- `ТочкаМаршрута.ИмяТочки` - точка маршрута бизнес-процесса;
- `ПустаяСсылка` - для получения пустой ссылки.

В случае необходимости получить значение системного перечисления параметр метода будет выглядеть следующим образом: `ИмяСистемногоПеречисления.ЗначениеСистемногоПеречисления`.

Например:

Копировать в буфер обмена

```
ТипДиаграммы = ПредопределенноеЗначение("ТипДиаграммы.ВогнутаяПоверхность");
```

В случае если требуется получить точку маршрута бизнес-процесса, строка, описывающая получаемое значение, будет выглядеть следующим образом:

Пример:

Копировать в буфер обмена

```
// Получение значения перечисления.
Вид = ПредопределенноеЗначение("Перечисление.ВидыТоваров.Товар");
// Получение значения пустой ссылки.
ПустаяСсылка =
    ПредопределенноеЗначение("Документ.РасходнаяНакл.ПустаяСсылка");
// Получение предопределенных данных справочника.
Элемент = ПредопределенноеЗначение("Справочник.Валюта.Рубль");
// Точка маршрута бизнес процесса
Точка =    ПредопределенноеЗначение("БизнесПроцесс.Согласование.
ТочкаМаршрута.Одобрение");
```

4.7.9. Синхронные и асинхронные методы работы

4.7.9.1. Общая информация

Сеанс работы с информационной базой определяет активного пользователя базы и поток управления этого пользователя. В рамках сеанса, программа на встроенном языке «1С:Предприятие» выполняется последовательно в одном потоке, таким образом - никакие два фрагмента кода на встроенном языке не могу выполняться одновременно. Встроенный язык в своем изначальном виде работает синхронно (в том числе и методы объектной модели). **Синхронный** вызов блокирует основной поток до тех пор, пока не завершится вызванный метод (со всеми вложенными). Это особенно неприятно в тех случаях, когда, например, выполняется ожидание какой-то реакции пользователя (ответ на вопрос). Веб-браузеры не поддерживают синхронное исполнение некоторых операций, которые во встроенном языке являются синхронными. Веб-браузер требует, чтобы эти операции были **асинхронными**. Асинхронные методы - это такие методы, завершение (и исполнение) которых не синхронно с кодом, из которого они вызваны. Асинхронные вызовы предназначены для решения одной задачи: в некоторых случаях предотвратить блокирование основного потока.

Для реализации асинхронной схемы использования в клиентском приложении, кроме основного потока существует очередь асинхронных задач. Задачи из этой очереди выполняются основным потоком в тот момент, когда основной поток сможет это сделать. Где конкретно выполняется асинхронный метод - является деталью реализации. Это не влияет на понимание и использование механизма. Однако важно заметить, что рассматриваемый механизм не имеет отношения к многопоточности.

Асинхронные (по сути) методы могут содержаться только в объектной модели встроенного языка системы «1С:Предприятие». Но наличие асинхронных методов в объектной модели требует, чтобы разработчик понимал, как работает эта конструкция.

Рассмотрим разницу между синхронной и асинхронной работой. Как уже было сказано выше, при синхронном подходе вызов какого-либо метода приведет к тому, что работа вызывающего кода будет остановлена до тех пор, пока не завершится вызов метода (со всеми вложенными вызовами). Любой вызов процедуры или функции встроенного языка всегда является синхронным вызовом. Но завершение метода зависит от используемой схемы работы: в синхронной схеме возврат из метода произойдет только при его фактическом завершении, а при асинхронной схеме - возможны несколько возвратов до фактического завершения работы.

Если рассмотреть пример с кодом, который показывает предупреждение, ожидает нажатия кнопки и выводит какой-то текст, то синхронный вариант будет выглядеть следующим образом:

Копировать в буфер обмена

```
Текст = "Текст предупреждения";
Предупреждение(Текст);
Сообщить("Закрыли предупреждение");
```

При исполнении этого кода мы увидим, что текст Закрыли предупреждение будет выведен после того, как в диалоге нажали кнопку Ок. Если вместо метода `Предупреждение()` будет вызов кого-то метода на встроенном языке, то исполнение функции `Сообщить()` произойдет только после того, как выполниться наш метод (со всеми вложенными вызовами, если таковые есть).

При асинхронном подходе вызов метода выполняется как обычно, но управление возвращается вызывающему коду до того, как асинхронный метод завершит свою работу. После этого вызывающий код продолжает свое выполнение. При этом вызванный (асинхронный) метод гарантирует, что после завершения своей работы он оповестит о своем завершении заинтересованное «лицо» (если такое оповещение требуется). Способ оповещения зависит от того, какую схему использования асинхронных методов использует разработчик. Все вышеописанное верно только для тех методов, для которых заявлена поддержка асинхронной работы.

