Додаток 3

Лістинг програми

namespace NextGenBase

{

[Service(typeof (IDataPresentationService), typeof (DefaultPresentationService))]

[DataService(typeof (DefaultDataService))]

[AuthService(typeof (DefaultAuthService))]

[Route("", "{controller}")]

public abstract partial class CRUDProvider<T, TEntity> : CRUDProvider, IRESTfull<T>, IDisposable

where T : class, new()

where TEntity : class, new()

{

#region Metadata

protected static readonly Dictionary<Type, ControllerMetadata<T, TEntity>> ControllerMetadatas

= new Dictionary<Type, ControllerMetadata<T, TEntity>>();

protected static readonly Dictionary<Interface, Type> InterfaceTypes

= new Dictionary<Interface, Type>

{

{Interface.Get, typeof (IRESTGet)},

{Interface.Put, typeof (IRESTPut<T>)},

{Interface.Post, typeof (IRESTPost<T>)},

{Interface.Patch, typeof (IRESTPatch)},

{Interface.Delete, typeof (IRESTDelete)},

{Interface.Search, typeof (IRESTSearch)},

{Interface.Reference, typeof (IRESTReference)}

};

private static readonly Dictionary<Type, Type> InterfacesChainTypes = new Dictionary<Type, Type>

{

{InterfaceTypes[Interface.Get], typeof (ActionContainer<IQueryable<TEntity>>)},

{InterfaceTypes[Interface.Put], typeof (ActionContainer<T>)},

{InterfaceTypes[Interface.Post], typeof (ActionContainer<T>)},

{InterfaceTypes[Interface.Patch], typeof (ActionContainer<T>)},

{InterfaceTypes[Interface.Delete], typeof (ActionContainer<T>)},

{InterfaceTypes[Interface.Search], typeof (ActionContainer<IQueryable<TEntity>>)}

};

private readonly ControllerMetadata<T, TEntity> \_metadata;

#endregion

#region Route

private static void Route(RouteCollection routes, HttpConfiguration config, Type x)

{

RouteStartPointAttribute tempAttribute;

IEnumerable<DisableRoutesAttribute> tempDisableRoutesAttribute = null;

//types.ForEach(x =>

x.Apply(d => tempDisableRoutesAttribute = CustomAttributeExtensions.GetCustomAttributes<DisableRoutesAttribute>((MemberInfo) d))

.GetMethods().ForEach(m =>

m.GetCustomAttributes<RouteAttribute>(true).Reverse()

.Apply<RouteAttribute>(o =>

{

if (typeof (IController).IsAssignableFrom(x))

{

var mappper = o.GetType().GetMethod("MapMvcRoute");

MethodInfo genericMethod = mappper.MakeGenericMethod(x, typeof (T));

genericMethod.Invoke(o,

new object[]

{

routes, x,

(tempAttribute = x.GetCustomAttribute<RouteStartPointAttribute>()) != null

? tempAttribute.StartPoint

: string.Empty

});

}

else

{

var mappper = o.GetType().GetMethod("MapHttpRoute");

MethodInfo genericMethod = mappper.MakeGenericMethod(x, typeof (T));

genericMethod.Invoke(o,

new object[]

{

x, m, config,

(tempAttribute = x.GetCustomAttribute<RouteStartPointAttribute>()) != null

? tempAttribute.StartPoint

: string.Empty

});

}

})

.Do());

}

#endregion

#region DataProvider

protected IDbSet<TEntity> DataProvider

{

get { return Repository.Set<TEntity>(); }

}

protected abstract IRepository Repository { get; }

#endregion

#region Data

protected PropertyInfo[] DataMetadata

{

get { return MetadataProvider.GetDataMetadata(typeof (T)); }

}

protected virtual T GetObject(params object[] keys)

{

return DataService.Map<TEntity, T>(DataProvider.Find(keys));

}

protected virtual TEntity GetEntity(params object[] keys)

{

return DataProvider.Find(keys);

}

#endregion

#region ctor

static CRUDProvider()

{

Mapper.Create<T, TEntity>();

Mapper.Create<TEntity, T>();

TypeOfView = JsonConvert.SerializeObject(Activator.CreateInstance(typeof(T)));

}

protected CRUDProvider()

{

\_metadata = ControllerMetadatas[GetType()];

