

Back-end разработка

Бизнес-логика объектов

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Бизнес-логика объектов	4
Механизм событийного слоя Entity	4
Асинхронность в событийном слое Entity	7

Бизнес-логика объектов



Способы настройки событий (сохранение, изменение, удаление и т. д.) объекта:

- На уровне интерфейса приложения через событийные подпроцессы дизайнера объекта.
- На уровне back-end части приложения с использованием средств разработки.

Механизм событийного слоя Entity

Назначение событийного слоя `Entity` — настройка обработчиков событий объекта на уровне back-end части приложения с использованием средств разработки. Приложение поддерживает работу только с обработчиками, которые определены в основной конфигурации и сборках файлового контента. Не поддерживаются обработчики из внешних сборок.

Важно. Механизм событийного слоя `Entity` срабатывает после выполнения событийных подпроцессов объекта.

События объекта, которые могут быть обработаны с использованием событийного слоя:

- `OnDeleted` — после удаления записи.
- `OnInserted` — после добавления записи.
- `OnInserting` — перед добавлением записи.
- `OnDeleting` — перед удалением записи.
- `OnSaved` — после сохранения записи.
- `OnSaving` — перед сохранением записи.
- `OnUpdated` — после обновления записи.
- `OnUpdating` — перед обновлением записи.

Составляющие, которые реализуют механизм событийного слоя `Entity`:

- `BaseEntityEventListener` — предоставляет методы-обработчики событий сущности.
- `EntityAfterEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется после возникновения события.
- Класс `EntityBeforeEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется до возникновения события.
- Атрибут `EntityEventListener` — регистрация слушателя.

Класс BaseEntityEventListener

Назначение класса `Terrasoft.Core.Entities.Events.BaseEntityEventListener` — предоставляет методы-обработчики различных событий сущности, которые представлены в таблице ниже.

Методы-обработчики событий сущности

Метод-обработчик	Описание метода
<code>OnDeleted(object sender, EntityAfterEventArgs e)</code>	Обработчик события после удаления записи.
<code>OnDeleting(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед удалением записи.
<code>OnInserted(object sender, EntityAfterEventArgs e)</code>	Обработчик события после добавления записи.
<code>OnInserting(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед добавлением записи.
<code>OnSaved(object sender, EntityAfterEventArgs e)</code>	Обработчик события после сохранения записи.
<code>OnSaving(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед сохранением записи.
<code>OnUpdated(object sender, EntityAfterEventArgs e)</code>	Обработчик события после обновления записи.
<code>OnUpdating(object sender, EntityBeforeEventArgs e)</code>	Обработчик события перед обновлением записи.

Параметры методов класса `BaseEntityEventListener` :

- `sender` — ссылка на экземпляр объекта, который генерирует событие.
- `e` — аргументы события. Может принимать значения `EntityAfterEventArgs` (после события) или `EntityBeforeEventArgs` (перед событием).

Последовательность вызова методов-обработчиков событий приведена в таблице ниже.

Последовательность вызова методов-обработчиков

Создание объекта	Изменение объекта	Удаление объекта
<code>OnSaving()</code>	<code>OnSaving()</code>	<code>OnDeleting()</code>
<code>OnInserting()</code>	<code>OnUpdating()</code>	<code>OnDeleted()</code>
<code>OnInserted()</code>	<code>OnUpdated()</code>	
<code>OnSaved()</code>	<code>OnSaved()</code>	

Экземпляр `UserConnection` в обработчиках событий необходимо получать из параметра `sender`. Пример получения `UserConnection` приведен ниже.

Пример получения `UserConnection`

```
[EntityEventListener(SchemaName = "Activity")]
public class ActivityEntityEventListener : BaseEntityEventListener
{
    public override void OnSaved(object sender, EntityAfterEventArgs e) {
        base.OnSaved(sender, e);
        var entity = (Entity) sender;
        var userConnection = entity.UserConnection;
    }
}
```

Класс `EntityAfterEventArgs`

Назначение класса `Terrasoft.Core.Entities.EntityAfterEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется после возникновения события.

Свойства класса `EntityAfterEventArgs`:

- `ModifiedColumnValues` — коллекция измененных колонок.
- `PrimaryColumnValue` — идентификатор записи.

Класс `EntityBeforeEventArgs`

Назначение класса `Terrasoft.Core.Entities.EntityBeforeEventArgs` — предоставляет свойства с аргументами метода-обработчика, который выполняется до возникновения события.

Свойства класса `EntityBeforeEventArgs`:

- `KeyValue` — идентификатор записи.
- `IsCanceled` — позволяет отменить дальнейшее выполнение события.
- `AdditionalCondition` — позволяет дополнительно описать условия фильтрации сущности перед действием.

Атрибут `EntityEventListener`

Назначение атрибута `EntityEventListener` — регистрация слушателя. Слушатель может быть связан со всеми объектами (`IsGlobal = true`) или с конкретным объектом (например, `SchemaName = "Contact"`). Один класс-слушатель можно помечать множеством атрибутов для определения необходимого набора "прослушиваемых" сущностей.

