

# СИСТЕМА ОРКЕСТРАЦИИ

Руководство по эксплуатации

Версия 9.2.2

nexign

Настоящая документация может быть использована только для поддержки работоспособности продуктов, установленных на основании договора с АО «Нэксайн». Документация может быть передана на основании договора, по которому производится (производилась или будет производиться) установка продуктов, или явно выраженного согласия АО «Нэксайн» на использование данной документации. Если данный экземпляр документации попал к вам каким-либо иным образом, пожалуйста, сообщите об этом в АО «Нэксайн» по адресу, приведенному ниже.

Все примеры, приведенные в документации (в том числе примеры отчетов и экранных форм), составлены на основании тестовой базы АО «Нэксайн». Любое совпадение имен, фамилий, названий компаний, банковских реквизитов и другой информации с реальными данными является случайным.

Все встречающиеся в тексте торговые знаки и зарегистрированные торговые знаки являются собственностью их владельцев и использованы исключительно для идентификации программного обеспечения или компаний.

Данная документация может не отражать некоторых модификаций программного обеспечения. Если вы заметили в документации ошибки или опечатки или предполагаете их наличие, пожалуйста, сообщите об этом в АО «Нэксайн».

Все имущественные авторские права сохраняются за АО «Нэксайн» в соответствии с действующим законодательством.

© АО «Нэксайн», 1992–2023

АО «Нэксайн»

Россия, 199155, Санкт-Петербург, ул. Уральская, д.4 лит.Б, помещение 22Н

Тел.: + 7 (812) 326-12-99; факс: + 7 (812) 326-12-98.

[office@nexign.com](mailto:office@nexign.com); [www.nexign.com](http://www.nexign.com)

# Содержание

<b>1. Общие сведения</b>	<b>5</b>
<b>2. Запуск и остановка</b>	<b>6</b>
<b>3. Администрирование CRAB через центр управления</b>	<b>7</b>
3.1. Проверка состояния компонентов CRAB_AKKA	7
3.2. Управление заявками	8
3.2.1. Поиск и просмотр зарегистрированных заявок	9
3.2.2. Добавление и клонирование заявок	13
3.2.3. Отмена и перезапуск исполнения заявок	16
<b>4. Обработка заявок</b>	<b>18</b>
4.1. Жизненный цикл заявки	18
4.2. Регистрация заявок	18
4.3. Контроль доступа к внешним ресурсам	19
4.3.1. Общие сведения	19
4.3.2. Последовательная обработка заявок	19
4.4. Распределение заявок	20
4.4.1. Управление очередностью исполнения заявок	20
4.4.2. Распределение заявок в соответствии с рейтами по сценариям	20
4.4.3. Автоматическое ограничение потоков заявок	21
4.4.4. Ручная остановка и возобновление потоков заявок	24
4.5. Исполнение сценариев	26
4.5.1. Общие сведения	26
4.5.2. Начало исполнения сценария	27
4.5.3. Исполнение workflow-сценариев	28
4.5.3.1. Общие сведения	28
4.5.3.2. Workflow с приостановкой и активацией исполнения заявок по тайм-ауту	29
4.5.3.3. Workflow с асинхронными взаимодействиями с ожиданием ответа	29
4.5.3.4. Опрос состояния внешней системы	31
4.5.3.5. Исполнение сценария при перезапуске	33
4.5.4. Автоматический переповтор заявок	33
4.5.4.1. Взаимодействие с внешними системами	33
4.5.4.2. Классификация ошибок синхронных взаимодействий	34
4.5.4.3. Классификация ошибок асинхронных взаимодействий	38
4.5.4.4. Переповторы синхронных взаимодействий	39
4.5.4.5. Переповторы асинхронных взаимодействий	41
4.6. Активация заявок	43
4.6.1. Изоляция SLEEP-заявок для активации по возрасту	43
4.7. Прерывание обработки заявок	44
4.8. Зависимая обработка заявок	44
4.8.1. Зависимости между заявками	44
4.8.2. Режимы зависимой обработки заявок	45
4.8.3. Активация режима reactive	46
<b>5. Обеспечение отказоустойчивости CRAB_AKKA</b>	<b>47</b>

5.1. Обеспечение отказоустойчивости распределителей . . . . .	47
5.2. Обеспечение целостности кластера . . . . .	48
5.3. Ограничения реализации . . . . .	49
5.4. Гарантированная активация заявок. . . . .	49

# 1. Общие сведения

Продукт «Система оркестрации (Nexign Orchestration Engine, **CRAB**)» и далее по тексту – **CRAB** является системой исполнения интеграционных сценариев и позволяет автоматизировать процессы в распределенной информационной среде, когда взаимодействующие программные продукты установлены на разных серверах, филиалах и центрах обработки данных.

Настоящее руководство – программный документ с описанием порядка и правил администрирования **CRAB**, включая процедуры:

- администрирования продукта через центр управления;
- обработки заявок;
- обеспечения отказоустойчивости компонентов **CRAB**.

Руководство предназначено для персонала эксплуатации, внедрения и служб технической поддержки продукта.

## 2. Запуск и остановка

Процедура запуска и остановки продукта описана в руководстве по установке [*CRAB-DOC\_INSTALL*].

## 3. Администрирование CRAB через центр управления

Центр управления (UI) **CRAB** предоставляет пользовательский интерфейс для работы с продуктом и позволяет:

- контролировать состояние ядра **CRAB** на одной площадке;
- управлять заявками на выполнение бизнес-сценариев.