Рассмотрим вышеприведенный пример в асинхронной схеме. Асинхронный вариант программы будет отличаться от синхронного варианта совсем немного. Но работать он будет абсолютно по-другому:

Копировать в буфер обмена

```
Текст = "Текст предупреждения";
ПоказатьПредупреждение( , Текст);
Сообщить("Закрыли предупреждение");
```

В этом примере текст Закрыли предупреждение будет показан до того, как в диалоге будет выполнено хоть какое-то действие. Таким образом, мы явным образом наблюдаем ранее описанную особенность асинхронного выполнения: исполнение на стороне вызывающего кода продолжится до того, как полностью закончилось исполнение вызванного метода. Под «исполнением кода» понимается полное выполнение метода (включая все вложенные) и выполнение всех положенных действий (включая интерактивные).

Для реализации «асинхронного программирования» платформа «1С:Предприятие» предлагает две схемы:

- Схему с реализацией методов обратного вызова (подробнее см. здесь).
- Схему с реализацией «обещаний» асинхронных методов (подробнее см. здесь).

Более подробно каждая из описанных схем будет описана далее.

В системе «1С:Предприятие» асинхронную технику можно использовать для работы:

1. С блокирующими окнами (см. здесь);
2. Расширением работы с файлами (см. здесь);
3. Расширением криптографии (см. здесь);
4. Внешними компонентами (см. здесь).

Все варианты предполагают асинхронное использование только на стороне клиентского приложения. На стороне сервера по-прежнему доступны синхронные техники работы, за исключением работы с блокирующими окнами (по причине отсутствия пользовательского интерфейса на стороне сервера).

4.7.9.2. Асинхронность через обратный вызов

При асинхронном подходе вызов метода выполняется как обычно, но управление возвращается вызывающему коду до фактического завершения асинхронного метода, и исполнение вызывающего кода продолжается. При этом вызванный метод гарантирует, что после завершения своей работы он оповестит о своем завершении заинтересованное «лицо» (если такое оповещение

требуется). Оповещение выполняется с помощью вызова специального метода обратного вызова, который передается в вызываемый метод объекта.

С точки зрения имен методов встроенного языка, асинхронные методы отличаются от синхронных тем, что имя асинхронного метода (как правило) начинается со слов `Начать` или `Показать`. Знание этой особенности позволит упростить поиск аналогов синхронных методов.

Рассмотрим организацию асинхронного вызова на примере отображения предупреждения. Для асинхронного вывода предупреждения используется метод `ПоказатьПредупреждение()`. Для того, чтобы сообщить о том, что диалог предупреждения закрыт, в этот метод передается **описание оповещения**. Описание оповещения содержит имя и месторасположения процедуры или функции, которая будет вызвана системой после закрытия диалога. Соответственно, метод `ПоказатьПредупреждение()` отличается от метода `Предупреждение()` тем, что в методе `ПоказатьПредупреждение()` существует еще один параметр, через который и передается описание оповещения. Эта разница означает, что метод поддерживает асинхронную работу.

Пример:

Копировать в буфер обмена

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    ОбратныйВызов = Новый ОписаниеОповещения("ПредупреждениеЗавершение", ЭтотОбъект);
    ПоказатьПредупреждение(ОбратныйВызов, "Закрытие формы обрабатывается отдельно");
КонецПроцедуры
&НаКлиенте
Процедура ПредупреждениеЗавершение(ДополнительныеПараметры) Экспорт
    // какие то действия
КонецПроцедуры
```

Для описания оповещения используется специальный объект `ОписаниеОповещения`. При его создании необходимо указать имя метода обработки оповещения, место расположения метода и дополнительные параметры, если необходимо. Дополнительные параметры могут использоваться методом обработки оповещения для своей работы. В приведенном выше примере дополнительные параметры не указываются.

Также следует отметить, что методы обработки оповещений (обратного вызова) могут располагаться только на стороне клиентского приложения в модуле формы, общем модуле и модуле команды. Процедура обратного вызова должна быть описана с ключевым словом `Экспорт`.

Если необходимо вызвать обработчик оповещения в каком-либо месте кода на встроенном языке, то можно использовать метод `ВыполнитьОбработкуОповещения()` (при этом описание вызываемого оповещения может быть создано и передано из других участков кода, например в качестве параметра метода). В качестве обработчика оповещения может выступать как процедура, так и функция. В случае, если обработчиком оповещения выступает функция, метод `ВыполнитьОбработкуОповещения()` вернет результат работы функции. Если обработчик оповещения - это процедура, то метод `ВыполнитьОбработкуОповещения()` вернет значение `Неопределено`.

Из примера, приведенного выше, видно, что работа с асинхронными методами существенно изменяет подходы к разработке. Основное изменение заключается в том, что единый фрагмент текста на встроенном языке (для случаев использования синхронных методов) разбивается на несколько изолированных фрагментов (в случае использования асинхронных методов).

Так, если в приведенном примере после отображения предупреждения должны были выполняться какие-то действия (именно после реакции пользователя, а не после вызова метода!), то эти действия следует перенести в процедуру `ПредупреждениеЗавершение()`.

Более сложные алгоритмы, очевидно, приведут и к более сложному рефакторингу исходного текста программы. Возможно, что переработки потребует сам алгоритм, а не только его реализация, которая может существенно усложниться. Например, в синхронном режиме, если алгоритму требуется какая-либо реакция пользователя, то эту реакцию получают непосредственно в том месте, где эта реакция требуется. Если перерабатывать такой алгоритм с использованием асинхронной техники, то логично разработать отдельный диалог, в котором пользователь ответит на все возможные вопросы, а затем результат этого «опроса» будет передан методу, который реализует нужный алгоритм. При этом вопросы «по месту» больше задавать не требуется, а вместо этого следует анализировать результаты «опроса» пользователя.

Кроме изменения подходов к разработке, при использовании асинхронных методов несколько изменяется и подход к обработке ошибок. Например, ошибки могут возникать в то время, когда код на встроенном языке не исполняется и нельзя использовать конструкцию `Попытка … Исключение`. Для обработки таких ситуаций при создании обработчика оповещения можно указать процедуру, которая будет вызываться системой при возникновении ошибки. Примером такой ошибки может быть ошибка, возникающая в процессе удаления файлов.

Пример:

Копировать в буфер обмена

```
&НаКлиенте
Процедура УдалитьВсеФайлыКаталога(ПутьККаталогу)
    ОбратныйВызов = Новый ОписаниеОповещения("УдалитьЗавершение", ЭтотОбъект, ,
"УдалитьОшибка", ЭтотОбъект);
    НачатьУдалениеФайлов(ОбратныйВызов, ПутьККаталогу, ПолучитьМаскуВсеФайлыКлиента());
КонецПроцедуры
&НаКлиенте
Процедура УдалитьЗавершение(ДополнительныеПараметры) Экспорт
    Сообщить("Удаление успешно завершено");
КонецПроцедуры
&НаКлиенте
Процедура УдалитьОшибка(ИнформацияОбОшибке, СтандартнаяОбработка,
ДополнительныеПараметры) Экспорт
    Сообщить("При удалении обнаружена ошибка: " +
КраткоеПредставлениеОшибки(ИнформацияОбОшибке));
КонецПроцедуры
```

В приведенной реализации есть одна особенность: если во время удаления файлов произойдет ошибка, то наряду с сообщением об ошибке в окне сообщений платформы, пользователь увидит также и системный диалог с сообщением об ошибке. В том случае, если системный диалог не нужен - следует в обработчике `УдалитьОшибка()` установить параметр `СтандартнаяОбработка` в значение `Ложь`.

Также следует учитывать, что указание обработчика ошибок в описании оповещения игнорируется при использовании асинхронных методов работы с блокирующими окнами (см. здесь).

Еще одной особенностью работы с асинхронными вызовами является их отладка. В данный момент отладчик умеет отлаживать действия, которые синхронно выполняются в предмете отладки (фактически, по инициативе отладчика).

В случае использования асинхронных вызовов асинхронный вызов выполняется в момент времени, отличный от фактического выполнения строки кода на встроенном языке, которая инициирует данный вызов. В связи с этим, если во время фактического исполнения асинхронного вызова в этом вызове случается ошибка, отладчик не может корректно определить состояние отлаживаемого приложения. Поэтому недоступна текущая строка исполнения, стек вызовов и так далее.

4.7.9.3. Асинхронность через обещания

4.7.9.3.1. Общая информация

Основу данной схемы работы составляет объект встроенного языка `Обещание`. Обещание представляет собой результат асинхронной функции, выполнение которой еще не завершено, но ее завершение ожидается в будущем (обещается). Объект `Обещание` возвращается асинхронной функцией в качестве результата своей работы.

Пример с отображением предупреждения и выводом сообщения пользователю после(!) закрытия окна предупреждения будет выглядеть следующим образом:

Копировать в буфер обмена

