上海越邦网络科技有限公司

秘级：公开

上海越邦网络科技有限公司

YuebonCore权限管理及快速开发框架

《前后端分离（Vue）版》

**技术白皮书**

**（V2.0）**

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件编号： | YB\_SE\_ABW\_01 |
| 文档名称： | YuebonCore FW权限管理及快速开发框架 |
| 编 写： | 陈青文 |
| 部 门： | 技术部 |
| 完成日期： |  |
| 审 核 |  |
| 审核日期： |  |

上海越邦网络科技有限公司【软件开发】项目组

**文档修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序 号 | 修改时间 | 修 改 人 | 审 核 人 | 备 注 |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |
| 14 |  |  |  |  |
| 15 |  |  |  |  |
| 16 |  |  |  |  |
| 17 |  |  |  |  |
| 18 |  |  |  |  |
| 19 |  |  |  |  |
| 20 |  |  |  |  |

目 录

[第1章 概述 4](#_Toc22767)

[1.1 项目简介 4](#_Toc5061)

[1.2 开发者信息 4](#_Toc19702)

[1.3 技术介绍 5](#_Toc31417)

[1.3.1 前端技术 5](#_Toc25115)

[1.3.2 后端技术 5](#_Toc9190)

[第2章 技术实现 6](#_Toc22751)

[2.1 体系结构 6](#_Toc25692)

[2.1.1 领域驱动设计整体架构 6](#_Toc13913)

[2.2 项目结构 8](#_Toc18421)

[2.2.1 核心基础项目 8](#_Toc3481)

[特别要注意的： 9](#_Toc22964)

[2.2.2 权限模块展示层 10](#_Toc14589)

[2.2.3 CMS示例模块 10](#_Toc16448)

[2.3 单点登录 11](#_Toc11085)

[2.3.1 IdentityServer4授权中心 11](#_Toc22294)

[2.3.2 MVC模式 11](#_Toc32410)

[2.4 用户登录 11](#_Toc29285)

[2.5 WebApi Token获取 11](#_Toc22321)

[2.6 权限控制 12](#_Toc10930)

[2.6.1 用户登录控制 12](#_Toc9005)

[2.6.2 控制器方法权限控制 12](#_Toc14007)

[2.6.3 界面元素权限控制 18](#_Toc17243)

[第3章 技术知识点 21](#_Toc19736)

[3.1 Session以及HttpContext对象 21](#_Toc20631)

[3.1.1 添加Session 21](#_Toc5347)

[3.1.2 修改startup.cs 21](#_Toc2259)

[3.1.3 Session扩展json存储和读取 21](#_Toc8955)

[3.1.4 IhttpContextAccessor注入 22](#_Toc2385)

[3.1.5 两种使用方法 23](#_Toc13268)

[3.2 Redis的使用 25](#_Toc26603)

[3.2.1 安装 25](#_Toc3124)

[3.2.2 Redis服务器配置 27](#_Toc2907)

[3.2.3 安装管理工具 Redis Desktop Manager(可选项) 27](#_Toc211)

[3.2.4 具体实现 29](#_Toc25299)

[第4章 手摸手使用YuebonCoreFW 32](#_Toc25609)

[4.1 添加业务数据层 32](#_Toc24437)

[4.1.1 实体类 32](#_Toc20979)

[第5章 Vue前端 33](#_Toc26216)

[5.1 环境搭建 33](#_Toc22192)

[5.2 常见错误及解决方法 33](#_Toc26770)

[5.2.1 ./src/main.js 33](#_Toc1964)

[Module build failed (from ./node\_modules/babel-loader/lib/index.js): 33](#_Toc30029)

[第6章 功能实现样例 34](#_Toc12303)

[6.1 打印与模板 34](#_Toc1930)

[6.1.1 功能描述 34](#_Toc29126)

[6.1.2 功能实现 35](#_Toc2365)

# 概述

YuebonCore基于.Net5.0的权限管理及快速开发框架，前后端分离，核心模块包括：组织机构、角色用户、权限授权、多系统、多应用管理、定时任务、业务单据编码规则、代码生成器等，整合应用最新技术包括Asp.NetCore MVC、EF、Dapper、WebAPI、Swagger、Vue等。它的架构易于扩展，是中小企业的首选。

YuebonCore其核心设计目标是开发迅速、代码量少、学习简单、功能强大、轻量级、易扩展，让Web开发更快速、简单，解决70%重复工作。轻松开发，专注您的业务，从YuebonCore开始！

## 项目简介

使用时请务必保留来源，请勿用于违反我国法律的web平台、如诈骗等非法平台网站。版权最终解释权归《YuebonCore团队》所有。

使用 MIT 协议，采用主流框架，容易上手，简单易学，学习成本低。可完全实现二次开发、基本满足80%项目需求。

代码生成器可以帮助解决.NET项目70%的重复工作，让开发更多关注业务逻辑。既能快速提高开发效率，帮助公司节省人力成本，同时又不失灵活性。

操作权限控制精密细致，对所有管理链接都进行权限验证，可控制到导航菜单、功能按钮。

数据权限（精细化数据权限控制，可以设置角色可以访问的数据范围，部门、工作组、公司数据）

常用类封装，日志、缓存、验证、字典、文件、邮件、,Excel。等等，目前兼容浏览器（IE11+、Chrome、Firefox、360浏览器等）

适用范围：可以开发OA、ERP、BPM、CRM、WMS、TMS、MIS、BI、电商平台后台、物流管理系统、快递管理系统、教务管理系统等各类管理软件。

在线体验地址：<http://netcore.ts.yuebon.com>（用户名：admin，密码：admin888）

官方文档：http://docs.v.yuebon.com

## 开发者信息

系统名称：YuebonCore快速开发平台

系统作者：YuebonCore团队

作者Q Q：381450948（微信同号）

QQ交流群：549652099

发布日期：2018年07月1日

版权所有：YuebonCore开发团队出品

开源协议：Mit

欢迎你加入我们一起共商、共建、共享技术成果！开源让我们进步，开源让我们开阔视野！

## 技术介绍

### 前端技术

Vue版前端技术栈 ：基于vue、vuex、vue-router 、vue-cli 、axios 和 element-ui。

前端采用vscode工具开发。

### 后端技术

核心框架：.Net5.0 + Web API + Dapper + EF + AutoMapper + swagger

定时计划任务：Quartz.Net组件

安全支持：过滤器、Sql注入、请求伪造

服务端验证：实体模型验证、自己封装Validator

缓存框架：微软自带Cache、Redis

日志管理：Log4net、登录日志、操作日志

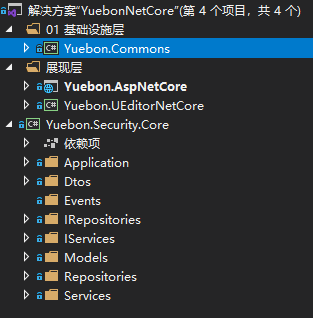
工具类：NPOI、验证码、丰富公共功能

性能分析：MiniProfiler组件

# 技术实现

## 项目结构

### 核心基础项目



Yuebon.Commons[基础类库]：包框架的核心组件，包含一系列快速开发中经常用到的Utility辅助工具功能，框架各个组件的核心接口定义，部分核心功能的实现；

Yuebon.Security.Core[权限管理类库]：以Security为基础实现以角色-功能、用户-功能的功能权限实现，以角色-数据，用户-数据的数据权限的封装；

Yuebon.AspNetCore[AspNetCore类库]，提供AspNetCore的服务端功能的封装，支持webapi和webmvc模式，同时支持插件式开发；