Доступ в UI задается параметрами пользовательского интерфейса **CRAB** и в зависимости от настроек может быть:

- неавторизованным без необходимости пользовательской аутентификации (режим `public`, задан по умолчанию);
- авторизованным через LDAP-аутентификацию (режим `ldap`).

Порядок переключения режимов с `public` на `ldap` см. в Приложении А руководства по установке.



**ВАЖНО!** Для входа в UI при установленном режиме доступа `ldap` требуется ввод данных доменной учетной записи.

### 3.1. Проверка состояния компонентов CRAB\_AKKA

Для проверки следует:

**Шаг 1.** На странице центра управления (UI) **CRAB\_AKKA** открыть раздел *Схема развертывания* ([Рис. 1](#)).


CRAB_AKKA 4.5.0			
Схема развертывания продукта			
Reception layer			
Наименование / IP	Изменение статуса	Прошло времени	
REST API			
srv7-namanta.net.billing.ru:8080	Вчера, 15:27	19 ч, 21 мин.	
AMQP CONSUMER			
srv7-namanta.net.billing.ru:2560	Вчера, 15:28	19 ч, 20 мин.	
Distribution layer			
Наименование / IP	Изменение статуса	Прошло времени	
DISPATCHER			
srv7-namanta.net.billing.ru:2552	Вчера, 15:27	19 ч, 20 мин.	
ACTIVATOR			
srv7-namanta.net.billing.ru:2570	Вчера, 15:28	19 ч, 20 мин.	
Execution layer			
Наименование / IP	Изменение статуса	Прошло времени	
WORKER			
srv7-namanta.net.billing.ru:2555	Вчера, 15:27	19 ч, 20 мин.	
Clustering layer			
Наименование / IP	Изменение статуса	Прошло времени	
SEED			
srv7-namanta.net.billing.ru:2552	Вчера, 15:27	19 ч, 20 мин.	

Рис. 1. Раздел *Схема развертывания* (UI) **CRAB\_AKKA**

Откроется страница «*Схема развертывания продукта*». На странице будет отображаться текущее состояние работающих узлов в виде таблиц, сгруппированных по уровням:

- Reception layer – уровень приема запросов от внешних продуктов и систем по протоколам HTTP и AMQP;
- Distribution layer – уровень передачи заявок на исполнение и обеспечения их асинхронного исполнения;
- Execution layer – уровень исполнения заявок и рассылки оповещений во внешние системы;
- Clustering layer – уровень/слой, отвечающий за формирование кластера.

В столбцах табличных форм каждого из уровней отображаются следующие данные.

Столбец	Содержание
Наименование / IP	Адрес и порт, которые использует узел
Изменение статуса	Дата и время последнего изменения статуса узла.  <div style="border: 1px solid #ccc; padding: 5px; background-color: #f0f0f0;">  <b>Внимание!</b> Остановленные и недоступные узлы в списке не отображаются                 </div>
Прошло времени	Время с момента последнего изменения статуса узла

**Шаг 2.** Убедиться, что требуемый узел отображается в заданном уровне/слое.

## 3.2. Управление заявками

Для управления заявками выберите раздел *Требования* центра управления CRAB (UI) ([Рис. 2](#)).

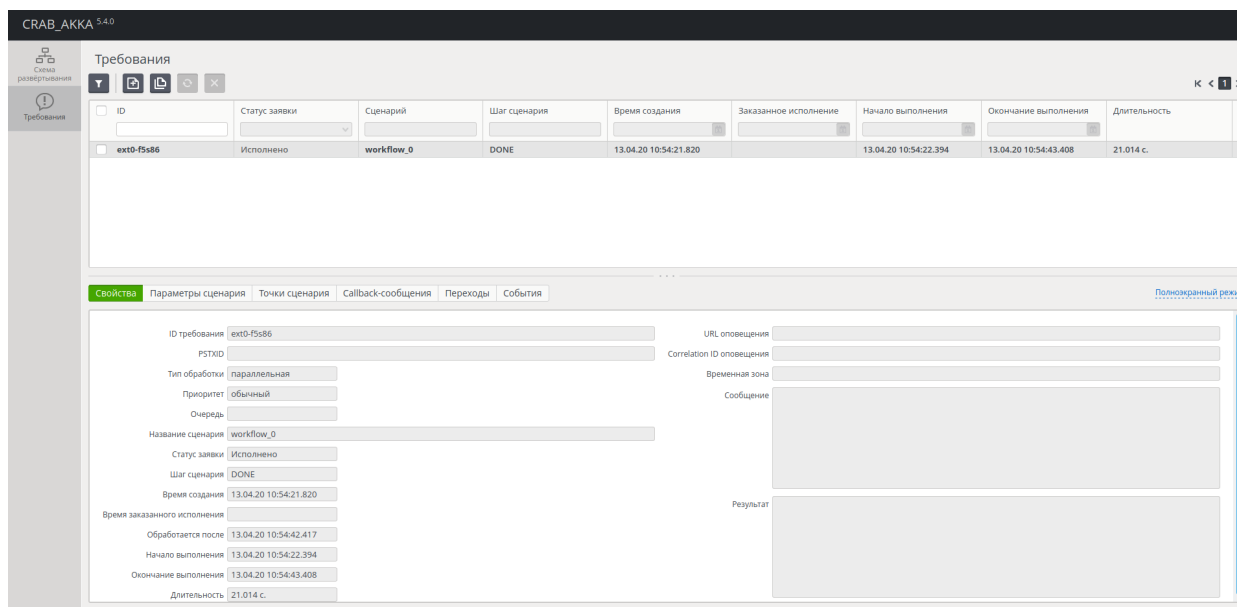


Рис. 2. Раздел Требования (UI) CRAB\_AKKA

Откроется страница «Требования» со строкой вкладок:

