[开发文档](http://106.14.124.170/doc/)

YiShaAdmin

基于.NET Core Web开发管理系统

YiShaAdmin 借鉴了很多开源项目的优点，让你开发Web管理系统和移动端Api更简单，所以我也把她开源了。

# [项目介绍](http://106.14.124.170/doc/#/?id=%e9%a1%b9%e7%9b%ae%e4%bb%8b%e7%bb%8d)

YiShaAdmin，那就先从名字说起。

* YiSha，取自诗句“一沙一世界，一叶一菩提”中的“一沙”。“一沙一世界”从一粒沙子里看出一个世界，意思是用心去观察，去体悟， 哪怕是一粒沙子，都可以是一个世界，容纳万物。
* Admin，是代表管理系统的意思。

YiShaAdmin 是一个开源的后台管理系统快速开发框架，基于.NET Core Web技术（ASP.NET Core Web，ASP.NET Core Web Api），主要目的让开发者专注业务， 降低技术难度，从而节省人力成本，缩短项目周期，提高软件安全质量。

[**在线体验**](http://106.14.124.170/doc/#/?id=%e5%9c%a8%e7%ba%bf%e4%bd%93%e9%aa%8c)

1. 项目地址：[**http://106.14.124.170/admin**](http://106.14.124.170/admin)
2. 账号密码：admin 123456

[**系统要求**](http://106.14.124.170/doc/#/?id=%e7%b3%bb%e7%bb%9f%e8%a6%81%e6%b1%82)

* .NET Core >= 2.2
* Microsoft SQL Server 2012+
* MySql 5.7+
* Oracle DB 11.2+

[**联系我们**](http://106.14.124.170/doc/#/?id=%e8%81%94%e7%b3%bb%e6%88%91%e4%bb%ac)

* YiShaAdmin技术交流群：~~满 428703155~~ [**851661600**](https://jq.qq.com/?_wv=1027&k=MQ6OdF8v)

# [快速了解](http://106.14.124.170/doc/#/quickstart)

[**简介**](http://106.14.124.170/doc/#/quickstart?id=%e7%ae%80%e4%bb%8b)

YiShaAdmin 是一款基于.NET Core Web + Bootstrap的企业级快速后台开发框架。内置模块如：用户管理、部门管理、菜单管理、角色权限设置、日志管理、新闻管理等。 在线定时任务配置；支持Sql Server、MySql和Oracle数据库。

[**主要特性**](http://106.14.124.170/doc/#/quickstart?id=%e4%b8%bb%e8%a6%81%e7%89%b9%e6%80%a7)

* 响应式布局，支持电脑端和移动端
* 强大的一键生成功能（包括视图、控制器、业务类、实体类、服务类、菜单）
* 支持Sql Server、MySql和Oracle数据库
* 基于角色的权限控制（Role-Based Access Control），可控制到按钮
* 对常用JS插件进行二次封装，使JS代码变得简洁，更加容易维护
* 完善的日志记录体系

[**技术框架**](http://106.14.124.170/doc/#/quickstart?id=%e6%8a%80%e6%9c%af%e6%a1%86%e6%9e%b6)

* 前端：Bootstrap
* 核心框架：.NET Core MVC
* 缓存层：Memory、Redis
* 持久层框架：Entity Framework Core
* 数据库支持：SqlServer、MySql、Oracle
* 定时任务：Quartz.Net
* API文档显示工具：Swagger UI

[**内置功能**](http://106.14.124.170/doc/#/quickstart?id=%e5%86%85%e7%bd%ae%e5%8a%9f%e8%83%bd)

* 员工管理：员工是系统操作者，该功能主要完成系统用户配置
* 部门管理：配置系统组织机构（公司、部门、小组）
* 职位管理：配置系统用户所担任职务
* 文章中心：管理新闻及公司案例
* 角色管理：角色菜单权限分配，角色对应权限，员工属于某个角色
* 菜单管理：配置系统菜单，操作权限，按钮权限标识
* 通用字典：系统里动态改变的数据，像文章类型，用字典进行维护，不变的数据，可以用枚举
* 行政区划：全国的省市县三级数据，展示树形数据的使用
* 系统日志（登录日志、操作日志、Api日志）：查看系统的登录、操作、Api调用日志
* 定时任务：在线（添加、修改、删除）任务调度，包含执行结果日志
* 系统Api：Api文档与测试工具
* 数据表管理：查看当前数据库所有表、字段及记录总数
* 代码生成：快速生成对单表的增删改查
* 服务器信息：查看服务器的基本信息

# [环境部署](http://106.14.124.170/doc/#/environment-setting)

[**准备工作**](http://106.14.124.170/doc/#/environment-setting?id=%e5%87%86%e5%a4%87%e5%b7%a5%e4%bd%9c)

1. Visual Studio 2017 及以上或者 Visual Studio Code
2. .Net Core 2.2 SDK或.Net Core 3.1 SDK(根据YiShaAdmin版本选择)
3. 数据库任选以下中的一种：

1). MySql >= 5.7 [**more detail**](https://dev.mysql.com/doc/connector-net/en/connector-net-entityframework-core.html)  
2). SqlServer >= Sql Server 2012 [**more detail**](https://docs.microsoft.com/en-us/ef/core/providers/sql-server/)  
3). Oracle >= Oracle DB 11.2 [**more detail**](https://docs.microsoft.com/en-us/ef/core/providers/)

[**运行系统**](http://106.14.124.170/doc/#/environment-setting?id=%e8%bf%90%e8%a1%8c%e7%b3%bb%e7%bb%9f)

1. 下载项目 ([**https://github.com/liukuo362573/YiShaAdmin**](https://github.com/liukuo362573/YiShaAdmin)) 解压到工作目录
2. 用Visual Studio 2017或者Visual Studio Code打开项目的解决方案文件YiShaAdmin.sln （如果你使用的是.Net Core 3.1版本，需要用Visual Studio 2019）
3. 创建YiShaAdmin数据库，执行根目录下Document/DatabaseScript对应数据库的脚本（当前仅有MySql和SqlServer数据库的脚本），每个库有2个脚本，一个是创建表，一个是初始化数据。
4. 右击YiSha.Admin.Web项目，设为启动项目，直接按F5键运行
5. 打开浏览器，输入: [**http://localhost:5000**](http://localhost:5000/) （默认账户 admin/123456） 若能正确显示登录页面，并能成功登录，菜单及页面显示正常，则表明开发环境搭建成功

建议使用Git克隆，因为克隆的方式可以和YiShaAdmin随时保持更新同步。使用Git命令克隆  
git clone ([**https://github.com/liukuo362573/YiShaAdmin.git**](https://github.com/liukuo362573/YiShaAdmin.git))

[**必要配置**](http://106.14.124.170/doc/#/environment-setting?id=%e5%bf%85%e8%a6%81%e9%85%8d%e7%bd%ae)

修改数据库连接  
编辑YiSha.Admin.Web和YiSha.Admin.WebApi目录下的appsetting.json文件，如果你的数据库是MySql，就按照下面MySql的配置，以此类推。

1. MySql 数据库
2. {
3. "DBProvider": "MySql",
4. "DBConnectionString": "server=localhost;database=YiShaAdmin;user=root;password=123456;port=3306;"

}

server: 数据库服务器地址  
database: 数据库名称  
user: 数据库连接用户名  
password: 数据库连接密码

1. SqlServer 数据库
2. {
3. "DBProvider": "SqlServer",
4. "DBConnectionString": "Server=localhost;Database=YiShaAdmin;User Id=sa;Password=123456;"

}

如果以上的SqlServer连接字符串不行，参考下这个"Server=localhost\\SQL2012;Database=YiShaAdmin;User Id=sa;Password=123456;"

如果带端口，可能是这样Server=localhost,1433;Database=YiShaAdmin;User Id=sa;Password=123456;

