nacos-sdk-csharp Documentation

Release 1.0.0

Catcher Wong

Aug 10, 2022

INTRODUCTION

nacos-sdk-csharp 是基于 C# (dotnet core) 实现 nacos 的版本

CHAPTER

ONE

快速上手

nacos-sdk-csharp 是基于 C# (dotnet core) 实现 nacos 的版本

1.1 安装 Nuget 包

选择您需要的包。

```
dotnet add package nacos-sdk-csharp
dotnet add package nacos-sdk-csharp.AspNetCore
dotnet add package nacos-sdk-csharp.Extensions.Configuration
dotnet add package nacos-sdk-csharp.YamlParser
dotnet add package nacos-sdk-csharp.IniParser
```

```
注：从 1.0.0 版本之后，包名里面的 `unofficial` 后缀已经被移除，
带 `unofficial` 的包已经不再维护更新，请尽早更新到最新版本。
```

1.2 功能特性

- 基本的 Open Api 接口封装
- 集成 ASP.NET Core 的配置系统
- 简易 ASP.NET Core 的服务注册和发现
- 和阿里云应用配置管理 (Application Configuration Management，简称 ACM) 集成使用
- ...

1.3 简易用法

1.3.1 配置

1. 在 *Program.cs* 进行如下配置

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, builder) =>
        {
            var c = builder.Build();

            // 从配置文件读取 Nacos 相关配置
            // 默认会使用 JSON 解析器来解析存在 Nacos Server 的配置
            builder.AddNacosV2Configuration(c.GetSection("NacosConfig"));
            // 也可以按需使用 ini 或 yaml 的解析器
            // builder.AddNacosV2Configuration(c.GetSection("NacosConfig"), Nacos.
↪IniParser.IniConfigurationStringParser.Instance);
            // builder.AddNacosV2Configuration(c.GetSection("NacosConfig"), Nacos.
↪YamlParser.YamlConfigurationStringParser.Instance);
        })
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        })
```

2. 修改 *appsettings.json*

```
{
    "NacosConfig": {
        "Listeners": [
        {
            "Optional": false,
            "DataId": "common",
            "Group": "DEFAULT_GROUP"
        },
        {
            "Optional": false,
            "DataId": "demo",
            "Group": "DEFAULT_GROUP"
        }
        ],
        "Namespace": "csharp-demo",
        "ServerAddresses": [ "http://localhost:8848/" ],
```

(continues on next page)

(continued from previous page)

```
        "UserName": "test2",
        "Password": "123456",
        "AccessKey": "",
        "SecretKey": "",
        "EndPoint": "acm.aliyun.com",
        "ConfigFilterAssemblies": ["YouPrefix.AssemblyName"],
        "ConfigFilterExtInfo": "some ext infomation"
    }
}
```

3. 用原生的.NET Core 方式来读取 Nacos 配置

```
[ApiController]
[Route("api/[controller]")]
public class ConfigController : ControllerBase
{
    private readonly IConfiguration _configuration;
    private readonly AppSettings _settings;
    private readonly AppSettings _sSettings;
    private readonly AppSettings _mSettings;

    public ConfigController(
        IConfiguration configuration,
        IOptions<AppSettings> options,
        IOptionsSnapshot<AppSettings> sOptions,
        IOptionsMonitor<AppSettings> _mOptions
        )
    {
        _logger = logger;
        _configuration = configuration;
        _settings = options.Value;
        _sSettings = sOptions.Value;
        _mSettings = _mOptions.CurrentValue;
    }

    [HttpGet]
    public string Get()
    {
        // ....

        return "ok";
    }

}
```

1.3.2 服务注册和发现

1. 服务注册

在 *Program.cs* 中配置

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        // ...

        services.AddNacosAspNet(Configuration, "nacos");
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        // ...
    }
}
```

修改 *appsettings.json*

```
"nacos": {
    "EndPoint": "sub-domain.aliyun.com:8080",
    "ServerAddresses": [ "http://localhost:8848" ],
    "DefaultTimeOut": 15000,
    "Namespace": "cs",
    "ListenInterval": 1000,
    "ServiceName": "App1",
    "GroupName": "DEFAULT_GROUP",
    "ClusterName": "DEFAULT",
    "Ip": "",
    "PreferredNetworks": "", // select an IP that matches the prefix as the service␣
↪registration IP
    "Port": 0,
    "Weight": 100,
    "RegisterEnabled": true,
```

(continues on next page)

(continued from previous page)

```
    "InstanceEnabled": true,
    "Ephemeral": true,
    "Secure": false,
    "AccessKey": "",
    "SecretKey": "",
    "UserName": "",
    "Password": "",
    "ConfigUseRpc": true,
    "NamingUseRpc": true,
    "NamingLoadCacheAtStart": "",
    "LBStrategy": "WeightRandom", //WeightRandom WeightRoundRobin
    "Metadata": {
    "aa": "bb",
    "cc": "dd"
    }
}
```