Yuebon.Cms.Core[CMS基础类库]，是文章管理模块，以此做案例给大家开发参考。

在以上4个项目放在单独的解决方案中，对以上4个项目单独构建包发布，方便在其他项目中引用。

Yuebon.WebApi[webapi接口]：为Vue版或其他三方系统提供接口服务。

DataBase是最新数据库备份文件，目前支持MS SQL Server和MySql。

Yuebon.Commons项目说明：

Attributes —— 一些常用的属性

Cache —— 缓存

CodeGenerator —— 代码生成器，用于生成Repository和Service层的代码

CodeTemplate —— Repository和Service层代码模板

Config —— 配置文件读取

Core —— 核心基础框架包括基础设施层

IRepositories —— Repository接口类，定义泛型接口,实体仓储模型的数据标准操作

IServices —— Service接口类，用来存放我们的对外服务接口

Repositories —— Repository层的父类，泛型仓储，实现泛型仓储接口

Services —— Service层的父类

Models —— 实体接口类，IBaseModel<TKey>

Moudle —— 模块应用管理

Extensions —— 一些常用的扩展方法

Filters —— 一些常用的拦截器

Helpers —— 一些常用的帮助类

Middlewares —— 中间件

Options —— 一些常用的配置类

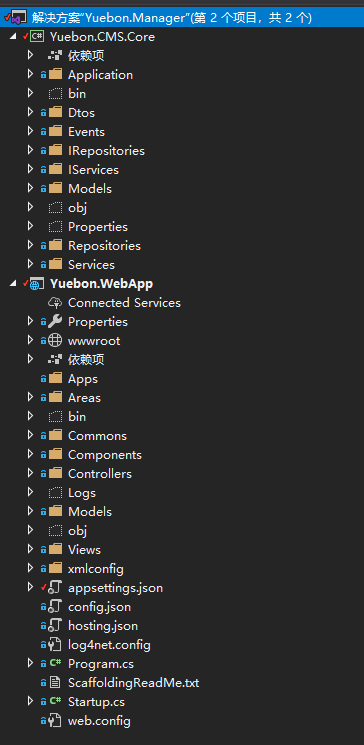
**特别要注意的：**

所有实体都需实现IBaseModel<TKey>接口（TKey是主键类型），如果需要在数据库中生成对应的数据表。

如果IRepositories、IServices、Repositories、Services这四个项目没有单独建立，调用代码生成器生成的代码将存在于调用项目的目录下。

利用代码生成器生成的代码文件需要手动添加到项目中。

### 权限模块展示层



为了更容易理解，我们将CMS模块作为示例一起介绍。

Yuebon.WebApp项目是权限管理模块展现层。采用基于asp.net core MVC模式实现，更简单、易上手。

### CMS示例模块

Yuebon.CMS.Core项目是CMS模块业务实现类库，放在这用户对比实现在Yuebon.WebApp项目基于核心模块的dll包引用和项目引用。

# 技术知识点

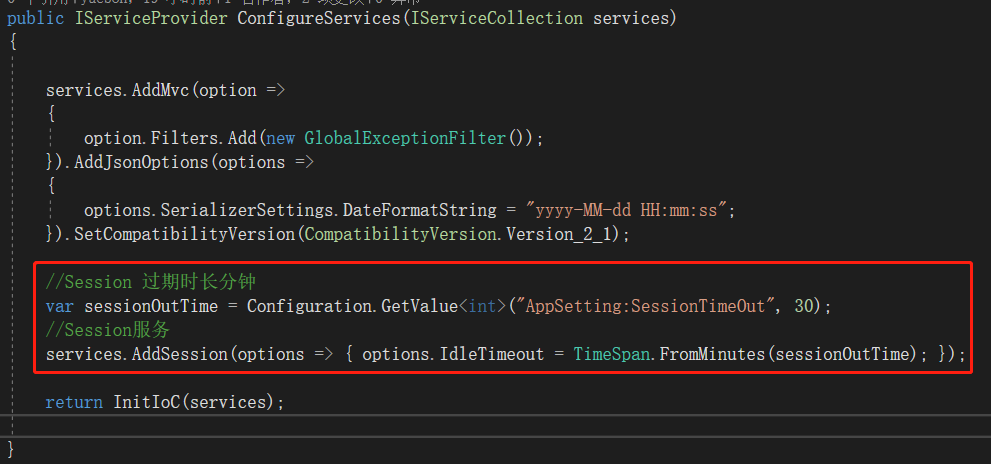
## Session以及HttpContext对象

### 添加Session

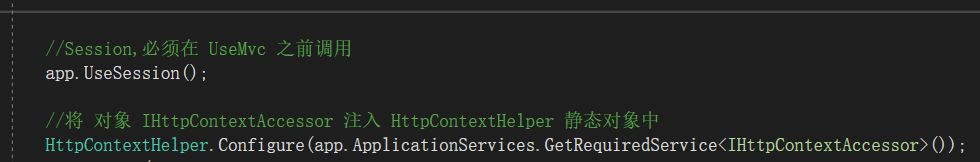
在你的项目上基于NuGet添加：Microsoft.AspNetCore.Session

### 修改startup.cs

在startup.cs找到方法ConfigureServices(IServiceCollection services) 注入Session(这个地方是Asp.net Core pipeline):services.AddSession()；如下图：



接下来我们要告诉Asp.net Core使用内存存储Session数据，在Configure(IApplicationBuilder app,...)中添加代码:app.UseSession()。所有中间件的配置顺序非常重要，必须在 UseSession 调用后才能访问 Session 。



### Session扩展json存储和读取

由于 Session 是基于 IDistributedCache 构建的，因此 Session 只能存储 byte[] 数据，这样使用起来很不方便，好在有很多扩展方法可以用来直接读取和保存。我们设置一个扩展类SessionExtensions直接将实体类序列化成json存储和读取。代码如下：

/// <summary>

/// Session 扩展函数,直接将实体类序列化成json存储和读取

/// </summary>

public static class SessionExtensions

{

/// <summary>

/// 设置session值

/// </summary>

/// <param name="session"></param>

/// <param name="key"></param>

/// <param name="value"></param>

public static void Set(this ISession session, string key, object value)

{

session.SetString(key, JsonConvert.SerializeObject(value));

}

/// <summary>

/// 获取session

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="session"></param>

/// <param name="key"></param>

/// <returns></returns>

public static T Get<T>(this ISession session, string key)

{

var value = session.GetString(key);

return value == null ? default(T) : JsonConvert.DeserializeObject<T>(value);

}

}

### IhttpContextAccessor注入

ASP.NET Core中提供了一个IHttpContextAccessor接口，HttpContextAccessor 默认实现了它简化了访问HttpContext。

它必须在程序启动时在IServicesCollection中注册，这样在程序中就能获取到HttpContextAccessor，并用来访问HttpContext。

修改starup.cs，在public void Configure(IApplicationBuilder app, IHostingEnvironment env)中添加如下代码：

//将 对象 IHttpContextAccessor 注入 HttpContextHelper 静态对象中 HttpContextHelper.Configure(app.ApplicationServices.GetRequiredService<IHttpContextAccessor>());

在private IServiceProvider InitIoC(IServiceCollection services)添加如下代码：

services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

HttpContext注入类代码如下：

/// <summary>

/// HttpContext注入类

/// </summary>

public static class HttpContextHelper

{

private static IHttpContextAccessor httpContextAccessor;

public static void Configure(IHttpContextAccessor \_httpContextAccessor)

{

httpContextAccessor = \_httpContextAccessor;

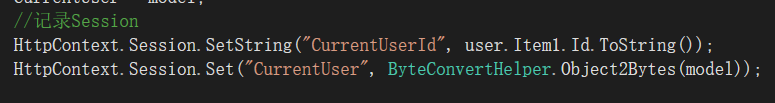
}

public static HttpContext HttpContext => httpContextAccessor.HttpContext;

