配置系统-案例

项目需求

1）网站用集群部署（多台服务器，访问由负载均衡服务器分配给具体服务器），如果用本地配置文件，每次修改都要挨个修改。有Apollo，Nacos等开源的配置中心以及云服务平台的配置服务，但是项目对于配置的要求没有非常复杂，因此决定再关系数据库中保存配置。不是从本地读取，是从中心服务器读取。

2）已经开源：https://github.com/yangzhongke/Zack.AnyDBConfigProvider

3）按照文档先使用一下，站在使用者角度体验一下，思考自己如何实现。亮点：value支持json格式。

.NET5 .NET Core用数据库做配置中心加载Configuration

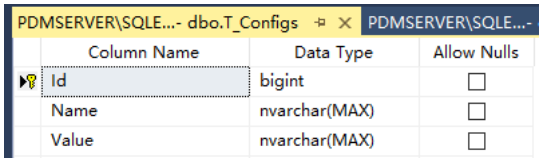
本文介绍了一个在.NET中用数据库做配置中心服务器的方式，介绍了读取配置的开源自定义ConfigurationProvider，并且讲解了主要实现原理。 ...

https://www.bilibili.com/read/cv10067951

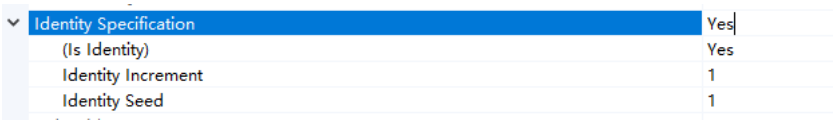
使用：

新建数据表

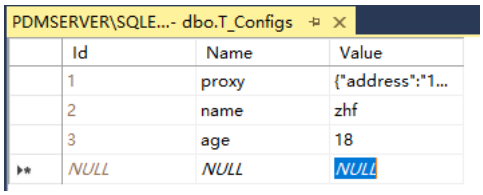
SQL中新建T_Configs表



Id为主键，设置自增



填写参数



NuGet安装

Install-Package zack.AnyDBConfigProvider

Install-Package Microsoft.Data.SqlClient

将读取json文件的配置替换成从中心服务器读取，这里我们将连接字符串写死

```
ConfigurationBuilder configBuilder = new ConfigurationBuilder();
//configBuilder.AddJsonFile("config.json", optional: true, reloadOnChange: true);

//从数据库读取配置， install-package zack.AnyDBConfigProvider
```

```
var connStr = @"Server=PDMSERVER\SQLEXPRESS; Database=StudentDB; User Id = sa; Password=Epdm2018;TrustServerCertificate=true";
configBuilder.AddDbConfiguration(() => new SqlConnection(connStr), reloadOnChange: true, reloadInterval: TimeSpan.FromSeconds(2));

var configRoot = configBuilder.Build();
```

其他的逻辑代码都不需要改变

阅读源代码

重点：

定时reload的实现，也可以考虑改成用触发器实时触发，

ReaderWriterLockSlim的使用

json解析为扁平结构

DBConfigurationSource

```
using Microsoft.Extensions.Configuration;

namespace Zack.AnyDBConfigProvider
{
    class DBConfigurationSource : IConfigurationSource
    {
        private DBConfigOptions options;
        public DBConfigurationSource(DBConfigOptions options)
        {
            this.options = options;
        }
        public IConfigurationProvider Build(IConfigurationBuilder builder)
        {
            return new DBConfigurationProvider(options);
        }
    }
}
```

DBConfigOptions

```
using System;
using System.Data;

namespace Microsoft.Extensions.Configuration
{
    public class DBConfigOptions
    {
        //连接那个数据库，通过回调的形式给用户一个数据库连接
        public Func<IDbConnection> CreateDbConnection { get; set; }
        //表的名字，默认为T_Configs
        public string TableName { get; set; } = "T_Configs";
        //数据更改后是否立即刷新
        public bool ReloadOnChange { get; set; } = false;
        //隔多长时间刷新
        public TimeSpan? ReloadInterval { get; set; }
    }
}
```

