Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
2020

A biblioteca **Webao** (*Web Access Object*) inspirada no conceito de *Data Access Object* para Web -- https://en.wikipedia.org/wiki/Data_access_object. Os *web access objects* disponibilizam métodos que retornam **objectos de domínio** (e.g. Student, Account, Artist, Track, Team, League, etc..) escondendo os detalhes de acesso à fonte de dados (e.g. base de dados).

A biblioteca **Webao** permite construir *web access objects* com métodos que retornam objectos de domínio com base na informação obtida de uma Web API Restful. Esta ideia é inspirada nos objectivos da biblioteca Retrofit. De forma simplificada e informal, uma Web API Restful pode ser entendida como uma API acedida via HTTP que retorna dados em formato JSON (https://www.json.org/). As classes `WebaoArtist` e `WebaoTrack` exemplificam dois tipos de *web access objects* com métodos para obtenção de artistas e músicas (i.e. `Artist` e `Track`).

A informação devolvida por estes *web access objects* é obtida neste exemplo a partir da API Restfull Last.fm. Os métodos destes *data acess objects* acedem aos seguintes URLs (deverá registar-se na Last.fm Web API e obter uma `api_key` para realizar os pedidos exemplificados):

- `GetInfo("muse")` -- http://ws.audioscrobbler.com/2.0/?method=artist.getinfo&artist=muse&api_key=***&format=json
- `Search("muse")` -- http://ws.audioscrobbler.com/2.0/?method=artist.search&format=json&artist=muse&api_key=***
- `GeoGetTopTracks("australia")` -- http://ws.audioscrobbler.com/2.0/?method=geo.gettoptracks&country=australia&api_key=***&format=json

Para que as instâncias destas classes realizem os pedidos exemplificados, estas devem ser definidas e anotadas da seguinte forma:

```
[BaseUrl("http://ws.audioscrobbler.com/2.0/")]
[AddParameter("format", "json")]
[AddParameter("api_key", "************")]
public class WebaoArtist : AbstractAccessObject
{
  public WebaoArtist(IRequest req) : base(req) {}

  [Get("?method=artist.getinfo&artist={name}")]
  [Mapping(typeof(DtoArtist), ".Artist")]
  public Artist GetInfo(string name) {
      return (Artist)Request(name);
  }
  [Get("?method=artist.search&artist={name}")]
  [Mapping(typeof(DtoSearch),
".Results.ArtistMatches.Artist")]
  public List<Artist> Search(string name) {
      return (List<Artist>)Request(name);
  }
}
```

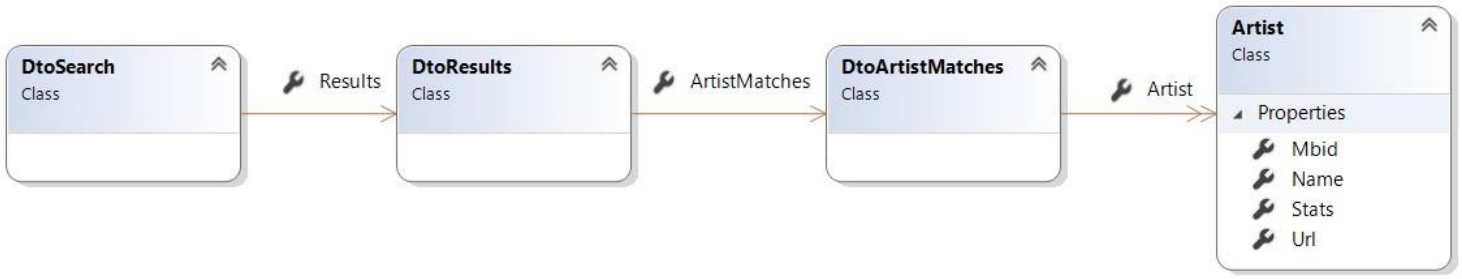
```
[BaseUrl("http://ws.audioscrobbler.com/2.0/")]
[AddParameterAttribute("format", "json")]
[AddParameterAttribute("api_key", "************")]
public class WebaoTrack : AbstractAccessObject
{
  public WebaoTrack(IRequest req) : base(req) { }


[Get("?method=geo.gettoptracks&country={country}")]
  [Mapping(typeof(DtoGeoTopTracks),
".Tracks.Track")]
  public List<Track> GeoGetTopTracks(string country)
{
      return (List<Track>) Request(country);
  }
}
```