1. Oracle 数据库 待补充

[**部署系统**](http://106.14.124.170/doc/#/environment-setting?id=%e9%83%a8%e7%bd%b2%e7%b3%bb%e7%bb%9f)

项目的根目录下有个Publish文件夹

运行publish -debug.bat会在当前目录生成 YiShaAdmin 和 YiShaAdminApi 两个文件夹，直接可以把这两个目录发布到线上服务器。

publish -release.bat比publish -debug.bat多了release选项，还有release模式下，会删除一些本地的配置文件， 防止发布到线上的时候，把线上服务器的配置文件覆盖掉。

当然第一次发布的时候，应该用publish -debug.bat，因为debug模式下才有所有的配置文件，debug模式发布之后，可以再用release模式覆盖一下。

[**常见问题**](http://106.14.124.170/doc/#/environment-setting?id=%e5%b8%b8%e8%a7%81%e9%97%ae%e9%a2%98)

1. 如果用nginx做反向代理，可以参考根目录下Document/nginx.conf的配置

如遇到问题到[**Issues**](https://github.com/liukuo362573/YiShaAdmin/issues)反馈

# [项目介绍](http://106.14.124.170/doc/#/introduction)

[**项目结构**](http://106.14.124.170/doc/#/introduction?id=%e9%a1%b9%e7%9b%ae%e7%bb%93%e6%9e%84)

YiShaAdmin

├─ Framework // 基础设施层

│ ├─ Cache // 数据缓存接口

| ├─ YiSha.Cache.Factory // 缓存工厂

| ├─ YiSha.Cache.Interface // 缓存抽象接口

| ├─ YiSha.Cache.MemoryCache // 缓存MemoryCache实现

| ├─ YiSha.Cache.RedisCache // 缓存Redis实现

| ├─ YiSha.Util // 框架公共方法

| ├─ YiSha.IdGenerator // Id生成器(使用Snowflake)

| ├─ YiSha.CodeGenerator // 代码生成器

├─ DataAccess // 数据库核心层

| ├─ YiSha.Data // 底层抽象接口

| ├─ YiSha.Data.EF // 数据库层实现

| ├─ YiSha.Data.Repository // 数据库层接口

├─ Business // 业务逻辑层

| ├─ YiSha.Business // 业务层方法

| ├─ YiSha.Business.AutoJob // 业务层定时任务

| ├─ YiSha.Business.Cache // 业务层缓存

| ├─ YiSha.Business.Service // 业务层服务

├─ Entity // 业务实体层

| ├─ YiSha.Entity // 数据库实体类

| ├─ YiSha.Enum // 业务枚举

| ├─ YiSha.Model // 业务层DTO对象

├─ Web // 前端

| ├─ YiSha.Admin.Web // 系统UI界面

| ├─ YiSha.Admin.WebApi // 系统Api接口

| ├─ YiSha.Web.Code // 管理用户登录状态

[**配置文件**](http://106.14.124.170/doc/#/introduction?id=%e9%85%8d%e7%bd%ae%e6%96%87%e4%bb%b6)

1. YiSha.Admin.Web 项目配置

appsetting.json 配置

"SystemConfig":

{

"Demo": false, // 是否是演示模式

"LoginMultiple": true, // 是否允许一个账户在多处登录

"LoginProvider": "Cookie", // 登录信息保存方式 Cookie Session WebApi

"SnowFlakeWorkerId": 1, // SnowFlake 节点序号

"ApiSite": "http://localhost:5001", // Api地址，例如可以上传文件到Api

"VirtualDirectory": "/admin", // 虚拟目录

"DBProvider": "MySql", // 数据库类型：SqlServer、MySql、Oracle

"DBConnectionString": "server=localhost;database=YiShaAdmin;user=root;password=123456;",

"DBCommandTimeout": 180, // 数据库超时时间，单位秒

"DBBackup": "", // 数据库备份路径

"CacheProvider": "Memory", // 缓存使用方式 Memory Redis

"RedisConnectionString": "127.0.0.1:6379"

}

此配置会在Startup.Configure(IApplicationBuilder app, IHostingEnvironment env)方法中，把配置映射为GlobalContext.SystemConfig， 这样在系统中需要使用配置的地方，就可以直接用GlobalContext.SystemConfig对象。

bundleconfig.json 配置

{

"outputFileName": "wwwroot/lib/bootstrap/3.3.7/js/bootstrap.min.js",

"inputFiles": [

"wwwroot/lib/bootstrap/3.3.7/js/bootstrap.js",

"wwwroot/lib/bootstrap/3.3.7/js/bootstrap.dropdown.js"

],

"minify": {

"enabled": true,

"renameLocals": true

}

}

系统使用BuildBundlerMinifier库，对js代码进行压缩和合并，在每次代码编译成功之后，就会应用bundleconfig.json文件里面的配置， outputFileName为inputFiles数组生成的一个新文件。

1. YiSha.Admin.WebApi 配置文件

appsetting.json 配置

"SystemConfig":

{

"LoginProvider": "WebApi", // 登录信息保存方式 Cookie Session WebApi

"SnowFlakeWorkerId": 2, // SnowFlake 节点序号

"AllowCorsSite": "http://localhost:5000", // 允许的其他站点访问Api

"DBProvider": "MySql",

"DBConnectionString": "server=localhost;database=YiShaAdmin;user=root;password=123456;",

"DBCommandTimeout": 180, // 数据库超时时间，单位秒

"DBBackup": "" // 数据库备份路径

}

# [前端手册](http://106.14.124.170/doc/#/frontend)

[**前端组件**](http://106.14.124.170/doc/#/frontend?id=%e5%89%8d%e7%ab%af%e7%bb%84%e4%bb%b6)

YiShaAdmin封装了一些常用的浏览器JS组件方法。

| **名称** | **使用** | **介绍** |
| --- | --- | --- |
| [**表格**](http://106.14.124.170/doc/#/frontend?id=%e8%a1%a8%e6%a0%bc) | $('#id').ysTable(options) | bootstrap table表格封装处理 |
| [**表格树**](http://106.14.124.170/doc/#/frontend?id=%e8%a1%a8%e6%a0%bc%e6%a0%91) | $('#id').ysTreeTable(options) | bootstrap table tree表格树封装处理 |
| [**树形选择**](http://106.14.124.170/doc/#/frontend?id=%e6%a0%91%e5%bd%a2%e9%80%89%e6%8b%a9) | $('#id').ysTree(options) | ztree封装处理 |

* |- |-

[**单选框**](http://106.14.124.170/doc/#/frontend?id=%e5%8d%95%e9%80%89%e6%a1%86) | $('#id').ysRadioBox(options) | 基础控件封装 | [**复选框**](http://106.14.124.170/doc/#/frontend?id=%e5%a4%8d%e9%80%89%e6%a1%86) | $('#id').ysCheckBox(options) | 基础控件封装 | [**下拉框**](http://106.14.124.170/doc/#/frontend?id=%e4%b8%8b%e6%8b%89%e6%a1%86) | $('#id').ysComboBox(options) | select2封装处理 | [**树形下拉框**](http://106.14.124.170/doc/#/frontend?id=%e6%a0%91%e5%bd%a2%e4%b8%8b%e6%8b%89%e6%a1%86) | $('#id').ysComboBoxTree(options) | ztree封装处理 |

* |- |-

[**弹出层**](http://106.14.124.170/doc/#/frontend?id=%e5%bc%b9%e5%87%ba%e5%b1%82) | ys.methodName | layer封装处理 | [**校验**](http://106.14.124.170/doc/#/frontend?id=%e6%a0%a1%e9%aa%8c) | $('#id').validate(options) | jquery validation 封装处理 | [**通用方法**](http://106.14.124.170/doc/#/frontend?id=%e9%80%9a%e7%94%a8%e6%96%b9%e6%b3%95) | ys.methodName | 通用方法封装处理 |

[**表格**](http://106.14.124.170/doc/#/frontend?id=%e8%a1%a8%e6%a0%bc)

