## Ns

Version 1.0.0

目录

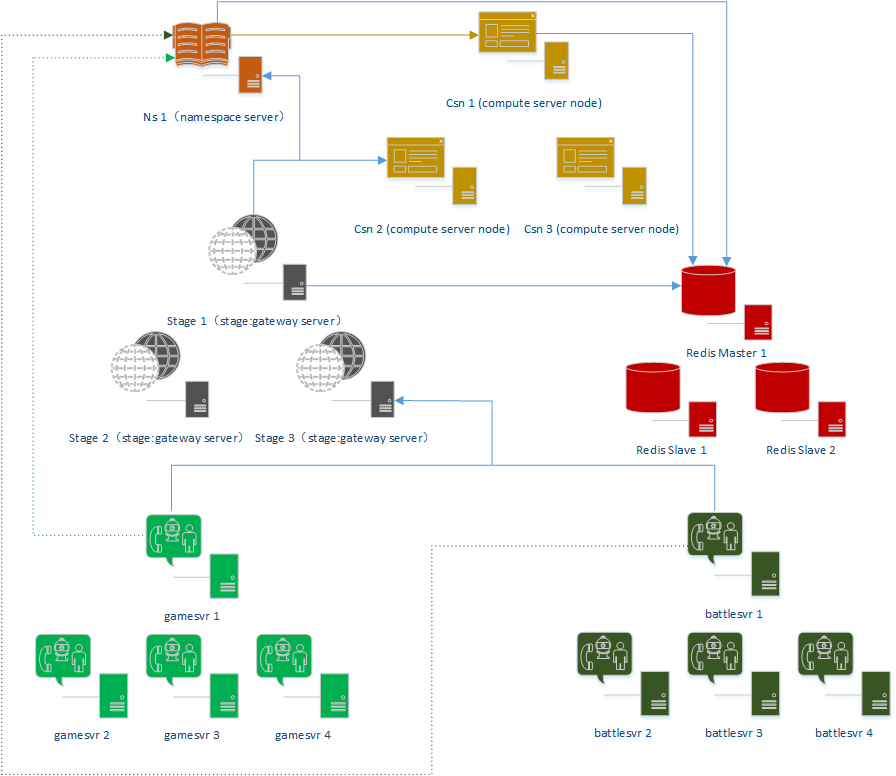
[1. 概述 1](#_Toc14607)

[2. 基础框架 1](#_Toc20340)

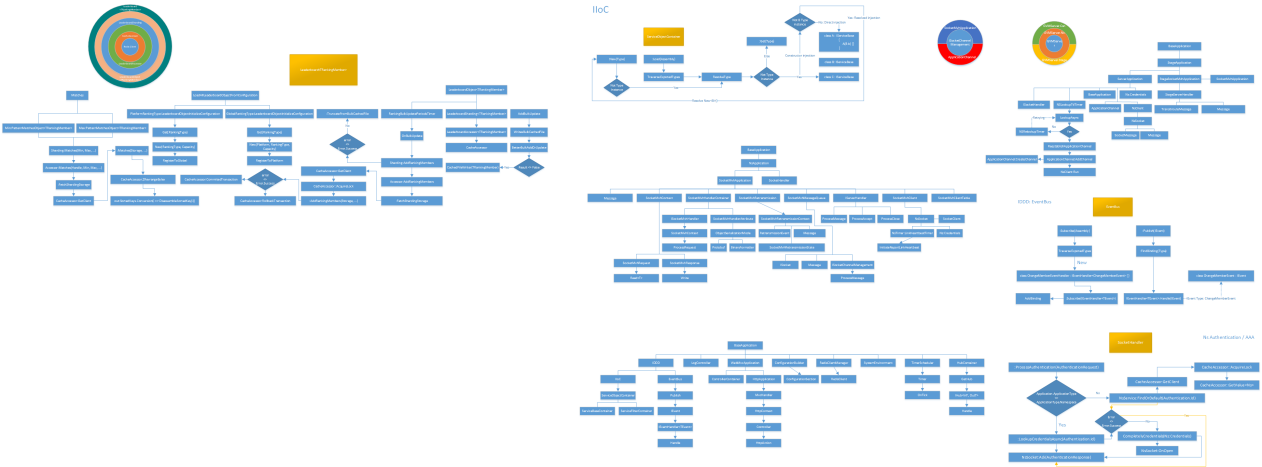
[3. 发布 8](#_Toc29237)