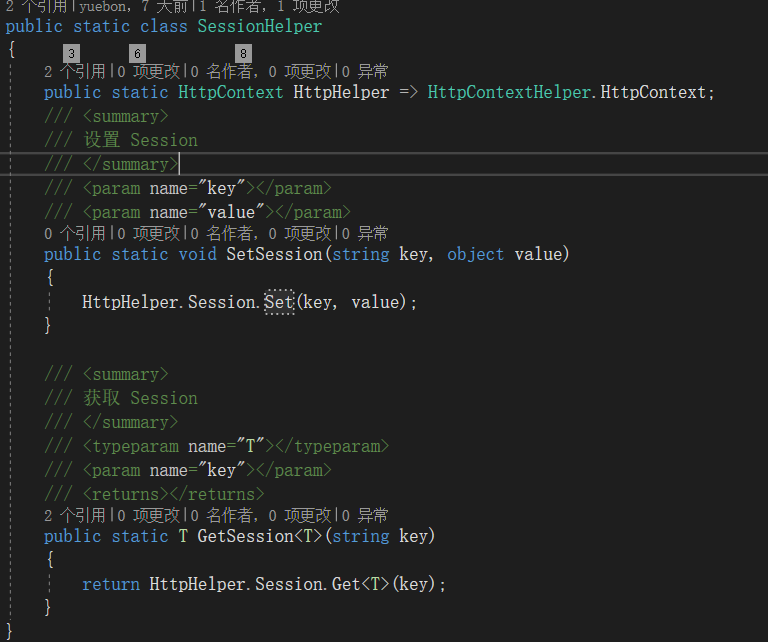
}

### 两种使用方法

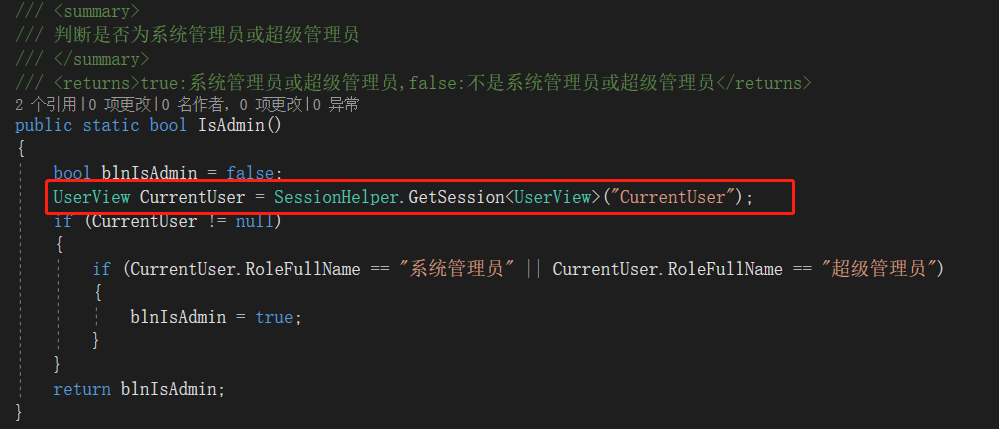
1、在MVC Controller里使用HttpContext.Session



2、如果不是在Controller里，依赖注入IHttpContextAccessor



在项目中应用例如根据用户校验是否是管理员时，需要获取当前登录用户session值，如下代码:



## Redis的使用

### 安装

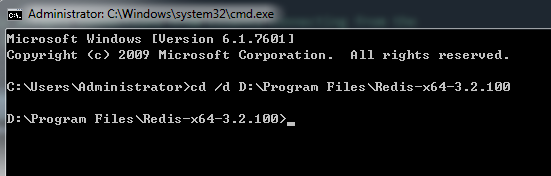
1.下载最新版redis，选择.zip则是免安装的版本

下载地址：<https://github.com/MicrosoftArchive/redis/releases>



2.解压到指定目录，并运行cmd命令

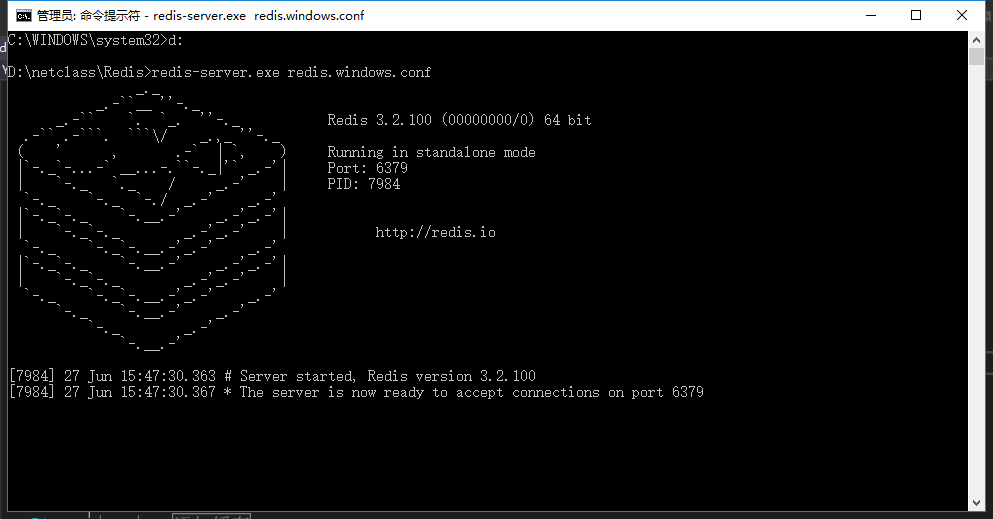
3.在该文件夹下运行命令：redis-server.exe redis.windows.conf





这里需要注意的是一定要带后面的启动参数，如果对配置文件修改完没有带启动参数并不会生效的，所以一定记得带参数。

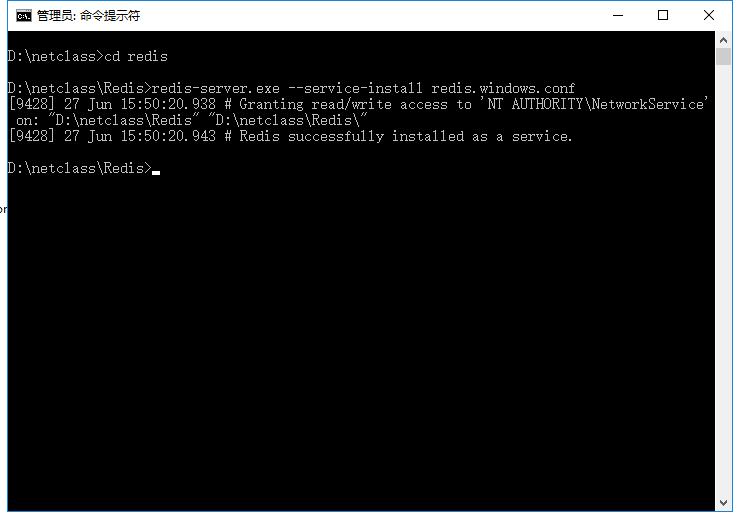
4.看到如下显示，则表示启动成功



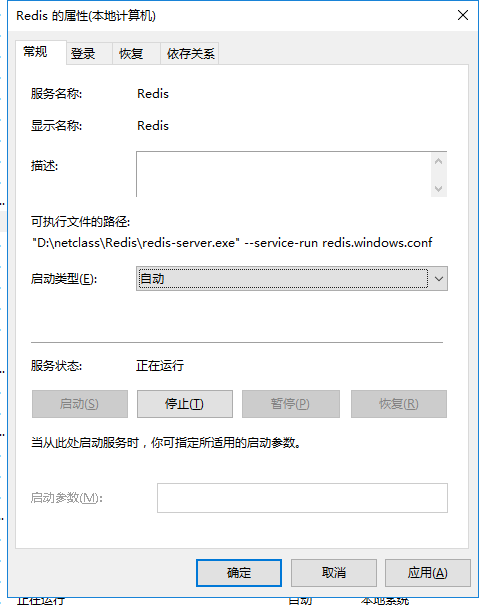
这样服务就算启动成功了，这个窗口是服务终端，如果窗口被关闭，对应的服务也会停止。

5.将redis安装成服务

在该文件夹下运行命令：redis-server.exe --service-install redis.windows.conf



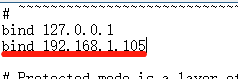
去服务列表查询服务，可以看到redis服务默认没有开启，开启redis服务（可以设置为开机自动启动），如下图：



### Redis服务器配置

#### Windows服务器配置

Windows系统配置文件是redis.windows.conf，系统初始默认绑定的IP地址为127.0.0.1，端口为6379。也可以自定义服务器IP地址,方式bind Ip地址。如图修改：



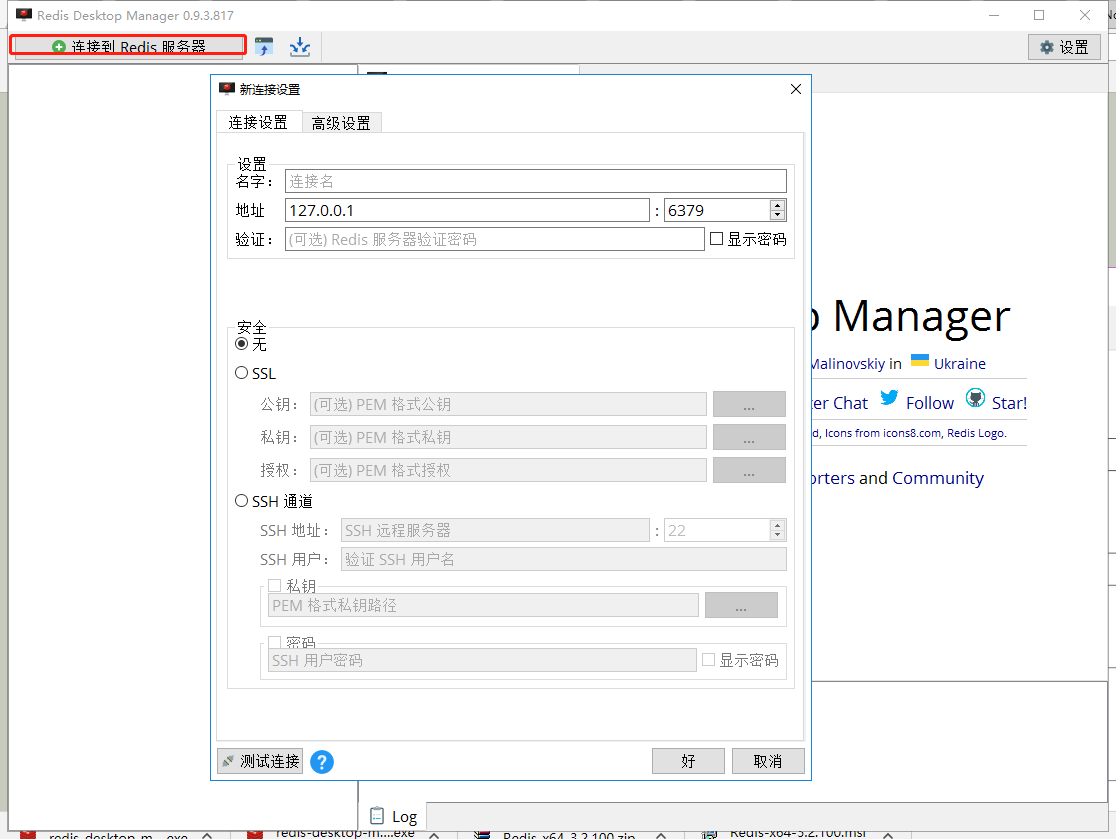
### 安装管理工具 Redis Desktop Manager(可选项)