```
Текст = "Текст предупреждения";
Ждать ПредупреждениеАсинх(Текст);
Сообщить("Закрыли предупреждение");
```

В результате своей работы асинхронная функция может или вернуть какое-то значение (результат работы функции) или сгенерировать исключение, если в процессе работы функции произошла ошибка. Для того, чтобы получить результат работы асинхронной функции, предназначен

оператор Ждать. У этого оператора есть единственный параметр - это объект типа Обещание. В результате система дожидается завершения работы асинхронной функции, которая вернула ожидаемое обещание и либо получает результат работы функции либо генерирует исключение. Но ожидание завершения происходит не в том месте кода, где использован оператор Ждать. При выполнении оператора Ждать произойдет возврат управления в вызывающий код. Подробнее про эту особенность поговорим чуть позже.

Важным моментом является тот факт, что объект типа Обещание является обычным объектом платформы. Его можно присваивать переменной, передавать в другие методы и использовать в выражениях. Другими словами: вызвать асинхронную функцию мы можем в одном месте, а дождаться завершения работы этой функции - совершенно в другом. Существенным моментом является то, что обещание не может быть передано на сервер. Получить обещание на сервере также не возможно.

Несколько переделаем предыдущий пример. Вместо отображения предупреждения, сделаем отображение вопроса: Продолжить выполнение?. В диалоге будут две кнопки Да и Нет.

Копировать в буфер обмена

```
Текст = "Продолжить работу?";
Результат = Ждать ВопросАсинх(Текст, РежимДиалогаВопрос.ДаНет);
Если Результат = КодВозвратаДиалога.Да Тогда
    // продолжаем работу
    Сообщить("Продолжаем работу");
Иначе
    // завершаем работу
    Сообщить("Завершаем работу");
    Возврат;
КонецЕсли;
```

Этот пример демонстрирует, что обещание нужно использовать «в комплекте» с оператором Ждать. Если получить обещание, но не использовать Ждать, то не будет переменной Результат. Следовательно - нельзя будет проверить, что ответил пользователь. После того, как асинхронная функция завершила свою работу, оператор Ждать получает из обещания фактическое возвращаемое значение (выполняет обещание).

Если асинхронная функция, во время своей работы, может сформировать исключение, то его также можно перехватить. Для этого, как и в синхронном коде, следует использовать конструкцию Попытка … Исключение. Но исключение будет перехвачено только в том случае, если обещание используется совместно с оператором Ждать. Например, вот так можно написать код поиска файлов по маске:

Копировать в буфер обмена

```
ОбещаниеФайлов = НайтиФайлыАсинх(Ждать КаталогВременныхФайловАсинх(), "*.xml");
Попытка
    Файлы = Ждать ОбещаниеФайлов;
Исключение
    // при поиске файлов случилась ошибка
КонецПопытки;
```

Если во время поиска файлов произойдет какое-либо исключение - это исключение будет перехвачено оператором Попытка.

В результате использования асинхронности через обещания, программист получает исходный код, который очень близок к своему синхронному аналогу. Асинхронность через обещания делает более простой адаптацию синхронного кода для использования в асинхронной схеме. Такой способ упрощает программирование некоторых алгоритмов на стороне клиентского приложения и делает их сопровождение более простой задачей.

4.7.9.3.2. Схема именования методов

Асинхронные методы платформы, которые используют обратные вызовы, выделяются префиксом Начать или Показать (см. здесь). Аналогично, платформенные функции, которые возвращают объект типа Обещание, выделяются с помощью суффикса Асинх. Но ключевым признаком того, что функция может использоваться для асинхронной работы с использованием обещаний - это тот факт, что функция возвращает объект типа Обещание. В результате некоторое множество функций платформы существуют в трех вариантах:

• Синхронный вариант - метод описывает совершаемое действие. Например, `Вопрос()` или `НайтиФайлы()`. Основной поток блокируется до момента завершения вызова метода.

• Асинхронный вариант через обратный вызов - название метода начинается с префикса `Начать` или `Показать`. Например, `ПоказатьВопрос()` или `НачатьПоискФайлов()`. О завершении метода выполняется оповещение с помощью метода обратного вызова, переданного в качестве параметра.

• Асинхронный вариант через обещания - название метода оканчивается на суффикс `Асинх`. Например, `ВопросАсинх()`, `НайтиФайлыАсинх()`. В этом случае используется объект `Обещание` и возможность дождаться завершения с помощью оператора `Ждать`.

Еще одной особенностью асинхронности через обещание является то, что использование оператора `Ждать` допустимо только в тех процедурах или функциях, которые объявлены с модификатором `Асинх`:

Копировать в буфер обмена

```
&НаКлиенте
Асинх Процедура ЖдатьПредупреждение(Команда)
    Текст = "Текст предупреждения";
    Ждать ПредупреждениеАсинх(Текст);
    Сообщить("Закрыли предупреждение");
КонецПроцедуры
```

Если у процедуры `ЖдатьПредупреждение()` убрать модификатор `Асинх`, при попытке сохранения модуля конфигуратор выдаст ошибку:

Копировать в буфер обмена