- *Свойства*;
- *Параметры сценария*;
- *Точки сценария*;



- *Callback сообщения;*
- *Переходы;*
- *События.*

### 3.2.1. Поиск и просмотр зарегистрированных заявок

Чтобы найти заявку и просмотреть её параметры:

**Шаг 1.** В центре управления (UI) **CRAB\_AKKA** откройте раздел *Требования*, введите в поле ID уникальный идентификатор заявки и нажмите на кнопку **Найти требования** (Рис. 3). Будет запущен поиск, результат которого отобразится в таблице зарегистрированных заявок.

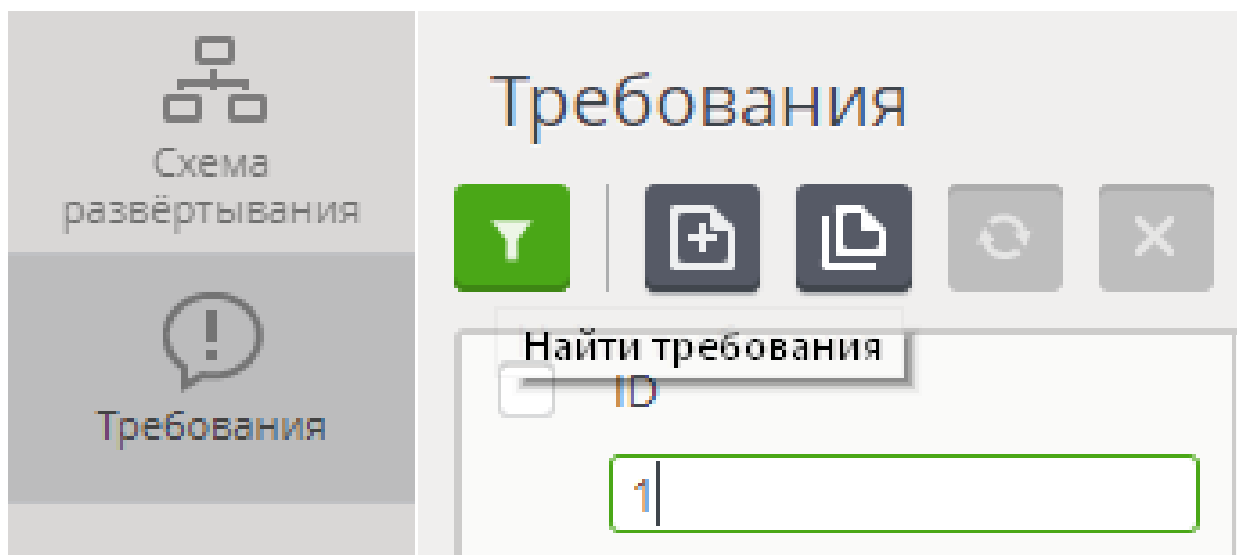


Рис. 3. Строка вкладок раздела Требования

**Шаг 2.** Откройте поочерёдно вкладки *Свойства* → *События* (Рис. 2).

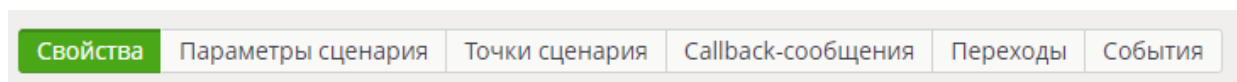


Рис. 4. Строка вкладок раздела Требования

Отобразятся параметры найденной заявки, описание которых приведено в [Табл. 1](#).

Табл. 1. Описание параметров заявки

Параметры	Описание
ID требования	Идентификатор заявки. Задается при ее добавлении в <b>CRAB</b> и используется для идентификации заказа на выполнение бизнес-сценария. Идентификатор должен быть уникальным в рамках внешней системы, для которой заказывается выполнение заявки, и в рамках <b>CRAB</b> . Идентификаторы не преобразуются, поэтому уникальность должна гарантироваться при регистрации заявок. Рекомендуется указывать краткое наименование внешней системы в качестве префикса в идентификаторе заявки.