#### 下载安装

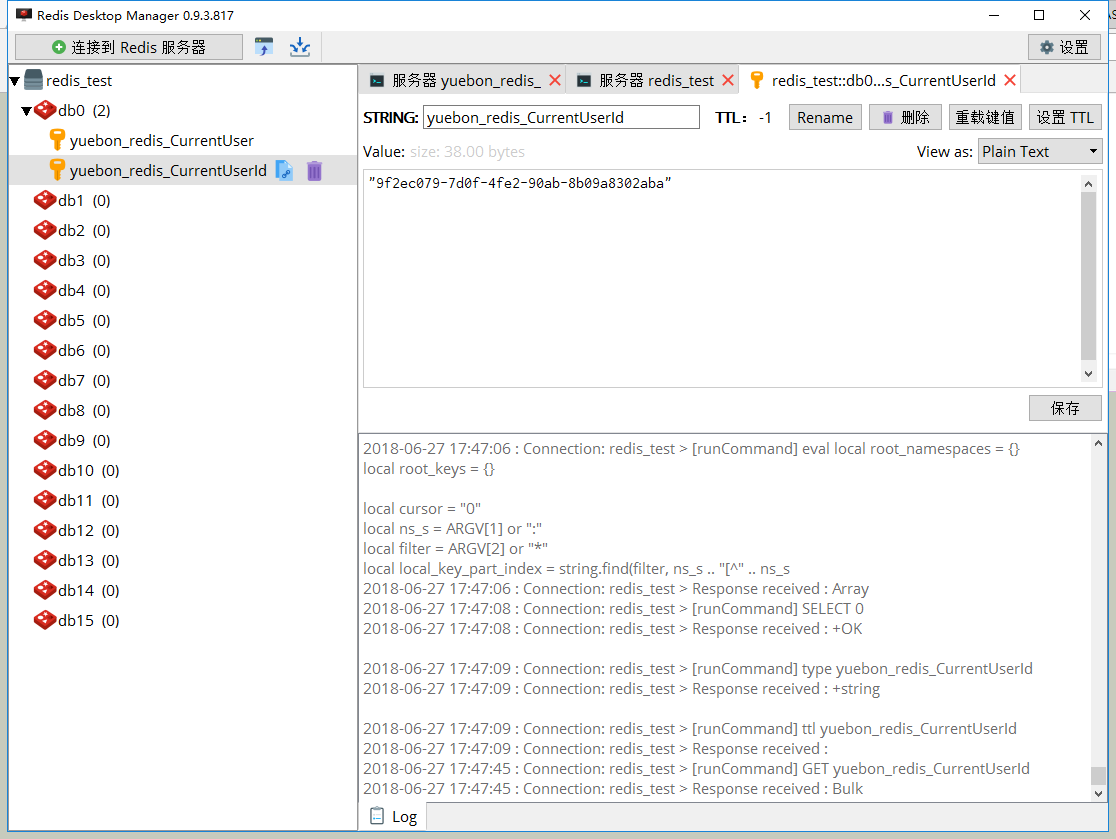
下载地址：<https://redisdesktop.com/download>

#### 使用

1、添加连接



2、查看数据



### 具体实现

#### 添加引用包

需要引用Microsoft.Extensions.Caching.Redis和StackExchange.Redis两个Redis包。

#### appsettings配置

在appsettings.json添加配置

"CacheProvider": {

"UseRedis": true,

"Redis\_ConnectionString": "127.0.0.1:6379,allowAdmin=true,password=123456,defaultdatabase=0",

"Redis\_InstanceName": "yuebon\_redis\_"

}

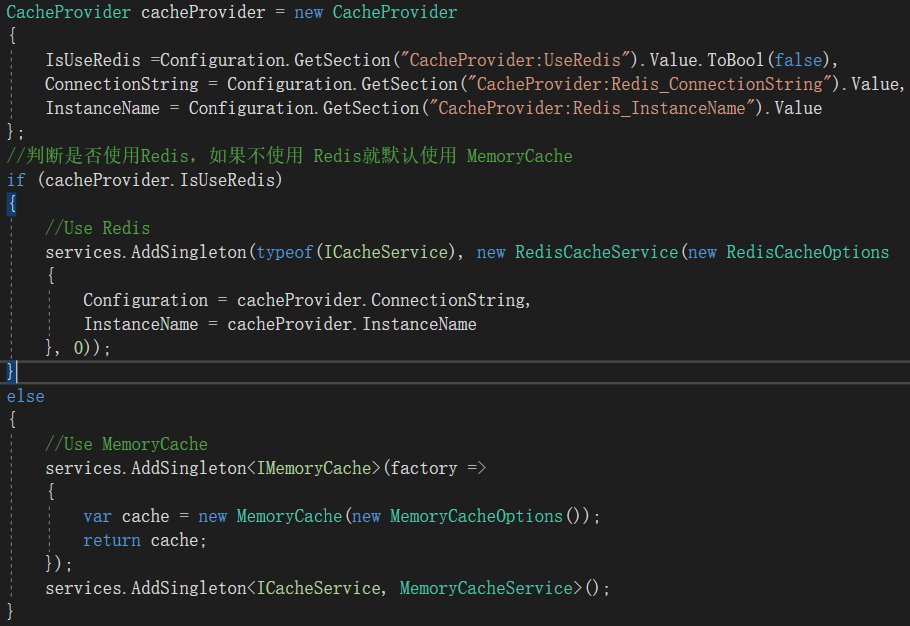
我们写一个类，来获取配置几个配置，CacheProvider说明：

1）、UseRedis：是否使用Redis缓存，不使用设置false即可；

2）、Redis\_ConnectionString：Redis连接，按示例参数进行配置，allowadmin、password、defaultdatabase都需要，其他配置参数请查阅相关文档；

3）、Redis\_InstanceName：Redis实例名称

#### 在startup添加redis服务



系统支持Redis和MemoryCache两种缓存方式，根据配置自动识别。

#### 用Redis存储Session

如果需要session保存在redis中需要添加服务如下：

services.AddDistributedRedisCache(options =>

{

options.Configuration = cacheProvider.ConnectionString;

options.InstanceName = cacheProvider.InstanceName;

});