```
Оператор Ждать (Await) может употребляться только в асинхронных процедурах или функциях
    <<?>>Ждать ПредупреждениеАсинх(Текст);
```

4.7.9.3.3. Особенности асинхронных методов

Асинхронной может быть как функция, так и процедура. Как было сказано выше, для того, чтобы метод стал асинхронным, его необходимо описать с использованием модификатора `Асинх`. Метод, описанный с таким модификатором, получает возможность использования оператора `Ждать`. При разработке асинхронных методов следует помнить, что параметры в асинхронные методы всегда передаются по значению. Таким образом, ключевое слово `Знач` в описании асинхронного метода игнорируется. Не поддерживается возможность передавать параметр асинхронного метода по ссылке - только по значению. В результате вышесказанного, следующие два описания методов полностью эквивалентны:

Копировать в буфер обмена

```
Асинх Функция КопироватьФайлыАсинх(ИсхКаталог, ЦелКаталог)
Асинх Функция КопироватьФайлыАсинх(Знач ИсхКаталог, Знач ЦелКаталог)
```

Асинхронные функции всегда возвращают объект типа `Обещание`. Асинхронные процедуры ничего не возвращают. Разберем эти особенности. Как уже было сказано выше, асинхронная функция всегда возвращает объект типа `Обещание`. Но ведь функция должна вернуть именно то значение, которое хочет вернуть автор функции. Для преобразования обещания в конкретное значение - служит оператор `Ждать`. Т. е. в том случае, когда нам необходимо получить фактическое значение результата работы функции, необходимо написать код, уже встречавшийся ранее в этом разделе (в следующей строке примера приведен синхронный аналог вызова):

Копировать в буфер обмена

```
Результат = Ждать АсинхроннаяФункция(Параметр1, Параметр2);
Результат = СинхроннаяФункция(Параметр1, Параметр2);
```

При обработке исключений следует понимать следующее: исключение, которое произошло при выполнении асинхронного метода невозможно перехватить привычным, синхронным способом. Другими словами, следующий пример не приведет к перехвату исключения:

Копировать в буфер обмена

```
&НаКлиенте
Асинх Процедура ОбработкаИсключения(Команда)
    Попытка
        ЕстьИсключение();
```

```
    Исключение
        Сообщить("Поймано исключение: " +
ОбработкаОшибок.ПодробноеПредставлениеОшибки(ИнформацияОбОшибке()));
    КонецПопытки;
КонецПроцедуры
&НаКлиенте
Асинх Функция ЕстьИсключение()
    ВызватьИсключение "Исключение в асинхронной функции";
КонецФункции
```

Обработать исключение можно одним из следующих способов:

- Обработать исключение в самом асинхронном методе, например, с помощью конструкции `Попытка … Исключение` вокруг кода, который может сформировать исключение. Другими словами можно не допустить ситуации, когда асинхронная функция закончится исключением.

- Поместив оператор `Ждать` с обещанием аснхронной функции в конструкцию `Попытка … Исключение` в коде, из которого вызвана асинхронная функция. Далее приведен пример такой обработки исключения:

Копировать в буфер обмена

```
&НаКлиенте
Асинх Процедура ОбработкаИсключения(Команда)
    Попытка
        Ждать ЕстьИсключение();
    Исключение
        Сообщить("Поймано исключение: " +
ОбработкаОшибок.ПодробноеПредставлениеОшибки(ИнформацияОбОшибке()));
    КонецПопытки;
КонецПроцедуры
&НаКлиенте
Асинх Функция ЕстьИсключение()
    ВызватьИсключение "Исключение в асинхронной функции";
КонецФункции
```

Также необходимо понимать, что перехватывать исключение необходимо не в месте вызова асинхронной функции, в месте использования оператора `Ждать` с обещанием, которое вернула асинхронная функция. Можно модифицировать предыдущий пример так, что отдельно будет получаться обещание и отдельно будем выполнено его ожидание:

Копировать в буфер обмена