}

void IDisposable.Dispose()

{

Repository.Dispose();

base.Dispose();

}

#endregion

#region Execution

protected ActionContainer<TC> CreateContainer<TC>(TC value = default(TC))

{

return new ActionContainer<TC> { Value = value };

}

protected ActionContainer CreateContainer(object value = null)

{

return new ActionContainer { Value = value };

}

private HttpResponseMessage \_invoke<TResult>(HttpRequestMessage request, Func<ActionContainer<TResult>, TResult> action,

HttpStatusCode statusCode,

Action failAction, IEnumerable<Type> interfaces)

{

if (AuthService.Methods.Any())

foreach (var @interface in interfaces.Where(i => AuthService.Methods.Contains(i)))

{

//IEnumerable<string> values;

//request.Headers.TryGetValues(AuthService.HeaderKey, out values);

var result = AuthService.Auth(request);

if (result.Success) continue;

if (failAction != null) failAction();

return request.CreateResponse(HttpStatusCode.BadRequest, DataPresenterService.MediaTypeFormatter);

}

var container = CreateContainer<TResult>();

var actionResult = action(container);

if (!container.Success)

{

return request.CreateResponse(HttpStatusCode.BadRequest, container.Exceptions, DataPresenterService.MediaTypeFormatter);

}

return request.CreateResponse(statusCode, DataPresenterService.Result(actionResult),

DataPresenterService.MediaTypeFormatter);

}

private HttpResponseMessage \_invoke(HttpRequestMessage request, Func<ActionContainer, HttpStatusCode> action,

HttpStatusCode statusCode,

Action failAction, IEnumerable<Type> interfaces)

{

foreach (var @interface in interfaces.Where(i => AuthService.Methods.Contains(i)))

{

//IEnumerable<string> values;

//request.Headers.TryGetValues(AuthService.HeaderKey, out values);

var result = AuthService.Auth(request);

if (result.Success) continue;

if (failAction != null) failAction();

return request.CreateResponse(HttpStatusCode.Unauthorized, AuthService.FailMessage,

DataPresenterService.MediaTypeFormatter);

}

var container = CreateContainer();

var actionResult = action(container);

if (!container.Success)

{

return request.CreateResponse(HttpStatusCode.BadRequest, container.Exceptions, DataPresenterService.MediaTypeFormatter);

}

return request.CreateResponse(actionResult, DataPresenterService.MediaTypeFormatter);

}

private Task<HttpResponseMessage> \_invokeAsync<TResult>(HttpRequestMessage request,

Func<ActionContainer<TResult>, Task<TResult>> action, HttpStatusCode statusCode,

Action failAction, IEnumerable<Type> interfaces)

{

if (AuthService.Methods.Any())

foreach (var @interface in interfaces.Where(i => AuthService.Methods.Contains(i)))

{

var result = AuthService.Auth(request);

if (result.Success) continue;

if (failAction != null) failAction();

return Task.FromResult(request.CreateResponse(HttpStatusCode.Forbidden, result.Exceptions, DataPresenterService.MediaTypeFormatter));

}

var container = CreateContainer<TResult>();

return action(container)

.ContinueWith(task =>

{

if (!container.Success)

{

return request.CreateResponse(HttpStatusCode.BadRequest, container.Exceptions, DataPresenterService.MediaTypeFormatter);

}

return request.CreateResponse(statusCode, DataPresenterService.Result(task.Result),

DataPresenterService.MediaTypeFormatter);

});

}

private Task<HttpResponseMessage> \_invokeAsync(HttpRequestMessage request, Func<ActionContainer, Task<HttpStatusCode>> action,

HttpStatusCode statusCode,

Action failAction, IEnumerable<Type> interfaces)

{

foreach (var @interface in interfaces.Where(i => AuthService.Methods.Contains(i)))

{

//IEnumerable<string> values;

//request.Headers.TryGetValues(AuthService.HeaderKey, out values);

var result = AuthService.Auth(request);

if (result.Success) continue;

if (failAction != null) failAction();

return

Task.FromResult(request.CreateResponse(HttpStatusCode.Unauthorized, AuthService.FailMessage,

DataPresenterService.MediaTypeFormatter));

}

var container = CreateContainer();

return action(container)

.ContinueWith(task =>

{

if (!container.Success)