[4. 部署 9](#_Toc15512)

1. 概述
   1. 服务器结构图



* 1. 基础架构图



1. 基础框架
   1. GVMServer
      1. DDD
         1. ServiceObjectContainer

领域服务对象容器是一种IoC[控制反转]管理容器，提供类型解析构造注入（含注入器）但它不仅仅提供IoC容器的基础体系的同时，还允许支持AOP动态代理服务过滤器，用于控制服务“具体到接口”请求放行与事务或日志处理。

IoC容器提供多组载入类型实例的办法，但建议仅直接加载程序集模块（Assembly），这样可以简化大量代码与服务器类型构造的直接或间接依赖，与具体类型实例构造器及签名依赖，由IoC容器解析器进行注入并且构造服务实例，你无须显示手动构造任何服务及依赖项。

2.1.1.2. EventBus

领域事件总线是一种的事件分离与驱动器，人们只须要把感兴趣的“事件源”订阅到“事件总线”而不必关心事件源从何而来，从何而去；事件处理器与事件源之间都仅仅依赖于“事件总线”。

对于“事件处理器”而言，它不需要知道“事件源”从何而来，何时发生，因为两者本身便没有任何的联系，反之亦然。

事件总线提供对于程序集模块（Assembly）的事件处理器与事件源类型进行订阅的方法，其目的与IoC相对类似为了减少用户代码之间的依赖与圈复杂度。

这带来了显著的好处，层与层之间的分解变得不在明显，架构可以具有更好的扩展性与灵活性这其实是属于Code First类型的一种设计理念。

这不同于“Union”，“三层架构（视图/业务/数据）”，“四层架构（视图/业务/聚合/数据）”、“五层架构（视图/业务/服务/聚合/数据）”。

2.1.1.3. HubContainer

领域集线器容器是一种集合条件组分离器，其作用为分解例如“责任链模式”、“状态模式”、“状态机器”与实际类型之间依赖，把依赖从具象类型上面转移到“集线器”与“集线器容器”本身。

* + 1. Collection

提供扩展集合容器，例如线程安全的“List<T>”或者“Set<T>”容器等。

* + 1. Hooking

Interceptor.NET 提供跨平台的.NET托管函数拦截器，允许运行时改变.NET托管函数的执行流程，详情参考：

<https://blog.csdn.net/liulilittle/article/details/105934959>

* + 1. IO

提供与I/O类型相关操作的辅助功能模块，例如文件辅助、目录辅助、LZ77算法压缩流辅助等。

* + 1. Linq

提供线程安全的常用的Linq扩展函数，减少用户使用Linq扩展函数库时大量编写无意义的判空检查。

* + 1. Log

提供日志控制、统计控制器的服务对象。

* + 1. Cryptography

提供加密算法工具。

* + 1. Threading

提供计时器、可重入自旋锁（T/TASLock）

* + 1. W3Xiyou

西游游戏平台对接空间，含GM/充值。

* + 1. Diagnostics

软件契约

* + 1. Database

数据库服务，例如数据库连接池、Mysql适配器、数据表格网关

* + 1. AOP

提供动态代理服务模块，例如数据表格（DataTable）动态模型代理，其次是数据表格与预编程静态定义模型的映射（ORM）。

* + 1. Net

提供基础的可靠实现的套接字客户端与侦听器通信层，同时提供与.ASP.NET接口类型的Web（C/S）框架类型库。

* + 1. Converter

提供二进制转换工具类。

* + 1. Sorting

提供排序算法的支持。

* + 1. Valuetype

提供值类型安全格式化，源类型工具。

* + 1. Serialization

提供序列化算法的基础类库，例如“二进制序列化【BinaryFormatter】”、“Json序列化”、“Xml序列化”、“Google Protobuf V2/V3序列化”、“Ssx/C++ BinaryFormatter 静态二进制序列化”等。