Параметры	Описание
PSTXID	Уникальный идентификатор бизнес-транзакции ( <i>pstixd</i> ). Общая метка на уровне Единого Биллинга, по которой администраторы группируют операции в разных системах. Максимальная длина – 1024 любых символа (согласно RFC822). <b>CRAB</b> не проверяет корректность идентификатора. При выполнении заявок с <b>pstxid CRAB</b> выводит идентификатор в системный журнал <b>CRAB_AKKA</b> (узлы с ролью <i>worker</i> , <i>consumer</i> или <i>webapp</i> )
Тип обработки	Порядок выполнения заявки. Задается при ее добавлении в <b>CRAB</b> . Виды передачи на исполнение: <ul style="list-style-type: none"> <li>• последовательная – заявки разбиваются по очередям и выполняются строго в порядке добавления в <b>CRAB</b>. Используется для комплексных операций, когда важно соблюсти последовательность выполнения бизнес-сценариев (например, замена SIM-карты);</li> <li>• параллельная – заявки выполняются независимо друг от друга</li> </ul>
Приоритет	Приоритет заявки (необязательный параметр). Возможные значения: <ul style="list-style-type: none"> <li>• Высокий;</li> <li>• Низкий;</li> <li>• Обычный (по умолчанию).</li> </ul> <p>Приоритеты учитываются и обрабатываются только <b>CRAB_AKKA</b> и обеспечивают более быстрое распределение приоритетных заявок в <b>CRAB</b> и заданные гарантии передачи на исполнение для менее приоритетных заявок. Заявки передаются исполнителям (<i>worker</i>) в соответствии с алгоритмом приоритизации пропорционально весам. Веса задаются при помощи параметров (см. «Настройка развертывания»)</p>
Очередь	Название очереди, в которую добавлена заявка. Указывается только для заявок с последовательной обработкой
Название сценария	Уникальный идентификатор бизнес-сценария, который требуется выполнить. Если указан <i>Workflow</i> , принимает значение <i>workflow</i>

Параметры	Описание
Статус заявки	Текущее состояние выполнения заявки. Возможные значения: <ul style="list-style-type: none"> <li>• Новое – новая заявка, ожидающая выполнения;</li> <li>• Выполняется – выполняется;</li> <li>• Исполнена – выполнена успешно;</li> <li>• `Ошибка` – выполнение завершилось ошибкой;</li> <li>• Ожидает` – заявка готова к возобновлению выполнения (внесены исправления в <b>CRAB</b> или внешнюю систему);</li> <li>• Отложена – исполнение заявки отложено до момента исполнения заданных условий для исполнения (по ЖЦ в поле lifecycle);</li> <li>• Отменена – исполнение заявки остановлено и возобновляться не будет</li> </ul>
Шаг сценария	Название последнего state сценария. Зависит от workflow исполненного сценария
Время создания	Дата и время регистрации заявки в <b>CRAB</b> . Указывается согласно часовому поясу на компьютере, с которого выполнен вход в <i>UI</i> . При регистрации заявок для одного и того же абонента время создания определяется с точностью до 1 миллисекунды.
Время заказанного исполнения	Запланированные дата и время для исполнения заявки. Используется для отложенного запуска бизнес-процессов. Например, для смены тарифа со следующего месяца. Указывается согласно часовому поясу на компьютере, с которого выполнен вход в <i>UI</i> .
Обрабатывается после	Дата и время для передачи заявки на исполнение. Раньше этого времени заявка не исполняется. Указывается согласно часовому поясу на компьютере, с которого выполнен вход в <i>UI</i>
Начало выполнения	Дата и время начала выполнения заявки. Значение остается пустым, пока заявка не передана на исполнение. Указывается согласно часовому поясу на компьютере, с которого выполнен вход в <i>UI</i> . Заявки выполняются в порядке добавления в <b>CRAB</b> с учетом типа обработки. Если несколько однотипных заявок регистрируются одновременно, то они выполняются в порядке увеличения идентификатора ID заявки (от a до z и от 0 до 9). Например, если t123 и a124 имеют одинаковое время создания, то первой на исполнение передается заявка a124
Окончание выполнения	Дата и время окончания выполнения заявки. Значение остается пустым, пока выполнение не закончено. Указывается согласно часовому поясу на компьютере, с которого выполнен вход в <i>UI</i>
Длительность	Длительность выполнения заявки (в секундах)

Если для зарегистрированной заявки заказано оповещение о выполнении, на вкладке Свойства отобразятся параметры, приведённые в [Табл. 2](#).

Табл. 2. Таблица параметров заявки при заказе оповещения

Параметры	Описание
URL оповещения	<p>Обратный адрес для публикации оповещения от <b>CRAB</b> об исполнении. Указывается в формате <code>amqp://?&lt;query&gt;</code>. Параметры в подстроке <code>&lt;query&gt;</code>:</p> <ul style="list-style-type: none"> <li>• <code>exchange</code> – точка обмена для публикации оповещения;</li> <li>• <code>queue</code> – очередь, которую проверяет внешняя система;</li> <li>• <code>key</code> – ключ маршрутизации (<code>routing_key</code>) для публикации оповещения.</li> </ul> <p>Адрес задается в одном из форматов:</p> <ul style="list-style-type: none"> <li>• имя очереди (<code>queue</code>);</li> <li>• ключ маршрутизации (<code>key</code>);</li> <li>• точка обмена (<code>exchange</code>) и ключ маршрутизации (<code>key</code>).</li> </ul> <p>В обратном адресе допускается не указывать точку обмена (<code>exchange</code>). В этом случае <b>CRAB</b> публикует оповещения через собственную точку обмена, название которой задается в настройках развертывания. Текущее значение хранится на общем для федерального решения конфигурационном сервере ZooKeeper <code>/ps/config/apps/crab/rabbitmq/defaultExchangeName</code> . <code>"amqp://?key=ps.crab_status.brt"</code></p> <p>При таком адресе сообщение публикуется в точку обмена по умолчанию с ключом маршрутизации <code>ps.crab_status.brt</code>. Атрибут используется только совместно с <code>Correlation ID</code> оповещения. Формат оповещений см. в <i>CRAB-DOC_API</i></p>
Correlation ID оповещения	<p>Уникальный идентификатор для оповещения об исполнении. Идентификатор задает внешняя система при регистрации в <b>CRAB</b> и с помощью него сопоставляет свой запрос и ответ <b>CRAB</b> при асинхронных операциях.</p> <p><code>"8480f192-fb94-4d79-84c7-4f7bf86f23ea"</code></p> <p>Атрибут используется только совместно с URL оповещения</p>
Временная зона	<p>Временная зона, согласно которой передаются данные в оповещениях. По умолчанию – UTC</p>
Сообщение	<p>Комментарий к результату исполнения. Поясняет, успешно ли выполнена операция. Заполняется сценарием <b>CRAB</b></p>
Результат	<p>Результат выполнения. Заполняется сценарием <b>CRAB</b></p>

