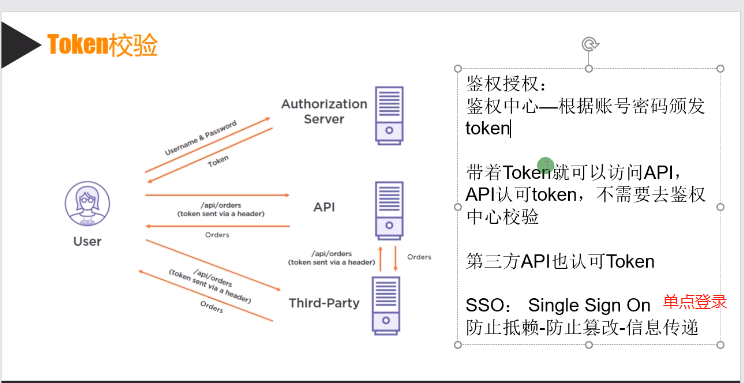
推荐使用 IdentityServer4

Token:官网 jwt.io

## JWT





## 使用：

### 原理

用户提交账号密码给服务器，成功的话，jwt创建一个token返回给用户，存储在客户端cookie中，jwt的字符串是使用.(点)连接的三段字符串，

token：eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

第一段是 header,json转成字符串以后baseurl64加密

{

"alg": "HS256",

"typ": "JWT"

}

第二段是 PAYLOAD 有效载荷，登录成功以后的用户相关信息，id,账号,过期时间等等，,json转成字符串以后baseurl64加密

{

"sub": "1234567890",

"name": "John Doe",

"iat": 1516239022,

“exp”:” 1516239022”

}

第三段是 VERIFY SIGNATURE 签名

将第一和第二段的密文拼接起来进行HS256加密+加盐，把生成的密文再次进行baseurl64加密

加盐：用户在输入密码注册后，随机生成一个 salt，一定要是随机生成的。然后按照一定的混淆规则，将 salt 撒入到明文密码中，最后将加盐后的密码做消息摘要，将 salt 和密码的消息摘要传给服务器。服务器如何匹配用户密码是否正确呢？用户用同样的方式对原始密码加盐再生成摘要，发送给服务器，服务器匹配摘要。因为盐是随机生成的，不用每次都生成盐，盐可以存在本地或者服务器，加盐的规则只要不泄漏出去就行，这样的处理比简单的对密码摘要要安全很多。

### 校验

由于token不存储在服务器，所以需要用户访问的时候(携带token)，去校验token

1.使用点(.)分割token,成3段

2.对第二段进行baseurl64解密，获取payload信息，检测token是否过期

3.将第一和第二段的密文拼接起来进行HS256加密+加盐，把生成的密文再次进行baseurl64加密

4.将token和密文比较，一样就是没有被更改的

1. 引入System.IdentityModel.Tokens.Jwt，Microsoft.AspNetCore.Authentication.JwtBearer包
2. 使用

[HttpPost]

[Route("login")]

[Authorize]

public string Login(string name, string password)

{

if ("zy".Equals(name) && "123456".Equals(password))

{

//账号密码正确，获取token

string token = \_jWTService.GetToken(name);

return JsonConvert.SerializeObject(new

{

result = true,

token

});

}

else {

return JsonConvert.SerializeObject(new

{

result = false,

token=""

});

}

}

1. Startup. ConfigureServices

//注入

services.AddScoped<IJWTService, JWTService>();

#region jwt鉴权授权

var audience = this.Configuration["audience"];

var issuer = this.Configuration["issuer"];

var SecurityKey = this.Configuration["SecurityKey"];

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)//默认授权机制

.AddJwtBearer(options=> {

options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters

{

ValidateIssuer = true,//是否验证

ValidateAudience=true,

ValidateLifetime=true,

ValidateIssuerSigningKey=true,

ValidAudience= audience,

ValidIssuer= issuer,

IssuerSigningKey=new SymmetricSecurityKey(Encoding.UTF8.GetBytes(SecurityKey))

};

});

1. Startup. Configure

//通过中间件来进行鉴权授权

app.UseAuthentication();

1. appsettings.json

"audience": "http://localhost:10086",

"issuer": "http://localhost:10086",

"SecurityKey": ""

1. 创建授权类JWTService

public interface IJWTService

{

string GetToken(string username);