2. 服务发现

```
[Route("api/[controller]")]
[ApiController]
public class ValuesController : ControllerBase
{
    private readonly Nacos.V2.INacosNamingService _svc;

    public ValuesController(Nacos.V2.INacosNamingService svc)
    {
        _svc = svc;
    }

    [HttpGet("test")]
    public async Task<IActionResult> Test()
    {
        // 这里需要知道被调用方的服务名
        var instance = await _svc.SelectOneHealthyInstance("App2", "DEFAULT_GROUP")
        var host = $"{instance.Ip}:{instance.Port}";

        var baseUrl = instance.Metadata.TryGetValue("secure", out _)
            ? $"https://{host}"
            : $"http://{host}";

        if(string.IsNullOrWhiteSpace(baseUrl))
        {
            return "empty";
```

(continues on next page)

(continued from previous page)

```
        }

        var url = $"{baseUrl}/api/values";

        using (HttpClient client = new HttpClient())
        {
            var result = await client.GetAsync(url);
            return await result.Content.ReadAsStringAsync();
        }
    }
}
```

CHAPTER

TWO

SDK 环境变量说明

环境变量 Key	默认值	说明
nacos.client.appKey	无	设定 sdk 的 AppKey
JM.SNAPSHOT.PATH	无	Naming 模块的缓存目录
push.receiver.udp.port	无	设定 client 的 udp 端口，针对服务端是 1.x
com.alibaba.nacos.client.naming.local.ip	无	已经废弃，请使用 com.alibaba.nacos.client.local.ip
com.alibaba.nacos.client.local.ip	无	设置 sdk 的 local ip
nacos.server.grpc.port.offset	1000	nacos 服务端 grpc 端口偏移值
nacos.server.port	8848	nacos 服务端的端口

CHAPTER

THREE

配置中心

nacos-sdk-csharp 操作 nacos 的配置提供了两个 nuget 包，一个是原生的 sdk 版本，一个是集成了 ASP.NET Core 配置体系的版本，大家可以根据自己的需要选择不同的版本。

```
注：请还在使用 nuget 包的名称里面还带有 `unofficial` 的朋友尽快升级到
不带 `unofficial` 的版本，新版本同时支持 nacos server 1.x 和 2.x
```

3.1 原生的 sdk 版本

原生 sdk 暴露出了下面几个方法

```
Task<string> GetConfig(string dataId, string group, long timeoutMs);

Task<string> GetConfigAndSignListener(string dataId, string group, long timeoutMs,␣
↪IListener listener);

Task AddListener(string dataId, string group, IListener listener);

Task<bool> PublishConfig(string dataId, string group, string content);

Task<bool> PublishConfig(string dataId, string group, string content, string type);

Task<bool> RemoveConfig(string dataId, string group);

Task RemoveListener(string dataId, string group, IListener listener);

Task<string> GetServerStatus();

Task ShutDown();
```

主要就是配置的 CURD 和监听。

监听可以让客户端实时获取 nacos 上面的最新配置，这个对实现配置的热加载/热更新是一个很重要的基石。

实现监听，需要自定义一个实现 *IListener* 的监听者。

```
public class CusConfigListen : Nacos.V2.IListener
{
    public void ReceiveConfigInfo(string configInfo)
    {
        // 这里会有配置变更的回调，在这里处理配置变更之后的逻辑。
        System.Console.WriteLine("config cb cb cb " + configInfo);
    }
}
```

添加监听和移除监听，要保证是同一个监听者对象，不然会造成监听一直存在，移除不了的情况。

对于重要配置被修改需要通知到部门群或者其他地方时，通过监听就可以实现一个 webhook 的功能了。

3.2 集成 ASP.NET Core 版本

推出这样一个版本很大程度是为了简化操作，只加几个配置，代码层级的用法几乎零改动，类似于 spring cloud 那样。

SDK 这一块通过实现了自定义的 **ConfigurationProvider** 和 **IConfigurationSource** 来达到了这个效果。

对于配置的热加载/热更新，SDK 则是实现了一个默认的监听者，在配置变更之后进行了 *OnReload* 的操作。

使用上需要在 *Program* 里面进行 **ConfigureAppConfiguration** 的设置。

```
Host.CreateDefaultBuilder(args)
    .ConfigureAppConfiguration((context, builder) =>
    {
        var c = builder.Build();
        builder.AddNacosV2Configuration(c.GetSection("NacosConfig"));
    })
    .ConfigureWebHostDefaults(webBuilder =>
    {
        webBuilder.UseStartup<Startup>();
    })
```

```
注：请注意 `IOptions<T>`、`IOptionsSnapshot<T>` 和 `IOptionsMonitor<T>` 的区别
避免出现配置变更了，应用读取不到最新配置的情况!!!
```