### 3.2.2. Добавление и клонирование заявок

Заявки подразделяются на два типа:

- *зависимые* – используются для комплексных операций, когда важна последовательность выполнения бизнес-сценариев;
- *независимые*.

Пример комплексной операции – обработка заявки на смену тарифного плана и одновременное подключение опций для нового плана. Сценарий смены тарифа запускается первым, потом добавляются опции, поэтому в **CRAB** регистрируются зависимые заявки.

Зависимые заявки помещаются в именные очереди согласно телефонному номеру абонента (MSISDN) и выполняются строго в порядке регистрации в **CRAB**.

Независимые заявки помещаются в общую очередь и выполняются независимо от порядка регистрации в **CRAB**.

#### Добавление независимых заявок

Чтобы добавить независимую заявку:

**Шаг 1.** В центре управления (UI) **CRAB\_АККА** откройте раздел *Требования* и нажмите **Добавить новое требование** ([Рис. 5](#)).

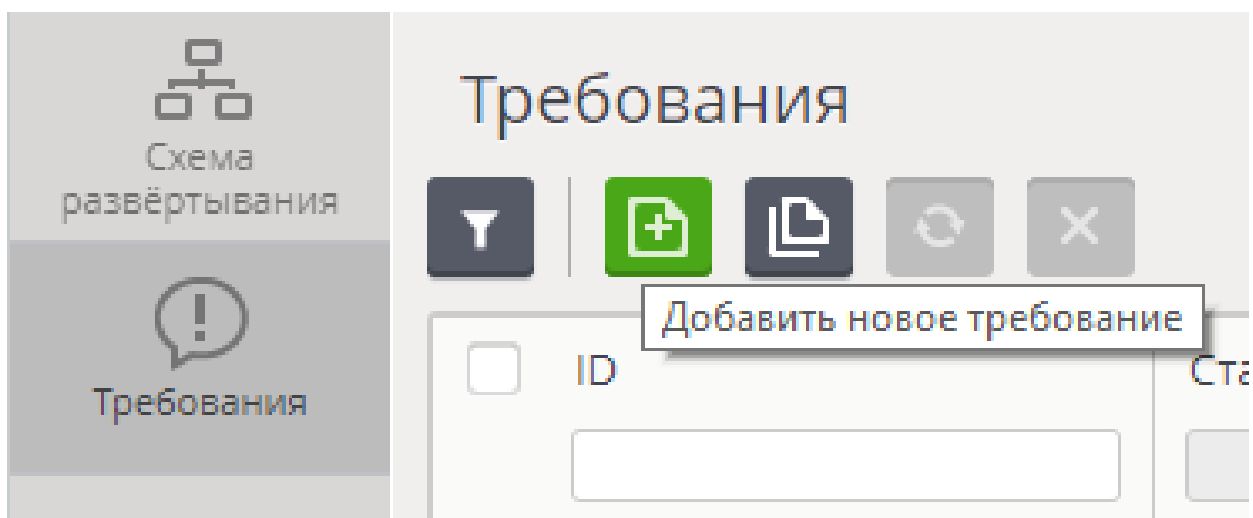


Рис. 5. Кнопка Добавить новое требование

Отобразится вкладка *Свойства* с параметрами новой заявки ([Рис. 6](#)).