```
&НаКлиенте
Асинх Процедура ОбработкаИсключения(Команда)
    Обещание = ЕстьИсключение();
    Попытка
        Ждать Обещание;
    Исключение
        Сообщить("Поймано исключение: " +
ОбработкаОшибок.ПодробноеПредставлениеОшибки(ИнформацияОбОшибке()));
    КонецПопытки;
КонецПроцедуры
&НаКлиенте
Асинх Функция ЕстьИсключение()
    ВызватьИсключение "Исключение в асинхронной функции";
КонецФункции
```

В данном примере исключение будет перехвачено, т. к. в попытку «обернуто» ожидание обещания. Но в качестве исходной строки с ошибкой будет указана не строка, где размещен оператор `Ждать`, а строка с фактическим вызовом асинхронной функции.

Еще одной особенностью работы с обещаниями является то, что процедура не возвращает никакого значения. Из этого следует, что процедура не может вернуть объект типа `Обещание`. Следовательно, перехватить исключение, которое возникает в асинхронной процедуре, за ее (процедуры) пределами - невозможно. Это исключение попадет в обработчик события `ОбработкаОтображенияОшибки`, а если этого обработчика нет - будет сразу показана пользователю. Таким образом, асинхронная процедура интересна, в первую очередь, как обработчик событий, которые формирует платформа (например, в форме). Если для процедуры не указан модификатор `Асинх` - в ней нельзя использовать оператор `Ждать`.

Сформулируем общие особенности использования асинхронных методов:

- При вызове асинхронного метода параметры всегда передаются по значению.

- Все обещания (объект типа `Обещание`), которые возвращают асинхронные функции, рекомендуется использовать в операторах `Ждать`. В противном случае будут потеряны возвращаемые значения функций и исключения, возникающие в функциях.

- Процедуру имеет смысл делать асинхронной в том случае, когда процедура является обработчиком события и в этой процедуре необходимо ожидать завершения исполнения асинхронных функций (оператор `Ждать`). Во всех остальных случаях рекомендуется делать асинхронные функции.

4.7.9.3.4. Как это работает

Перед рассмотрением примера использования асинхронных методов, имеет смысл упомянуть одну особенность работы этих методов, на которой ранее не делалось явного акцента. Когда в методе используется оператор `Ждать`, можно сказать, что такой метод имеет не только несколько промежуточных возвратов, которые возникают в тот момент, когда оператор `Ждать` ожидает выполнения обещания. Такой метод имеет и несколько «промежуточных» точек входа в метод. «Промежуточная» точка входа - это фактически, тот же оператор `Ждать`, но который «дождался» выполнения обещания. Происходит выполнение обещания, результат работы асинхронной функции становится доступен для использования или происходит генерация исключения, которое произошло в асинхронной функции, но было «спрятано» в обещании. Другими словами, именно отсюда будет продолжено выполнение асинхронного метода после того, как выполнится обещание. Демонстрирует такое поведение простой пример:

Копировать в буфер обмена

```
Асинх Процедура КопированиеФайлов(Команда)
    Текст = "Выполнить копирование файлов?";
    Обещание = ВопросАсинх(Текст, РежимДиалогаВопрос.ДаНет);
    Результат = Ждать Обещание;
    Если Результат = КодВозвратаДиалога.Да Тогда
        // продолжаем работу
        КопироватьФайлАсинх(ФайлИсточник, ФайлПриемник);
        Попытка
            Ждать ОбещаниеКопирования;
            Сообщить("Файл скопирован");
        Исключение
            // при копировании произошла ошибка
        КонецПопытки;
    КонецЕсли;
КонецПроцедуры
```

В процедуре `КопированиеФайлов()` будет две точки промежуточного возврата и две промежуточных точки входа:

- `Результат = Ждать Обещание` - это точка промежуточного возврата и она же - промежуточная точка входа.

- `Ждать ОбещаниеКопирования` - это промежуточная точка возврата и эта же строка будет промежуточной точкой входа.

Теперь можно рассмотреть простой пример, поясняющий схему работы «асинхронного программирования». В этом примере будут перебираться все файлы, которые размещены в каталоге временных файлов, над каждым файлом выполняется какое-то действие и в качестве результата будет возвращено количество обработанных файлов. Пример сделан для пояснения смысла, а не для создания оптимального алгоритма. В примере, для упрощения описания работы, выполнена нумерация строк:

Копировать в буфер обмена

```
01|&НаКлиенте
02|Асинх Процедура ВыполнитьКоманду(Команда)
03|    ЧислоФайлов = Ждать ПоказатьФайлыАсинх();
04|    Сообщить("Число файлов = " + ЧислоФайлов);
05|КонецПроцедуры
06|
```