CHAPTER

FOUR

配置加解密

配置加解密这个功能是衍生功能，目前和 nacos 服务端联系还不大，是 SDK 对配置加密后再传输，解密后再使用。

如果需要使用这个功能，需要做下面两件事。

1. 自定义 ConfigFilter
2. 配置 ConfigFilter

4.1 自定义 ConfigFilter

把配置加解密的逻辑放到自定义的 ConfigFilter 里面。ConfigFilter 可以有多个，不同 ConfigFilter 的执行顺序是按他们的 Order 属性来决定的。

```
public class MyNacosConfigFilter : IConfigFilter
{
    public void DoFilter(IConfigRequest request, IConfigResponse response,␣
↪IConfigFilterChain filterChain)
    {
        if (request != null)
        {
            // 这里是请求的过滤，也就是在这里进行加密操作


            // 不要忘了在这里覆盖请求的内容!!!!!
            request.PutParameter(Nacos.V2.Config.ConfigConstants.ENCRYPTED_DATA_KEY,␣
↪encryptedDataKey);
            request.PutParameter(Nacos.V2.Config.ConfigConstants.CONTENT, content);
        }

        if (response != null)
        {
            // 这里是响应的过滤，也就是在这里进行解密操作
```

(continues on next page)

(continued from previous page)

```
            // 不要忘了在这里覆盖响应的内容!!!!!
            response.PutParameter(Nacos.V2.Config.ConfigConstants.CONTENT, content);
        }
    }

    public string GetFilterName() => nameof(MyNacosConfigFilter);

    public int GetOrder() => 1;

    public void Init(NacosSdkOptions options)
    {
        // 做一些初始化操作
    }
}
```

4.2 配置 ConfigFilter

这一步主要是告诉 nacos-sdk-csharp 去那个程序集找 ConfigFilter 的实现，以及它的实现需要什么参数。

```
{
    "NacosConfig": {
        "ConfigFilterAssemblies": [ "XXXX.CusLib" ],
        "ConfigFilterExtInfo": "{\"JsonPaths\":[\"ConnectionStrings.Default\"],\
↪"Other\":\"xxxxxx\"}"
    }
}
```

这里主要是两个配置，一个是 *ConfigFilterAssemblies* 指定实现类所在的程序集，一个是 *ConfigFilterExtInfo* 指定实现类可能需要的参数。

CHAPTER

FIVE

服务发现

nacos-sdk-csharp 操作 nacos 的配置提供了两个 nuget 包，一个是原生的 sdk 版本，一个是集成了 ASP.NET Core 配置体系的版本，大家可以根据自己的需要选择不同的版本。

```
注：请还在使用 nuget 包的名称里面还带有 `unofficial` 的朋友尽快升级到
不带 `unofficial` 的版本，新版本同时支持 nacos server 1.x 和 2.x
```

5.1 原生的 sdk 版本

原生 sdk 暴露出了下面几个方法

```
Task RegisterInstance(string serviceName, string ip, int port);

Task RegisterInstance(string serviceName, string groupName, string ip, int port);

Task RegisterInstance(string serviceName, string ip, int port, string clusterName);

Task RegisterInstance(string serviceName, string groupName, string ip, int port,␣
↪string clusterName);

Task RegisterInstance(string serviceName, Instance instance);

Task RegisterInstance(string serviceName, string groupName, Instance instance);

Task DeregisterInstance(string serviceName, string ip, int port);

Task DeregisterInstance(string serviceName, string groupName, string ip, int port);

Task DeregisterInstance(string serviceName, string ip, int port, string clusterName);

Task DeregisterInstance(string serviceName, string groupName, string ip, int port,␣
↪string clusterName);
```

(continues on next page)

(continued from previous page)

```
Task DeregisterInstance(string serviceName, Instance instance);

Task DeregisterInstance(string serviceName, string groupName, Instance instance);

Task<List<Instance>> GetAllInstances(string serviceName);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName);

Task<List<Instance>> GetAllInstances(string serviceName, bool subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, bool␣
↪subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, List<string> clusters);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, List
↪<string> clusters);

Task<List<Instance>> GetAllInstances(string serviceName, List<string> clusters, bool␣
↪subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, List
↪<string> clusters, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, bool healthy);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, bool␣
↪healthy);

Task<List<Instance>> SelectInstances(string serviceName, bool healthy, bool␣
↪subscribe);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, bool␣
↪healthy, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, List<string> clusters, bool␣
↪healthy);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, List
↪<string> clusters, bool healthy);

Task<List<Instance>> SelectInstances(string serviceName, List<string> clusters, bool␣
```

(continues on next page)