}

public class JWTService : IJWTService

{

private readonly IConfiguration \_iConfiguration;

public JWTService(IConfiguration iConfiguration)

{

\_iConfiguration = iConfiguration;

}

public string GetToken(string username)

{

/\*\*

1. \* claims Payload 有效载荷，包含了一些和token有关的信息，建议配置在 配置文件appsettings.json

中

\* 这些字段是关键字，除了这些可以包含其他任何json可以兼容的字段

\* issuer：给谁的

\* sub:主题

\* iat：时间戳

\* jti: token唯一标识

\* audience:签发人

\*/

Claim[] claims = new[] {

new Claim(ClaimTypes.Name,username),

new Claim("name", "zy"),

new Claim("age", "18"),

new Claim("money","5000000") //其他字段信息

};

SymmetricSecurityKey key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_iConfiguration["SecurityKey"]));

SigningCredentials creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

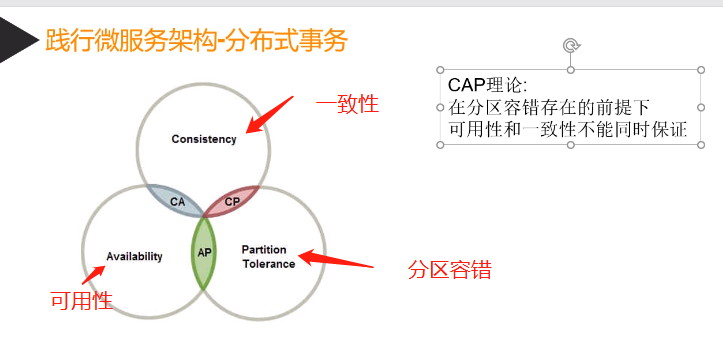
var token = new JwtSecurityToken(issuer: \_iConfiguration["issuer"],audience: \_iConfiguration["audience"],claims: claims,expires:DateTime.Now.AddMinutes(5),signingCredentials: creds);

string returntoken = new JwtSecurityTokenHandler().WriteToken(token);

return returntoken;

