# Genericita

FE = FilterElement

Ve zmenovadle chtel Filip abych predelal FilterElement tak aby bral jako genericky argument ValueDataContract. Nakonec to musel udelat sam a to nasledovne:

* Pro FE udelal rozhrani. Vsude ve VM kde doted byl odkaz na FE ted bude rozhrani IfilterElement

V rozhrani jsou definovane vsechny public property z FE. Zajimave je ze predelal SelectedValues coz byl List<ValueDataContract> na IList.

Field je List: private List<T> m\_selectedValues = new List<T>();

* FE je ted genericky:

public class FilterElement<T> : PropertyChangedBase, IFilterElement

where T : ValueDataContract, new()

Pri vytvareni instance bude potreba mu predat nejaky genericky argument. Klauzule where rika jakeho typu argument bude.

Omezení konstruktorem **new()** zajistí, že typ použitý jako typový argument pro T má veřejný bezparametrový konstruktor. Tím povolí generické třídě použití new T() k vytvoření instance jejího typu. Omezení typových parametrů je nutné používat obezřetně, neboť také můžeme omezit možné vhodné využití generického typu.

[‎1/‎25/‎2018 9:38 AM] Peter Hlavenka:

nejvic me udivuje, ze se da vytvorit instance abstraktni tridy

Filip Čálek:

kde?

Peter Hlavenka:

d(new T {Name = m\_filter});

] Filip Čálek:

no ale to se nevytvari instance abstraktni tridy

where T : ValueDataContract, new()

Peter Hlavenka:

ValueDC je abstract co dela teda new ()?

Filip Čálek:

tim new() prave rikam,ze ocekavam neabstraktni tridu cili nejakeho jeho potomka

a pak muzu udelat presne toto

d(new T {Name = m\_filter});

Dalsi fazi bylo vytvoreni generickeho requestu ktery chce jako parameter nejakeho potomka abstraktni tridy ValueDataContract:

public abstract class ValueDataContract

Aby jsme mohli predavat potomky requestu ma kazda instance FE svuj vlastni objekt ktery dedi od ValueDataContractu:

public class OwnerDataContract : ValueDataContract

Genericky argument FE dostava pri tvorbe instance FE:

Filters.Add(new FilterElement<**MotiveVersionNameDataContract**>(m\_publisher, MultiComboOpened, IsNewValuesFilterPart, CreateParameters,));

Filters.Add(new FilterElement<**OwnerDataContract**>(m\_publisher, MultiComboOpened, IsNewValuesFilterPart, CreateParameters));

Filters.Add(new FilterElement<**CompanyBrandDataContract**>(m\_publisher, MultiComboOpened, IsNewValuesFilterPart, CreateParameters,));

**Reflexe:**  Kolekce Filters<IFilterElement> obsahuje kolekci rozhrani jenz schovava FE. Aby bylo mozne zjistit jakeho typu je genericky argument v konkretnim FE, musime se zeptat :

MotiveNames = Filters.FirstOrDefault(d => d.SelectedValues != null && **d.GetType().IsAssignableFrom(typeof(MotiveVersionNameDataContract)))?.**SelectedValues.Cast<MotiveVersionNameDataContract>().Select(d => d.Name).ToList(),

**Cast: A nasledne musime objekty z kolekce IList SelectedValues precastit na typ ktery potrebujeme. SelectedValues ma totiz uvnitr List<T> kde T je pro kazdy FE nejaky typ ValueDataContractu. Jaky, to prave urcuje genericky argument FE.**

MotiveNames = Filters.FirstOrDefault(d => d.SelectedValues != null && d.GetType().IsAssignableFrom(typeof(MotiveVersionNameDataContract)))?.**SelectedValues.Cast<MotiveVersionNameDataContract>().**Select(d => d.Name).ToList(),

# **Genericka metoda:**

**Do hranatych zavorek pridavame zastupny znak <T> jak k navratovemu typu (pokud je genericky) tak za nazev metody.**

public List**<T1>** CreateValueComboItems**<T1>()** where T1 : ValueDataContract

{

//**todo genericky request**

return ValueComboItems.Cast<T1>().ToList();

}

**Konstruktor uz <T> nedeklaruje:**

I kdyz deklarace tridy ma zastupny znak T:

public class FilterElement<T> : PropertyChangedBase, IFilterElement where T : ValueDataContract, new()

Konstruktor je bez nej:

public FilterElement(IClientToServicePublisher publisher, Func<GetComboBoxValuesRequest<T>, List<T>> valueComboPressed)

Pro vytvoreni instance se musi zadat genericky argument:

var neco = new FilterElement<**MotiveVersionNameDataContract**>(m\_publisher, MultiComboOpened,));

# **Genericky request (zmenovadlo):**

Coz teda znamena ze kdyz chci vytvorit genericky request tak request musi pozadovat abstraktni typ ValueDataContract . Ten mu samozrejme davat nebudu jelikoz jeho instanci nemuzu vytvorit. Misto toho mu da nejakeho jeho potomka. Potomek je totiz taky ValueDataContract a tim je podminka splnena.