表格组件基于bootstrap table组件进行封装，轻松实现数据表格。

表格初始化$('#id').ysTable(options)

表格参数(Table options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String |  | 请求后台的Url |
| method | String | GET | 请求方式 |
| toolbar | String | #toolbar | 工具按钮用哪个容器 |
| striped | Boolean | true | 是否显示行间隔色 |
| cache | Boolean | false | 是否使用缓存，默认为false，所以一般情况下需要设置一下这个属性 |
| pagination | Boolean | true | 是否显示分页 |
| columns | Array | [] | 默认空数组，在JS里面定义参考列的各项(Column options) |
| queryParams | Object | {} | 附加在Url后面的QueryString参数 |
| sortable | Boolean | true | 是否启用排序 |
| sortStable | Boolean | true | 设置为 true 将获得稳定的排序 |
| sortName | String | id | 排序列名称 |
| sortOrder | String | desc | 排序方式 |
| sidePagination | String | server | 分页方式：client客户端分页，server服务端分页 |
| pageNumber | Number | 1 | 初始化加载第一页，默认第一页,并记录 |
| pageSize | Number | 10 | 每页的记录行数 |
| pageList | String | 10, 25, 50, 100 | 可供选择的每页的行数 |
| uniqueId | String | Id | 每一行的唯一标识，一般为主键列 |
| totalField | String | TotalCount | json中的总记录数属性 |
| dataField | String | Result | json中的数据属性 |
| search | Boolean | false | 是否显示表格搜索 |
| clickToSelect | Boolean | true | 是否启用点击选中行 |
| showColumns | Boolean | true | 是否显示所有的列 |
| showRefresh | Boolean | true | 是否显示刷新按钮 |
| showToggle | Boolean | true | 是否显示详细视图和列表视图的切换按钮 |
| height | Number | undefined | 行高，如果没有设置height属性，表格自动根据记录条数觉得表格高度 |
| minimumCountColumns | Number | 2 | 最少允许的列数 |
| cardView | Boolean | false | 是否显示详细视图 |
| detailView | Boolean | false | 是否显示父子表 |
| onLoadSuccess | Number | Null | 当所有数据被加载时触发处理函数 |
| onLoadError | Number | Null | 数据加载失败时触发 |

表格列的参数(Column options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| radio | Boolean | false | 默认false不显示radio（单选按钮），设为true则显示，radio宽度是固定的 |
| checkbox | Boolean | false | 默认false不显示checkbox（复选框），设为true则显示，checkbox的每列宽度已固定 |
| field | String | Null | 是每列的字段名，不是表头所显示的名字，通过这个字段名可以给其赋值，相当于key，表内唯一 |
| title | String | Null | 这个是表头所显示的名字，不唯一，如果你喜欢，可以把所有表头都设为相同的名字 |
| titleTooltip | String | true | 当悬浮在某控件上，出现提示 - 参考 Bootstrap 提示工具（Tooltip）插件 |
| class | boolean | false | 表格列样式 |
| rowspan | Number | true | 每格所占的行数 |
| colspan | Number | true | 每格所占的列数 |
| align | String | true | 每格内数据的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| halign | String | true | table header（表头）的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| falign | String | true | table footer（表脚，的对齐方式，有：left（靠左）、right（靠右）、center（居中） |
| valign | String | true | 每格数据的对齐方式，有：top（靠上）、middle（居中）、bottom（靠下） |
| width | Number | Null | 每列的宽度。如果没有自定义宽度自适应 |
| sortable | Boolean | false | 默认false就默认显示，设为true则会被排序 |
| order | String | asc | 默认的排序方式为"asc（升序）"，也可以设为"desc（降序）"。 |
| visible | Boolean | true | 默认为true显示该列，设为false则隐藏该列 |
| cardVisible | Boolean | true | 默认为true显示该列，设为false则隐藏。 |
| switchable | Boolean | true | 默认为true显示该列，设为false则禁用列项目的选项卡。 |
| clickToSelect | Boolean | true | 默认true不响应，设为false则当点击此行的某处时，不会自动选中此行的checkbox（复选框）或radiobox（单选按钮） |
| formatter | Function | Null | 某格的数据转换函数，需要三个参数： -value： field（字段名） -row：行的数据 -index：行的（索引）index |
| footerFormatter | Function | Null | 某格的数据转换函数，需要一个参数： -data： 所有行数据的数组 函数需要返回（return）footer某格内所要显示的字符串的格式 |
| events | Object | Null | 当某格使用formatter函数时，事件监听会响应，需要四个参数： -event：-value：字段名 -row：行数据 -index：此行的index |
| sorter | Function | Null | 自定义的排序函数，实现本地排序，需要两个参数： - a：第一个字段名 - b：第二个字段名 |
| sortName | String | Null | 排序列名称 |
| cellStyle | Function | Null | 对某列中显示样式改变 |
| searchable | Boolean | true | 默认true，表示此列数据可被查询 |
| searchFormatter | Boolean | true | 默认true，可使用格式化的数据查询 |
| escape | Boolean | false | 是否转义HTML字符串 |

[**表格树**](http://106.14.124.170/doc/#/frontend?id=%e8%a1%a8%e6%a0%bc%e6%a0%91)

表格树组件基于bootstrap table tree组件进行封装，快速实现树形表格。

树形表格初始化$('#id').ysTreeTable(options)

树形表格参数(Table Tree options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String |  | 请求后台的Url |
| method | String | GET | 请求方式 |
| toolbar | String | #toolbar | 工具按钮用哪个容器 |
| code | String | Id | 选取记录返回的值,用于设置父子关系 |
| parentCode | String | ParentId | 用于设置父子关系 |
| rootIdValue | String | Null | 设置根节点id值----可指定根节点，默认为null,"",0,"0" |
| data | Object | Null | 构造table的数据集合 |
| columns | Array | [] | 参考上面的表格列的参数 |
| ajaxParams | Object | {} | 请求数据的ajax的data属性 |
| expandColumn | Object | Null | 在哪一列上面显示展开按钮 |
| expandAll | Boolean | true | 是否全部展开 |
| expandFirst | Boolean | true | 是否默认第一级展开--expandAll为false时生效 |
| striped | Boolean | false | 是否各行渐变色 |
| bordered | Boolean | true | 是否显示边框 |
| hover | Boolean | true | 是否鼠标悬停 |
| condensed | Boolean | false | 是否紧缩表格 |
| height | Number | 0 | 表格高度 |
| showTitle | Boolean | true | 是否采用title属性显示字段内容（被formatter格式化的字段不会显示） |
| showColumns | Boolean | true | 是否显示内容列下拉框 |
| showRefresh | Boolean | true | 是否显示刷新按钮 |
| expanderExpandedClass | String | glyphicon glyphicon-chevron-down | 展开的按钮的图标 |
| expanderCollapsedClass | String | glyphicon glyphicon-chevron-right | 缩起的按钮的图标 |

常用方法

| **方法** | **参数** | **描述** |
| --- | --- | --- |
| search | target,param | 刷新数据 |
| expandRowById | target,id | 展开指定的行 |

[**树形选择**](http://106.14.124.170/doc/#/frontend?id=%e6%a0%91%e5%bd%a2%e9%80%89%e6%8b%a9)

树形选择组件基于ztree组件进行封装。

树形选择初始化$('#id').ysTree(options)

常用参数(Tree options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String |  | 请求后台的Url |
| async | Boolean | false | 是否是异步请求 |
| maxHeight | String | 300px | 容器的最大高度，内容超出这个高度，容器会显示垂直滚动条 |
| expandLevel | Number | 0 | 展开的级数，0表示展开一级，1表示展开二级，以此类推 |
| check | Object |  | enable属性表示是否显示复选框 |
| view | Object |  | selectedMulti属性表示你是否可以多选 |

