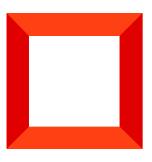


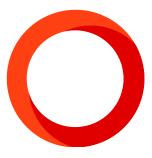
Back-end разработка

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	
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 методыобработчики различных событий сущности, которые представлены в таблице ниже.

Методы-обработчики событий сущности

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

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.
- Kласс 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. Тореration — 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 асихронность в событийном слое

Чтобы реализовать асихронность 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.

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);
    }
}
```