Ocelot 中文文档二

作者：loogn
原文链接：https://ld246.com/article/1562115708078
来源网站：链滴

GraphQL

好吧！你明白我的意思Ocelot并不直接支持GraphQL，但有这么多人问起它，我想表明整合graphql-otnet库是多么容易

请参阅示例项目OcelotGraphQL。 结合使用graphql-dotnet项目和Ocelot的DelegatingHandler功，这很容易实现。 不过，我现在不打算更加密切地与GraphQL集成。 查看示例的自述文件，应该给你足够的指导如何去做！

祝你好运，玩得开心：>

服务发现

Ocelot允许您指定服务发现提供程序，并使用它来查找Ocelot正在将请求转发给下游服务的主机和端。目前，这仅在GlobalConfiguration部分中受支持，这意味着所有ReRoute将使用相同的服务发现供程序，以便在ReRoute级别指定ServiceName。

Consul

GlobalConfiguration中需要以下内容。 提供者是必需的，如果你没有指定主机和端口，默认使用Consul。

```
"ServiceDiscoveryProvider": {
    "Host": "localhost",
    "Port": 9500
}
```

将来我们可以添加一个功能，允许ReRoute配置服务发现提供程序。

为了告诉Ocelot一个ReRoute需要使用服务发现提供程序来发现下游主机和端口，您必须在下游请求置中添加ServiceName，UseServiceDiscovery和LoadBalancer。 目前Ocelot有RoundRobin(轮询和LeastConnection(最少连接)两个负载均衡的算法。 如果没有指定负载均衡器，Ocelot将不会均衡求。

```
{
    "DownstreamPathTemplate": "/api/posts/{postId}",
    "DownstreamScheme": "https",
    "UpstreamPathTemplate": "/posts/{postId}",
    "UpstreamHttpMethod": [ "Put" ],
    "ServiceName": "product",
    "LoadBalancer": "LeastConnection",
    "UseServiceDiscovery": true
}
```

如此设置之后，Ocelot将从服务发现提供程序查找下游主机和端口，并通过可用服务平衡请求。

ACL Token

如果您使用Consul的ACL，Ocelot也支持添加X-Consul-Token头。 为了实现ACL访问，您必须添加面的附加属性Token。

```
"ServiceDiscoveryProvider": {
```

```
    "Host": "localhost",
    "Port": 9500,
    "Token": "footoken"
}
```

Ocelot会将这个令牌添加到用来发出请求的consul客户端，然后用于后续的每个请求。

Eureka

这个功能是作为问题 262的一部分被提出。为Netflix的Eureka服务发现提供程序添加支持。 主要原是它是Steeltoe的一个关键部分，Steeltoe又与Pivotal有关！ 反正背景很牛逼。

为了使Eureka工作，需要在 ocelot.json中添加如下配置..

```
"ServiceDiscoveryProvider": {
    "Type": "Eureka"
}
```

遵循这里的指导，您可能还需要添加一些内容到appsettings.json。 例如，下面的json告诉steeltoe/键服务在哪里寻找服务发现服务器，以及服务是否应该向其注册。

```
"eureka": {
    "client": {
    "serviceUrl": "http://localhost:8761/eureka/",
    "shouldRegisterWithEureka": false,
    "shouldFetchRegistry": true
    }
}
```

我被告知，如果shouldRegisterWithEureka是false，那么shouldFetchRegistry将会默认为true，以你不需要显式地将它留在这里。

现在Ocelot将在启动时注册所有必要的服务，并且如果配置有上述json，则会将其注册到Eureka。其一项服务每30秒（默认）轮询一次Eureka获取最新的服务状态并将其保留在内存中。当Ocelot要求供给定的服务时，它会从内存中检索出来，因此性能不是一个大问题。注意，此代码由Pivotal.Discovery.Client 的NuGet包提供，所以非常感谢他们的辛勤工作。