常用方法

| **方法** | **参数** | **描述** |
| --- | --- | --- |
| getCheckedNodes | target,column | 获取组件的值（第一个参数为组件，第二个为数据列，缺省是id） 使用示例 $('#id').ysTree("getCheckedNodes") |
| setCheckedNodes | target,ids | 设置组件的值（第一个参数为组件，第二个为值） 使用示例 $('#id').ysTree('setCheckedNodes', value) |

[**单选框**](http://106.14.124.170/doc/#/frontend?id=%e5%8d%95%e9%80%89%e6%a1%86)

单选框初始化 $('#id').ysRadioBox(options)

常用参数 (RadioBox options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String |  | 数据源url，支持ajax获取数据源 |
| key | String | Key | 数据源里面有很多属性的时候，key表示对应选项的值 |
| value | String | Value | 数据源里面有很多属性的时候，value表示对应的选项，界面上看到的数据 |
| data | Object |  | 当url为空的时候，可以设置此属性作为组件的数据源 |
| dataName | String | Result | 当数据源中的某一个属性作为实际要绑定的组件的数据源，dataName就是那个属性的值 |

常用方法

| **方法** | **参数** | **描述** |
| --- | --- | --- |
| getValue | target | 获取组件的值（第一个参数为组件） 使用示例 $('#id').ysRadioBox('getValue') |
| setValue | target,value | 设置组件的值（第一个参数为组件，第二个为值） 使用示例 $('#id').ysRadioBox('setValue', value) |

[**复选框**](http://106.14.124.170/doc/#/frontend?id=%e5%a4%8d%e9%80%89%e6%a1%86)

复选框初始化 $('#id').ysCheckBox(options)

常用参数 (CheckBox options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String | false | 数据源url，支持ajax获取数据源 |
| key | String | Key | 数据源里面有很多属性的时候，key表示对应选项的值 |
| value | String | Value | 数据源里面有很多属性的时候，value表示对应的选项，界面上看到的数据 |
| data | Object |  | 当url为空的时候，可以设置此属性作为组件的数据源 |
| dataName | String | Result | 当数据源中的某一个属性作为实际要绑定的组件的数据源，dataName就是那个属性的值 |

常用方法

| **方法** | **参数** | **描述** |
| --- | --- | --- |
| getValue | target | 获取组件的值（第一个参数为组件） 使用示例 $('#id').ysCheckBox('getValue') |
| setValue | target,value | 设置组件的值（第一个参数为组件，第二个为值） 使用示例 $('#id').ysCheckBox('setValue', value) |

[**下拉框**](http://106.14.124.170/doc/#/frontend?id=%e4%b8%8b%e6%8b%89%e6%a1%86)

下拉框组件基于select2组件进行封装。

下拉框初始化 $('#id').ysComboBox(options)

常用参数 (ComboBox options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String |  | 数据源url，支持ajax获取数据源 |
| key | String | Key | 数据源里面有很多属性的时候，key表示对应选项的值 |
| value | String | Value | 数据源里面有很多属性的时候，value表示对应的选项，界面上看到的数据 |
| maxHeight | String | 160px | 容器的最大高度，内容超出这个高度，容器会显示垂直滚动条 |
| class | String |  | 下拉框的样式 |
| multiple | Boolean | false | 组件是否可以多选，true 可以多选，false 不能多选 |
| data | Object |  | 当url为空的时候，可以设置此属性作为组件的数据源 |
| dataName | String | Result | 当数据源中的某一个属性作为实际要绑定的组件的数据源，dataName就是那个属性的值 |
| onChange | Function |  | 当组件选项改变的时候触发 |

常用方法

| **方法** | **参数** | **描述** |
| --- | --- | --- |
| getValue | target | 获取组件的值（第一个参数为组件） 使用示例 $('#id').ysComboBox('getValue') |
| setValue | target,value | 设置组件的值（第一个参数为组件，第二个为值） 使用示例 $('#id').ysComboBox('setValue', value) |

[**树形下拉框**](http://106.14.124.170/doc/#/frontend?id=%e6%a0%91%e5%bd%a2%e4%b8%8b%e6%8b%89%e6%a1%86)

树形下拉框组件基于ztree组件进行封装。

树形下拉框初始化 $('#id').ysComboBoxTree(options)

常用参数 (ComboBoxTree options)

| **参数** | **类型** | **默认值** | **描述** |
| --- | --- | --- | --- |
| url | String |  | 数据源url，支持ajax获取数据源 |
| async | Boolean | false | 是否是异步请求 |
| maxHeight | String | 200px | 容器的最大高度，内容超出这个高度，容器会显示垂直滚动条 |
| expandLevel | Number | 0 | 展开的级数，0表示展开一级，1表示展开二级，以此类推 |
| check | Object |  | enable属性表示是否显示复选框 |
| view | Object |  | selectedMulti属性表示你是否可以多选 |
| callback | Object |  | 事件回调，例如 callback: { customOnClick: function (event, treeId, treeNode) { alert(treeId); } } |

常用方法

| **方法** | **参数** | **描述** |
| --- | --- | --- |
| getValue | target | 获取组件的值（第一个参数为组件） 使用示例 $('#id').ysComboBoxTree('getValue') |
| setValue | target,value | 设置组件的值（第一个参数为组件，第二个为值） 使用示例 $('#id').ysComboBoxTree('setValue', value) |

[**弹出层**](http://106.14.124.170/doc/#/frontend?id=%e5%bc%b9%e5%87%ba%e5%b1%82)

弹出层组件基于 layer 组件进行封装，提供了弹出、消息、提示、确认、遮罩处理等功能。

* 提供成功、错误、警告弹出层
* ys.msgSuccess('成功')
* ys.msgError('错误或失败')

ys.msgWarning('警告')

* 提供成功、错误、警告弹出层，需要点击关闭
* ys.alertSuccess('成功')
* ys.alertError('错误或失败')

ys.alertWarning('警告')

* 提供页面弹出层
* ys.openDialog({title:'页面标题',content:'页面url',width:'768px',height:'600px',callback}) // 弹出层打开url
* ys.openDialogContent({content:html,width:'768px',height:'600px',callback}) // 弹出层显示html，比如一张图片

ys.closeDialog()

* 提供遮罩层
* ys.showLoading('显示内容')

ys.closeLoading()

* 修改页面数据示例，Index页面调用showSaveForm方法弹出修改Form页面，Form页面调用saveForm方法保存数据到服务端

1.Index 页面

function showSaveForm(bAdd) {

var id = 0;

if (!bAdd) {

var selectedRow = $("#gridTable").bootstrapTable("getSelections");

if (!ys.checkRowEdit(selectedRow)) {

return;

}

else {

id = selectedRow[0].Id;

}

}

ys.openDialog({

title: id > 0 ? "编辑文章" : "添加文章",

content: '@Url.Content("~/OrganizationManage/News/NewsForm")' + '?id=' + id,

width: "768px",

height: "600px",

callback: function (index, layero) {

var iframeWin = window[layero.find('iframe')[0]['name']];

iframeWin.saveForm(index);

}

});

}

2.Form 页面

function saveForm(index) {

if ($("#form").validate().form()) {

var postData = $("#form").getWeb'#id's({ Id: id });

ys.ajax({

url: '@Url.Content("~/OrganizationManage/News/SaveFormJson")',

type: "post",

data: postData,

success: function (obj) {

if (obj.Tag == 1) {

ys.msgSuccess(obj.Message);

parent.refreshGrid();

parent.layer.close(index);

}

else {

ys.msgError(obj.Message);

}

}

});

}

}

[**校验**](http://106.14.124.170/doc/#/frontend?id=%e6%a0%a1%e9%aa%8c)

校验组件沿用jquery validation组件验证逻辑。

* 校验初始化
* $("#form").validate({
* rules: {
* userName: { required: true },
* password: {
* required: true,
* minlength: 6,
* maxlength: 20
* },
* mobile: { isPhone: true},
* email: { email: true}
* }

});

* 保存时校验

$("#form").validate().form()