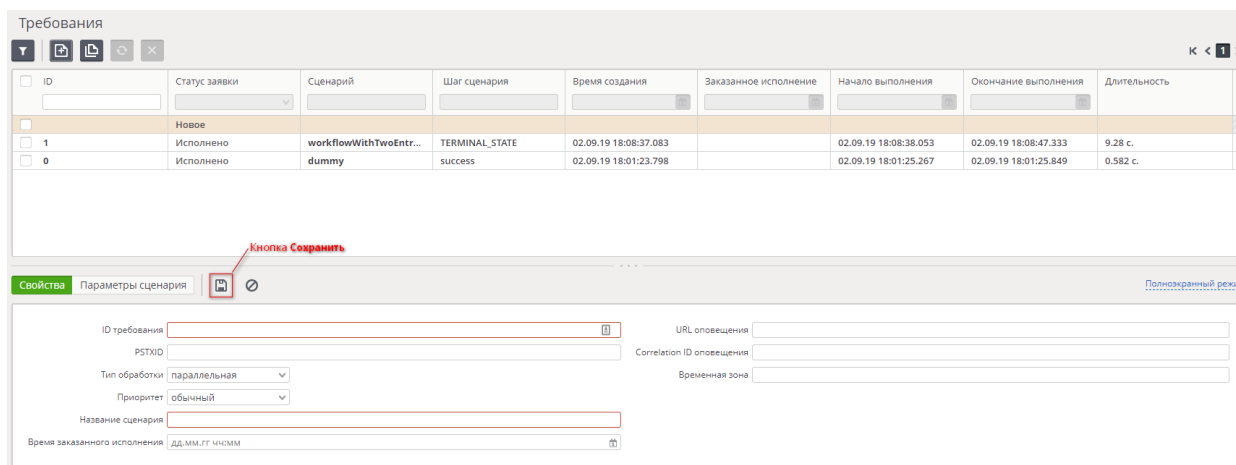


Рис. 6. Вкладка *Свойства* с параметрами новой заявки

**Шаг 2.** Заполните обязательные поля ID заявки, Название сценария, в поле Тип обработки выберите параллельная (задано по умолчанию), при необходимости в поле Время заказанного исполнения укажите отсрочку для выполнения заявки.

**Шаг 3.** При необходимости для заказа оповещения о выполнении заявки заполните поля:

- URL оповещения;
- Correlation ID оповещения;
- Временная зона.

**Шаг 4.** Если для бизнес-сценария требуются дополнительные параметры, на вкладке *Параметры сценария* (Рис. 7) заполните поля Параметр, Тип, Значение и нажмите **Добавить параметр**.

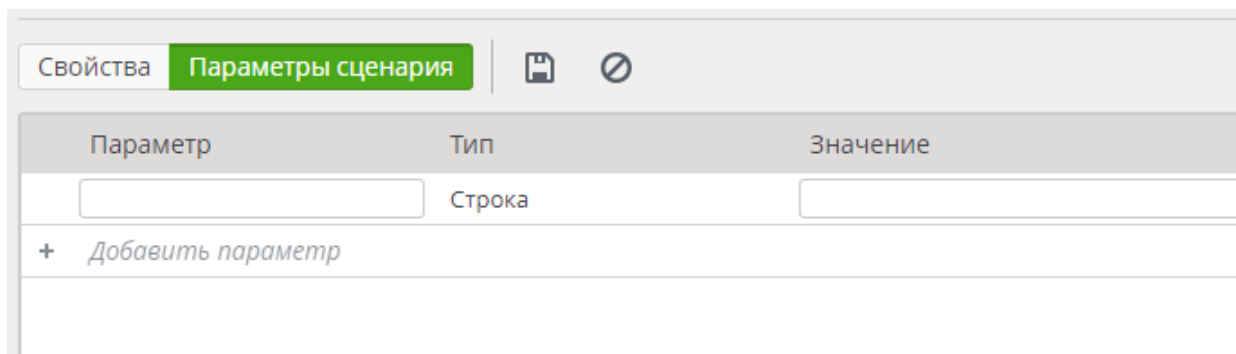


Рис. 7. Вкладка *Параметры сценария*

**Шаг 5.** Нажмите на кнопку **Сохранить**. Новая зарегистрированная заявка будет добавлена в список автоматически.

Для удаления заявки без регистрации в **CRAB** нажмите кнопку **Отмена** и подтвердите операцию.

### Добавление зависимых заявок

Чтобы добавить зависимую заявку:

**Шаг 1.** В центре управления (UI) **CRAB\_AKKA** откройте раздел *Требования* и нажмите **Добавить новое требование** (см. Рис. 5). Отобразится вкладка *Свойства с параметрами новой заявки* (см. Рис. 6).

**Шаг 2.** Заполните обязательные поля ID требования, Название сценария, в поле Тип

обработки **выберите** последовательная (задано по умолчанию), при необходимости в поле *Время заказанного исполнения* укажите отсрочку для выполнения заявки.

**Шаг 3.** При необходимости для заказа оповещения о выполнении заявки заполните поля:

- URL оповещения;
- Correlation ID оповещения;
- Временная зона.

**Шаг 4.** Если для бизнес-сценария требуются дополнительные параметры, на вкладке *Параметры сценария* (см. [Рис. 7](#)) заполните поля *Параметр*, *Тип*, *Значение* указанными в таблице значениями и нажмите **Добавить параметр**.

Атрибут	Значение
Параметр	SubscriberId
Тип	Строка
Значение	Номер абонента (MSISDN), к которому относится данная заявка

**Шаг 5.** Нажмите на кнопку **Сохранить**. Новая зарегистрированная заявка будет добавлена в список автоматически.

Для удаления заявки без регистрации в **CRAB** нажмите на кнопку **Отмена** и подтвердите операцию.

### Клонирование заявок

Чтобы клонировать заявку:

**Шаг 1.** В центре управления (UI) **CRAB\_AKKA** откройте раздел *Требования*, выберите заявку и нажмите **Клонировать требование** ([Рис. 8](#)).