微服务ServiceFabric

如果您在Service Fabric中部署了服务，则通常会使用命名服务来访问它们。

以下示例展示如何设置一个ReRoute以便在在Service Fabric中工作。 最重要的是ServiceName，它Service Fabric应用程序名称和特定服务名称组成的。 我们还需要将UseServiceDiscovery设置为tru，并在GlobalConfiguration中设置ServiceDiscoveryProvider。 这里的例子显示了一个典型的配置。它假定Service Fabric在本地主机上运行，并且命名服务位于19081端口上。

下面的例子来例子本文件夹，所以请检查它配置是否与你的实际情况相符！

```
    "ReRoutes": [
        {
        "DownstreamPathTemplate": "/api/values",
        "UpstreamPathTemplate": "/EquipmentInterfaces",
        "UpstreamHttpMethod": [
            "Get"
```

```
        ],
        "DownstreamScheme": "http",
        "ServiceName": "OcelotServiceApplication/OcelotApplicationService",
        "UseServiceDiscovery" :  true
        }
    ],
    "GlobalConfiguration": {
        "RequestIdKey": "OcRequestId",
        "ServiceDiscoveryProvider": {
            "Host": "localhost",
            "Port": 19081,
            "Type": "ServiceFabric"
        }
    }
}
```

如果您使用无状态/ guest exe服务，ocelot将能够通过命名服务进行代理，而无需其他任何操作。是，如果您使用的是有状态/actor服务，则客户端请求必须发送PartitionKind和PartitionKey查询字串。

```
GET http://ocelot.com/EquipmentInterfaces?PartitionKind=xxx&PartitionKey=xxx
```

Ocelot无法为您解决这个问题。

认证

为了验证ReRoutes并随后使用Ocelot的任何基于声明的功能，如授权或使用令牌中的值修改请求。户必须像往常一样在他们的Startup.cs中注册认证服务，但他们给每个注册提供了一个方案（认证提商密钥），例如

```
{
    var authenticationProviderKey = "TestKey";

    services.AddAuthentication()
        .AddJwtBearer(authenticationProviderKey, x =>
        {
        });
}
```

在此示例中，TestKey是此提供程序已注册的方案。 然后，我们将其映射到配置中的ReRoute，例如

```
"ReRoutes": [{
        "DownstreamHostAndPorts": [
            {
                "Host": "localhost",
                "Port": 51876,
            }
        ],
        "DownstreamPathTemplate": "/",
        "UpstreamPathTemplate": "/",
        "UpstreamHttpMethod": ["Post"],
        "ReRouteIsCaseSensitive": false,
        "DownstreamScheme": "http",
        "AuthenticationOptions": {
```

```
        "AuthenticationProviderKey": "TestKey",
        "AllowedScopes": []
    }
}]
```

当Ocelot运行时，它会查看此ReRoutes的AuthenticationOptions.AuthenticationProviderKey并查是否存在给定密钥注册的身份验证提供程序。 如果没有，那么Ocelot不会启动，如果有的话ReRoue将在执行时使用该提供者。

如果ReRoute配置了认证，Ocelot在执行认证中间件时将调用与其相关的任何验证方案。 如果请求证失败，Ocelot返回http状态码401。

JWT令牌

如果您想使用JWT令牌进行身份验证，例如Auth0等提供商，您可以使用正常的方式注册你的身份验中间件。

```
public void ConfigureServices(IServiceCollection services)
{
    var authenticationProviderKey = "TestKey";

    services.AddAuthentication()
        .AddJwtBearer(authenticationProviderKey, x =>
        {
            x.Authority = "test";
            x.Audience = "test";
        });

    services.AddOcelot();
}
```