* + 1. Planning

提供服务器策划档（Xml格式）映射模型的工具服务模块。

* 1. GVMServer.Ns
     1. Collections

提供分布式环境下原子性安全的集合容器，例如分布式的“Set”、“Dictionary”等；同时提供分布式环境下的“Monitor（监视同步对象）”。

* + 1. Deployment
       1. ApplicationChannel

服务器应用通道管理器，处理交换来自多个不同服务器之间的消息；驱动服务器应用虚拟套接字（例如：gamesvr会被映射为一个虚拟的应用套接字通道）；与C/C++版本STAGE原理是类似的。

* + - 1. ApplicationVirtualSocket

服务器应用虚拟套接字，提供虚拟的套接字层屏蔽物理套接字链路，减少服务器对于管理物理套接字的的性能开销，保证链接的有效性与防止物理套接字句柄泄漏；详情参考C/C++版本STAGE文档。

* + - 1. PerformanceCounter

性能计数器，仅支持Windowws32版本；目前服务器代码内未应用到此基础类型；确保代码跨平台时请不要利用此类型。

* + - 1. ServerApplication

服务器应用实例，它确保可以自动与自任何一个有效的节点STAGE服务器之间建立最优先通道；服务器之间为网状结构，其次会逐步的建立与多个不同STAGE服务器之间建立可靠信道。

一个服务器应用首先会向NS（名称节点）服务器进行本节点备案；远过程调用服务的“PutOnRecord”接口，在确保与NS服务器建立了正确的备案时，在询问Ns（名称节点）服务器分配到有效的STAGE服务器时【使用：Lookup 接口】。

* + - 1. SystemEnvironment

提供跨平台的服务器环境信息获取方法，例如获取CPU利用率、内存使用率等；仅支持Linux、Windows系统，须知暂不支持MacOS。

* + 1. Enum
       1. Commands

定义公共NS框架与不同类型应用服务器之间的“套接字通信层”网络命令代码。

* + - 1. Error

定义公共NS框架的错误代码，即任何与NS服务或接口相关的返回的错误代码，建议统一定义在此处。

* + - 1. Event

定义的NS框架通用事件源，例如“策划档加载”、“策划档加载到集群节点”、“策划当预加载”、“释放全部策划档”、“读入全部策划档”等。

* + - 1. EventHandler

定义的NS框架一些事件源的处理器。

* + - 1. Functional

提供NS框架基础的功能模块，

例如：

1. 缓存访问者
2. NS/Web控制器
   1. Lookup 以负载均衡的形式查找约定类型的实例
   2. LookupAll 查找全部约定类型的实例
   3. PutOnRecord 服务器应用类型备案（NS凭证由此接口生成）
   4. Get 检索约定条件的NS凭证信息
3. NsLoadbalancing