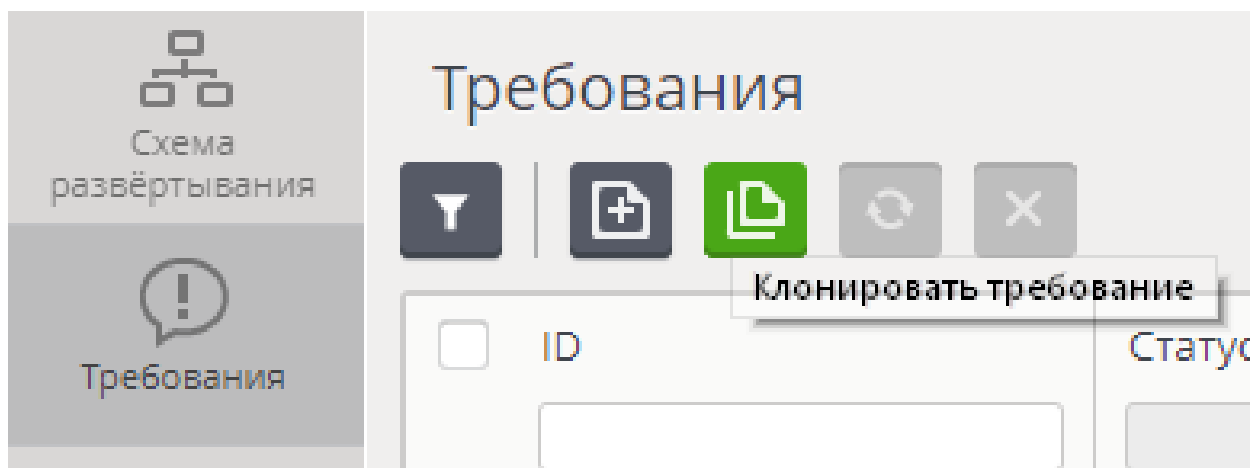


Рис. 8. Кнопка Клонировать требование

Параметры клонируемой заявки отобразятся на вкладке *Свойства* (см. [Рис. 6](#)).

**Шаг 2.** При необходимости отредактируйте параметры и нажмите на кнопку **Сохранить** (см. [Рис. 6](#)). Склонированная заявка будет добавлена в список автоматически.

Для удаления заявки без регистрации в **CRAB** нажмите кнопку **Отмена** и подтвердите операцию.

### 3.2.3. Отмена и перезапуск исполнения заявок

Если в ходе исполнения заявки возникли ошибки, и она перешла в статус *Ошибка (Fail)*, можно:

- отменить заявку;
- устранить причины ошибки и перезапустить исполнение заявки.

Новая или перезапущенная заявка может перейти в статус *Отложено (Postponed)* до наступления условий исполнения, заданных в поле ЖЦ (*lifecycle*).

Пример условия: переход одной или нескольких других заявок в определенное состояние по Workflow. Идентификаторы заданных заявок и целевые состояния определяются полем *lifecycle*. Посмотреть его значение можно в теле ответа на запрос `/orders/{externalId}` (доступно через консоль браузера при работе с UI).

#### Отмена заявки

В центре управления (UI) **CRAB\_AKKA** откройте раздел *Требования*, установите флажок в строке заявки в статусе *Ошибка* и нажмите **Отменить выполнение** (Рис. 9).

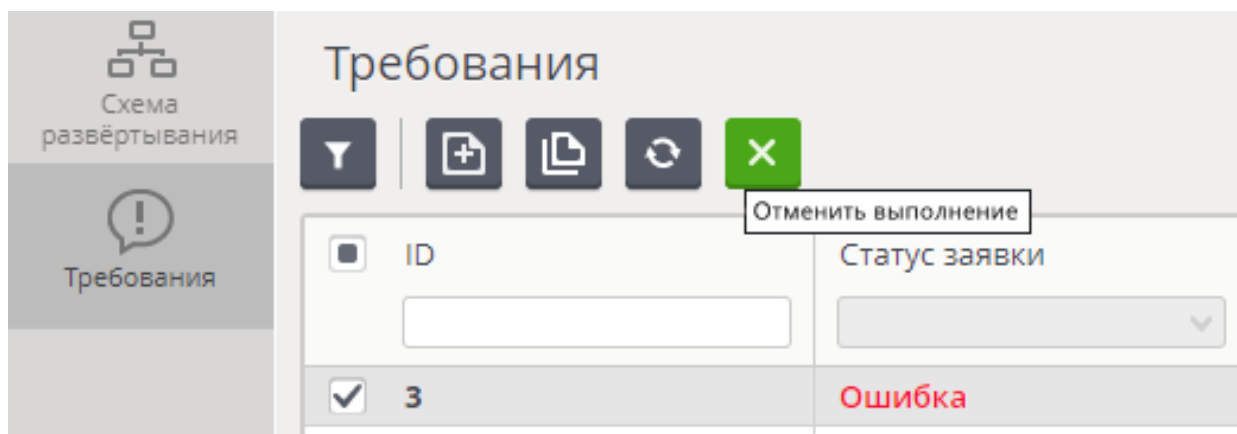


Рис. 9. Кнопка Отменить выполнение

Статус заявки в списке будет изменён автоматически на *Отменено* (Рис. 10).