(continued from previous page)

```
↪healthy, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, List
↪<string> clusters, bool healthy, bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName);

Task<Instance> SelectOneHealthyInstance(string serviceName, bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, bool␣
↪subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, List<string> clusters);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, List
↪<string> clusters);

Task<Instance> SelectOneHealthyInstance(string serviceName, List<string> clusters,␣
↪bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, List
↪<string> clusters, bool subscribe);

Task Subscribe(string serviceName, IEventListener listener);

Task Subscribe(string serviceName, string groupName, IEventListener listener);

Task Subscribe(string serviceName, List<string> clusters, IEventListener listener);

Task Subscribe(string serviceName, string groupName, List<string> clusters,␣
↪IEventListener listener);

Task Unsubscribe(string serviceName, IEventListener listener);

Task Unsubscribe(string serviceName, string groupName, IEventListener listener);

Task Unsubscribe(string serviceName, List<string> clusters, IEventListener listener);

Task Unsubscribe(string serviceName, string groupName, List<string> clusters,␣
↪IEventListener listener);
```

(continues on next page)

(continued from previous page)

```
Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, string␣
↪groupName);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, AbstractSelector␣
↪selector);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, string groupName,
↪ AbstractSelector selector);

Task<List<ServiceInfo>> GetSubscribeServices();

Task<string> GetServerStatus();

Task ShutDown();
```

可以看到有很多重载的方法，主要也是离不开服务的 CURD 和监听。

5.2 集成 ASP.NET Core 版本

推出这样一个版本很大程度是为了简化操作，可以在应用启动的时候注册到 nacos 的注册中心，应用停止的时候可以注销。

SDK 内部是通过实现了一个后台服务 (IHostedService) 来达到这个效果的。

无论是注册还是注销，都会有重试的机制，目前最多重试 3 次。

查询服务实例则需要通过上面 **INacosNamingService** 提供的方法来实现。

CHAPTER

SIX

集成微服务引擎 MSE

6.1 前置条件

注册阿里云账号，并开通和购买微服务引擎。

使用 SDK 操作的话，只需要将 ServerAddresses 替换成 mse 实例对应的地址即可。

示例如下：

6.2 配置模块

```
{
    "NacosConfig": {
        "Listeners": [
            {
                "Optional": false,
                "DataId": "common",
                "Group": "DEFAULT_GROUP"
            },
            {
                "Optional": false,
                "DataId": "demo",
                "Group": "DEFAULT_GROUP"
            }
        ],
        "Namespace": "0138xxxx-yyyy-zzzz-1111-000000000000",
        "ServerAddresses": [ "http://mse-xxxxxxxxxxx-nacos-ans.mse.aliyuncs.com:8848"
↪]
    }
}
```

6.3 服务注册发现模块

```
{
    "nacos": {
        "ServerAddresses": [ "http://mse-xxxxxxxxxx-nacos-ans.mse.aliyuncs.com:8848"␣
↪],
        "DefaultTimeOut": 15000,
        "Namespace": "0138xxxx-yyyy-zzzz-1111-000000000000",
        "ListenInterval": 1000,
        "ServiceName": "App1",
        "GroupName": "DEFAULT_GROUP",
        "ClusterName": "DEFAULT",
        "Weight": 100,
        "RegisterEnabled": true,
        "InstanceEnabled": true,
        "Ephemeral": true
    }
}
```

CHAPTER

SEVEN

一些博客

7.1 2022

- 微服务配置中心 Nacos .Net 5 【2022-05-17】
- 待挖掘 【2022-xx-xx】

7.2 2021

- 聊一聊 Yarp 结合 Nacos 完成服务发现 【2021-12-27】
- Nacos 配置中心 +ASP.NET Core 【2021-12-26】
- 聊一聊基于 Nacos 的 metadata 完成服务间的 AB 测试 【2021-12-13】
- Asp.Net5 WebAPI 使用 NacOS 作为配置中心的方法 【2021-11-19】
- 聊一聊声明式接口调用与 Nacos 的结合使用 【2021-11-12】
- 聊一聊.NET Core 结合 Nacos 实现配置加解密 【2021-06-15】
- 聊一聊如何在.NET Core 中使用 Nacos 2.0 【2021-03-22】
- 聊一聊和 Nacos 2.0.0 对接那些事 【2021-03-08】

7.3 2020

- ASP.NET Core 集成 Nacos 配置中心之适配多格式配置 【2020-10-02】
- 手动造轮子——为 Ocelot 集成 Nacos 注册中心 【2020-07-21】
- 搭建一套 ASP.NET Core+Nacos+Spring Cloud Gateway 项目 【2020-07-03】
- ASP.NET Core 使用 Nacos SDK 访问阿里云 ACM 【2020-06-07】
- 在.NET Core 中用最原生的方式读取 Nacos 的配置 【2020-04-26】

7.4 2019

- ASP.NET Core 使用 Nacos 作为配置中心的多环境问题 【2019-11-21】

CHAPTER

EIGHT

一些组件

- Ocelot.Provider.Nacos 【Ocelot 集成 Nacos 注册中心组件】
- nacos-csharp-extensions 【Nacos 的一些扩展 (声明式接口调用集成，YARP 集成等)】