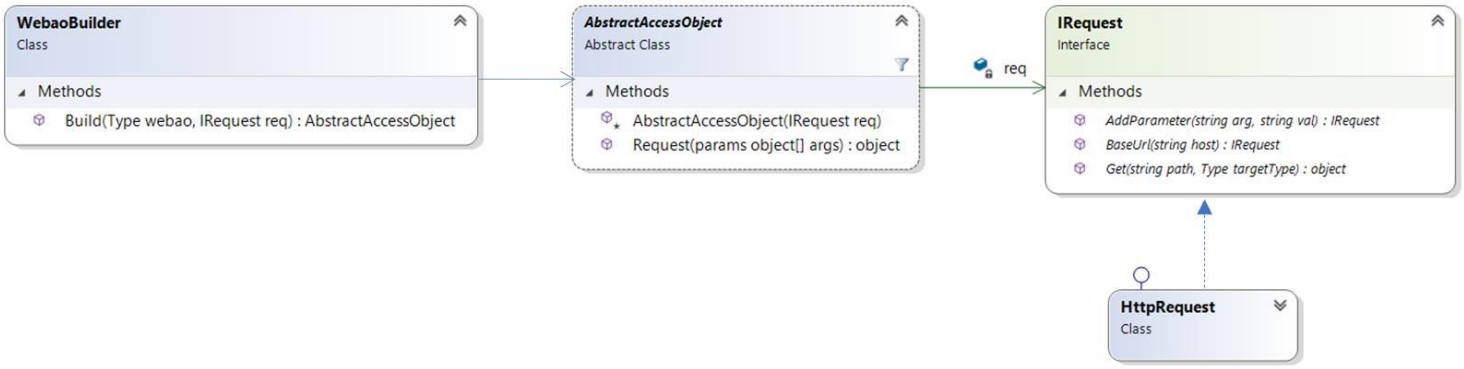
As anotações têm o seguinte papel:

- `BaseUrl` - DNS e caminho a incluir em todos os pedidos.
- `AddParameter` - *query-string parameter* a adicionar a todos os pedidos.
- `Get` -- especifica o path a concatenar com o `BaseUrl` para formar o URL do pedido.
- `Mapping` - especifica: 1) o tipo para o qual deve ser convertido a resposta JSON (e.g. `typeof(DtoArtist)`) e 2) o grafo de propriedades que têm que ser acedidas para obter o resultado a retornar.

Por exemplo no caso do método `search()` o caminho `.Results.ArtistMatches.Artist` especifica as propriedades que devem ser percorridas no grafo da imagem seguinte desde `DtoSearch` até chegar a `List<Artist>`:



Cabe à classe `WebaoBuilder` instanciar os *web access objects* (i.e. `WebaoArtist` e `WebaoTrack`) e processar as informações anotadas nessas classes **configurando devidamente estas instâncias bem como o objecto `IRequest`**.
De notar que os *data access objects* requerem uma instância de `IRequest` através da qual o seu tipo base `AbstractAccessObject` realizará os pedidos HTTP à Restful API, conforme ilustra o diagrama de classes seguinte.



O método `Get` da instância de `IRequest` recebe como parâmetro o URL do pedido (`path`) e o tipo (`targetType`) para o qual será convertido a resposta JSON. Ou seja, no exemplo anterior do método `search()` o parâmetro `targetType` receberá o tipo presente na anotação correspondente a `typeof(DtoSearch)`.