然后将身份验证提供程序密钥映射到配置中的ReRoute，例如

```
"ReRoutes": [{
        "DownstreamHostAndPorts": [
            {
                "Host": "localhost",
                "Port": 51876,
            }
        ],
        "DownstreamPathTemplate": "/",
        "UpstreamPathTemplate": "/",
        "UpstreamHttpMethod": ["Post"],
        "ReRouteIsCaseSensitive": false,
        "DownstreamScheme": "http",
        "AuthenticationOptions": {
            "AuthenticationProviderKey": "TestKey",
            "AllowedScopes": []
        }
    }]
```

Identity Server 承载令牌

为了使用IdentityServer承载令牌，请按照惯例在ConfigureServices 中使用方案（密钥）注册您的IdentityServer服务。 如果您不明白如何操作，请访问IdentityServer文档。

```
public void ConfigureServices(IServiceCollection services)
{
    var authenticationProviderKey = "TestKey";
    var options = o =>
        {
            o.Authority = "https://whereyouridentityserverlives.com";
            o.ApiName = "api";
            o.SupportedTokens = SupportedTokens.Both;
            o.ApiSecret = "secret";
        };

    services.AddAuthentication()
        .AddIdentityServerAuthentication(authenticationProviderKey, options);

    services.AddOcelot();
}
```

然后将身份验证提供程序密钥映射到配置中的ReRoute，例如

```
"ReRoutes": [{
        "DownstreamHostAndPorts": [
            {
                "Host": "localhost",
                "Port": 51876,
            }
        ],
        "DownstreamPathTemplate": "/",
        "UpstreamPathTemplate": "/",
        "UpstreamHttpMethod": ["Post"],
        "ReRouteIsCaseSensitive": false,
        "DownstreamScheme": "http",
        "AuthenticationOptions": {
            "AuthenticationProviderKey": "TestKey",
            "AllowedScopes": []
        }
    }]
```

允许的范围

如果将范围添加到AllowedScopes，Ocelot将获得类型范围的所有用户声明（从令牌中），并确保户具有列表中的所有范围。

这是一种基于范围限制对ReRoute访问的方式。

授权

Ocelot支持基于声明的授权。 这意味着如果您有要授权的路由，您可以将以下内容添加到您的ReRoue配置中。

```
"RouteClaimsRequirement": {
```

```
    "UserType": "registered"
}
```

在此示例中，授权中间件被调用时，Ocelot将检查用户是否拥有声明类型UserType以及该声明的值否是registered。 如果不是，那么用户将不会被授权，并且响应将被禁止。

Websockets

Ocelot额外支持代理websockets。这个功能在问题 212中被提出。

为了是Ocelot代理websocket，你需要做如下事情。

在你的Configure方法中，你要告知应用程序使用WebSockets。

```
 Configure(app =>
{
    app.UseWebSockets();
    app.UseOcelot().Wait();
})
```

然后在你的ocelot.json中添加如下代码，用于配置websockets代理一个ReRoute。

```
{
    "DownstreamPathTemplate": "/ws",
    "UpstreamPathTemplate": "/",
    "DownstreamScheme": "ws",
    "DownstreamHostAndPorts": [
        {
            "Host": "localhost",
            "Port": 5001
        }
    ],
 }
```

使用这个配置，Ocelot将匹配所有进入的Websocket流量并将其代理到localhost:5001/ws。 更清晰解释一下就是，Ocelot将接收来自上游客户端的消息，并将它们代理到下游服务，从下游服务接收消并将这些消息代理到上游客户端。

已支持的

1. 负载均衡
2. 路由
3. 服务发现

这意味着您可以运行websockets的下游服务，并在您的ReRoute配置中使用多个DownstreamHostAdPorts，或将您的ReRoute挂接到服务发现提供程序上，然后负载均衡上游请求.....我认为这很酷:)

不支持的

不幸的是，很多Ocelot的功能都是非websocket所特有的，比如header和http客户端填充。 我列出以下不适用的功能。

1. 跟踪
2. 请求Id
3. 请求聚合
4. 流量限制
5. 服务质量
6. 中间件注入
7. 转换Header
8. 委托处理程序
9. 转换声明
10. 缓存
11. 认证 - 如果有人请求它，我们可能可以使用基本身份验证做些事情
12. 授权