[**通用方法**](http://106.14.124.170/doc/#/frontend?id=%e9%80%9a%e7%94%a8%e6%96%b9%e6%b3%95)

通用方法放在yisha.js文件中，以ys.方法名访问，不用记住，使用的时候，去文件里面通过函数名查找就可以了。

| **名称** | **介绍** |
| --- | --- |
| [**字符串处理**](http://106.14.124.170/doc/#/frontend?id=string_common) | 字符串封装处理 |
| [**Url请求封装处理**](http://106.14.124.170/doc/#/frontend?id=request_common) | Url请求封装处理 |

**字符串处理**

| **方法名** | **描述** |
| --- | --- |
| isNullOrEmpty | 判断字符串是否是Null或者空 |
| getJson | Html.Raw()方法会提示语法错误，所以用这个函数包装一下 |
| getGuid | 客户端获取类似的guid |
| getValueByKey | 获取json里面key的值 |
| formatDate | 格式为 yyyy-MM-dd HH:mm:ss |
| trimStart | 去掉字符串开始的部分 |
| trimEnd | 去掉字符串结束的部分 |
| toString | JS的toString()，如果是Null，返回空 |

**Url请求封装处理**

| **方法名** | **描述** |
| --- | --- |
| ajax | 对jquery的ajax进行的封装 |
| ajaxUploadFile | 上传文件或图片 |
| exportExcel | 导出excel |
| request | 获取url里面的某个参数值 |
| getHttpFileName | 获取url最后的文件名 |
| getFileNameWithoutExtension | 获取文件名，不带扩展名 |
| changeURLParam | 修改url里面的某个参数 |

# [后端手册](http://106.14.124.170/doc/#/backend)

[**数据格式**](http://106.14.124.170/doc/#/backend?id=%e6%95%b0%e6%8d%ae%e6%a0%bc%e5%bc%8f)

为了统一，后端响应的数据类型都是TData或者泛型TData<T>类型的json数据，T为泛型，可以是List类型或是Object类型的数据。

* 如果是简单数据的状态返回，可以使用TData类型，有Tag、Message、Description这3个属性。
* public class TData
* {
* /// <summary>
* /// 操作结果，Tag为1代表成功，0代表失败，其他的验证返回结果，可根据需要设置
* /// </summary>
* public int Tag { get; set; }
* /// <summary>
* /// 提示信息或异常信息
* /// </summary>
* public string Message { get; set; }
* /// <summary>
* /// 扩展Message
* /// </summary>
* public string Description { get; set; }

}

* 如果是复杂数据的状态返回，可以使用泛型类型的TData<T>，它继承自TData。 当需要返回一条记录的时候，TotalCount属性不使用，当需要返回分页数据的时候，TotalCount属性为总记录数，Result属性存放需要返回的数据。
* public class TData<T> : TData
* {
* /// <summary>
* /// 列表的记录数
* /// </summary>
* public int TotalCount { get; set; }
* /// <summary>
* /// 数据
* /// </summary>
* public T Result { get; set; }

}

注意看类属性上面的注释，对各个属性的作用都有解释，比如属性Tag成功为1，失败为0（int型默认就是0，失败时可以不用赋值，成功时才要赋值）， 这是一种默认的约定，实际开发可以按照这种约定，方便统一控制。

[**分页实现**](http://106.14.124.170/doc/#/backend?id=%e5%88%86%e9%a1%b5%e5%ae%9e%e7%8e%b0)

1. 前端通过组件ysTable来显示表格

$(function () {

initGrid();

});

function initGrid() {

var queryUrl = '@Url.Content("~/OrganizationManage/Position/GetPageListJson")';

$('#gridTable').ysTable({

url: queryUrl,

sortName: 'PositionSort',

sortOrder: 'Asc',

columns: [

{ checkbox: true, visible: true },

{ field: 'Id', title: 'Id', visible: false },

{ field: 'PositionName', title: '职位名称', width: "15%", sortable: true },

{ field: 'PositionSort', title: '显示顺序', width: "15%", sortable: true },

{

field: 'BaseModifyTime', title: '创建时间', formatter: function (value, row, index) {

return ys.formatDate(value, "yyyy-MM-dd HH:mm:ss");

}

}

],

queryParams: function (params) {

// 获取分页参数

var pagination = $('#gridTable').ysTable('getPagination', params);

// 获取界面搜索条件

var queryString = $("#searchDiv").getWebControls(pagination);

return queryString;

}

});

}

searchDiv 是一个大的div，里面是搜索条件控件。

1. 后端Controller

[HttpGet]

[AuthorizeFilter("organization:position:search")]

public async Task<IActionResult> GetPageListJson(PositionListParam param, Pagination pagination)

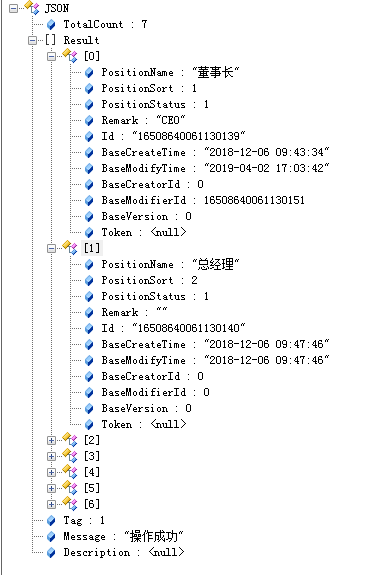
{

TData<List<PositionEntity>> obj = await sysPositionBLL.GetPageList(param, pagination);

return Json(obj);

}

* AuthorizeFilter继承自ActionFilterAttribute，作用是对Action进行权限校验，organization:position:search为GetPageListJson这个Action对应的权限标识。
* GetPageListJson方法的第一个参数PositionListParam会自动映射传入的搜索条件参数，第二参数PositionListParam会自动映射传入的分页条件。
* 响应的数据为Json格式：



* 搜索条件映射成Sql语句有2种写法，第一种是把输入条件参数转成Expression表达式树，适合简单的Sql查询，第二种是拼接原生Sql，适合复杂的Sql查询。

1 ) 第一种转成Expression表达式树写法

private Expression<Func<PositionEntity, bool>> ListFilter(PositionListParam param)

{

var expression = LinqExtensions.True<PositionEntity>();

if (param != null)

{

if (!string.IsNullOrEmpty(param.PositionName))

{

expression = expression.And(t => t.PositionName.Contains(param.PositionName));

}

if (!string.IsNullOrEmpty(param.PositionIds))

{

long[] positionIdArr = CommonHelper.SplitToArray<long>(param.PositionIds, ',');

expression = expression.And(t => positionIdArr.Contains(t.Id.Value));

}

}

return expression;

}

调用方法如下：

public async Task<List<PositionEntity>> GetPageList(PositionListParam param, Pagination pagination)

{

var expression = ListFilter(param);

var list = await this.BaseRepository().FindList(expression, pagination);

return list.ToList();

}

2 ) 第二种拼接原生Sql写法

private List<DbParameter> ListFilter(LogApiListParam param, StringBuilder strSql)

{

strSql.Append(@"SELECT a.id as Id,

a.base\_modify\_time as BaseModifyTime,

a.base\_modifier\_id as BaseModifierId,

a.log\_status as LogStatus,

FROM sys\_log\_api a

LEFT JOIN sys\_user b ON a.base\_modifier\_id = b.id

LEFT JOIN sys\_department c ON b.department\_id = c.id

WHERE 1 = 1");

var parameter = new List<DbParameter>();

if (param != null)

{

if (!string.IsNullOrEmpty(param.UserName))

{

strSql.Append(" AND b.user\_name like @UserName");

parameter.Add(DbParameterExtension.CreateDbParameter("@UserName", '%' + param.UserName + '%'));

}

}

return parameter;

}

调用方法如下：

public async Task<List<LogApiEntity>> GetPageList(LogApiListParam param, Pagination pagination)

{

var strSql = new StringBuilder();