关键在使用AddDistributedRedisCache。其他这不能自动保存。

#### 具体实现

1、配置参数类，代码如下：



2、缓存接口 ICacheService

不论使用Redis还是MemoryCache都需要实现该接口的方法。具体代码参考项目中的ICacheService.cs

3、Redis实现ICacheService

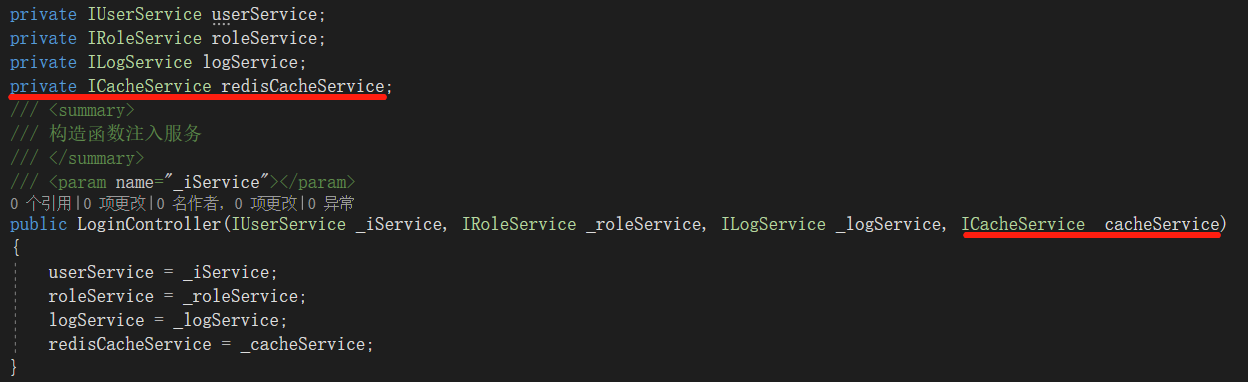
具体代码参考项目中的RedisCacheService.cs

4、MemoryCache实现ICacheService

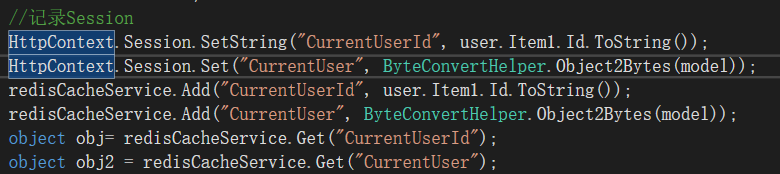
具体代码参考项目中的MemoryCacheService.cs

#### 使用

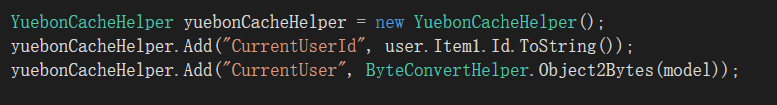
我们以登录为例说明：



具体使用：



从列子可以看出每次使用都需要注册服务和区分使用MemoryCache或Redis进行缓存，为此我们封装了YuebonCacheHelper类，会自动适配缓存类型。使用列子如下：



也可以在Startup中注入使用如：

services.AddTransient<MemoryCacheService>();

## Token和参数签名

系统从外部获取数据时，通常采用API接口调用的方式来实现，我们通过http Post或者Get方式请求服务器的时候，请求方和接口提供方之间的通信过程，有这几个安全性问题需要考虑：

1、请求参数是否被篡改？

2、请求来源是否合法？

3、请求是否具有唯一性(不可复制)，防止请求被恶意攻击？

为了保证数据在通信时的安全性，我们可以采用TOKEN+参数签名的方式来进行相关验证，是WebApi接口服务最重要的安全防护手段之一。结合YuebonCore项目中实际使用情况，介绍下前后端Token+参数签名校验实现方案。

在YuebonCore中约定所有接口默认都需要进行Token验证、签名验证和登录访问。目前API的访问控制，分为三种：

1. 匿名访问AllowAnonymousAttribute：表示当前功能不需要登录即可访问，无视登录状态、角色、Token要求。
2. 不需要签名验证NoSignRequiredAttribute：表示不需要参数签名验证，其他验证仍然需要遵守。
3. 不需要登录访问NoPermissionRequiredAttribute：表示当前功能不需要登录访问，但需要验证token。
4. 功能权限YuebonAuthorizeAttribute：表示当前功能需要登录并且用户拥有指定功能权限，才能访问，未登录或者登录但未拥有指定功能权限，拒绝访问。

### Token获取

access\_token是接口的全局唯一接口调用凭据，接口调用各接口时都需使用access\_token。开发者需要进行妥善保存。access\_token的存储至少要保留512个字符空间。access\_token的有效期目前为2个小时，需定时刷新，重复获取将导致上次获取的access\_token失效。

目前access\_token的有效期通过返回的expire\_in来传达，目前是7200秒之内的值。中控服务器需要根据这个有效时间提前去刷新新access\_token。在刷新过程中，中控服务器可对外继续输出的老access\_token，此时接口平台后台会保证在5分钟内，新老access\_token都可用，这保证了第三方业务的平滑过渡。

#### 应用获取Token

我们在前后端独立项目中每一个需要访问接口的应用程序我们都称其为一个应用。应用在访问接口时须根据分配的应用Id（AppId）和应用密钥（Appsecret）来获取token。我们推荐为不同应用分配不同的AppId和Appsecret，具体可以在应用管理模块获取，同时需要对调用接口的请求地址进行授权，未获得许可的请求地址将不能获取token值，在“开发者-应用管理”进行设置。

**接口调用请求说明**

https请求方式: GET //xxx/api/Token

**参数说明**

|  |  |  |
| --- | --- | --- |
| **参数** | **是否必须** | **说明** |
| grant\_type | 是 | 获取access\_token填写client\_credential |
| appid | 是 | 第三方用户唯一凭证 |
| secret | 是 | 第三方用户唯一凭证密钥，即appsecret |

**返回说明**

正常情况下，接口会返回下述JSON数据包给前端。

{"access\_token":"ACCESS\_TOKEN","expires\_in":7200}

**参数说明**

|  |  |
| --- | --- |
| **参数** | **说明** |
| access\_token | 获取到的凭证 |
| expires\_in | 凭证有效时间，单位：秒 |

错误时接口会返回错误码等信息，JSON数据包示例如下（该示例为获取access\_token时AppID或AppSecret错误）:

{"errcode":40001,"errmsg":"invalid appid"}

**返回码说明**

|  |  |
| --- | --- |
| **返回码** | **说明** |
| -1 | 系统繁忙，此时请开发者稍候再试 |
| 0 | 请求成功 |
| 40001 | 获取access\_token时AppID或AppSecret错误。请开发者认真比对appid和AppSecret的正确性，或查看是否正在为恰当的应用调用接口 |
| 40002 | 调用接口的服务器URL地址不正确，请联系供应商进行授权 |
| 40003 | 请确保grant\_type字段值为client\_credential |

生成token的核心算法由TokenProvider类提供，它提供了应用获取token的方法GenerateToken()、根据用户信息获取token的方法LoginToken()和验证token的方法ValidateToken()。

#### 登录获取Token

应用获取的token不包含用户信息，用户登录获取token在登录接口中调用TokenProvider的LoginToken()生成。

参考代码如下：

|  |
| --- |
| **User user = userLogin.Item1;//获取用户信息**  JwtOption jwtModel = App.GetService<JwtOption>();  TokenProvider tokenProvider = new TokenProvider(jwtModel);  //根据用户信息和appid获取新的token  **TokenResult tokenResult = tokenProvider.LoginToken(user, appId);**  YuebonCurrentUser currentSession = new YuebonCurrentUser  {  UserId = user.Id,  Name = user.RealName,  AccessToken = tokenResult.AccessToken,  AppKey = appId,  CreateTime = DateTime.Now,  Role = \_roleService.GetRoleEnCode(user.RoleId),  ActiveSystemId = systemType.Id,  CurrentLoginIP = strIp,  IPAddressName = ipAddressName  }; |

### 参数签名

通过参数签名实现放置请求参数是否被篡改和请求唯一性。这是目前较为主流的一种安全保护方式。

#### 参数签名规则

它要求调用方按照约定好的算法生成签名字符串，作为请求的一部分，接口提供方验算签名即可知是否合法。步骤通常如下：

①接口提供方给出appid和appsecret

②调用方根据appid和appsecret、timestamp、nonce以及请求参数，按照一定算法生成签名sign

③接口提供方验证签名

生成签名的步骤如下：

1、将所有业务请求参数按字母先后顺序排序

2、参数名称和参数值链接成一个字符串A

3、发起请求时的时间戳timestamp（单位：毫秒）

4、随机字符串nonce，由数字和字母组成32位字符串。

5、由timestamp+nonce+字符串A+appsecret组成一个新字符串B

对字符串B进行md5得到签名sign

假设请求的参数为：f=1,b=23,k=33，排序后为b=23,f=1,k=33，参数名和参数值链接后为b23f1k33，尾部加上appsecret后md5：md5(timestampnoncekey1value1key2value2...secret)。

#### 前端签名

**1、前端Axios请求统一拦截处理**

我们前端采用vue家族技术，客户端http请求以Axios进行统一的拦截处理。axios的请求拦截是最好用的。

|  |
| --- |
| // request拦截器  service.interceptors.request.use(    config => {      if (config.headers['Content-Type'] === undefined) { config.headers['Content-Type'] = 'application/json;charset=UTF-8' }      const token = getToken()      if (token) {        config.headers['Authorization'] = 'Bearer ' + token      }      const timeStamp = new Date().getTime().toString().substr(0, 10)      const nonce = GetRandomString()      const iSSign = config.headers['sign']      if (iSSign || iSSign === undefined) {        config.headers['appId'] = store.getters.appId        config.headers['nonce'] = nonce        config.headers['timeStamp'] = timeStamp        config.headers['signature'] = sign(config, nonce, timeStamp, store.getters.appSecret)      }      return config    },    error => {      // do something with request error      return Promise.reject(error)    }  ) |