{

return request.CreateResponse(HttpStatusCode.BadRequest, container.Exceptions, DataPresenterService.MediaTypeFormatter);

}

return request.CreateResponse(statusCode, DataPresenterService.Result(task.Result),

DataPresenterService.MediaTypeFormatter);

});

}

protected HttpResponseMessage ValidateAndInvoke<TResult>(HttpRequestMessage request, Func<ActionContainer<TResult>, TResult> action,

HttpStatusCode statusCode,

Action failAction = null, params Type[] interfaces)

{

try

{

var disableAttrs = this.GetType().GetCustomAttributes<DisableRoutesAttribute>().ToArray();

if (!disableAttrs.Any())

{

return \_invoke(request, action, statusCode, failAction, interfaces);

}

if (interfaces.Any(Interface => !disableAttrs.Where(o => o.RouteInterface != Interface)

.Where(o => !Interface.GetInterfaces().Contains(o.RouteInterface))

.Any()))

{

if (failAction != null) failAction();

return request.CreateResponse(HttpStatusCode.NotFound, "HTTP Error 404 - Page Not Found",

DataPresenterService.MediaTypeFormatter);

}

return \_invoke(request, action, statusCode, failAction, interfaces);

}

catch (Exception e)

{

if (failAction != null) failAction();

return request.CreateResponse(HttpStatusCode.BadRequest, DataPresenterService.Result(e.Message),

DataPresenterService.MediaTypeFormatter);

}

}

protected HttpResponseMessage ValidateAndInvoke(HttpRequestMessage request, Func<ActionContainer, HttpStatusCode> action,

HttpStatusCode statusCode,

Action failAction = null, params Type[] interfaces)

{

try

{

var disableAttrs = this.GetType().GetCustomAttributes<DisableRoutesAttribute>().ToArray();

if (!disableAttrs.Any())

{

return \_invoke(request, action, statusCode, failAction, interfaces);

}

if (interfaces.Any(Interface => !disableAttrs.Where(o => o.RouteInterface != Interface)

.Where(o => !Interface.GetInterfaces().Contains(o.RouteInterface))

.Any()))

{

if (failAction != null) failAction();

return request.CreateResponse(HttpStatusCode.NotFound, "HTTP Error 404 - Page Not Found",

DataPresenterService.MediaTypeFormatter);

}

return \_invoke(request, action, statusCode, failAction, interfaces);

}

catch (Exception e)

{

if (failAction != null) failAction();

return request.CreateResponse(HttpStatusCode.BadRequest, DataPresenterService.Result(e.Message),

DataPresenterService.MediaTypeFormatter);

}

}

protected Task<HttpResponseMessage> ValidateAndInvokeAsync<TResult>(HttpRequestMessage request,

Func<ActionContainer<TResult>, Task<TResult>> action, HttpStatusCode statusCode,

Action failAction = null, params Type[] interfaces)

{

try

{

var disableAttrs = this.GetType().GetCustomAttributes<DisableRoutesAttribute>().ToArray();

if (!disableAttrs.Any())

{

return \_invokeAsync(request, action, statusCode, failAction, interfaces);

}

if (interfaces.Any(Interface => !disableAttrs.Where(o => o.RouteInterface != Interface)

.Where(o => !Interface.GetInterfaces().Contains(o.RouteInterface))

.Any()))

{

if (failAction != null) failAction();

return

Task.FromResult(request.CreateResponse(HttpStatusCode.NotFound,

"HTTP Error 404 - Page Not Found", DataPresenterService.MediaTypeFormatter));

}

return \_invokeAsync(request, action, statusCode, failAction, interfaces);

}

catch (Exception e)

{

if (failAction != null) failAction();

return

Task.FromResult(request.CreateResponse(HttpStatusCode.BadRequest,

DataPresenterService.Result(e.Message), DataPresenterService.MediaTypeFormatter));

}

}

protected Task<HttpResponseMessage> ValidateAndInvokeAsync(HttpRequestMessage request,

Func<ActionContainer, Task<HttpStatusCode>> action, HttpStatusCode statusCode,

Action failAction = null, params Type[] interfaces)

{

try

{

var disableAttrs = this.GetType().GetCustomAttributes<DisableRoutesAttribute>().ToArray();

if (!disableAttrs.Any())

{

return \_invokeAsync(request, action, statusCode, failAction, interfaces);