List<DbParameter> filter = ListFilter(param, strSql);

var list = await this.BaseRepository().FindList<LogApiEntity>(strSql.ToString(), filter.ToArray(), pagination);

return list.ToList();

}

[**Excel导入导出**](http://106.14.124.170/doc/#/backend?id=excel%e5%af%bc%e5%85%a5%e5%af%bc%e5%87%ba)

Excel导入导出使用开源项目NPOI进行处理，以下把用户导入和用户导出作为示例。

Excel导入

Excel导入分两步，第一步把需要导入的文件通过FileController上传到后台，第二步把第一步返回的文件相对地址传给导入处理方法。

1. 前端写法

<input id="importFile" type="file">

// 把需要导入的文件通过FileController上传到后台

var filePath = undefined;

$("#importFile").fileinput({

language: 'zh',

'uploadUrl': '@Url.Content("~/File/UploadFile")' + '?fileModule=@UploadFileType.Import.ParseToInt()',

showPreview: false,

allowedFileExtensions: ['xls', 'xlsx']

}).on("fileuploaded", function (event, data) {

var obj = data.response;

if (obj.Tag == 1) {

filePath = obj.Result;

}

else {

filePath = '';

}

});

// 把第一步返回的文件相对地址传给导入处理方法

function saveForm(index) {

if (!filePath) {

ys.alertError('文件未上传或者上传失败');

return;

}

var postData =$("#form").getWebControls();

postData.FilePath = filePath;

ys.ajax({

url: '@Url.Content("~/OrganizationManage/User/ImportUserJson")',

type: "post",

data: postData,

success: function (obj) {

if (obj.Tag == 1) {

ys.msgSuccess('导入成功');

parent.searchGrid();

parent.layer.close(index);

}

else {

ys.msgError(obj.Message);

}

}

});

}

1. 后端写法

[HttpPost]

public async Task<IActionResult> ImportUserJson(ImportParam param)

{

List<UserEntity> list = new ExcelHelper<UserEntity>().ImportFromExcel(param.FilePath);

TData obj = await userBLL.ImportUser(param, list);

return Json(obj);

}

方法ImportFromExcel会根据传入的Excel文件相对路径，找到文件的绝对路径。Excel的第一行是标题，第二行是列名，列名可以是实体类的属性名称或者实体类属性名称的Description描述。

Excel导出

Excel导出可以根据界面的搜索条件，后台设置需要导出的列，导出所有符合条件的记录。

1. 前端写法

<a id="btnExport" class="btn btn-warning" onclick="exportForm()"><i class="fa fa-download"></i> 导出</a>

function exportForm() {

var url = '@Url.Content("~/OrganizationManage/User/ExportUserJson")';

var postData = $("#searchDiv").getWebControls();

ys.exportExcel(url, postData);

}

1. 后端写法

[HttpPost]

public async Task<IActionResult> ExportUserJson(UserListParam param)

{

TData<string> obj = new TData<string>();

TData<List<UserEntity>> userObj = await userBLL.GetList(param);

if (userObj.Tag == 1)

{

string file = new ExcelHelper<UserEntity>().ExportToExcel("用户列表.xls",

"用户列表",

userObj.Result,

new string[] { "UserName", "RealName", "Gender", "Mobile", "Email" });

obj.Result = file;

obj.Tag = 1;

}

return Json(obj);

}

方法ExportToExcel的第一个参数是Excel文件名，第二个参数是Excel的表头，第三个参数是数据源，第四个参数是导出的列（列名会自动映射为实体类上面的Description属性）

[**文件或图片上传**](http://106.14.124.170/doc/#/backend?id=%e6%96%87%e4%bb%b6%e6%88%96%e5%9b%be%e7%89%87%e4%b8%8a%e4%bc%a0)

管理后台上传的图片或文件可能也需要在Api项目能够访问，考虑到文件的统一管理，后台的文件默认是上传到Api项目里的。

YiSha.Admin.Web项目里的配置文件appsetting.json中的ApiSite节点就是配置管理后台使用的Api地址，所以如果管理后台和Api 不在一个域名下，就涉及到跨域上传。当然，Api项目通过配置是支持跨域请求的， 在Api项目的配置文件appsetting.json中的AllowCorsSite节点就是配置允许的跨域站点来请求Api。

使用流程

1. 前端图片上传
2. function uploadThumbImage(file, callback) {
3. var formdata = new FormData();
4. formdata.append("fileList", file);
5. ys.ajaxUploadFile({
6. url: '@GlobalContext.SystemConfig.ApiSite' + '/File/UploadFile?fileModule=@UploadFileType.News.ParseToInt()',
7. data: formdata,
8. success: function (obj) {
9. if (obj.Tag == 1) {
10. if (callback) {
11. callback('@GlobalContext.SystemConfig.ApiSite' + obj.Result);
12. }
13. }
14. else {
15. ys.msgError(obj.Message);
16. }
17. }
18. })

}

完成代码，可以参考News/NewsForm.cshtml页面，方法uploadThumbImage的参数file就是<input type='file' /> 控件的属性files数组的值。

1. 后端上传处理

上面代码里url中的/File/UploadFile就是后端处理上传的Controller，代码如下，参数fileModule表示上传到哪个模块，就是按模块存放文件。

[HttpPost]

public async Task<TData<string>> UploadFile(int fileModule, IFormCollection fileList)

{

TData<string> obj = await FileHelper.UploadFile(fileModule, fileList.Files);

return obj;

}

[**日志和异常处理**](http://106.14.124.170/doc/#/backend?id=%e6%97%a5%e5%bf%97%e5%92%8c%e5%bc%82%e5%b8%b8%e5%a4%84%e7%90%86)

日志分三类，登录日志、操作日志、Api日志。

日志记录采用异步的方式 Insert 到日志表中，每个调用的耗时也会记录下来，方便查看哪些调用比较耗时。 如果调用异常了，会插入一条失败的日志记录，并且异常信息会记录下来，方便debug ^\_^

1. 登录日志：用户登录或退出系统，都会插入一条日志记录，对应的表为sys\_log\_login。
2. 操作日志：用户在后台查看、新增、修改、删除数据都会插入一条日志记录，对应的表为sys\_log\_operate。
3. Api日志：Api调用每个方法的调用都会插入一条日志记录，对应的表为sys\_log\_api。

[**权限控制**](http://106.14.124.170/doc/#/backend?id=%e6%9d%83%e9%99%90%e6%8e%a7%e5%88%b6)

系统是由页面组成的，每个页面对应一个权限标识，页面包含按钮，每个按钮同样对应一个权限标识，下面以OrganizationMange模块下的News页面为例。

1. 页面权限

页面OrganizationManage/News/NewsIndex，页面可见对应的权限标识为organization:news:view。

1. 按钮权限

页面包含增删改查按钮，查询对应的权限标识为organization:news:search，增加对应的权限标识为organization:news:add， 修改对应的权限标识为organization:news:edit 删除对应的权限标识为organization:news:delete。

注意事项

1. 前台权限限制

当用户进入到系统的主页面/Home/Index时，通过yisha-data.js会把当前用户所有的权限加载进来，针对toolbar类型的按钮， 如果没有权限，按钮会被自动移除掉。按钮的命名需要和权限标识对应起来，例如权限标识organization:news:add对应在页面的按钮是btnAdd， 遵循这样的命名，框架会根据按钮的id找权限，找不到权限，按钮会被自动移除掉。

这是前台的权限限制，一般来说，在前台做这种限制，可以限制在系统操作的正常用户，如果有不友好的朋友，模拟抓包请求没有权限的接口， 所以后台也需要做权限限制。

<div class="btn-group-sm hidden-xs" id="toolbar">

<a id="btnAdd" class="btn btn-success" onclick="showSaveForm(true)"><i class="fa fa-plus"></i> 新增</a>

<a id="btnEdit" class="btn btn-primary disabled" onclick="showSaveForm(false)"><i class="fa fa-edit"></i> 修改</a>