负载均衡采样记录服务，NS套接字每间隔一段时间（1秒）向服务器采样上报NS套接字所在服务器的“内存”、“CPU”负载等相关信息；告知NS服务器。（分布式的）

1. NsPlanningConfiguration

策划配置档案服务，提供从本地主机载入配置档到本地服务器，（加载远程主机是由事件处理器提供的服务）同时提供从集群之中访问策划配置档的能力。（分布式的）

1. NsService

NS服务器核心服务，提供对于各个动态接入服务器的NS凭证的生成备案的数据查询与存储。（分布式的）

* + - 1. Integers

提供Int128、Int256超大整数类型运算的支持，.NET运行时含默认“Int16、Int32、Int64”等类型。

* + - 1. Model

定义公共模型的空间，默认存在“服务器节点采样”模型。

* + - 1. Mq

提供RabbitMQ消息队列中间件交互的包装器及连接管理器的模块。

* + - 1. Net

提供NS套接字通信层基础框架。

* + - * 1. Handler

定义服务器处理器、套接字处理器；并且实现公共的套接字处理器基础类型（核心功能：鉴权处理、心跳采样等、发起心跳请求、发起鉴权请求等）。

* + - * 1. Model

定义公共NS框架底层的网络交换消息模型，例如鉴权、心跳、游戏服间路由、节点间路由等。

* + - * 1. Mvh

面向NS框架套接字层的Mvh（模型、视图、处理器）框架类库，管理套接字请求的处理器路由，套接字上下文封装、重传等一系列的功能。

* + - 1. BaseApplication

整个NS框架提供面向用户提供的可扩展基础应用实例类型，它管理着整个NS服务器之间的数据交换与各类型NS框架提供的基础服务的生命周期。

* 1. GVMServer.Stage
     1. StageServerHandler

基于NS框架的扩展，处理来自游戏服务器或者CSN节点之间的数据交换；具有轻微的负载均衡机制。

* + 1. StageSocketHandler

基于NS框架的扩展，主要目的用于STAGE服务器进行负载均衡采样的记录。

* 1. GVMServer.Csn
     1. EventHandler

主要实现“加载全部策划配置档”的“事件处理器”，因为只有“CSN”节点是需要加载策划配置档案的。

* + 1. Handler
       1. AllServerAverageRankingUpdateHandler

全服平均排名上传套接字层处理器，提供对于“指定平台全服或全区全服平均排名的接口服务”，目前约定“战力排行”、“等级排行”前50名平均值需要进行“全区全服全平台/指定平台全区全服”战力排名。（运行在完全分布式环境下的。）

* + - 1. MatchAvavailableServerHandler

匹配可用服务器接口处理器，提供欲查询的服务器应用类型，CSN返回约定类型首选在线可用的服务器。

* + - 1. MatchRankingGamePlayersHandler

匹配排名游戏玩家处理器，允许查询“指定平台指定服”上榜集群的玩家的数据，也支持查询“全平全区全服”上榜玩家排名数据。

QueryAllAvailableServersHandler  
 查询全部可用服务器接口处理器，提供欲查询的服务器应用类型，CSN返回约定类型在线可用的服务器列表。

* + - 1. RankingMemberInformationBulkUpdateHandler

游戏服务器玩家排名数据批量上传接口，目前仅支持“战力排行”、“等级排行”指定平台/全部平台/全区全服。

* + 1. Modules

目前仅提供“Netif”网络接口模块，其模块提供关于广播、发送、或许不在时允许重传的虚拟套接字等功能。

* + 1. Planning

定义跨服节点使用的策划档模型，目前仅有“活动配置档”；须知配置档模型需要开发人员手动定义。

* + 1. Protobuf

定义与游戏服务器之间交换的基于Google Protobuf V2/V3协议的请求、响应、通知等数据模型。

* + 1. NodeEnvironment

定义分布式容器、分布式监视锁（Monitor）的实例的根ID，目前CSN节点仅仅定义了以下几个根节点项；可以按需求调整根节点之间范围；

public const long WorkingActivitiesSet = 1000000000000000; // 分布式工作活动Set 容器根ID

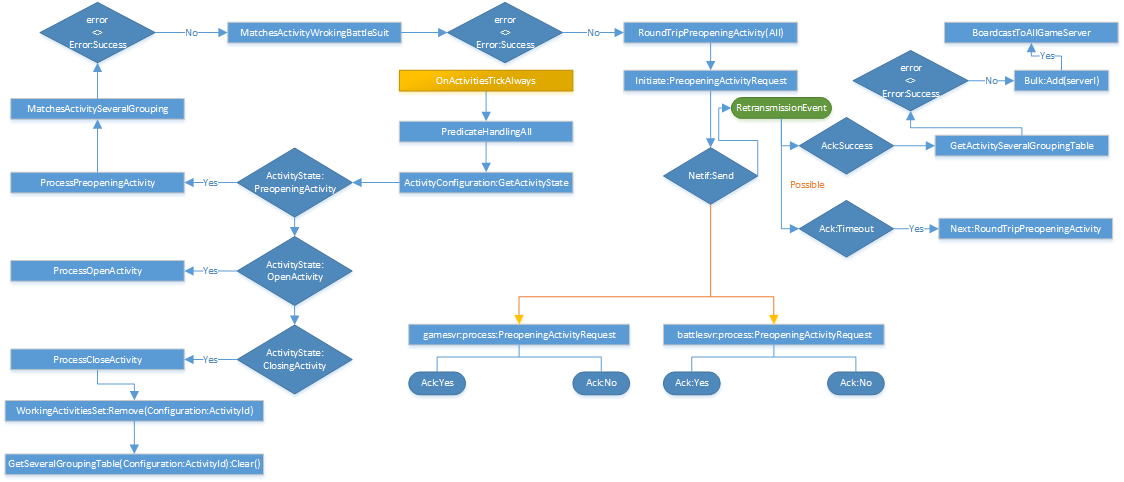
public const long ActivitySeveralGroupingTable = 1000000000100001; // 分布式活动 游戏跨服分组表容器根ID【分布式字典】（保留 < 十万个）

