Объектная модель подключаемых модулей

Последние изменения: 2024-03-26

Рассмотрим объектную модель подключаемых модулей. Базовым интерфейсом для всех подключаемых модулей является IConnectivityBase:

[C#]

```
/// <summary>
/// Базовый интерфейс для подключаемых модулей
/// </summary>
public interface IConnectivityBase
{
/// <summary>
/// Уникальный идентификатор модуля.
/// </summary>
string Id
{
get;
}
/// <summary>
/// Флаг, указывающий разрешена или нет работа модуля.
/// </summary>
bool Enabled
{
get;
set;
}
/// <summary>
/// Инициализация (запуск) модуля. После вызова этой функции он должен перейти в  рабочее
состояние
/// </summary>
void Initialize();
/// <summary>
/// Инициализировано соединение или нет.
/// </summary>
bool Initialized
{
get;
}
/// <summary>
/// Функция проверки дополнительных лицензионных ограничений модуля
/// </summary>
param name="supportedSystems">Список внешних модулей, упомянутых в лицензионном
файле</param>
 void
CheckLicenseLimitations(System.Collections.Generic.List<Cleverence.Licensing.LicenseExternalSystem

supportedSystems);
}
```

От интерфейса IConnectivityBase наследуется интерфейс соединения с внешней системой IConnector и интерфейс расширения (плагина) IPlugin.

[C#]

```
/// <summary>
/// Интерфейс соединения с внешней системой. Важное отличие - наличие метода Invoke, позволяющего вызывать любые
какие-то методы из внешней системы
/// </summary>
public interface IConnector: IConnectivityBase
{
/// <summary>
/// Тайм-аут при вызове функций внешней системы через коннектор.
/// </summary>
int Timeout
{
get;
set;
}
/// <summary>
/// Поведение при наступлении тайм-аута. ThrowException - вызывать исключение, ReInvoke - завершить попытку вызова,
переинициализировать подключение и сделать снова вызов.
/// </summary>
TimeoutBehavaior TimeoutBehavaior
{
get;
set;
}
/// <summary>
/// Сообщает, что коннектор сам внутри обрабатывает таймауты, и внешняя обработка не требуется
/// </summary>
bool IsSelfTimeoutBehavior { get; }
/// <summary>
/// Для ActiveMQ коннектора добавляет в аргументы DeviceInfo.
/// </summary>
bool IsSupportDeviceInfoInArgs
{
get;
}
/// <summary>
/// Вызов метода внешней системы.
/// </summary>
/// <param name="methodName">Имя метода.</param>
/// <param name="args">Параметры.</param>
/// <returns>Результат (null, если метод ничего не возвращает).</returns>
object InvokeMethod(string methodName, object[] args);
}
/// <summary>
/// Базовый интерфейс плагина.
/// Немного расширяет стандартный IConnectivityBase явным методом остановки модуля.
/// </summary>
public interface IPlugin: IConnectivityBase
```

```
{
/// <summary>
/// Деинициализация (остановка) модуля. После вызова этой функции он должен остановить
свою работу.
/// </summary>
void Deinitialize();
}
```

Основное отличие между внешним соединением (коннектором) и расширением (плагином) в том, что коннектор имеет функцию InvokeMethod, с помощью которой выполняется вызов внешней системы при запросах с ТСД и при обработке событий сервера. Свойства Timeout, TimeoutBehavaior, IsSelfTimeoutBehavior позволяют установить тайм-аут при вызовах и задать поведение при наступлении тайм-аута. Плагин, в отличии от коннектора, не поддерживает вызовы. И коннектор и плагин запускаются при вызове функции Initialize сервером Mobile SMARTS. Вызов Initialize происходит при старте сервера базы данных Mobile SMARTS (если в настройках базы включена опция «Инициализировать подключения к внешним системам при старте») или при обращении к коннектору (вызов с ТСД или обработка события сервера). Конкретная реализация Initialize определяется особенностями системы, к которой выполняется подключение. Например, может быть выполнено создание COM-объекта для работы со внешней системой и выполнено подключение к базе данных системы с заданным логином/паролем. В случае плагина Initialize запускает модуль в работу. Например, может начаться слежение за папкой или запущен таймер для выполнения периодических операций. Плагин имеет функцию Deinitialize, при вызове которой работа плагина прекращается. Работа плагина может быть остановлена через панель управления (контекстное меню узла плагина -> «Остановить»), также остановка происходит при остановке сервера базы данных Mobile SMARTS.

Еще одним интерфейсом, производным от **IConnectivityBase**, является **IEventsProcessor**:

```
[C#]

/// <summary>
/// Базовый интерфейс для модуля, умеющего обрабатывать события.
/// Для реализации обработки событий необходимо пользоваться
/// атрибутами EventProcessorAttribute, DocumentEventProcessorAttribute
/// </summary>
public interface IEventsProcessor: IConnectivityBase
{
}
```

Интерфейс не добавляет каких-либо функций в **IConnectivityBase**, а служит маркером для модулей, которые используются при обработке событий сервера.

Существует базовая реализация интерфейса **IConnectivityBase**, от которой можно наследоваться при реализации своих коннекторов — класс **ConnectorBase**.

Если разрабатываемый коннектор будет использоваться для обработки событий сервера, рекомендуется в качестве базового использовать другой класс — **ConnectorTypical**, производный от **ConnectorBase**. Данный класс позволяет указывать имена обработчиков событий сервера в свойствах самого коннектора.