<a id="btnDelete" class="btn btn-danger disabled" onclick="deleteForm()"><i class="fa fa-remove"></i> 删除</a>

</div>

1. 后台权限限制

在每个Action方法上加AuthorizeFilter("")这个Attribute，多个权限标识字符串用逗号隔开。

[HttpPost]

[AuthorizeFilter("organization:news:add,organization:news:edit")]

public async Task<IActionResult> SaveFormJson(NewsEntity entity)

{

TData<string> obj = await newsBLL.SaveForm(entity);

return Json(obj);

}

[**定时任务**](http://106.14.124.170/doc/#/backend?id=%e5%ae%9a%e6%97%b6%e4%bb%bb%e5%8a%a1)

在实际开发Web项目时，有一类不可缺少的，那就是定时任务。定时任务的场景可以说非常广泛，比如某些视频网站，购买会员后，每天会给会员送成长值， 每月会给会员送一些电影券；比如定时备份数据库、发送邮件、清理数据等。所以我们提供方便友好的Web界面，实现动态管理任务，可以达到控制定时任务 启动、暂停、重启、删除、添加、修改等操作，极大地方便了开发过程。

使用流程

1. 新建定时任务数据库记录

在 系统管理-> 定时任务 中新建一个任务，任务名称 用来唯一的区分一个任务。

1. 新建定时任务类

在YiSha.Business.AutoJob项目的Job文件夹，建一个任务类，继承自接口IJobTask，在任务类里面写业务代码。

1. 加入到 JobExecute

在 JobExecute 类的 Execute 方法，加一个case分支，case的名称就是第一步建的任务名称。

switch (context.JobDetail.Key.Name)

{

case "数据库备份":

obj = await new DatabasesBackupJob().Start();

break;

case "任务名称":

// do job

break;

}

[**Api接口**](http://106.14.124.170/doc/#/backend?id=api%e6%8e%a5%e5%8f%a3)

Api 对应的项目为YiSha.Admin.WebApi，使用Swagger来查看文档及测试。

1. 在控制层 Controller 中添加特定的Attribute来描述接口信息或是验证。
2. [Route("[controller]/[action]")]
3. [ApiController]
4. [AuthorizeFilter]

public class NewsController : ControllerBase

AuthorizeFilter是自定义的Attribute，用来验证调用者是否有整个Controller的权限。

1. 在Action中配置方法的Request Method
2. [HttpGet]
3. public async Task<TData<List<NewsEntity>>> GetPageList([FromQuery] NewsListParam param, [FromQuery] Pagination pagination)
4. [HttpPost]

public async Task<TData<string>> SaveForm([FromBody]NewsEntity entity)

1. 在 系统管理-> 系统Api 中查看Api文档及测试接口。

[**数据库表**](http://106.14.124.170/doc/#/backend?id=%e6%95%b0%e6%8d%ae%e5%ba%93%e8%a1%a8)

为了方便处理，数据表和实体之间的命名保持一致，采用帕斯卡命名法（也叫大驼峰命名法Upper Camel Case）。 数据库表采用 模块名+表名 的规则命名，比如系统角色表，SysRole，如果你有自己的业务模块，客户管理（Crm），客户表就可以这样命名CrmCustomer。 模块可以和MVC的Areas不是完全对应的，一切以自己的业务需要为基准。

数据库表字段的命名和表名的命名类似。还是以系统角色表为例，比如字段RoleName，就代表角色名，比如字段Remark，就表示备注。

每个表根据需要继承自不同的实体，目前可供继承的实体有BaseEntity、BaseCreateEntity、BaseModifyEntity、BaseExtensionEntity。如果你的表里只有Id字段，可以继承BaseEntity； 如果你的表里需要创建时间BaseCreateTime字段和创建人BaseCreatorId字段，可以继承BaseCreateEntity，其他以此类推。 继承合适的实体在于框架会给这些实体自动赋值，这样就不用在自己的业务模块里去操作了。

下面以MySql数据库为准，框架支持的表基础字段。

| **字段名** | **类型** | **描述** |
| --- | --- | --- |
| Id | BIGINT | 主键，非自增 |
| BaseIsDelete | INT | 删除字段标记，1表示删除，默认是0 |
| BaseCreateTime | DATETIME | 创建时间 |
| BaseModifyTime | DATETIME | 修改时间 |
| BaseCreatorId | BIGINT | 创建人 |
| BaseModifierId | BIGINT | 修改人 |
| BaseVersion | INT | 数据更新版本，控制并发 |

[**代码生成**](http://106.14.124.170/doc/#/backend?id=%e4%bb%a3%e7%a0%81%e7%94%9f%e6%88%90)

大部分项目里其实有很多代码是重复的，几乎每个基础模块的代码都有增删改查（CRUD）的功能，而这些功能都是大同小异的， 每个功能对应的View、Controller、Business、Service、Model文件结构也是一致的。 如果这些功能都要自己动手写，将会大大浪费我们的精力降低效率，所以这种重复性的代码和创造文件的工作可以使用代码生成。

注意事项

代码生成只生成单表的增删改查，如果需要复杂的多表操作，可以在生成的单表代码基础上修改。

使用流程

1. 新建数据库表结构

手动创建或是用脚本创建表，创建之后，可以在 系统工具-> 代码生成 页面看到创建的表。

1. 生成代码基本配置

选择刚刚创建的表，点击生成，会弹出一个生成代码界面，控件的 \* 表示为必填项。

类名前缀 如果没有特殊需求，可以默认，不用修改，它是根据数据库表名生成的。

输出到所在模块 此下拉框里面的值，是读取YiSha.Admin.Web项目的Areas目录里面的文件夹，如果要生成到一个新模块，可以添加一个新文件夹，注意模块名的后缀是Manage。

1. 列表页编辑

在这个页面，可以选择列表页是否需要搜索、是否需要工具栏、列表需要显示哪些字段、列表是否需要分页等，设置好之后，代码生成工具会按需生成。

1. 表单页编辑

在这个页面，可以选择表单页需要显示哪些字段、显示方式等，设置好之后，代码生成工具会按需生成。

1. 代码预览

在这个页面，可以看到即将生成的所有代码，代码会自动生成到对应模块的目录下，如果文件已经存在，就不产生新文件。  
菜单会生成在菜单根目录，在实际使用时，需要移动菜单的位置，在 系统管理-> 菜单管理 里面选择父级菜单即可。

# [常见问题](http://106.14.124.170/doc/#/question)

[**YiShaAdmin可以商用吗？**](http://106.14.124.170/doc/#/question?id=yishaadmin%e5%8f%af%e4%bb%a5%e5%95%86%e7%94%a8%e5%90%97%ef%bc%9f)

1. 可以的，YiShaAdmin是开源免费的，遵守MIT开源协议。

[**YiShaAdmin界面为什么和若依很像？**](http://106.14.124.170/doc/#/question?id=yishaadmin%e7%95%8c%e9%9d%a2%e4%b8%ba%e4%bb%80%e4%b9%88%e5%92%8c%e8%8b%a5%e4%be%9d%e5%be%88%e5%83%8f%ef%bc%9f)

1. [**若依**](http://www.ruoyi.vip/)是一个开源的Java后台开发框架，整体的界面设计是很好的，YiShaAdmin里面的CSS样式是借鉴若依的，此处应有感谢。
2. YiShaAdmin的前端控件封装和页面JS控制，和若依是不一样的，后端那就完全不一样了。

[**部署到Windows Server或Linux服务器，不能访问？**](http://106.14.124.170/doc/#/question?id=%e9%83%a8%e7%bd%b2%e5%88%b0windows-server%e6%88%96linux%e6%9c%8d%e5%8a%a1%e5%99%a8%ef%bc%8c%e4%b8%8d%e8%83%bd%e8%ae%bf%e9%97%ae%ef%bc%9f)