DBConfigurationProvider

最核心的代码,Load

```
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Diagnostics;
using System.Text.Json;
using System.Threading;

namespace Zack.AnyDBConfigProvider
{
    public class DBConfigurationProvider : ConfigurationProvider,IDisposable
    {
        private DBConfigOptions options;
```

```
        //allow multi reading and single writing
        //允许并发读，不允许并发写的读写锁
        private ReaderWriterLockSlim lockObj = new ReaderWriterLockSlim();
        private bool isDisposed=false;
        public DBConfigurationProvider(DBConfigOptions options)
        {
            this.options = options;
            TimeSpan interval = TimeSpan.FromSeconds(3);
            if(options.ReloadInterval!=null)
            {
                interval = options.ReloadInterval.Value;
            }
            if (options.ReloadOnChange)
            {
                ThreadPool.QueueUserWorkItem(obj => {
                    while (!isDisposed)
                    {
                        Load();
                        Thread.Sleep(interval);
                    }
                });
            }
        }

        public void Dispose()
        {
            this.isDisposed = true;
        }

        public override IEnumerable<string> GetChildKeys(IEnumerable<string> earlierKeys, string parentPath)
        {
            lockObj.EnterReadLock();
            try
            {
                return base.GetChildKeys(earlierKeys, parentPath);
            }
            finally
            {
                lockObj.ExitReadLock();
            }
        }

        public override bool TryGet(string key, out string value)
        {
            lockObj.EnterReadLock();
            try
            {
                return base.TryGet(key, out value);
            }
            finally
            {
                lockObj.ExitReadLock();
            }
        }

        public override void Load()
        {
            base.Load();
            IDictionary<string, string> clonedData=null;
            try
            {
                lockObj.EnterWriteLock();
                clonedData = Data.Clone();
                //获取表名
                string tableName = options.TableName;
                Data.Clear();
                using (var conn = options.CreateDbConnection())
                {
                    conn.Open();
                    //核心代码
                    DoLoad(tableName, conn);
                }
            }
            catch(DbException)
            {
                //if DbException is thrown, restore to the original data.
                this.Data = clonedData;
                throw;
            }
            finally
            {
                lockObj.ExitWriteLock();
            }
            //OnReload cannot be between EnterWriteLock and ExitWriteLock, or "A read lock may not be acquired with the write lock hel
            if (Helper.IsChanged(clonedData, Data))
            {
```

```
                OnReload();
            }
        }

        private void DoLoad(string tableName, System.Data.IDbConnection conn)
        {
            using (var cmd = conn.CreateCommand())
            {
                //从数据库中select所有的配置内容，如果name重名，则取Id最大的
                cmd.CommandText = $"select Name,Value from {tableName} where Id in(select Max(Id) from {tableName} group by Name)";
                using (var reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        string name = reader.GetString(0);
                        string value = reader.GetString(1);
                        if(value==null)
                        {
                            this.Data[name] = value;
                            continue;
                        }
                        value = value.Trim();
                        //if the value is like [...] or {} , it may be a json array value or json object value,
                        //so try to parse it as json
                        if(value.StartsWith("[") && value.EndsWith("]")
                            || value.StartsWith("{") && value.EndsWith("}"))
                        {
                            TryLoadAsJson(name, value);
                        }
                        else
                        {
                            this.Data[name] = value;
                        }
                    }
                }
            }
        }
        //将json格式做扁平化处理
        private void LoadJsonElement(string name, JsonElement jsonRoot)
        {
            if (jsonRoot.ValueKind == JsonValueKind.Array)
            {
                int index = 0;
                foreach (var item in jsonRoot.EnumerateArray())
                {
                    //https://andrewlock.net/creating-a-custom-iconfigurationprovider-in-asp-net-core-to-parse-yaml/
                    //parse as "a:b:0"="hello";"a:b:1"="world"
                    string path = name + ConfigurationPath.KeyDelimiter + index;
                    LoadJsonElement(path, item);
                    index++;
                }
            }
            else if (jsonRoot.ValueKind == JsonValueKind.Object)
            {
                foreach (var jsonObj in jsonRoot.EnumerateObject())
                {
                    string pathOfObj = name + ConfigurationPath.KeyDelimiter + jsonObj.Name;
                    LoadJsonElement(pathOfObj, jsonObj.Value);
                }
            }
            else
            {
                //if it is not json array or object, parse it as plain string value
                this.Data[name] = jsonRoot.GetValueForConfig();
            }
        }

        private void TryLoadAsJson(string name, string value)
        {
            var jsonOptions = new JsonDocumentOptions { AllowTrailingCommas = true, CommentHandling = JsonCommentHandling.Skip };
            try
            {
                var jsonRoot = JsonDocument.Parse(value, jsonOptions).RootElement;
                LoadJsonElement(name,jsonRoot);
            }
            catch (JsonException ex)
            {
                //if it is not valid json, parse it as plain string value
                this.Data[name] = value;
                Debug.WriteLine($"When trying to parse {value} as json object, exception was thrown. {ex}");
            }
        }
    }
}
```

DBConfigurationProviderExtensions

```
using System;
using System.Data;
using Zack.AnyDBConfigProvider;

namespace Microsoft.Extensions.Configuration
{
    public static class DBConfigurationProviderExtensions
    {
        public static IConfigurationBuilder AddDbConfiguration(this IConfigurationBuilder builder,
            DBConfigOptions setup)
        {
            return
                builder.Add(new DBConfigurationSource(setup));
        }

        public static IConfigurationBuilder AddDbConfiguration(this IConfigurationBuilder builder, Func<IDbConnection> createDbConnect
        {
            return AddDbConfiguration(builder, new DBConfigOptions {CreateDbConnection=createDbConnection,TableName=tableName,
                ReloadOnChange = reloadOnChange, ReloadInterval = reloadInterval});
        }
    }
}
```

Helper

```
using System.Collections.Generic;

namespace Zack.AnyDBConfigProvider
{
    static class Helper
    {
        public static IDictionary<string,string> Clone(this IDictionary<string, string> dict)
        {
            IDictionary<string, string> newDict = new Dictionary<string, string>();
            foreach(var kv in dict)
            {
                newDict[kv.Key] = kv.Value;
            }
            return newDict;
        }

        public static bool IsChanged(IDictionary<string, string> oldDict,
            IDictionary<string, string> newDict)
        {
            if(oldDict.Count!=newDict.Count)
            {
                return true;
            }
            foreach(var oldKV in oldDict)
            {
                var oldKey = oldKV.Key;
                var oldValue = oldKV.Value;
                var newValue = newDict[oldKey];
                if(oldValue!=newValue)
                {
                    return true;
                }
            }
            return false;
        }
    }
}
```

JsonElementExtensions

```
using System.Text.Json;

namespace Zack.AnyDBConfigProvider
{
    static class JsonElementExtensions
    {
        public static string GetValueForConfig(this JsonElement e)
        {
            if(e.ValueKind== JsonValueKind.String)
            {
                //remove the quotes, "ab"-->ab
                return e.GetString();
            }
```

```
            else if (e.ValueKind == JsonValueKind.Null
                || e.ValueKind == JsonValueKind.Undefined)
            {
                //remove the quotes, "null"-->null
                return null;
            }
            else
            {
                return e.GetRawText();
            }
        }
    }
}
```