CHAPTER

NINE

V1.2.2 (NOV 7TH, 2021) 发布记录

1. [NAMING] Improve server push empty pretection (#155)
2. [NAMING] Fixed gzip compression (#161 #162)

---

1. [NAMING] 优化服务端推空保护 (#155)
2. [NAMING] 修复 gzip 压缩问题 (#161 #162)

CHAPTER

TEN

V1.3.0 (DEC 9TH, 2021) 发布记录

1. [ALL] Support net6 for TargetFrameworks (#164)
2. [NAMING] Support reading nacos server port offset from environment variable (#167)
3. [ALL] remove RequestStream in proto file (#169)

---

1. [ALL] TargetFrameworks 添加 net6 支持 (#164)
2. [NAMING] 支持从环境变量读取 nacos server 的端口偏移 (#167)
3. [ALL] 从 proto 文件移除 RequestStream 方法 (#169)

CHAPTER

ELEVEN

V1.3.1 (JAN 20TH, 2022) 发布记录

1. [NAMING] Fix ServiceListRequest pageSize error
2. [NAMING] Fix service push when new data lastRefTime lessthan old data lastRefTime
3. [NAMING] Fix judgement for is subscribe services.
4. [CONFIG] Fix thread safe issue of yaml and ini parser.

---

1. [NAMING] 修复分页请求 pageSize 参数传递问题
2. [NAMING] 修复服务推送时，lastRefTime 新数据比老数据的小的情况
3. [NAMING] 修复是否订阅服务的判断
4. [CONFIG] 修复 yaml 和 ini 解析器的线程安全问题

CHAPTER

TWELVE

V1.3.2 (MAR 15TH, 2022) 发布记录

1. [NAMING] Fix QueryInstancesOfService error cluster param (#180 #181)
2. [NAMING] Fix Empty namespace issue (#187 #189)

---

1. [NAMING] 修复 QueryInstancesOfService 请求参数错误 (#180 #181)
2. [NAMING] 修复空命名空间默认值问题 (#187 #189)

CHAPTER

THIRTEEN

V1.3.3 (MAY 30TH, 2022) 发布记录

1. [NAMING] Fix IsSubscribed exception when NamingUseRpc is false (#194 #196)
2. [CONFIG] Fix ReceiveConfigInfo dict get item error in startup (#200 #201)
3. [CONFIG] Improve Microsoft.Extensions.Configuration Integration (#203 #204)
4. [CONFIG] Config cache path from JM.SNAPSHOT.PATH env at first (#205 #206)
5. [CORE] Add package readme file (#207)
6. [CI] Update version of Nacos Server

---

1. [NAMING] 修复 NamingUseRpc 设置成 false 是 IsSubscribed 抛异常的问题 (#194 #196)
2. [CONFIG] 修复启动时 ReceiveConfigInfo 字典操作异常问题 (#200 #201)
3. [CONFIG] 优化 Microsoft.Extensions.Configuration 的集成 (#203 #204)
4. [CONFIG] 优化配置缓存的路径读取方式 (#205 #206)
5. [CORE] 添加 nuget 包说明文件 (#207)
6. [CI] 更新 Nacos Server 版本

CHAPTER

FOURTEEN

GUIDELINES FOR UPGRADING TO VERSION 1.0

14.1 Background

Before *nacos-sdk-csharp* v1.0.0, it was mainly connected to Nacos server 1. X.

With the release of Nacos server 2.0.0, *nacos-sdk-csharp* has been reconstructed and adapted to dock.

Because the protocol of Nacos server from 1.X to 2.X is changed, 1.X is mainly HTTP, 2.X is mainly grpc.

Before, the SDK mainly encapsulated and supplemented some contents of open API.

In version 1.0.0, the SDK method will align with the Java version of the SDK. The adjustment here will be relatively large, but several versions of the previous method will be retained.

14.2 Upgrade

The premise is that the version of nacos server is 2.0.0 or higher.

Modify the referenced nuget package

```
    <ItemGroup>
-       <PackageReference Include="nacos-sdk-csharp-unofficial" Version="0.8.5" />
+       <PackageReference Include="nacos-sdk-csharp" Version="1.0.0-alphaxxxx" />
    </ItemGroup>
```

> NOTE：The suffix “unofficial” has been removed from the package name. The version 1.0.0 is still pre-released, so there will be alpha, which will not be available after the official release. If you depend on the extension package, you need to do the corresponding processing.

The basic configuration is as follows:

```
{
    "EndPoint": "",
    "ServerAddresses": [ "http://localhost:8848" ],
```

(continues on next page)

(continued from previous page)

```
    "DefaultTimeOut": 15000,
    "Namespace": "cs",
    "ListenInterval": 1000,
    "AccessKey": "",
    "SecretKey": "",
    "UserName": "",
    "Password": "",
    "ConfigUseRpc": true,
    "NamingUseRpc": true,
    "NamingLoadCacheAtStart": ""
}
```

Only when ‘ConfigUseRpc’ and ‘NamingUseRpc’ are set to true, gPRC will be used to interact with Nacos server, otherwise HTTP will be used.