[C#]

Код класса приведен не полностью

```
/// <summary>
/// Базовый класс коннектора, позволяющий указывать имена обработчиков событий в свойствах самого коннектора.
/// При обработке событий вызываются функции класса, помеченные атрибутом EventProcessor или DocumentEventProcessor.
/// При разработке новых коннекторов рекомендуется наследоваться от данного класса.
/// </summary>
public abstract class ConnectorTypical : ConnectorBase
{
/// <summary>
/// Имя функции-обработчика события "Документ добавляется".
/// Событие вызывается перед добавлением документа в базу Mobile SMARTS из внешней системы.
/// </summary>
[EventHandlerProperty(EventType.DocumentAdding)]
public string DocumentAddingHandler { get; set; }
.....
/// <summary>
/// Обработчик вызывается перед добавлением документа на сервер из внешней системы.
/// </summary>
/// <param name="di"></param>
/// <param name="documentTypeName"></param>
/// <param name="doc"></param>
/// <returns>Возвращается переданный документ или null, если требуется отклонить добавление документа (будет вызвано исключение и документ не будет сохранен на сервере).
</returns>
[DocumentEventProcessor(DocumentType = "*", EventType = EventType.DocumentAdding)]
public virtual object DocumentAdding(DeviceInfo di, string documentTypeName, Document doc)
...
}
```

Имена обработчиков в панели управления указываются в свойствах коннектора:

Указание обработчиков событий сервера в свойствах коннектора позволяет организовать вызов нескольких обработчиков для одного события. Порядок вызова обработчиков редактируется в панели управления в свойствах узла «События сервера»:

Класс ConnectorTypical содержит виртуальные функции, вызываемые при обработке событий сервера, данные функции помечены атрибутом DocumentEventProcessor для событий документов и атрибутом EventProcessor для остальных событий (номенклатуры и таблиц). Например:

```
[C#]

<summary>
/// Обработчик вызывается сервером при запросе документа с ТСД.
/// </summary>
/// <returns>Возвращается документ (Document), может в сериализованном в xml-строку виде.
Если документ не был получен, то возвращается null. Также может возвращаться InvokeResult.
</returns>
[DocumentEventProcessor(DocumentType = "*", EventType = EventType.GetDocument)]
public virtual object GetDocument(DeviceInfo di, string documentTypeName, string  identity,
GetDocumentMode mode)
```

Атрибут DocumentEventProcessor имеет параметры: DocumentType, позволяет ограничить вызов данного обработчика определенными типами документов, «*» — вызов будет выполнен для всех типов документов, EventType — тип события. Атрибут EventProcessor имеет один параметр: EventType — тип события. При реализации своих обработчиков в производных классах также следует использовать атрибуты DocumentEventProcessor и EventProcessor.

Типы событий, задаваемые перечислением EventType:

События документов
DocumentAdding
Документ добавляется. **Событие, возникающее в процессе добавления документа на сервере. Вызывается при выгрузке документа в базу Mobile SMARTS из внешней системы.**
DocumentAdded
Документ добавлен. **Событие, возникающее после добавления документа на сервере. Вызывается при выгрузке документа в базу Mobile SMARTS из внешней системы.**
DocumentAppointing
Документ назначается пользователю. **Событие о том, что документ готов передаваться на мобильное устройство. Вызывается при запросе с ТСД получения документа для работы.**
DocumentAppointed
Документ назначен пользователю. **Событие о том, что документ захвачен на обработку. Вызывается в момент, когда документ был передан на ТСД для работы с ним.**
DocumentChanged
Документ изменен. **Событие об изменении документа. Вызывается при сохранении документа на сервер в процессе работы на ТСД при использовании в конфигурации Mobile SMARTS действия «Сохранение документа на сервер».**
DocumentFinished
Документ завершен. **Событие о завершении обработки документа. Вызывается при получении сервером завершенного документа с ТСД.**

DocumentReleased
Документ возвращен с ТСД без обработки. Вызывается когда документ с терминала был возвращен пользователем вызовом release (вернуть документ без изменений).
DocumentRemoved
Документ удален. Событие об удалении документа с сервера. Удаление выполняется внешней системой.
ListDocuments
Получить список документов. Вызывается при входе в список документов на ТСД. Используется в паре с событием ПолучитьДокумент для того, чтобы реализовать выбор документов прямо из учетной системы, без предварительной выгрузки.
GetDocument
Получить документ. Вызывается при выборе в списке документов на ТСД того документа, который еще не был выгружен (попал в этот список с помощью события ПолучитьСписокДокументов).
События номенклатуры
ProductNotFound
Получить товар. Событие о том, что продукт был запрошен терминалом по коду (штрихкод, артикул, код) с сервера и не был найден в серверном справочнике.
ProductsNotFound
Получить товары по списку Id. Событие о том, что несколько товаров были запрошены терминалом по идентификаторам с сервера и не были найден в серверном справочнике. Позволяет реализовывать получение сразу нескольких товаров из базы учетной системы без предварительной выгрузки.
GetProductByPartOfName
Получить товар по части наименования. Вызывается при вводе в строку поиска в списке номенклатуры на ТСД. Используется для того, чтобы реализовать поиск номенклатуры по части наименования прямо из учетной системы.

GetCellById
Получить ячейку по идентификатору
GetCellByBarcode
Получить ячейку по штрихкоду
ListProducts
Получить список товаров. Вызывается при входе в список номенклатуры на ТСД. Используется для того, чтобы реализовать выбор номенклатуры прямо из учетной системы, без предварительной выгрузки.
События таблиц
TableRequest
Обработать запрос. Событие вызывается при запросе данных из таблицы, определенной в конфигурации Mobile SMARTS. Позволяет получать строки таблицы он-лайн по запросу с ТСД.

Не нашли что искали?

Задать вопрос в техническую поддержку