通过代码可以看出headers中传递sign参数来控制是否需要签名验证。当sign为true或者没有设置sign时默认需要签名。设置sign为false时不需要签名。

1. **签名实现**

按照签名规则进行签名实现，包含了随机字符串nonce生成和签名加密。

import md5 from 'js-md5'

/\*\*

 \* 获取一个32位随机字符串

 \* @returns

 \*/

const GetRandomString = () => {

  var chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz123456789'

  var maxPos = chars.length

  var pwd = ''

  for (var i = 0; i < 32; i++) {

    pwd += chars.charAt(Math.floor(Math.random() \* maxPos))

  }

  return pwd

}

/\*\*

\* 签名

\* @param config 请求配置

\* @param nonce 随机字符串

\* @param timestamp 签名时间戳

\* @param appSecret 应用密钥

\* @param method 请求方式

\*/

const sign = (config, nonce, timestamp, appSecret) => {

  // 签名格式： timestamp + nonce + data(字典升序)+appSecret

  const ret = []

  if (config.params) {

    const pArray = []

    const data = config.params

    for (const it in data) {

      pArray.push(it)

    }

    // 字典升序

    const sArray = pArray.sort()

    for (const it of sArray) {

      let val = data[it]

      if (typeof val === 'object' && //

        (!(val instanceof Array) || (val.length > 0 && (typeof val[0] === 'object')))) {

        val = JSON.stringify(val)

      }

      ret.push(it + val)

    }

  } else {

    ret.push(JSON.stringify(config.data))

  }

  const signsrc = timestamp + nonce + ret.join('') + appSecret

  return md5(signsrc)

}

export {

  sign,

  GetRandomString

}

#### API签名验证

我们都知道http请求有两种传参形式。

1、通过url传参，最常见的就是get请求(实际上post,put,delete都可以使用这种传参方式)，如:

<http://api.XXX.com/getproduct?id=value1>

2、通过request body传参，最常见的就是post请求。

我们针对于以上两种传参方式，采用不同的签名校验规则(注:签名算法规则仅供参考)。WebApi是不支持通过url和body同时传参数的,所以在服务端可以通过Request.QueryString获取到form参数进行判断,执行不同逻辑。

|  |
| --- |
| .....  //根据请求类型拼接参数  NameValueCollection form = HttpUtility.ParseQueryString(request.QueryString.ToString());  var data = string.Empty;  if (form.Count > 0)  {  data = GetQueryString(form);  }  else  {  request.EnableBuffering();  Stream stream = request.Body;  StreamReader streamReader = new StreamReader(stream);  data = streamReader.ReadToEnd();  request.Body.Position = 0;  }  ...... |

1、url传参Get请求方式参数处理

按照请求参数名称将所有请求参数按照字母先后顺序排序得到：keyvaluekeyvalue...keyvalue。字符串如：将arong=1,mrong=2,crong=3 排序为：arong=1, crong=3,mrong=2；然后将参数名和参数值进行拼接得到参数字符串：arong1crong3mrong2。

代码实现：

|  |
| --- |
| /// <summary>  /// get请求查询参数， url上直接接参数时,通过此方法获取  /// </summary>  /// <param name="form">请求参数</param>  /// <returns></returns>  public static string GetQueryString(NameValueCollection form)  {  //第一步：取出所有get参数  Dictionary<string, string> parames = new Dictionary<string, string>();  for (int f = 0; f < form.Count; f++)  {  var key = form.Keys[f];  if (key != null)  parames.Add(key, form[key]);  }  // 第二步：把字典按Key的字母顺序排序  IDictionary<string, string> sortedParams = new SortedDictionary<string, string>(parames);  IEnumerator<KeyValuePair<string, string>> dem = sortedParams.GetEnumerator();  // 第三步：把所有参数名和参数值串在一起  StringBuilder query = new StringBuilder(""); //签名字符串  if (parames == null || parames.Count == 0) return query.ToString();  while (dem.MoveNext())  {  string key = dem.Current.Key;  string value = dem.Current.Value;  if (!string.IsNullOrEmpty(key)) query.Append(key).Append(value);  }  return query.ToString();  } |

2、post请求验证

将获取request.Body，即传给接口的json序列化参数或请求参数。代码如下：

|  |
| --- |
| request.EnableBuffering();  Stream stream =request.Body;  StreamReader streamReader = new StreamReader(stream);  data = streamReader.ReadToEnd();  request.Body.Position = 0; |

1. 签名验证

通过上述逻辑之后，data变量中存储的就是接口参数内容。以下是实际签名校验逻辑。

|  |
| --- |
| /// <summary>  /// 签名验证  /// </summary>  /// <param name="timeStamp">时间戳</param>  /// <param name="nonce">随机字符串</param>  /// <param name="appSecret">客户端应用密钥</param>  /// <param name="data">接口参数内容</param>  /// <param name="signature">当前请求内容的数字签名</param>  /// <returns></returns>  public static bool Validate(string timeStamp,string nonce,string appSecret, string data,string signature)  {  var signStr = timeStamp + nonce + data + appSecret;  string signMd5 = MD5Util.GetMD5\_32(signStr);  return signMd5 == signature;  } |

这里使用了md5的算法进行签名，也可以自行选择其他签名方式，例如RSA，SHA等。

1. Action拦截器验证

YuebonCore将在权限验证拦截器YuebonAuthorizationFilter中进行统一拦截进行签名验证。代码如下:

|  |
| --- |
| ......  #region 签名验证  bool boolSign = context.HttpContext.Request.Headers["sign"].SingleOrDefault().ToBool(true);  var isSign = controllerActionDescriptor.MethodInfo.GetCustomAttributes(inherit: true)  .Any(a => a.GetType().Equals(typeof(NoSignRequiredAttribute)));  //需要签名验证  if (!isSign && boolSign)  {  CommonResult resultSign = SignHelper.CheckSign(context.HttpContext);  if (!resultSign.Success)  {  context.Result = new JsonResult(resultSign, options);  return;  }  }  #endregion  ...... |

针对部分接口我们不需要签名验证，将通过自定义的NoSignRequiredAttribute属性进行标注或前端header中设置sign为false即可。我们在拦截器中对未标注[NoSignRequired]且sing为true的才进行验证。需要特别注意的时header中未包含sign时默认为需要验证签名。

以上签名方法安全有效地解决了参数被篡改和身份验证的问题，如果参数被篡改，没事，因为别人无法知道appsecret，也就无法重新生成新的sign。

md5签名方法可以保证来源及请求参数的合法性，但是请求链接一旦泄露，可以反复请求，对于某些拉取数据的接口来说并不是一件好事，相当于是泄露了数据。

#### 请求唯一性保证

在请求中带上时间戳，并且把时间戳也作为签名的一部分，在接口提供方对时间戳进行验证，只允许一定时间范围内的请求，例如1分钟。因为请求方和接口提供方的服务器可能存在一定的时间误差，建议时间戳误差在5分钟内比较合适。允许的时间误差越大，链接的有效期就越长，请求唯一性的保证就越弱。所以需要在两者之间衡量。

字符串nonce是一个唯一且为随机字符串，用于唯一标识每个已签名的请求。通过每个请求具有唯一标识符，服务提供商能够防止请求多次使用。由于签名中包含 nonce 值，因此攻击者在不知道共享机密的情况下无法更改该值。对于服务提供商来说，使用 nonce 的成本可能非常昂贵，因为他们需要持续存储收到的所有 nonce 值。为了更轻松地实现，OAuth 向每个请求添加了一个时间戳值，允许服务提供商在有限的时间内仅保留 nonce 值。

从安全角度来看，真正的 timeStamp时间戳值和 nonce 字符串的组合。只有一起提供永久的唯一价值，攻击者永远不会再使用。

目前我们采用 timeStamp时间戳值和 nonce 字符串的组合，缓存了2小时，可以确保2小时内不可在使用该请求。

#### 秘钥的保存

在签名的过程中，起到决定性作用之一的是appsecret，因此如何保存成为关键。我们分类讨论。

接口调用方的代码跑在服务器的情况比较好办，除非服务器被攻陷，否则外接无法知道appsecret，当然，要注意不能往日志里写入appsecret的值，其他敏感值也禁止写入日志，如账号密码等信息。