A listagem seguinte ilustra o exemplo de como realizar pedidos através de `HttpRequest` às rotas de pesquisa de artistas (`method=artist.search`) e *top tracks* (`method=geo.gettoptracks`) do Last.fm Web API.

```
HttpRequest req = new HttpRequest();
req.BaseUrl("http://ws.audioscrobbler.com/2.0");
req.AddParameter("format", "json");
req.AddParameter("api_key", "************");
/*
 * Search for band Muse
 */
DtoSearch dto = (DtoSearch) req.Get(
    "?method=artist.search&artist=muse",
    typeof(DtoSearch));
```

```
Assert.AreEqual("Muse", dto.Results.ArtistMatches.Artist[0].Name);
Assert.AreEqual("Mouse on Mars", dto.Results.ArtistMatches.Artist[3].Name);
/*
 * Get top tracks from Australia
 */
DtoGeoTopTracks aus = (DtoGeoTopTracks) req.Get(
    "?method=geo.gettoptracks&country=australia",
    typeof(DtoGeoTopTracks));
List<Track> tracks = aus.Tracks.Track;
Assert.AreEqual("The Less I Know the Better", tracks[0].Name);
Assert.AreEqual("Mr. Brightside", tracks[1].Name);
Assert.AreEqual("The Killers", tracks[1].Artist.Name);
```

A classe `WebaoDynBuilder` tem o mesmo comportamento de `WebaoBuilder`, mas que pretende **simplificar** a definição e obter **melhor desempenho** dos *web access objects*.
Para tal os *web access objects* são definidos por interfaces, conforme o exemplo seguinte:

```
[BaseUrl("http://ws.audioscrobbler.com/2.0/")]
[AddParameter("format", "json")]
[AddParameter("api_key", "************")]
public interface WebaoDynArtist
{
    [Get("?method=artist.getinfo&artist={name}")]
    [Mapping(typeof(DtoArtist), ".Artist")]
    Artist GetInfo(string name);

    [Get("?method=artist.search&artist={name}&page={page}")]
    [Mapping(typeof(DtoSearch), ".Results.ArtistMatches.Artist")]
    List<Artist> Search(string name, int page);
}
```

```
Request req = new HttpRequest();
WebaoDynArtist webaoDyn =
  (WebaoDynArtist) WebaoDynBuilder
    .Build(typeof(WebaoDynArtist), req);
```

`WebaoDynBuilder` gera em tempo de execução (dinamicamente) uma implementação de uma classe que implementa a interface especificada. **ATENÇÃO** a classe gerada dinamicamente:

1. **NÃO** faz operações de reflexão.
2. **NÃO** estende de `AbstractAccessObject`.

Por exemplo, a obtenção das propriedades `Results`, `ArtistMatches` e `Artist` que é feita para o método `Search()` por via de reflexão em `AbstractAccessObject` é feita com acesso directo às respectivas propriedades em código IL gerado dinamicamente por via de `System.Reflection.Emit`.
APENAS é usada reflexão na própria classe `WebaoDynBuilder` e suas auxiliares e **NÃO** nas classes que este gera dinamicamente.

O projecto **WebaoBench** compara o desempenho entre as implementações Reflect e Emit.

O *custom mattribute* `Mapping` guarda o método responsável por extrair o valor do *dto* (e.g. uma função `GetArtistsList` que está previamente definida em `DtoSearch`, ou noutra classe qualquer). Esta utilização é alternativa à do segundo parâmetro do construtor de `Mapping` referente ao grafo de propriedades que têm que ser acedidas para obter o resultado a retornar. Assim a classe `Mappingattribute` tem dois construtores:

```
[AttributeUsage(AttributeTargets.Method, AllowMultiple = false)]
public class MappingAttribute : Attribute
{
    ...
    public MappingAttribute(Type dto, string path) {...}
    public MappingAttribute(Type dto) {...}
    public string With { set { ... } }
    ...
```

O utilizador poderá usar o *custom attribute* `Mapping` da seguinte forma:

```
[Get("...")]
[Mapping(typeof(DtoSearch), With = "Webao.Test.Dto.DtoSearch.GetArtistsList")]
List<Artist> Search(string name, int page);
```

Neste caso `GetArtistsList` é o nome do método da classe `DtoSearch` que faz: `return this.Results.ArtistMatches.Artist;`.

