedisPool的底层实现，使其连接pool池支持跟据 key,value的构造方法，跟据不同ShardInfos，创建不同的jedis连接客户端，达到分区的效果，供应用层调用

4.模块的说明

脏数据处理模块，处理失败执行的缓存操作。

       屏蔽监控模块，对于jedis操作的异常监控，当某结点出现异常可控制redis结点的切除等操作。

整个分布式模块通过hornetq，来切除异常redis结点。对于新结点的增加，也可以通过reload方法实现增加。（此模块对于新增结点也可以很方便实现）

对于以上分布式架构的实现满足了项目的需求。另外使用中对于一些比较重要用途的缓存数据 可以单独设置一些redis结点，设定特定的优先级。另外对于缓存接口的设计，也可以跟据需求，实现基本接口与一些特殊逻辑接口。对于cas相关操作，以 及一些事物操作可以通过其watch机制来实现。（参考我以前写的redis事物介绍）

以上是基于redis分布式架构的介绍！但是应用中读写都是在一起的。 相关写是在应用操作后flush或者update的，有一定的耦合。为了使读写分离，以及缓存模块跟应用的耦合更小，考虑使用mysql binlog来刷新缓存。以下是基于binlog刷新可性行分析以及实现过程中需要注意的地方。

[**事件总线,分布式事务解决方案：CAP**](https://www.cnblogs.com/savorboard/p/cap.html)

CAP 是一个在分布式系统中（SOA，MicroService）实现事件总线及最终一致性（分布式事务）的一个开源的 C# 库，她具有轻量级，高性能，易使用等特点。

你可以轻松的在基于 .NET Core 技术的分布式系统中引入CAP，包括但限于 ASP.NET Core 和 ASP.NET Core on .NET Framework。

CAP 以 NuGet 包的形式提供，对项目无任何入侵，你仍然可以以你喜爱的方式来构建分布式系统。

CAP 具有 Event Bus 的所有功能，并且CAP提供了更加简化的方式来处理EventBus中的发布/订阅。

CAP 具有消息持久化的功能，也就是当你的服务进行重启或者宕机时，她可以保证消息的可靠性。

CAP 实现了分布式事务中的最终一致性，你不用再去处理这些琐碎的细节。

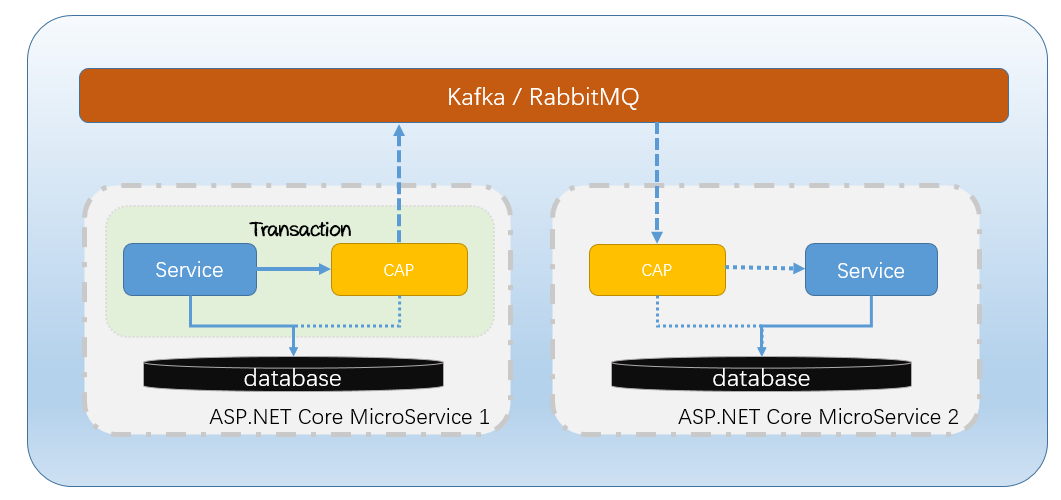
CAP 提供了基于 Microsoft DI 的 API 服务，她可以和你的 ASP.NET Core 系统进行无缝结合，并且能够和你的业务代码集成支持强一致性的事务处理。

CAP 是开源免费的。CAP基于MIT协议开源，你可以免费的在你的私人或者商业项目中使用，不会有人向你收取任何费用。

CAP 同时支持使用 RabbitMQ，Kafka，Azure Service Bus 等进行底层之间的消息发送，你不需要具备这些消息队列的使用经验，仍然可以轻松的集成到项目中。

CAP 目前支持使用 Sql Server，MySql，PostgreSql，MongoDB 数据库的项目。

CAP 同时支持使用 EntityFrameworkCore 和 ADO.NET 的项目，你可以根据需要选择不同的配置方式

下面是CAP在系统中的一个不完全示意图****

[**弹性和瞬态故障处理**](https://www.cnblogs.com/CreateMyself/p/7589397.html)

Polly是一种.NET弹性和瞬态故障处理库，允许我们以非常顺畅和线程安全的方式来执诸如行重试，断路，超时，故障恢复等策略。 Polly针对对.NET 4.0，.NET 4.5和.NET Standard 1.1以及.NET Core实现，该项目作者现已成为.NET基金会一员

Polly解决了哪些问题

有些故障不是必然会发生的，而是偶然可能会发生的，比如网络偶尔会突然出现不稳定或无法访问这种故障为应对这种情况我们需要灵活处理故障问题，polly能很好的处理好这种偶发性情况。如果我们遇到这种偶发性问题 polly可以指定下次5s，10s，20s，进行尝试，如果多次尝试都不能正常运行，否则认为网络不正常，则抛出异常，进行日志记录。

**Jekins自动化部署**

Jenkins是一个开源的、可扩展的持续集成、交付、部署（软件/代码的编译、打包、部署）的基于web界面的平台。允许持续集成和持续交付项目，无论用的是什么平台，可以处理任何类型的构建或持续集成

选择jekins理由

是所有CI产品中在安装和配置上最简单的。

基于Web访问，用户界面非常友好、直观和灵活，在许多情况下，还提供了AJAX的即时反馈。

Jenkins是基于Java开发的(如果你是一个Java开发人员，这是非常有用的)，但它不仅限于构建基于Java的软件。

 Jenkins拥有大量的插件。这些插件极大的扩展了Jenkins的功能；它们都是开源的，而且它们可以直接通过web界面来进行安装与管理

Jekin工作原理

1. 开发者检入代码到源代码仓库。

2. CI系统会为每一个项目创建了一个单独的工作区。当预设或请求一次新的构建时，它将把源代码仓库的源码存放到对应的工作区。

3. CI系统会在对应的工作区内执行构建过程。

4. （配置如果存在）构建完成后，CI系统会在一个新的构件中执行定义的一套测试。完成后触发通知(Email,RSS等等)给相关的当事人。

5. （配置如果存在）如果构建成功，这个构件会被打包并转移到一个部署目标(如应用服务器)或存储为软件仓库中的一个新版本。软件仓库可以是CI系统的一部分，也可以是一个外部的仓库，诸如一个文件服务器或者像Java.net、 SourceForge之类的网站。

6. CI系统通常会根据请求发起相应的操作，诸如即时构建、生成报告，或者检索一些构建好的构件