public const long ActivityConfigurationLockId = 1000000000200001; // 分布式活动 配置档监视视同步锁根ID（保留 < 十万个）

public const long PreopeningActivityAsyncLockId = 1000000000300001; // 分布式活 动预开启活动监视同步锁根ID（保留 < 十万个）

* + 1. Services
       1. ActivitiesService

跨服节点活动服务，服务定期执行“OnActivitiesTickAlways（活动一直Tick检查)”目前为“写死的2分钟”，这里应该是每“1~3S”进行一次实时Tick，记住修改处理的代码（当时是为了好与gamesvr那边进行联合调试弄的）。



OnActivitiesTickAlways函数会一直检查存储在“分布式集群”上面“活动策划档”进行“活动预开启”、“活动关闭”等流程的处理。

注意一点，每次活动Tick进行处理时是保持分布式环境下面一致性的，同时有一点在没有真的改变分布式环境下的数据时（注意使用事务），是不可能出现程序崩溃数据会被改变的。

MatchesActivitySeveralGrouping 按策划配置档案（活动）匹配活动所需要的服务器阵营列表。

MatchesActivityWrokingBattleSuit 按策划配置档案（活动）匹配活动所需要的服务器“gamesvr”、“battlesvr”战斗服，（目前是约定三个“gamesvr”一个“battlesvr”具体看策划配置文件）

RoundTripPreopeningActivity函数是往返确认gamesvr与battlesvr能够接受当前跨服节点请求的活动“预开启”的函数，如果“gamesvr”与“battlesvr”告诉服务器它不接受这个活动预开启的请求时，此时跨服节点服务器会重新匹配新的“gamesvr”或“battlesvr”进行重试；直到成功或者全都失败时，重新预开启活动的行为则只能等下次TICK到达的时候在处理；当然在多个跨服节点工作的情况下，可能下个在处理的就不是本次处理的跨服节点也是很正常的。

* + 1. Ranking

实现完全分布式下的高性能排行榜；支持“无限制位十进制时间（允许千位万位）”、“积分”在分布式环境下的排行榜排名服务。

1. LeaderboardAccessor

排行榜访问者与分布式缓存之间进行直接操作的数据层，内部实现过于复杂；在排行榜未见到大的问题之前不建议修改；若需需求请寻找对于分布式环境具有良好经验或者技术功底的人进行解决。

1. LeaderboardSharding

排行版分片，它管理排行榜约定范围分片内的全部排名数据，其本身对于LeaderboardAccessor的简要封装形式。

1. LeaderboardObject

排行榜对象，提供批量排行榜数据录入查询、匹配等一系列功能；同时它尽可能确保排行榜数据不被丢失；具有批量导入排名列表信息本地持久化的机制。

1. PatternMatchesObject

排行榜匹配规则对象，用于描述匹配排行榜对象时的规则；以下为匹配用例。

private Error MatchesOffPlatform(string platform, int rankingType, long minScore, long maxScore, int maxMatching, out IEnumerable<TRankingMember> matches, bool offPlatform)

{

matches = null;