A biblioteca **Webao** oferece ainda uma forma **alternativa** à configuração por *custom attributes*.

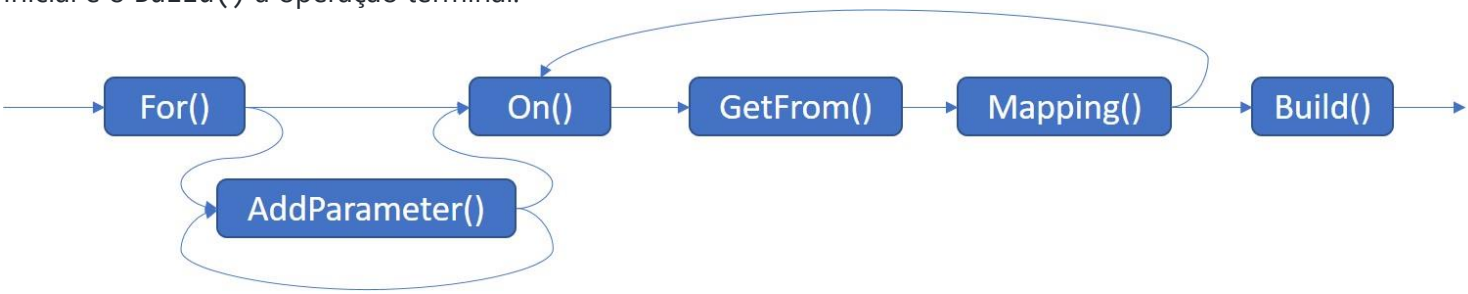
Assim a interface que especifica um Webao (e.g. `WebaoDynTrack`, `WebaoDynArtist, etc`) pode ser definida como no exemplo seguinte:

```
interface WebaoDynArtist {
    Artist GetInfo(string name);
    List<Artist> Search(string name, int page);
}
```

A partir da interface anterior pode ser gerada uma instância de uma implementação dinâmica através de:

```
WebaoDynArtist webao = WebaoDynBuilder
    .For<WebaoDynArtist>("http://ws.audioscrobbler.com/2.0/")
    .AddParameter("format", "json")
    .AddParameter("api_key", "************")
    .On("GetInfo")
    .GetFrom("?method=artist.getinfo&artist={name}")
    .Mapping<DtoArtist>(dto => dto.Artist)
    .On("Search")
    .GetFrom("?method=artist.search&artist={name}&page={page}")
    .Mapping<DtoSearch>(dto => dto.Results.ArtistMatches.Artist)
    .Build(req);
```

Os métodos anteriores podem ser encadeados pela seguinte ordem sendo que o `For()` será sempre o método inicial e o `Build()` a operação terminal:



Destes métodos o único que reside no `WebaoDynbBuilder` é o `For()`, sendo que os restantes são definidos em classes auxiliares.

Os serviços de Webao disponibilizam agregados de objectos de domínio. Estes serviços podem ser customizados com uma instância de uma implementação de `IRequest` com a qual são criadas as instâncias dos Webao auxiliares. Por exemplo:

```
class ServiceTracks
{
    private readonly WebaoTrack webao;
    public ServiceTracks() : this(new HttpRequest()) {}
    public ServiceTracks(IRequest req) {....}
    public IEnumerable<Track> TopTracksFrom(string country) {...}
}
```

Neste caso `TopTracksFrom()` retorna uma sequência **LAZY** com todas as músicas do *top tracks* para o país dado, i.e. `country`. Note que este método deve juntar numa única sequência *lazy* as músicas obtidas das várias páginas de resultados da Web Api da Last.fm.

A mesma abordagem é seguida para outros métodos de serviços que agreguem numa única sequência os resultados paginados de outras Web API.