Настроить обработчик события объекта

Чтобы **настроить обработчик** события объекта (наследника класса `Entity`):

1. Создайте класс-наследник класса `BaseEntityEventListener`.
2. Декорируйте класс атрибутом `[EntityEventListener]` с указанием имени сущности, для которой необходимо выполнить подписку событий.
3. Переопределите метод-обработчик настраиваемого события.

Пример переопределения метода-обработчика события

```
/* Слушатель событий сущности "Активность". */
@EntityEventListener(SchemaName = "Activity")]
public class ActivityEntityEventListener : BaseEntityEventListener
{
    /* Переопределение обработчика события сохранения сущности. */
    public override void OnSaved(object sender, EntityAfterEventArgs e) {
        /* Вызов родительской реализации. */
        base.OnSaved(sender, e);
        /* Дополнительные действия.
        ... */
    }
}
```

Асинхронность в событийном слое Entity

Дополнительная бизнес-логика на объекте продолжительна во времени и выполняется последовательно. Это замедляет производительность front-end части приложения, например, при сохранении или изменении сущности. Для решения этой проблемы разработан **механизм асинхронного выполнения операций**, который основан на событийном слое `Entity`.

Составляющие, которые реализуют асинхронность в событийном слое `Entity`:

- Интерфейс `IEntityEventAsyncExecutor` — предоставляет метод для асинхронного выполнения операций.
- Интерфейс `IEntityEventAsyncOperation` — предоставляет метод для запуска асинхронной операции.
- Класс `EntityEventAsyncOperationArgs` — экземпляры класса используются в качестве аргумента для передачи в асинхронную операцию.

Интерфейс IEntityEventAsyncExecutor

Назначение интерфейса `Terrasoft.Core.Entities.AsyncOperations.Interfaces.IEntityEventAsyncExecutor` — предоставляет метод для асинхронного выполнения операций.

`ExecuteAsync<TOperation>(object parameters)` — типизированный метод для запуска операции с параметрами. `TOperation` — конфигурационный класс, который реализует интерфейс `IEntityEventAsyncOperation`.

Интерфейс IEntityEventAsyncOperation

Назначение интерфейса `Terrasoft.Core.Entities.AsyncOperations.Interfaces.IEntityEventAsyncOperation` — предоставляет метод для запуска асинхронной операции.

`Execute(UserConnection userConnection, EntityEventAsyncOperationArgs arguments)` — метод для запуска.

Важно. В классе, который реализует интерфейс `IEntityEventAsyncOperation`, не рекомендуется реализовывать логику изменения основной сущности. Это может привести к неверному формированию данных. Также не стоит выполнять легковесные операции (например, подсчет значения поля), поскольку создание отдельного потока может занимать больше времени, чем выполнение самой операции.

Класс EntityEventAsyncOperationArgs

Назначение класса `Terrasoft.Core.Entities.AsyncOperations.EntityEventAsyncOperationArgs` — экземпляры класса используются в качестве аргумента для передачи в асинхронную операцию.

Свойства класса `EntityEventAsyncOperationArgs`:

- `EntityId` — идентификатор записи.
- `EntitySchemaName` — название схемы.
- `EntityColumnValues` — словарь текущих значений колонок сущности.
- `OldEntityColumnValues` — словарь старых значений колонок сущности.

Реализовать асинхронность в событийном слое

Чтобы **реализовать асинхронность в событийном слое**:

1. Создайте класс, который реализует интерфейс `IEntityEventAsyncOperation`. В нем реализуйте дополнительную логику, которая может выполняться асинхронно.

Пример класса, который реализует дополнительную логику, приведен ниже.

Пример класса, который реализует дополнительную логику

```
/* Класс, который реализует асинхронный вызов операций. */
public class DoSomethingActivityAsyncOperation: IEntityEventAsyncOperation
{
    /* Стартовый метод класса. */
    public void Execute(UserConnection userConnection, EntityEventAsyncOperationArgs argument)
    {
        /* ... */
    }
}
```

2. В методе-обработчике слушателя объекта активности используйте [фабрику классов](#).

Назначение фабрики классов:

- Получить экземпляр класса, который реализует интерфейс `IEntityEventAsyncExecutor`.
- Подготовить параметры.
- Передать на выполнение класс, который реализует дополнительную логику.

Пример вызова асинхронной операции в событийном слое приведен ниже.

Пример вызова асинхронной операции в событийном слое

```
[EntityEventListener(SchemaName = "Activity")]
public class ActivityEntityEventListener : BaseEntityEventListener
{
    /* Метод-обработчик события после сохранения сущности. */
    public override void OnSaved(object sender, EntityAfterEventArgs e) {
        base.OnSaved(sender, e);
        /* Экземпляр класса для асинхронного выполнения. */
        var asyncExecutor = ClassFactory.Get<IEntityEventAsyncExecutor>(
            new ConstructorArgument("userConnection", ((Entity)sender).UserConnection));
        /* Параметры для асинхронного выполнения. */
        var operationArgs = new EntityEventAsyncOperationArgs((Entity)sender, e);
        /* Выполнение в асинхронном режиме. */
        asyncExecutor.ExecuteAsync<DoSomethingActivityAsyncOperation>(operationArgs);
    }
}
```