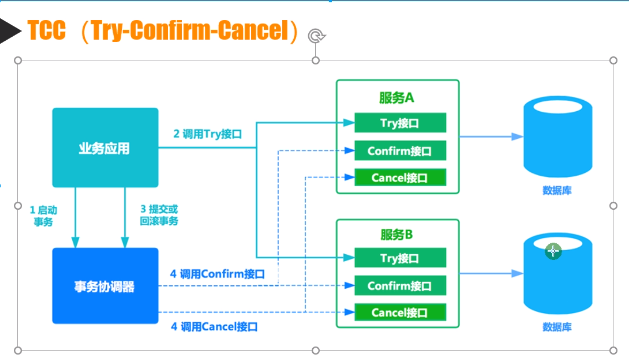
}

}

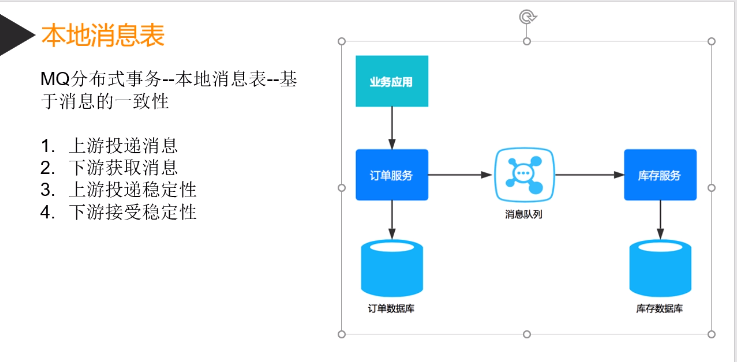
2pc/3pc

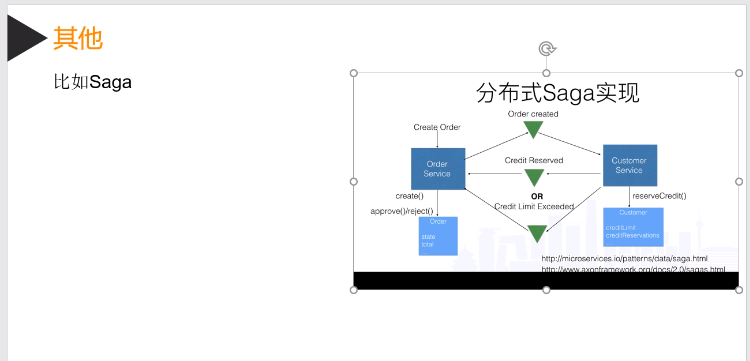
Base理论：分区容错前提下，优先保证可用性，然后最终结果一致就行。

Tcc: 银行用的多



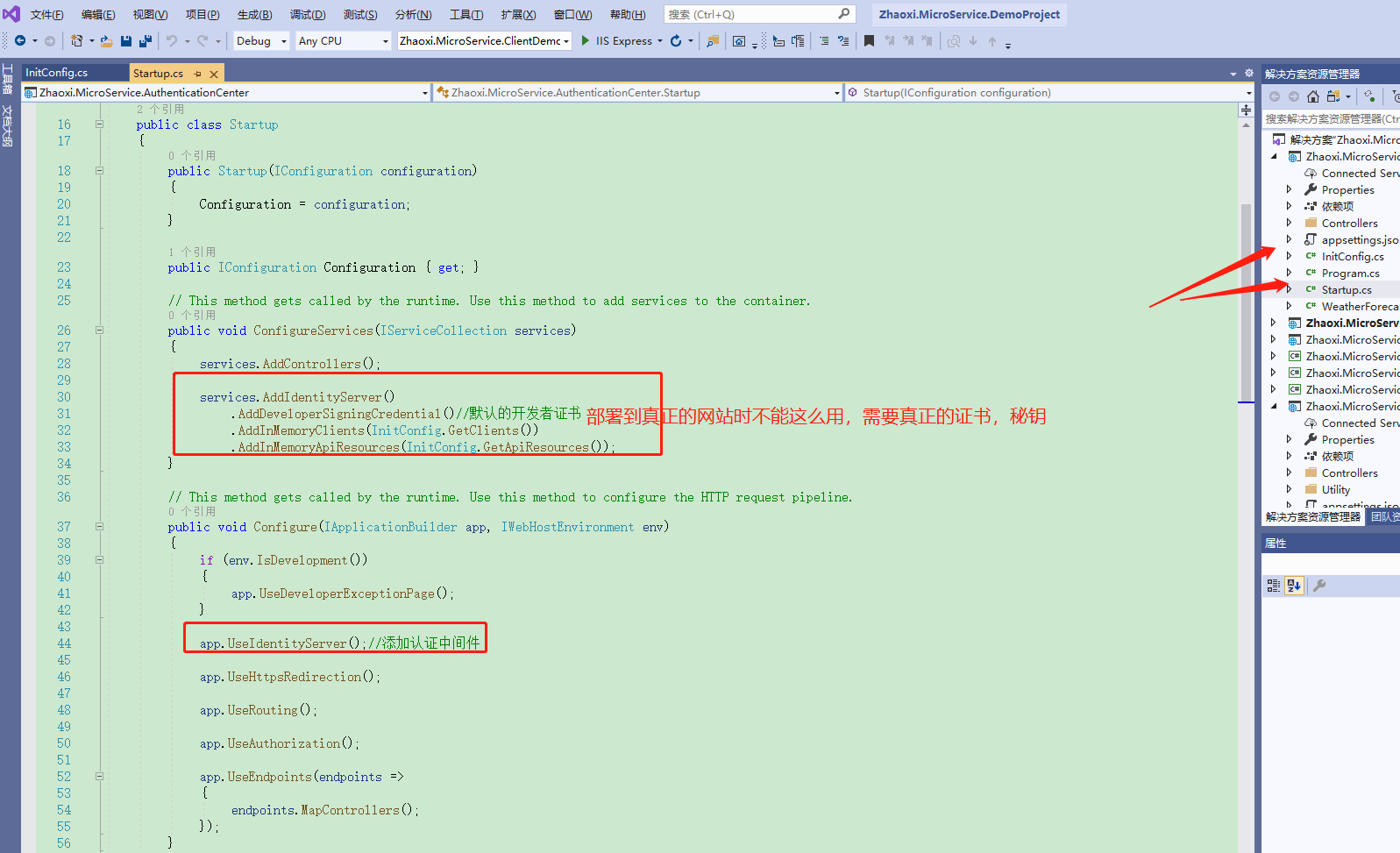
互联网常用:



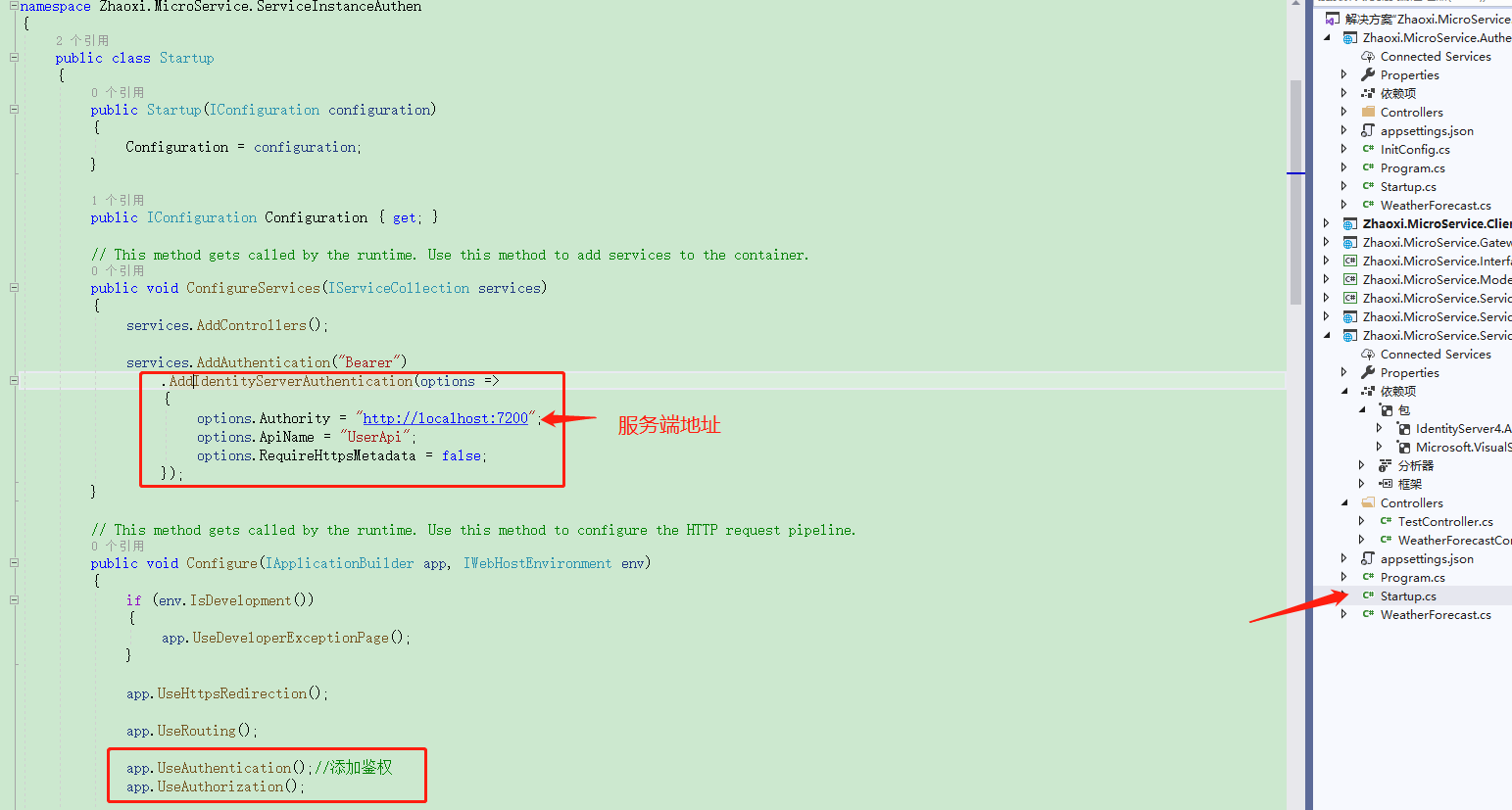


## IdentityServer4

添加IdentityServer4包 (服务端)



客户端 添加IdentityServer4.AccessTokenValidation 包：



然后在需要的地方添加授权特性[Authorize] ，没有授权，返回401

[Authorize]

[HttpGet]

[Route("IndexA")]

public IActionResult IndexA()

{

return new JsonResult(new

{

message = "This is TestControllerIndex",

Port = this.\_iConfiguration["port"],

Time = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss fff")

});

}