}

if (interfaces.Any(Interface => !disableAttrs.Where(o => o.RouteInterface != Interface)

.Where(o => !Interface.GetInterfaces().Contains(o.RouteInterface))

.Any()))

{

if (failAction != null) failAction();

return

Task.FromResult(request.CreateResponse(HttpStatusCode.NotFound,

"HTTP Error 404 - Page Not Found", DataPresenterService.MediaTypeFormatter));

}

return \_invokeAsync(request, action, statusCode, failAction, interfaces);

}

catch (Exception e)

{

if (failAction != null) failAction();

return

Task.FromResult(request.CreateResponse(HttpStatusCode.BadRequest,

DataPresenterService.Result(e.Message), DataPresenterService.MediaTypeFormatter));

}

}

#endregion

#region General

protected static Linker<TAction> \_chainBuilder<TAction>(Type @interface,

ControllerMetadata<T, TEntity> controllerMetadata)

{

Func<TAction, TAction> func = null;

var chain = func.X();

controllerMetadata.AdditionalMethods[@interface]

.Cast<Func<TAction, TAction>>()

.ForEach(o => chain = chain > o);

controllerMetadata.MethodChains[@interface] = chain;

return chain;

}

protected TAction \_chainInvoker<TAction>(Type @interface, TAction results)

{

return (\_metadata.MethodChains[@interface] as Linker<TAction>) > results;

}

protected virtual IQueryable<TEntity> \_search(string filterString)

{

Dictionary<string, Func<Expression, Expression, BinaryExpression>> separetorPairs =

new Dictionary<string, Func<Expression, Expression, BinaryExpression>>

{

{"&&", Expression.And},

{"||", Expression.Or}

};

var separators = new string[] {"&&", "||"};

var filters = filterString.Split(separators, StringSplitOptions.None)

.Select(o => o.Trim());

Queue<string> separetorList = new Queue<string>();

var reg = new Regex(@"(\&\&|\|\|)");

var matches = reg.Matches(filterString);

foreach (Match match in matches)

{

separetorList.Enqueue(match.Value);

}

ParameterExpression pe = Expression.Parameter(typeof (TEntity), "o");

var enumerable = filters as string[] ?? filters.ToArray();

Expression e1 = ExpressionBuilder(enumerable.First(), pe);

enumerable.Skip(1)

.ForEach(f => e1 = separetorPairs[separetorList.Dequeue()](e1, ExpressionBuilder(f, pe)));

MethodCallExpression whereCallExpression = Expression.Call(

typeof (Queryable),

"Where",

new Type[] {DataProvider.ElementType},

DataProvider.Expression,

Expression.Lambda<Func<TEntity, bool>>(e1, pe));

IQueryable<TEntity> results = DataProvider.Provider.CreateQuery<TEntity>(whereCallExpression);

return results;

}

private ActionContainer<IQueryable<TEntity>> \_\_search(string filterString)

{

return \_chainInvoker(InterfaceTypes[Interface.Get], CreateContainer(\_search(filterString)));

}

#endregion

#region Helpers

private static void \_initControllerMetadata(Type type)

{

type.Apply(BuildControllerMetadata)

.Apply(BuildChains)

.Apply(BuildAdditionalMethodChains)

.Apply(BuildCustomServiceChains);

//BuildChains(BuildAdditionalMethodChains(BuildCustomServiceChains(BuildControllerMetadata(type))));

}

private static void BuildAdditionalMethodChains(Type controller)

{

MethodInjectionAttribute tempAttr = null;

controller.GetMethods()

.Where(m => (tempAttr = m.GetCustomAttribute<MethodInjectionAttribute>(true)) != null)

.ForEach(x =>

{

if (!ControllerMetadatas[controller].AdditionalMethods.ContainsKey(tempAttr.Interface))

{

ControllerMetadatas[controller].AdditionalMethods.Add(tempAttr.Interface,

new LinkedList<Delegate>());

}

var parameters = x.GetParameters()

.Select(p => Expression.Parameter(p.ParameterType, p.Name))

.ToArray();

var call = Expression.Call(null, x, parameters);

var m = Expression.Lambda(call, parameters).Compile();

ControllerMetadatas[controller].AdditionalMethods[tempAttr.Interface].AddLast(m);

});

}

private static void BuildCustomServiceChains(Type controller)