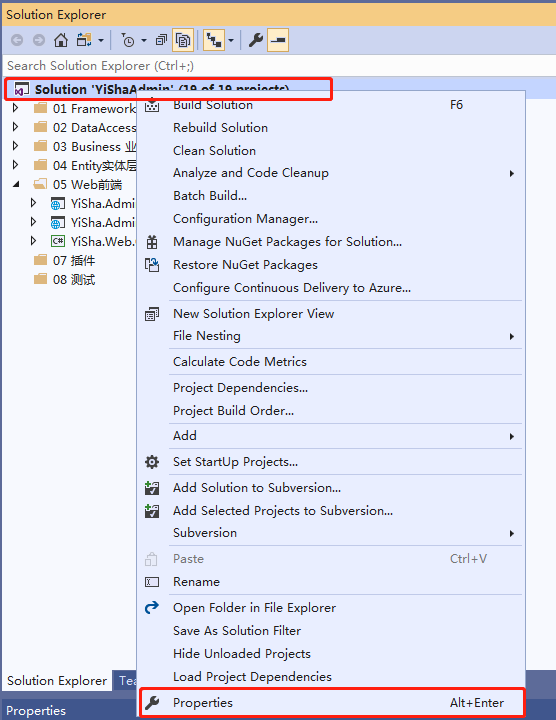
请确保你安装了.NET Core Runtime，根据自己项目使用的.NET Core版本，下载对应的版本 [**下载链接**](https://dotnet.microsoft.com/download/dotnet-core)

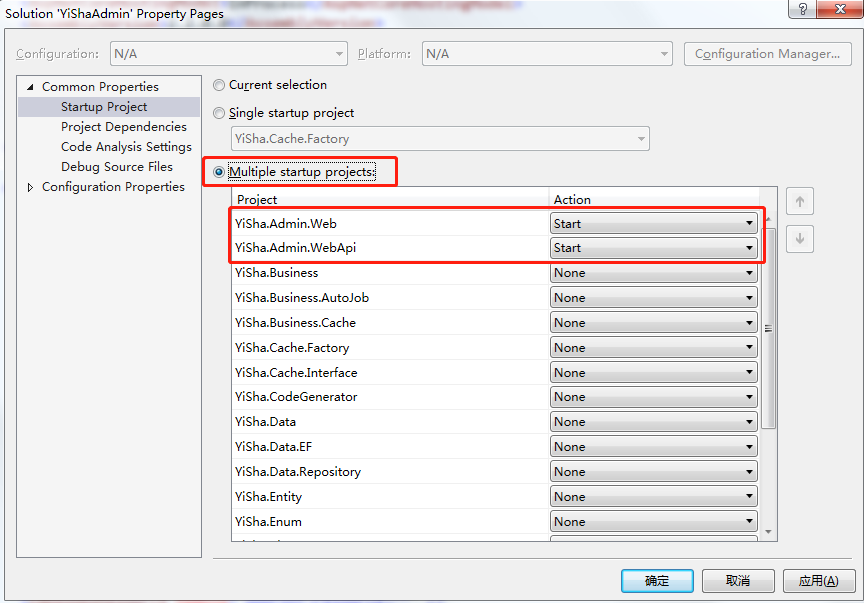
[**部署到CentOS 7服务器，验证码不显示？**](http://106.14.124.170/doc/#/question?id=%e9%83%a8%e7%bd%b2%e5%88%b0centos-7%e6%9c%8d%e5%8a%a1%e5%99%a8%ef%bc%8c%e9%aa%8c%e8%af%81%e7%a0%81%e4%b8%8d%e6%98%be%e7%a4%ba%ef%bc%9f)

安装GDI依赖 sudo yum install libgdiplus

[**如何用Visual Studio同时启动2个项目？**](http://106.14.124.170/doc/#/question?id=%e5%a6%82%e4%bd%95%e7%94%a8visual-studio%e5%90%8c%e6%97%b6%e5%90%af%e5%8a%a82%e4%b8%aa%e9%a1%b9%e7%9b%ae%ef%bc%9f)

1. 首先在解决方案上右击-> 属性



1. 然后在弹出的窗口中选择启动多个项目，选择可以启动的项目，YiSha.Admin.Web和YiShaAdmin.WebApi，Action 选择 Start，点击确定完成。 

[**'OFFSET' 附近有语法错误。在 FETCH 语句中选项 NEXT 的用法无效。**](http://106.14.124.170/doc/#/question?id=_39offset39-%e9%99%84%e8%bf%91%e6%9c%89%e8%af%ad%e6%b3%95%e9%94%99%e8%af%af%e3%80%82%e5%9c%a8-fetch-%e8%af%ad%e5%8f%a5%e4%b8%ad%e9%80%89%e9%a1%b9-next-%e7%9a%84%e7%94%a8%e6%b3%95%e6%97%a0%e6%95%88%e3%80%82)

目前EF Core支持Sql Server的最低版本是2012，如果你用的数据库是Sql Server 2012以下，是不支持的。

[**主键Id为什么返回前端需要转换成字符串？**](http://106.14.124.170/doc/#/question?id=%e4%b8%bb%e9%94%aeid%e4%b8%ba%e4%bb%80%e4%b9%88%e8%bf%94%e5%9b%9e%e5%89%8d%e7%ab%af%e9%9c%80%e8%a6%81%e8%bd%ac%e6%8d%a2%e6%88%90%e5%ad%97%e7%ac%a6%e4%b8%b2%ef%bc%9f)

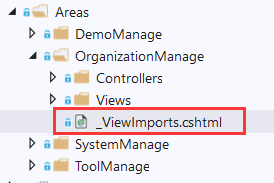
主键Id在后端用的是long类型，C#中long的数据范围是-2^63到2^63-1，但是在JavaScript中，Number能表示的范围是-2^53到2^53-1，比后端C#的小， 所以后端的long，在前端需要转成string。

[**IIS如何配置多个SSL证书？**](http://106.14.124.170/doc/#/question?id=iis%e5%a6%82%e4%bd%95%e9%85%8d%e7%bd%ae%e5%a4%9a%e4%b8%aassl%e8%af%81%e4%b9%a6%ef%bc%9f)

要想在IIS里面配置多个SSL证书，IIS的版本至少要在IIS 8及以上。具体可以参考 [**链接**](https://serverfault.com/questions/761089/iis-7-5-cant-configure-two-ssl-certificates-under-one-ip-adress)。

[**.cshtml文件里如何引用命名空间？**](http://106.14.124.170/doc/#/question?id=cshtml%e6%96%87%e4%bb%b6%e9%87%8c%e5%a6%82%e4%bd%95%e5%bc%95%e7%94%a8%e5%91%bd%e5%90%8d%e7%a9%ba%e9%97%b4%ef%bc%9f)

首先可以在页面头部使用@using 命名空间引入，这种引入的命名空间，里面的类只能在当前页面使用，如果想让Views文件下所有的.cshtml文件都可以使用， 在Views同级目录新建一个\_ViewImports.cshtml文件，把需要的命名空间放到新建的这个文件里即可，参考下图。



[**多表关联查询如何写？**](http://106.14.124.170/doc/#/question?id=%e5%a4%9a%e8%a1%a8%e5%85%b3%e8%81%94%e6%9f%a5%e8%af%a2%e5%a6%82%e4%bd%95%e5%86%99%ef%bc%9f)

由于DbContext里面没有添加DBSet这样的实体，所以多表关联查询不能使用linq，只能使用原生的sql写法(参考LogLoginService.GetPageList)或者使用视图。 另一方面考虑到EF Core生成的多表关联查询，字段会比较啰嗦，影响性能。

# [捐赠支持](http://106.14.124.170/doc/#/donate)

## [捐赠支持](http://106.14.124.170/doc/#/donate?id=%e6%8d%90%e8%b5%a0%e6%94%af%e6%8c%81)

**Donate**

**如果你觉得这个项目帮助到了你，你可以请作者喝杯咖啡表示鼓励 ☕️**