```
07|&НаКлиенте
08|Асинх Функция ПоказатьФайлыАсинх()
09|    Каталог = Ждать КаталогВременныхФайловАсинх();
10|    СписокФайлов = Ждать НайтиФайлыАсинх(Каталог, ПолучитьМаскуВсеФайлы());
11|    Счетчик = 0;
12|    Для каждого Файл Из СписокФайлов Цикл
13|        // что-то делаем
14|        Счетчик = Счетчик + 1;
15|    КонецЦикла;
16|    Возврат Счетчик;
17|КонецФункции
```

Процедура `ВыполнитьКоманду()` является обработчиком события `ПриНажатии` кнопки формы. Исполнение код начинается с нажатия пользователем на эту кнопку:

- Выполнение начинается со строки 02. Затем, в строке 03, вызывается функция `ПоказатьФайлыАсинх()`.

- В строке 09 происходит вызов асинхронной функции платформы `КаталогВременныхФайловАсинх()`. Т. к. это асинхронная функция, то она возвращает обещание своего результата, которое является аргументом оператора `Ждать`. Исполнение оператора `Ждать` приводит к тому, что выполнение функции `ПоказатьФайлыАсинх()` прерывается и управление возвращается в процедуру `ВыполнитьКоманду()`, на строку 03.

- В силу того, что `ПоказатьФайлыАсинх()` возвращает `Обещание`, а вызов функции `ПоказатьФайлыАсинх()` является параметром оператора `Ждать`, выполнение процедуры `ВыполнитьКоманду()` в этот момент завершается.

- В результате выполнения встроенного языка в системе существуют два ожидания:

  1. Когда завершится выполнение функции `ПоказатьФайлыАсинх()` (строка 03).

  2. Когда завершится выполнение функции `КаталогВременныхФайловАсинх()` (строка 09).

- Ровно в тот момент, как завершилось выполнение функции `КаталогВременныхФайловАсинх()`, управление возвращается в строку 09 (включая восстановление локальных переменных), оператор `Ждать` в этой строке «дожидается» обещанного завершения работы функции. Ожидание прекратилось, в переменную `Каталог` помещен каталог временных файлов пользователя. Управление передано на строку 10.

- В строке 10 вызывается асинхронный метод платформы `НайтиФайлыАсинх()`. Логика работы с ним аналогична логике работы с методом `КаталогВременныхФайловАсинх()`. Управление передается системе «1С:Предприятие». После «выполнения» строки 10 в системе по-прежнему остается два ожидания:

  3. Когда завершится выполнение функции `ПоказатьФайлыАсинх()` (строка 03).

  4. Когда завершится выполнение функции `НайтиФайлыАсинх()` (строка 10).

- В предыдущем пункте слово «выполнение» взято в кавычки потому, что собственно выполнение еще не завершилось. Платформа определила необходимость ожидать завершения работы асинхронного кода (оператор `Ждать`) и приостановила работу функции `ПоказатьФайлыАсинх()`. В связи с тем, что фактического завершения работы функции `ПоказатьФайлыАсинх()` не произошло - ожидание в строке 03 продолжается. Т. к. более никакого кода на встроенном языке выполнять не требуется - управление возвращается платформе.

- После того, как поиск файлов завершился, управление передается на строку 10, выполняется восстановление контекста (значения локальных переменных), оператор `Ждать` в строке 10 завершает ожидание исполнения. В переменную `СписокФайлов` помещается массив с файлами из каталога временных файлов.

- Теперь в системе имеется только одно ожидание: на строке 03 система ждет фактического завершения работы функции `ПоказатьФайлыАсинх()`.

- Управление передается на строку 11 и далее по тексту функции `ПоказатьФайлыАсинх()`.

• В строке 16 происходит фактическое завершение работы функции `ПоказатьФайлыАсинх()` и происходит возврат количества обработанных файлов.

• После выполнения фактического возврата, происходит завершение ожидания в строке 03 и выполнение передается на строку 04.

• На экран выводится сообщение о количестве обработанных файлов, и работа обработчика логически завершается в строке 05.

В этой схеме важно понимать следующее: когда происходит возобновление исполнения встроенного языка после завершения работы оператора `Ждать` (строки 03, 09 и 10), инициатором вызова (и, фактически, вершиной стека вызовов) будет интерпретатор встроенного языка платформы. Но для того, чтобы программисту было проще понять, где он находится, в отладчике будет выполняться реконструкция того стека вызовов, который должен был привести к выполнению той или иной строки встроенного языка. Другими словами, отладчик пытается показать стек таким, каким стек был-бы при выполнении аналогичной, но полностью синхронной программы.

4.7.9.3.5. Примеры использования

Удаление файлов (аналог примера с обратными вызовами)

Для того чтобы удалить все файлы во временном каталоге, необходимо использовать примерно следующий код:

Копировать в буфер обмена

```
&НаКлиенте
Асинх Процедура УдалитьФайлыВременногоКаталога(Команда)
    Попытка
        Ждать УдалитьФайлыАсинх(Ждать КаталогВременныхФайловАсинх(),
ПолучитьМаскуВсеФайлы());
    Исключение
        Сообщить("При удалении файлов обнаружена ошибка: " +
ОбработкаОшибок.ПодробноеПредставлениеОшибки(ИнформацияОбОшибке()));
    КонецПопытки;
КонецПроцедуры
```

Копирование файлов

Ниже приведена реализация клиентского метода копирования всех файлов между каталогами, которые указаны на форме в реквизитах `КаталогОткуда` и `КаталогКуда`. Кнопка формы называется `ВыполнитьКопирование` и в модуле формы расположен следующий код:

Копировать в буфер обмена