LeaderboardObject<TRankingMember> leaderboard = null;

if (string.IsNullOrEmpty(platform))

{

leaderboard = this.Get(rankingType);

}

else

{

leaderboard = this.Get(platform, rankingType);

}

if (leaderboard == null)

{

return Error.Error\_UnableToFetchATheLeaderboardObjectInstance;

}

if (maxMatching <= 0)

{

maxMatching = -1;

}

PatternMatchesObject<TRankingMember> min = null;

PatternMatchesObject<TRankingMember> max = null;

if (0 != minScore)

{

min = new PatternMatchesObject<TRankingMember>()

{

AchievementScore = minScore

};

if (!offPlatform)

{

max.Platform = platform;

}

}

if (0 != maxScore)

{

max = new PatternMatchesObject<TRankingMember>()

{

AchievementScore = maxScore,

};

if (!offPlatform)

{

max.Platform = platform;

}

}

Error error = leaderboard.Sharding.Matches(min, max, maxMatching, out matches);

if (error != Error.Error\_Success)

{

return error;

}

return Error.Error\_Success;

}

1. 发布
   1. 打开>= Visual Studio 2017
   2. 打开（GVMServer）解决方案
   3. 右键需求发布项目
   4. 发布（&B）
   5. 发布配置文件设置
      1. 配置（Debug | Any CPU）
      2. 目标框架（>= netcoreapp2.0）
      3. 部署模式（框架依赖 / 基于框架部署）
      4. 目标运行时（可移植 / 无需指定运行时平台）
      5. 目标位置（发布程序资源的目录）
   6. 发布（&U）
2. 部署
   1. Os

>= Windows Vista (NT 6.0)

>= Windows Server 2003

>= CentsOS 7

>= Deepin 9

>= Ubuntu 16.04

>= Fedora 29

>= SLES 12

>= RHEL 7

>= OpenSUSE 15

* 1. .Net-Core

sudo yum update

sudo yum install libunwind libicu

mkdir dotnet

cd dotnet

curl -sSL -o dotnet.tar.gz https://download.microsoft.com/download/1/B/4/1B4DE605-8378-47A5-B01B-2C79D6C55519/dotnet-sdk-2.0.0-linux-x64.tar.gz

sudo mkdir -p /opt/dotnet

sudo tar -zvxf dotnet.tar.gz -C /opt/dotnet

sudo ln -s /opt/dotnet/dotnet /usr/local/bin

* 1. Mysql
     1. 创建“sa”管理用户【权限：ALL；访问：%(任何主机)】密码“sa”
     2. keepalived、mysql-proxy 配置读写分写
     3. 刷新数据库设置包括权限设定
     4. 新建数据库“warea”执行我们提供的T/SQL数据库脚本生成表结构视图。
  2. Install

例如：stage

sudo mkdir ~/app/stage

cd ~/app/stage

rz stage.zip

unzip stage.zip

rm -rf stage.zip

配置：

cd ~/app/stage

nano appsettings.json

测试：

cd ~/app/stage

dotnet GVMServer.Stage.dll

Ctrl+C #……终止

守护：

编写：

sudo nano /etc/systemd/system/nsstage.service

Ctrl+V

[Unit]

Description=NsStage App running on CentOS 7

[Service]

WorkingDirectory=/home/yanfawubu/app/stage

ExecStart=/opt/dotnet/dotnet/home/yanfawubu/app/stage/GVMServer.Web.dl l

Restart=always

# Restart service after 10 seconds if the dotnet service crashes:

RestartSec=3

KillSignal=SIGINT

SyslogIdentifier=dotnet-nsstage-app

User=yanfawubu

Environment=ASPNETCORE\_ENVIRONMENT=Production

Environment=DOTNET\_PRINT\_TELEMETRY\_MESSAGE=false

[Install]

WantedBy=multi-user.target

运行：

cd /etc/systemd/system

sudo systemctl enable nsstage.service

sudo chmod a+x nsstage.service # chmod 777

sudo systemctl daemon-reload

sudo systemctl start nsstage.service

sudo systemctl status nsstage.service