我不是100%确定这个功能在进入高速运转时会发生什么，所以请确保您彻底测试过！

管理

Ocelot支持在运行时通过一个认证的Http API修改配置。有两种方式对其验证， 使用Ocelot的内置IdentityServer（仅用于向管理API验证请求）或将管理API验证挂接到您自己的IdentityServer中。

提供你自己的 IdentityServer

挂接到你自己的IdentityServer，你只需要添加一下代码到你的ConfigureServices 方法中。

```
public virtual void ConfigureServices(IServiceCollection services)
{
   Action<IdentityServerAuthenticationOptions> options = o => {
         // o.Authority = ;
         // o.ApiName = ;
         // etc....
      };

   services
      .AddOcelot()
      .AddAdministration("/administration", options);
}
```

您现在需要从你的IdentityServer获取令牌，并在后续请求Ocelot的管理API时使用。

这个功能是对问题 228 的实现。这个功能很有用，因为IdentityServer认证中间件需要IdentityServe的URL。 如果您使用内置IdentityServer，则可能无法获得Ocelot URL。

内置 IdentityServer

管理API使用您从Ocelot请求的持票人令牌进行身份验证。这是由我已经使用了几年的非常了不起的Identity Server项目提供的。 您可以去看一下。

为了启用管理部分，您需要做一些操作。 首先将此添加到您的初始化文件Startup.cs中。

管理路径可以是任何值，显然不能使用将要通过Ocelot路由的url，因为这是行不通的。管理功能使用sp.net core的MapWhen功能，并且所有到{root} /administration的请求将被发送到那里，而不是Oelot中间件。

secret是Ocelot内置IdentityServer用于验证对管理API请求的客户端密钥。你可以随意填写！

```
public virtual void ConfigureServices(IServiceCollection services)
{
    services
        .AddOcelot()
        .AddAdministration("/administration", "secret");
}
```

现在，如果您使用上述配置选项并想要访问API，可以使用解决方案中名为ocelot.postman_collection.json的postman脚本来更改Ocelot配置。 显然，如果Ocelot运行在不同与http://localhost:5000的URL上，则需要改一下。

这些脚本向您展示了如何从ocelot请求bearer令牌，然后使用它来获取现有配置和修改配置。

如果您在群集中运行多个Ocelot，则需要使用证书对用于访问管理API的bearer令牌签名。

为了做到这一点，您需要为集群中的每个Ocelot再添加两个的环境变量。

OCELOT_CERTIFICATE

用于签名令牌的证书路径。 证书必须是X509类型，显然Ocelot要能够访问它。

OCELOT_CERTIFICATE_PASSWORD

证书的密码。

通常Ocelot只使用临时签名凭证，但如果您设置了这些环境变量，那么它将使用设置的证书。 如果群中的所有其他Ocelot都具有相同的证书，那很棒，这样就对了！

管理 API

POST {adminPath}/connect/token

这会使用我们上面讨论的客户端证书得到一个用于管理区域的令牌。 在这种情况下，这将调用Ocelo中托管的IdentityServer。

请求体是from-data,允许一下数据：

client_id 设置为admin

client_secret 设置为您在设置管理服务时使用的内容

scope 设置为admin

grant_type 设置为client_credentials

GET {adminPath}/configuration

获得当前的Ocelot配置。 这与我们先前设置Ocelot的JSON完全相同。

POST {adminPath}/configuration

这会覆盖现有的配置（可能应该是put更合适！）。 我建议用GET获取您的配置，进行更改后使用此ap发回...。

这个请求体的JSON，格式和我们使用文件系统设置Ocelot的 FileConfiguration.cs格式相同

DELETE {adminPath}/outputcache/{region}

这将清空特定区域的缓存。如果您使用空白region，它将清除缓存的所有实例！赋予您运行Ocelots群，并将其全部缓存在内存中，并同时清除所有缓存（仅使用分布式缓存）的能力。