Deklarace :

public class GetComboBoxValuesRequest<T> : ChangingRequestBase<ListResponse<T>> where T: ValueDataContract

Konstruktor:

public GetComboBoxValuesRequest( nejake parametry ) { … }

Vytvoreni instance :

var request = new GetComboBoxValuesRequest<T>(nejake parametry);

var request = new GetComboBoxValuesRequest<OwnerDataContract>(nejake parametry);

Protoze ve zmenovadle vytvarim instanci ve FE kde uz mam jasno co je T predavam jen T. Jinak by to mohlo vypadat takhle:

var request = new GetComboBoxValuesRequest<OwnerDataContract>(nejake parametry);

**Genericky request neumime automaticky subscribnout:**

**Musime vytvorit rucniho podepisovatele, ktery bude mit private propertu typu IserviceActionSubscriber, jehoz instanci si nechame vywindsorovat v konstruktoru a nasledne v metode Subscribe si podepiseme genericky request pro vsechny argumenty ktere muze mit (request je jen jeden ale muze mit deset ruznych argumentu, teda budeme mit deset podpisu ):**

public class CustomServiceActionSubscriber

{

private IServiceActionSubscriber ServiceActionSubscriber { get; }

public CustomServiceActionSubscriber(IServiceActionSubscriber serviceActionSubscriber)

{

ServiceActionSubscriber = serviceActionSubscriber;

}

public void Subscribe()

{

ServiceActionSubscriber.Subscribe<GetComboBoxValuesRequest<**OwnerDataContract**>, ListResponse<OwnerDataContract>>(typeof(GetOwnersByParametersAction));

ServiceActionSubscriber.Subscribe<GetComboBoxValuesRequest<**ProductBrandDataContract**>,ListResponse<ProductBrandDataContract>>(typeof(GetProductBrandsByParametersAction));

}

}

**Request je jeden ale servisnich akci k nemu je tolik, kolik muze mit generickych argumentu.**

**PS: Hlaseni ze servisni akce musi by IserviceAction je proto, ze jsem zapomnel v serv. Akci vsude typ ValueDatacontract misto toho aby tam byl jeho potomek.**

# **Potreboval jsem pro kazdy filter, pokud ma vybrany nejaky Item zjistit, jestli se tento item shoduje s propertou na wrapperu. Volam 10x gen. metodu CheckFilter ktere predam pokazde jiny argument.**

private void CheckConditions(ComposedWrapper wrapper)

{

wrapper.AllConditionsMet = true;

CheckFilter<FilterElement<MotiveNameDataContract>>(wrapper, wrapper.IsMotiveNameSearched);

CheckFilter<FilterElement<OwnerDataContract>>(wrapper, wrapper.IsOwnerSearched);

CheckFilter<FilterElement<CompanyBrandDataContract>>(wrapper, wrapper.IsCompanyBrandSearched);

CheckFilter<FilterElement<ProductBrandDataContract>>(wrapper, wrapper.IsProductBrandSearched);

CheckFilter<FilterElement<ProductDetailDataContract>>(wrapper, wrapper.IsProductDetailSearched);

CheckFilter<FilterElement<CategoryDataContract>>(wrapper, wrapper.IsCategorySearched);

CheckFilter<FilterElement<RoleDataContract>>(wrapper, wrapper.IsRoleSearched);

CheckFilter<FilterElement<MarketDataContract>>(wrapper, wrapper.IsMarketSearched);

CheckFilter<FilterElement<GenderDataContract>>(wrapper, wrapper.IsGenderSearched);

CheckFilter<FilterElement<PlatformDataContract>>(wrapper, wrapper.IsPlatformSearched);

}

**Argument se v metode porovna, vybere se spravny filter, a podle podminky zmenime propertu AllConditionsMet. (nebo taky nezmenime).**

private void CheckFilter<T>(ComposedWrapper wrapper, bool searched) where T : IFilterElement

{

if (Filters.First(d => d.GetType() == typeof(T)).SelectedValues.Any() && (searched == false || wrapper.AllConditionsMet == false))

{

wrapper.AllConditionsMet = false;

}

}

# **Pokud chci primo zjistit typ generickeho argumentu pouziju :**

var motiveNameFilter = newValues.First(d => d.GetType().GetGenericElementType().IsAssignableFrom(typeof(MotiveNameDataContract)));

**(ted uz nemusim davat typeof(FilterElement<MotiveNameDataContract>)) (zmizelo to FilterElement)**

# Multiple where constraint

private List<IPricedItemWrapper> LoadPricedItems<T, T1>(List<int> inputIds, T loader)

where T : IPricedItemWrapperLoader<T1>

where T1: IPricedItemWrapper

{

var result = loader.Load(inputIds).Cast<IPricedItemWrapper>().ToList();

return result;

}

**Volani metody je pak :**

LoadPricedItems<objekt gen. argumentu T, objekt gen. argumentu T1 > (objekt podle parametru metody , objekt podle parametru metody);

# **Factory aneb jak predat genericky argument jako parametr metody :**

**Volani:**

var response = m\_publisher.Publish(GetRenameMotivletPropertyRequest(SelectedVariable, Id, newName));

**Metoda:**

private RenameMotivletPropertyRequest<T, T1> GetRenameMotivletPropertyRequest<T, T1>(T parameter, T1 id, string name) where T:ValueDataContract

{

return new RenameMotivletPropertyRequest<T, T1>(id, name);

}

**Pricemz request je genericky a bere T jako svuj genericky argument:**

public class RenameMotivletPropertyRequest<T, T1>: AuditableRequestBase<DoneResponse> where T: ValueDataContract