```
&НаКлиенте
Асинх Процедура ВыполнитьКопирование(Команда)
    Попытка
        СколькоФайлов = Ждать КопироватьФайлыАсинх(КаталогОткуда, КаталогКуда);
        Сообщить("Скопировано файлов: " + СколькоФайлов);
    Исключение
        ОписаниеОшибки =
ОбработкаОшибок.ПредставлениеОшибкиДляПользователя(ИнформацияОбОшибке());
        ПредупреждениеАсинх("При копировании файлов произошла ошибка: " + Символы.ПС +
ОписаниеОшибки);
    КонецПопытки
КонецПроцедуры
&НаКлиенте
Асинх Функция КопироватьФайлыАсинх(ИсхКаталог, ЦелКаталог)
    СписокФайлов = Ждать НайтиФайлыАсинх(ИсхКаталог, ПолучитьМаскуВсеФайлы(), Ложь);
    Счетчик = 0;
    Для Каждого Файл Из СписокФайлов Цикл
        ИсхФайл = ИсхКаталог + ПолучитьРазделительПути() + Файл.Имя;
        ЦелФайл = ЦелКаталог + ПолучитьРазделительПути() + Файл.Имя;
        Ждать КопироватьФайлАсинх(ИсхФайл, ЦелФайл);
        Счетчик = Счетчик+1;
    КонецЦикла;
    Возврат Счетчик;
КонецФункции
```

4.7.9.4. Работа в веб-клиенте

Работа в веб-клиенте имеет некоторые особенности. Эти отличия вызваны особенностями реализации веб-браузеров (отсутствие поддержки модальных окон, синхронных вызовов, особенности модели безопасности). При работе в веб-клиенте не поддерживается синхронная техника работы (во всех вариантах). Для работы с файлами (см. здесь) и криптографией (см. здесь) требуется наличие специальных расширений, установленных в используемом веб-браузере. При использовании веб-браузера Google Chrome или Mozilla Firefox, перед установкой расширений платформы, необходимо выполнить установку расширения веб-браузера Расширение для работы с 1С:Предприятием, которое устанавливается из магазина расширений соответствующего веб-браузера. При необходимости, переход на страницу установки будет выполнен автоматически. Без установки этого расширения будет невозможно использование расширений платформы.

Для работы с расширениями необходимо:

- Соответствующим образом настроить веб-браузер (см. здесь).
- Расширение работы с файлами:
  - Установить расширение - с помощью одного из методов `УстановитьРасширениеРаботыСФайламиАсинх()`, `УстановитьРасширениеРаботыСФайлами()` или `НачатьУстановкуРасширенияРаботыСФайлами()`. Это интерактивное действие, которое необходимо выполнить один раз для каждого пользователя локального компьютера, использующего расширение.
  - Подключить расширение - с помощью метода `ПодключитьРасширениеРаботыСФайламиАсинх()`, `ПодключитьРасширениеРаботыСФайлами()` или `НачатьПодключениеРасширенияРаботыСФайлами()`.
- Расширение работы с криптографией:
  - Установить расширение - с помощью метода `УстановитьРасширениеРаботыСКриптографиейАсинх()`, `УстановитьРасширениеРаботыСКриптографией()` или `НачатьУстановкуРасширенияРаботыСКриптографией()`. Это интерактивное действие, которое необходимо выполнить один раз для каждого пользователя локального компьютера, использующего расширение.
  - Подключить расширение - с помощью метода `ПодключитьРасширениеРаботыСКриптографиейАсинх()`, `ПодключитьРасширениеРаботыСКриптографией()` или `НачатьПодключениеРасширенияРаботыСКриптографией()`.
- Внешние компоненты - более подробно о работе с внешними компонентами см. здесь.

4.8. Особенности различных вариантов запуска системы

Система «1С:Предприятие» может использоваться в файловом и клиент-серверном вариантах, во внешнем соединении, а также в виде Интернет-сервисов.

Конфигуратор позволяет настроить использование процедур и функций общих модулей и модулей объектов для каждого варианта.

4.8.1. Исполнение процедур и функций

Для указания разрешения применения процедур и функций различных модулей (про виды модулей см. здесь) используют инструкции препроцессору и директивы компиляции.

4.8.1.1. Различие инструкций препроцессора и директив компиляции

Инструкции препроцессора и директивы компиляции предназначены для того, чтобы оставить в скомпилированном модуле только то, что действительно должно присутствовать в том или ином контексте. При этом инструкции препроцессора действуют на исходный текст модуля (т. е. удаляют из модуля текст, который не может в нем находиться), а директивы компиляции