region是您在Ocelot配置的FileCacheOptions部分中针对region字段设置的内容。

流量控制

感谢@catcherwong 的文章激励我最终写出了这个文档

Ocelot支持上游的请求限制，以便您的下游服务不会过载。 此功能是由GitHub上的@geffzhang添！ 非常感谢。

好了，为了让ReRoute获得流量限制，你需要添加下面的json到ReRoute中。

```
"RateLimitOptions": {
    "ClientWhitelist": [],
    "EnableRateLimiting": true,
    "Period": "1s",
    "PeriodTimespan": 1,
    "Limit": 1
}
```

ClientWhitelist - 是一个包含客户端白名单列表的数组。这意味着在白名单里的客户端不受流量限制影响。

EnableRateLimiting - 该值指定流量限制是否开启。

Period - 该值指定时间段，例如1s，5m，1h，1d等。

PeriodTimespan - 该值指定我们可以在一定的秒数后重试。

Limit - 该值指定一个客户端可以在定义的时间段(Period)内允许的最大请求数。

您还可以在ocelot.json的GlobalConfiguration部分中设置以下内容

```
"RateLimitOptions": {
  "DisableRateLimitHeaders": false,
  "QuotaExceededMessage": "Customize Tips!",
  "HttpStatusCode": 999,
  "ClientIdHeader" : "Test"
}
```

DisableRateLimitHeaders - 该值指定是否禁用X-Rate-Limit和Rety-After头。

QuotaExceededMessage - 该值指定超出限制时返回的消息。

HttpStatusCode - 该值指定超出限制时返回的HTTP状态代码。

ClientIdHeader - 允许您指定应该用于标识客户端的头。 默认是“ClientId”

缓存

目前Ocelot使用CacheManager项目提供了一些非常基本的缓存。这是一个了不起的项目，它解决了多缓存问题。 我会推荐这个软件包来做Ocelot缓存。 如果你看看这里的例子，你可以看到如何设置存管理器，然后传入Ocelot的AddOcelotOutputCaching配置方法。 您可以使用CacheManager软包支持的任何设置，只需传入即可。

无论如何，Ocelot目前支持对下游服务的URL进行缓存，并可以设置一个以秒为单位的TTL使缓存过。 您也可以通过调用Ocelot的管理API来清除某个Region的缓存。

为了在路由中使用缓存，需要在ReRoute中加上如下设置。

```
"FileCacheOptions": { "TtlSeconds": 15, "Region": "somename" }
```

示例中ttl设置为15表示缓存在15秒后过期。

Qos服务质量

目前Ocelot支持一种QoS功能。 如果您希望在请求向下游服务时使用断路，则可以在ReRoute中进设置。 这个功能使用了一个名为Polly的.NET库，这个库很棒，在这里可以找到它。

添加如下配置块到一个ReRoute配置中。

```
"QoSOptions": {
  "ExceptionsAllowedBeforeBreaking":3,
  "DurationOfBreak":5,
  "TimeoutValue":5000
}
```

为了实现这个规则，你必须设置一个大于0的数字给ExceptionsAllowedBeforeBreaking。DurationOBreak是断路器跳闸后保持断开的时间。TimeoutValue表示如果请求超过5秒钟，它将自动超时。

你可以单独设置TimeoutValue选项，而不设置ExceptionsAllowedBeforeBreaking 和DurationOfBrak 。

```
"QoSOptions": {
  "TimeoutValue":5000
}
```

单独设置另外两个选项其中之一是没有意义的，因为他们两个相互影响。

如果您不添加QoS部分，QoS将不会被使用，但Ocelot默认将所有下游请求的超时时间设置为90秒。如果有人需要这个90秒是可配置，请提出问题。