And *Namespace* should be set to the Namespace Id in the nacos console, not the Namespace name!!!!!!

14.3 Configuration changes

SDK

Adjust *INacosConfigClient* to *INacosConfigService*， and it providers the following methods.

```
Task<string> GetConfig(string dataId, string group, long timeoutMs);

Task<string> GetConfigAndSignListener(string dataId, string group, long timeoutMs,␣
↪IListener listener);

Task AddListener(string dataId, string group, IListener listener);

Task<bool> PublishConfig(string dataId, string group, string content);

Task<bool> PublishConfig(string dataId, string group, string content, string type);

Task<bool> RemoveConfig(string dataId, string group);

Task RemoveListener(string dataId, string group, IListener listener);

Task<string> GetServerStatus();

Task ShutDown();
```

Integrate ASP.NET Core

```
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, builder) =>
        {
            var c = builder.Build();
-            builder.AddNacosConfiguration(c.GetSection("NacosConfig"));
+            builder.AddNacosV2Configuration(c.GetSection("NacosConfig"));
        })
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        })
```

14.4 Service changes

SDK

Adjust *INacosNamingClient* to *INacosNamingService*, and it providers the following methods.

```
Task RegisterInstance(string serviceName, string ip, int port);

Task RegisterInstance(string serviceName, string groupName, string ip, int port);

Task RegisterInstance(string serviceName, string ip, int port, string clusterName);

Task RegisterInstance(string serviceName, string groupName, string ip, int port,␣
↪string clusterName);

Task RegisterInstance(string serviceName, Instance instance);

Task RegisterInstance(string serviceName, string groupName, Instance instance);

Task DeregisterInstance(string serviceName, string ip, int port);

Task DeregisterInstance(string serviceName, string groupName, string ip, int port);

Task DeregisterInstance(string serviceName, string ip, int port, string clusterName);

Task DeregisterInstance(string serviceName, string groupName, string ip, int port,␣
↪string clusterName);

Task DeregisterInstance(string serviceName, Instance instance);

Task DeregisterInstance(string serviceName, string groupName, Instance instance);
```

(continues on next page)

(continued from previous page)

```
Task<List<Instance>> GetAllInstances(string serviceName);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName);

Task<List<Instance>> GetAllInstances(string serviceName, bool subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, bool␣
↪subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, List<string> clusters);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, List
↪<string> clusters);

Task<List<Instance>> GetAllInstances(string serviceName, List<string> clusters, bool␣
↪subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, List
↪<string> clusters, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, bool healthy);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, bool␣
↪healthy);

Task<List<Instance>> SelectInstances(string serviceName, bool healthy, bool␣
↪subscribe);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, bool␣
↪healthy, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, List<string> clusters, bool␣
↪healthy);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, List
↪<string> clusters, bool healthy);

Task<List<Instance>> SelectInstances(string serviceName, List<string> clusters, bool␣
↪healthy, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, List
↪<string> clusters, bool healthy, bool subscribe);
```

(continues on next page)

(continued from previous page)

```
Task<Instance> SelectOneHealthyInstance(string serviceName);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName);

Task<Instance> SelectOneHealthyInstance(string serviceName, bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, bool␣
↪subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, List<string> clusters);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, List
↪<string> clusters);

Task<Instance> SelectOneHealthyInstance(string serviceName, List<string> clusters,␣
↪bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, List
↪<string> clusters, bool subscribe);

Task Subscribe(string serviceName, IEventListener listener);

Task Subscribe(string serviceName, string groupName, IEventListener listener);

Task Subscribe(string serviceName, List<string> clusters, IEventListener listener);

Task Subscribe(string serviceName, string groupName, List<string> clusters,␣
↪IEventListener listener);

Task Unsubscribe(string serviceName, IEventListener listener);

Task Unsubscribe(string serviceName, string groupName, IEventListener listener);

Task Unsubscribe(string serviceName, List<string> clusters, IEventListener listener);

Task Unsubscribe(string serviceName, string groupName, List<string> clusters,␣
↪IEventListener listener);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, string␣
↪groupName);
```

(continues on next page)

(continued from previous page)

```
Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, AbstractSelector␣
↪selector);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, string groupName,
↪ AbstractSelector selector);

Task<List<ServiceInfo>> GetSubscribeServices();

Task<string> GetServerStatus();

Task ShutDown();
```

Integrate ASP.NET Core

Modify *Startup*

```
        public void ConfigureServices(IServiceCollection services)
        {
-            services.AddNacosAspNetCore(Configuration);
+            services.AddNacosAspNet(Configuration);
            services.AddControllers();
        }
```

From *INacosServerManager* to *INacosNamingService*。

Details：

```
- var baseUrl = await _serverManager.GetServerAsync("App2");

+ var instance = await _svc.SelectOneHealthyInstance("App2", "DEFAULT_GROUP");
+ var host = $"{instance.Ip}:{instance.Port}";

+ var baseUrl = instance.Metadata.TryGetValue("secure", out _)
+     ? $"https://{host}"
+     : $"http://{host}";
```

On the basis of the original service configuration, three options are added: **InstanceEnabled**, **Ephemeral**, **Secure**.