假如是客户端请求接口，就需要多想一步了。假如把appsecret硬编码到客户端，会有反编译的风险，特别是android。可以在客户端登陆验证成功后，返回给客户端的信息中带上appsecret(当然，返回的数据也可能被拦截，真是防不胜防啊。。。)。特别说明一下，在android开发中，假如硬要把appsecret硬编码，建议把appsecret放到NDK中编译成so文件，app启动后去读取。

# 手摸手使用YuebonCore

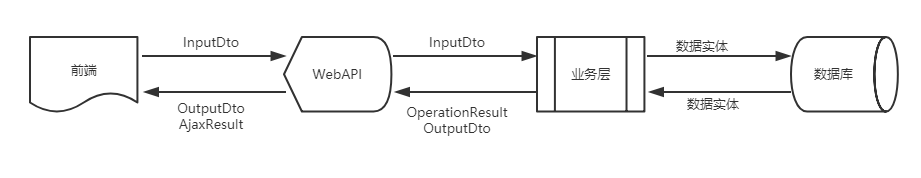
## 添加业务数据层

一个模块的数据层，主要包括如下几个方面：

数据实体：数据实体是服务层向数据库进行数据持久化的数据载体，也是数据层进行业务处理的数据载体，数据层与数据层的数据交换都应通过实体类来完成。

输入DTO：InputDto用于前端向API层输入数据，API层向服务层输入数据

输出DTO：OutputDto主要用于数据查询服务，是API层向前端输出数据的载体



数据实体映射配置：CodeFirst的开发模式，数据库的设计细节完全靠代码来完成，数据实体映射配置正是负责这项工作的，针对一个实体，可以在这里配置其在数据库中的数据表关系、数据约束及各个数据字段的每一个细节配置。

对象Mapper映射：数据传输对象DTO 与 实体类Entity 之间的转换与更新，如果都要通过手写代码来进行属性的一一对应赋值，是件很累人的事，通过 对象Mapper映射（例如AutoMapper）功能，只需进行一次配置，即可很方便的实现不同类型对象的数据转换与更新。

### 实体类

# Vue前端

## 环境搭建

# 克隆项目

git clone https://github.com/PanJiaChen/vue-element-admin.git

# 进入项目目录

cd vueUI

# 安装依赖

npm install

# 建议不要用 cnpm 安装 会有各种诡异的bug 可以通过如下操作解决 npm 下载速度慢的问题

npm install --registry=https://registry.npm.taobao.org

# 本地开发 启动项目

npm run dev

# 解决node-sass安装不成功的问题

npm i node-sass --sass\_binary\_site=https://npm.taobao.org/mirrors/node-sass

项目打包命令：npm run build:prod --report

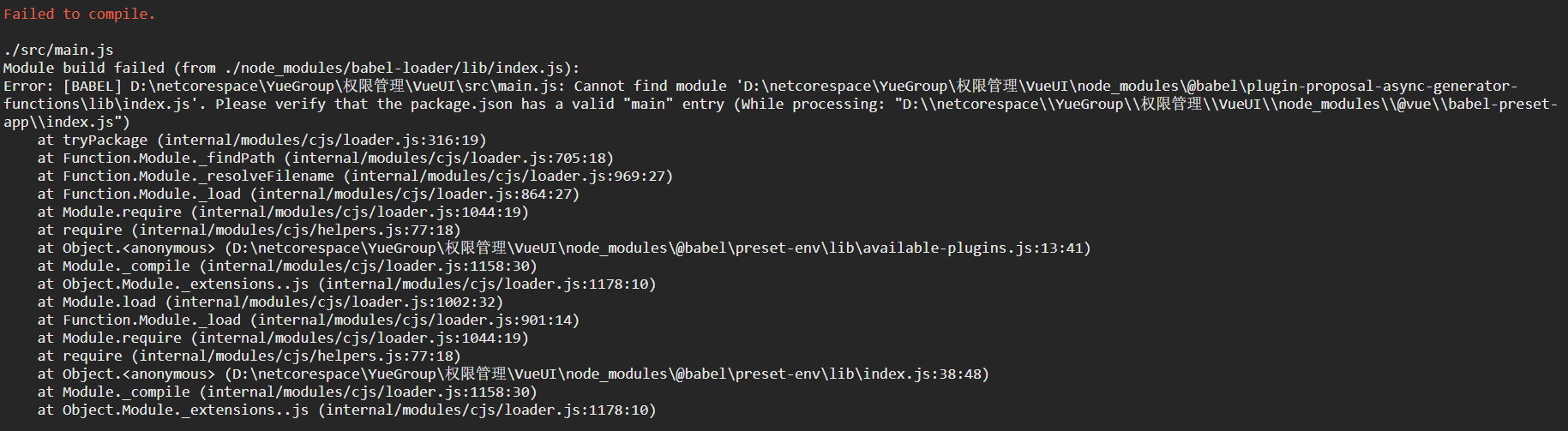
## 常见错误及解决方法

### ./src/main.js

启动后出现如下图错误：

### Module build failed (from ./node\_modules/babel-loader/lib/index.js):

解决方法：



控制台输入  npm install @babel/core @babel/preset-env 命令

# 功能实现样例

## 打印与模板

### 功能描述

通过富文本编辑器编辑打印模板来实现自定义打印模板，模板必须采用表格“<table></table>”标签以及相关标签进行定义。

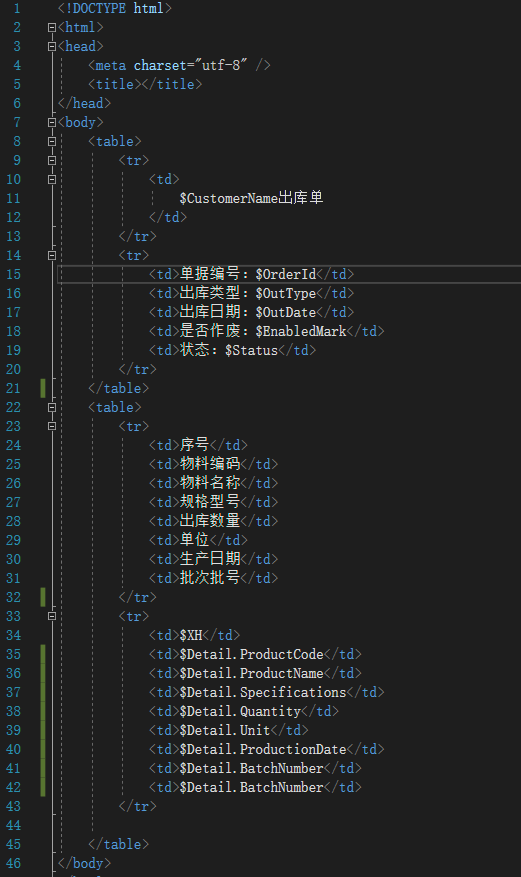
用户自定义模板内容保存为独立网页文件，以.html结尾。

在定义模板时所有变量采用“$”加模块类属性名称进行定义，明细列表采用“$Detail.”+明细对象类属性名称定义，如果没有明细列表可以不定义，一般情况下一个模板只能定义一个明细列表，需要定多个明细要重新定义明细变量，结合业务模块实现。

明细列表的如果需要序号请“$XH”进行定义。需要二维码请用“$PrintBarCode”定义。

### 功能实现

#### 打印模板例子



#### 服务端代码实现

一、实现步骤流程

1. 读取模板文件
2. 替换模板定义变量。要优先替换主表信息，然后在替换明细列表。
3. 在替换时针对不能直接使用的属性值需要提前处理并替换掉，然后在通过方法ObjectReplaceString()进行替换。

二、关键代码

/// <summary>

/// 出库单详情打印

/// </summary>

/// <param name="idList"></param>

/// <returns></returns>

public async Task<CommonResult> PrintOutStockDetail(string idList)

{

CommonResult result = new CommonResult();

try

{

if (!string.IsNullOrEmpty(idList))

{

string soureFile = AppContext.BaseDirectory + "wwwroot\\printtemplate\\outstock\\PrintOutStockDetail.html";

if (!File.Exists(soureFile))

{

result.ErrMsg = "打印模板不存在";

return result;

}

string returnStrHtml = "";

string where = "id in ('" + idList.Trim(',').Replace(",", "','") + "')";

IEnumerable<OutStock> list = await \_repository.GetListWhereAsync(where);

var doc = new HtmlDocument();

doc.Load(soureFile);

string templateStr = doc.DocumentNode.OuterHtml;

foreach (OutStock info in list)

{

string pagehtml = templateStr;

pagehtml = pagehtml.Replace("<body>", "<body> <div id=\"printHtml" + info.Id+">");

pagehtml = pagehtml.Replace("</body>", " </div></body>");