{

//MethodInjectionAttribute tempAttr = null;

controller.GetCustomAttributes<InjectionServiceAttribute>()

.ForEach(x =>

ControllerMetadatas[controller]

.CustomeServicesChain[x.ServiceInterface]

.Compose(x.Service

.GetMethod("Invoke")

.CreateBoundedDelegate<Func<ActionContainer, ActionContainer>>()));

}

private static Dictionary<Type, LinkedList<Delegate>>

BuildAdditionalMethods(Dictionary<Type, LinkedList<Delegate>> dictionary)

{

Enum.GetValues(typeof(Interface))

.As<IEnumerable<int>>()

.ForEach(i => dictionary.Add(InterfaceTypes[(Interface)i], new LinkedList<Delegate>()));

return dictionary;

}

private static Dictionary<Type, Func<ActionContainer, ActionContainer>>

BuildCustomServiceDictionary(Dictionary<Type, Func<ActionContainer, ActionContainer>> dictionary)

{

Enum.GetValues(typeof(Interface))

.As<IEnumerable<int>>()

.ForEach(i => dictionary.Add(InterfaceTypes[(Interface)i], container => container));

return dictionary;

}

private static void BuildChains(Type type)

{

//if (type.BaseType == null) return type;

var m = type.BaseType.GetMethod("\_chainBuilder", BindingFlags.NonPublic | BindingFlags.Static);

foreach (var methodChain in InterfacesChainTypes)

{

MethodInfo genericMethod = m.MakeGenericMethod(InterfacesChainTypes[methodChain.Key]);

genericMethod.Invoke(null, new object[] {methodChain.Key, ControllerMetadatas[type]});

}

//return type;

}

private static void BuildControllerMetadata(Type type)

{

if (ControllerMetadatas.ContainsKey(type)) return;

var tuple = new ControllerMetadata<T, TEntity>

{

AdditionalMethods = BuildAdditionalMethods(new Dictionary<Type, LinkedList<Delegate>>()),

CustomeServicesChain = BuildCustomServiceDictionary(new Dictionary<Type, Func<ActionContainer, ActionContainer>>()),

MethodChains = InitChains(new Dictionary<Type, object>())

};

ControllerMetadatas.Add(type, tuple);

}

private static Expression ExpressionBuilder(string s, ParameterExpression par)

{

Func<Expression, Expression, BinaryExpression> SmallerThan =

(expression, expression1) => Expression.GreaterThan(expression1, expression);

Func<Expression, Expression, BinaryExpression> SmallerThanOrEqual =

(expression, expression1) => Expression.GreaterThanOrEqual(expression1, expression);

var splitters = new[] {"==", ">", "<", ">=", "<=", "!="};

Dictionary<string, Func<Expression, Expression, BinaryExpression>> splitterPairs =

new Dictionary<string, Func<Expression, Expression, BinaryExpression>>

{

{"==", Expression.Equal},

{"!=", Expression.NotEqual},

{">", Expression.GreaterThan},

{">=", Expression.GreaterThanOrEqual},

{"<", SmallerThan},

{"<=", SmallerThanOrEqual}

};

var p = s.Split(splitters, StringSplitOptions.None);

var pair = new {Key = p.First().Trim(), Value = p.Last().Trim()};

var splitter = s.Replace(pair.Key, string.Empty).Replace(pair.Value, string.Empty).Trim();

Expression left = Expression.Property(par, typeof (TEntity).GetProperty(pair.Key));

//GetMethod("ToLower", System.Type.EmptyTypes));

Expression right;

int value;

right = int.TryParse(pair.Value, out value) ? Expression.Constant(value) : Expression.Constant(pair.Value);

return splitterPairs[splitter](left, right);

}

private static Dictionary<Type, object>

InitChains(Dictionary<Type, object> dictionary)

{

dictionary.Add(InterfaceTypes[Interface.Get], null);

dictionary.Add(InterfaceTypes[Interface.Patch], null);

dictionary.Add(InterfaceTypes[Interface.Put], null);

dictionary.Add(InterfaceTypes[Interface.Delete], null);

dictionary.Add(InterfaceTypes[Interface.Post], null);

dictionary.Add(InterfaceTypes[Interface.Search], null);

return dictionary;

}

#endregion

#region Services

#endregion

}

}