CHAPTER

FIFTEEN

升级到 1.0 版本指引

15.1 背景

nacos-sdk-csharp v1.0.0 版本之前主要是对接 nacos server 1.x。

随着 nacos server 2.0.0 即将发布，*nacos-sdk-csharp* 也已经进行了一次重构和适配来对接。

由于 nacos server 从 1.x 到 2.x 客户端对接的协议有所变更，1.x 主要是 HTTP，2.x 主要是 gRPC。

之前 sdk 主要是对 Open Api 进行了封装和补充了部分内容。

在 1.0.0 版本，sdk 的方法会对齐 java 版的 sdk，这里的调整会比较大，但是之前用的方法还会保留几个版本。

15.2 升级

大前提是 nacos server 版本已经是 2.0.0 或更高版本。

修改引用的 nuget 包

```
    <ItemGroup>
-       <PackageReference Include="nacos-sdk-csharp-unofficial" Version="0.8.5" />
+       <PackageReference Include="nacos-sdk-csharp" Version="1.0.0-alphaxxxx" />
    </ItemGroup>
```

> 注：包名已经移除了 *unofficial* 的后缀，1.0.0 的版本目前还是 prerelease，所以会有 alpha 的字样，正式发布后是没有的。如果依赖了扩展包，也需要做对应的处理。

核心的基础配置如下：

```
{
    "EndPoint": "",
    "ServerAddresses": [ "http://localhost:8848" ],
    "DefaultTimeOut": 15000,
    "Namespace": "cs",
```

(continues on next page)

(continued from previous page)

```
    "ListenInterval": 1000,
    "AccessKey": "",
    "SecretKey": "",
    "UserName": "",
    "Password": "",
    "ConfigUseRpc": true,
    "NamingUseRpc": true,
    "NamingLoadCacheAtStart": ""
}
```

只有当 *ConfigUseRpc* 和 *NamingUseRpc* 设置为 true 的时候，才会用 gRPC 去和 nacos server 交互，反之还是 HTTP。

另外，Namespace 字段填写的值是控制台中的命名空间 Id，不是命名空间名称!!!!

15.3 配置变化

SDK

代码层使用的 *INacosConfigClient* 需要调整成 *INacosConfigService*，提供了如下的方法。

```
Task<string> GetConfig(string dataId, string group, long timeoutMs);

Task<string> GetConfigAndSignListener(string dataId, string group, long timeoutMs,␣
↪IListener listener);

Task AddListener(string dataId, string group, IListener listener);

Task<bool> PublishConfig(string dataId, string group, string content);

Task<bool> PublishConfig(string dataId, string group, string content, string type);

Task<bool> RemoveConfig(string dataId, string group);

Task RemoveListener(string dataId, string group, IListener listener);

Task<string> GetServerStatus();

Task ShutDown();
```

集成 ASP.NET Core

主要是变更 *ConfigureAppConfiguration* 里面的 AddNacosConfiguration，其余的不需要变化。

```
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, builder) =>
        {
            var c = builder.Build();
-            builder.AddNacosConfiguration(c.GetSection("NacosConfig"));
+            builder.AddNacosV2Configuration(c.GetSection("NacosConfig"));
        })
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        })
```

15.4 服务变化

SDK

代码层使用的 *INacosNamingClient* 需要调整成 *INacosNamingService*，提供了如下的方法。

```
Task RegisterInstance(string serviceName, string ip, int port);

Task RegisterInstance(string serviceName, string groupName, string ip, int port);

Task RegisterInstance(string serviceName, string ip, int port, string clusterName);

Task RegisterInstance(string serviceName, string groupName, string ip, int port,␣
↪string clusterName);

Task RegisterInstance(string serviceName, Instance instance);

Task RegisterInstance(string serviceName, string groupName, Instance instance);

Task DeregisterInstance(string serviceName, string ip, int port);

Task DeregisterInstance(string serviceName, string groupName, string ip, int port);

Task DeregisterInstance(string serviceName, string ip, int port, string clusterName);

Task DeregisterInstance(string serviceName, string groupName, string ip, int port,␣
↪string clusterName);

Task DeregisterInstance(string serviceName, Instance instance);

Task DeregisterInstance(string serviceName, string groupName, Instance instance);
```

(continues on next page)