#region 表头的替换

//先处理枚举类型数据和关联数据,如客户名称、仓库名称，出库类型，替换后在做如下操作

string creatorUser = \_userRepository.Get(info.CreatorUserId).NickName;

pagehtml = pagehtml.Replace("$CreatorUser", creatorUser);

if (!string.IsNullOrEmpty(info.CheckUserId))

{

string checkUser = \_userRepository.Get(info.CheckUserId).NickName;

pagehtml = pagehtml.Replace("$CheckUser", checkUser);

}

if (!string.IsNullOrEmpty(info.LastModifyUserId))

{

string lastModifyUser = \_userRepository.Get(info.LastModifyUserId).NickName;

pagehtml = pagehtml.Replace("$LastModifyUser", lastModifyUser);

}

string status = string.Empty;

if (info.Status=="0")

{

status = "待审核";

}

else if (info.Status == "1")

{

status = "待分拣";

}

else if (info.Status == "2")

{

status = "分拣中";

}

else if (info.Status == "3")

{

status = "打包中";

}

else if (info.Status == "4")

{

status = "待出库";

}

else if (info.Status == "5")

{

status = "已出库";

}

else if (info.Status == "7")

{

status = "已作废";

}

pagehtml = pagehtml.Replace("$Status", status);

//其他可以直接用属性值替换的内容采用如下批量处理

pagehtml = ObjectReplaceHtmlHelper.ObjectReplaceString(info, pagehtml);

#endregion

#region 分录明细处理

if (pagehtml.Contains("$Detail."))

{

HtmlNodeCollection htmlNodes = doc.DocumentNode.SelectNodes("//table//tr");

where = string.Format("OrderId='{0}' order by EntryId asc", info.OrderId);

string resultTr = string.Empty;

IEnumerable<OutStockDetail> outStockDetailList = await \_dRepository.GetListWhereAsync(where);

string itemListStr = string.Empty;//用于保存分录数据字符串

string trReplaceStr = "";//用于保存需要替换的tr字符串

foreach (HtmlNode nnoder in htmlNodes)

{

if (nnoder.OuterHtml.Contains("$Detail."))

{

trReplaceStr = nnoder.OuterHtml;

break;

}

}

int xu = 1;//序号

foreach (OutStockDetail itemdetail in outStockDetailList)

{

itemListStr += ObjectReplaceHtmlHelper.ObjectReplaceString(itemdetail, trReplaceStr, "Detail.");

itemListStr=itemListStr.Replace("$XH", xu.ToString());

xu++;

}

pagehtml = pagehtml.Replace(trReplaceStr, itemListStr);

}

#endregion

returnStrHtml += pagehtml;

}

result.ResData= returnStrHtml;

result.Success = true;

}

else

{

result.ErrMsg = "请选择要打印的出库单";

}

}

catch (Exception ex)

{

Log4NetHelper.Error("打印出库单发生异常", ex);

}

return result;

}

/// <summary>

/// 用实体属性替换相关的字符串，主要应用于打印和导出

/// 方法将日期时间型属性值截取为日期型，格式“YYYY-MM-DD”，

/// 将布尔型属性值调整为是或否

/// </summary>

/// <param name="objInfo">实体对象</param>

/// <param name="strReplace">要替换的原字符串</param>

/// <param name="prefix">变量前缀</param>

/// <returns></returns>

public string ObjectReplaceString(object objInfo, string strReplace,string prefix= "")

{

string result = string.Empty;

string nowReplace = strReplace;

Type type = objInfo.GetType();//获得该类的Type

foreach (PropertyInfo pi in type.GetProperties())

{

string name = pi.Name;//获得属性的名字,后面就可以根据名字判断来进行些自己想要的操作

var value = pi.GetValue(objInfo, null);//用pi.GetValue获得值

var propertyType = value?.GetType() ?? typeof(object);//获得属性的类型

string replaceOld = "$" + prefix + name;

string newStrValue = "";

if (value != null)

{

//将日期时间型和布尔型数据进行处理，其他枚举数据提前处理

if (propertyType.Name == "DateTime")//如果是时间型取日期

{

newStrValue = value.ToString().Substring(0, 10);

}

else if (propertyType.Name == "Boolean")//布尔型转为是或否

{

bool blvalue;

if (bool.TryParse(value.ToString(), out blvalue))

{

newStrValue = "是";

}

else

{

newStrValue = "否";

}

}

else

{

newStrValue = value.ToString();

}

}

nowReplace = nowReplace.Replace(replaceOld, newStrValue);

}

result += nowReplace;

return result;

}

#### Vue前端实现

引用lodop打印组件

import { getLodop } from '@/utils/LodopFuncs'

var LODOP

在页面定义一个隐藏标签用于保存打印html内容

<div id="printHtmlDiv" style="display:none" v-html="printHtml" />

以下是批量打印和单页面打印代码

/\*\*

     \* 列表批量打印

     \*/

    handlesPrintOutStock: function() {

      if (this.currentSelected.length === 0) {

        this.$alert('请先选择要操作的数据', '提示')

        return false

      } else {

        var currentIds = ''

        this.currentSelected.forEach(element => {

          currentIds += element.Id + ','

        })

        const data = {

          idList: currentIds

        }

        PrintOutStockDetail(data).then(res => {

          if (res.Success) {

            if (res.ResData !== '') {

              this.printHtml = res.ResData

              LODOP = getLodop()

              LODOP.PRINT\_INIT('出库单打印')

              LODOP.NewPage()

              LODOP.SET\_PRINT\_PAGESIZE(1, 0, 0, 'A4')

              var \_self = this

              setTimeout(function() {

                \_self.createPrintPage()

              }, 600)

            }

          } else {

            this.$message({

              message: res.ErrMsg,

              type: 'error'

            })

          }

        })

      }

},

/\*\*

     \* 构建打印页面内容

     \*/

    createPrintPage: function() {

      this.currentSelected.forEach(element => {

        LODOP.NewPage()

        LODOP.SET\_PRINT\_MODE('FULL\_WIDTH\_FOR\_OVERFLOW', true)

        LODOP.SET\_PRINT\_MODE('FULL\_HEIGHT\_FOR\_OVERFLOW', true)

        LODOP.SET\_PRINT\_MODE('POS\_BASEON\_PAPER', true)

        LODOP.ADD\_PRINT\_HTM('10mm', '12.65mm', '100%', 2970, document.getElementById('printHtml' + element.Id).outerHTML)

        LODOP.ADD\_PRINT\_BARCODE(30, 625, 180, 60, 'EAN128A', element.Id)

      })

      // LODOP.SET\_PREVIEW\_WINDOW(0, 1, 0, 800, 600, '')// 打印前弹出选择打印机的对话框

      LODOP.SET\_PRINT\_MODE('AUTO\_CLOSE\_PREWINDOW', 1)// 打印后自动关闭预览窗口

      LODOP.PREVIEW()

      this.printHtml = ''

    },

    /\*\*

     \* 详情页打印

     \*/

    handlesPrintOutStockDetail: function() {

      if (this.currentId === '') {

        this.$alert('请先选择要操作的数据', '提示')

        return false

      } else {

        const data = {

          idList: this.currentId

        }

单页打印

        PrintOutStockDetail(data).then(res => {

          if (res.Success) {

            if (res.ResData !== '') {

              this.printHtml = res.ResData

              LODOP = getLodop()

              LODOP.PRINT\_INIT('出库单打印')

              // LODOP.NewPage()

              LODOP.SET\_PRINT\_PAGESIZE(1, 0, 0, 'A4')

              LODOP.SET\_PRINT\_MODE('FULL\_WIDTH\_FOR\_OVERFLOW', true)

              LODOP.SET\_PRINT\_MODE('FULL\_HEIGHT\_FOR\_OVERFLOW', true)

              LODOP.SET\_PRINT\_MODE('POS\_BASEON\_PAPER', true)

              LODOP.ADD\_PRINT\_HTM('10mm', '12.65mm', '100%', 2970, this.printHtml)

              LODOP.ADD\_PRINT\_BARCODE(30, 625, 180, 60, 'EAN128A', this.currentId)

              LODOP.SET\_PREVIEW\_WINDOW(0, 1, 0, 800, 600, '')// 打印前弹出选择打印机的对话框

              LODOP.SET\_PRINT\_MODE('AUTO\_CLOSE\_PREWINDOW', 1)// 打印后自动关闭预览窗口

              LODOP.PREVIEW()

            }

          } else {

            this.$message({

              message: res.ErrMsg,

              type: 'error'

            })

          }

        })

      }

    },