(continued from previous page)

```
Task<List<Instance>> GetAllInstances(string serviceName);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName);

Task<List<Instance>> GetAllInstances(string serviceName, bool subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, bool␣
↪subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, List<string> clusters);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, List
↪<string> clusters);

Task<List<Instance>> GetAllInstances(string serviceName, List<string> clusters, bool␣
↪subscribe);

Task<List<Instance>> GetAllInstances(string serviceName, string groupName, List
↪<string> clusters, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, bool healthy);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, bool␣
↪healthy);

Task<List<Instance>> SelectInstances(string serviceName, bool healthy, bool␣
↪subscribe);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, bool␣
↪healthy, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, List<string> clusters, bool␣
↪healthy);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, List
↪<string> clusters, bool healthy);

Task<List<Instance>> SelectInstances(string serviceName, List<string> clusters, bool␣
↪healthy, bool subscribe);

Task<List<Instance>> SelectInstances(string serviceName, string groupName, List
↪<string> clusters, bool healthy, bool subscribe);
```

(continues on next page)

(continued from previous page)

```
Task<Instance> SelectOneHealthyInstance(string serviceName);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName);

Task<Instance> SelectOneHealthyInstance(string serviceName, bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, bool␣
↪subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, List<string> clusters);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, List
↪<string> clusters);

Task<Instance> SelectOneHealthyInstance(string serviceName, List<string> clusters,␣
↪bool subscribe);

Task<Instance> SelectOneHealthyInstance(string serviceName, string groupName, List
↪<string> clusters, bool subscribe);

Task Subscribe(string serviceName, IEventListener listener);

Task Subscribe(string serviceName, string groupName, IEventListener listener);

Task Subscribe(string serviceName, List<string> clusters, IEventListener listener);

Task Subscribe(string serviceName, string groupName, List<string> clusters,␣
↪IEventListener listener);

Task Unsubscribe(string serviceName, IEventListener listener);

Task Unsubscribe(string serviceName, string groupName, IEventListener listener);

Task Unsubscribe(string serviceName, List<string> clusters, IEventListener listener);

Task Unsubscribe(string serviceName, string groupName, List<string> clusters,␣
↪IEventListener listener);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, string␣
↪groupName);
```

(continues on next page)

(continued from previous page)

```
Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, AbstractSelector␣
↪selector);

Task<ListView<string>> GetServicesOfServer(int pageNo, int pageSize, string groupName,
↪ AbstractSelector selector);

Task<List<ServiceInfo>> GetSubscribeServices();

Task<string> GetServerStatus();

Task ShutDown();
```

集成 ASP.NET Core

调整 *Startup*

```
        public void ConfigureServices(IServiceCollection services)
        {
-            services.AddNacosAspNetCore(Configuration);
+            services.AddNacosAspNet(Configuration);
            services.AddControllers();
        }
```

用到 *INacosServerManager* 的地方需要换成 *INacosNamingService*。

具体使用如下：

```
- var baseUrl = await _serverManager.GetServerAsync("App2");

+ var instance = await _svc.SelectOneHealthyInstance("App2", "DEFAULT_GROUP");
+ var host = $"{instance.Ip}:{instance.Port}";

+ var baseUrl = instance.Metadata.TryGetValue("secure", out _)
+     ? $"https://{host}"
+     : $"http://{host}";
```

服务的配置，在原先的基础上添加了，**InstanceEnabled**，**Ephemeral**，**Secure** 三个内容。

CHAPTER

SIXTEEN

常见问题

这个文档记录了一些 SDK 和 nacos server 对接使用上的常见问题。

16.1 1. 命名空间问题

命名空间可以说是第一要素，如果这个没有设置对，那么在控制台里面会看不到对应命名空间下面的数据。

在新建命名空间时，是可以指定命名空间 Id 的，不指定的话会自动生成一个 UUID。

在 SDK 的配置里面，配置的一定是命名空间 Id。如果是 public 命名空间，配置的是一个空字符串。

16.2 2. nacos server 端口开放问题

SDK 在 v1.x 版本之后，就是默认用 grpc 的方式和 nacos server 对接，这个时候会出现下面几种情况。

- a. nacos server 是 1.x 版本，SDK 版本 >= 1.0
- b. nacos server 是 2.x 版本，基于 docker/k8s 部署，只暴露了默认的 8848 端口，没有暴露 grpc 的端口，SDK 版本 >= 1.0

这个时候会出现 *Client not connected,current status: STARTING* 的错误

针对 a 的情况，需要把 xxxUseRpc 设置为 false。

针对 b 的情况，需要把 9848 暴露出来。

如果修改了默认端口或者是通过环境变量设置了偏移，自行调整对应端口，参考 https://nacos.io/zh-cn/docs/2.0.0-compatibility.html

16.3 3. nacos-sdk-csharp 版本与 nacos server 版本关系

nacos server 目前主要有 1.x 版本和 2.x 版本

nacos-sdk-csharp 有 0.x unofficial 版本和 1.x 版本

nacos-sdk-csharp 0.x unofficial 版本只能应用于 nacos server 1.x 版本

nacos-sdk-csharp 1.x 版本可以同时应用于 nacos server 1.x 版本和 2.x 版本