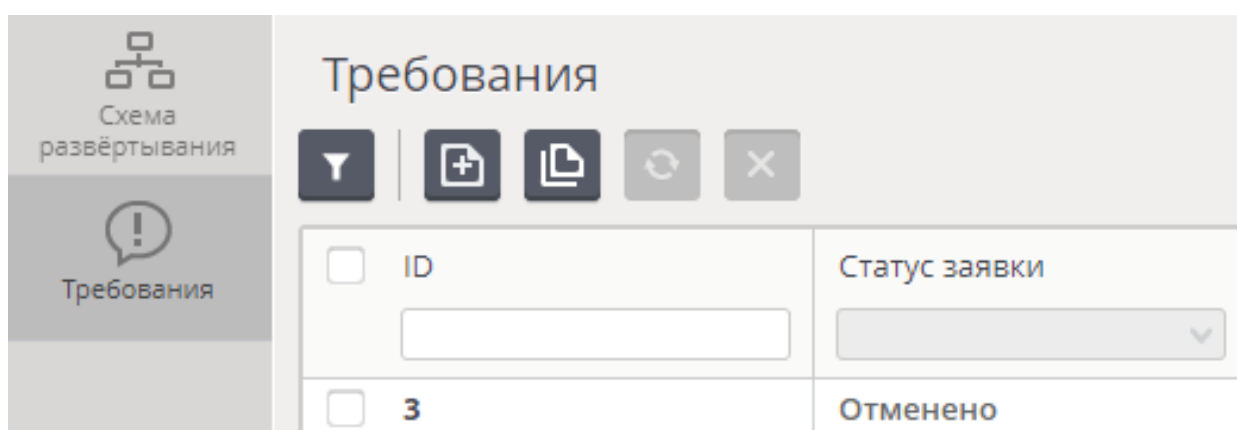


Рис. 10. Статус заявки «Отменено»

#### Перезапуск исполнения заявки

В центре управления (UI) **CRAB\_AKKA** откройте раздел *Требования*, установите флажок в



строке заявки в статусе **Ошибка** и нажмите **Перезапустить** ([Рис. 11](#)).

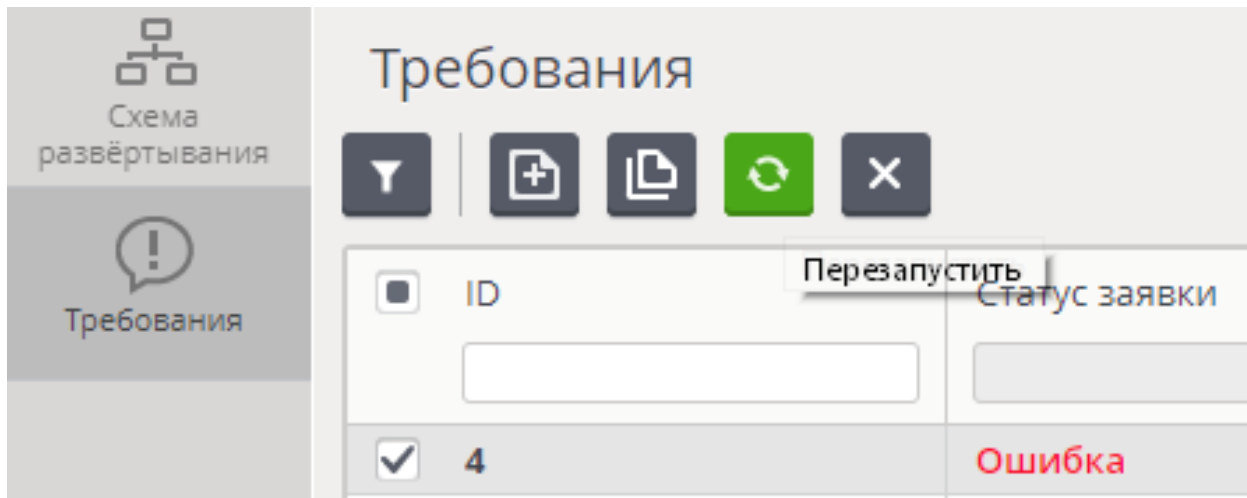


Рис. 11. Кнопка Перезапустить

Исполнение заявки будет возобновлено с последнего успешного шага сценария. Статус заявки в списке будет изменён автоматически на **Перезапущено** ([Рис. 12](#)).

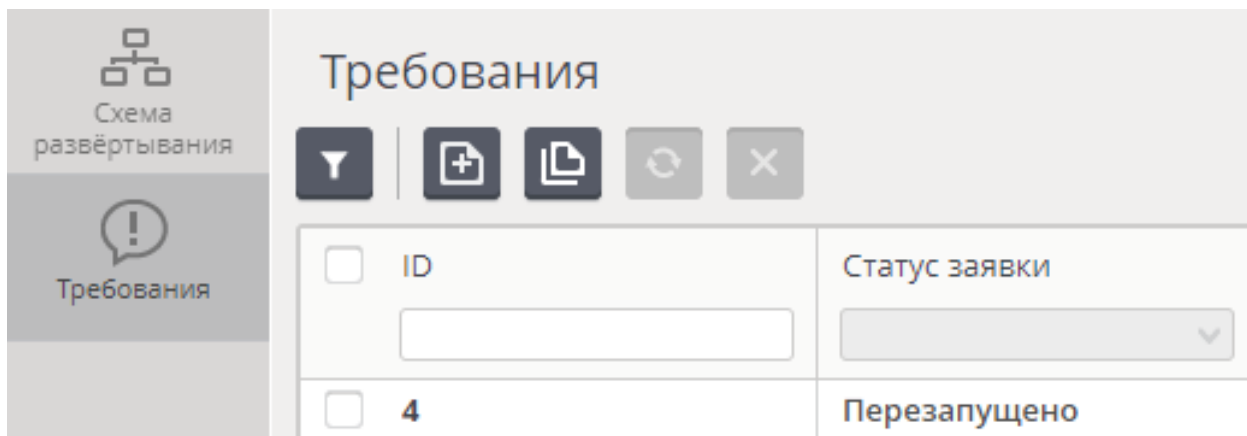


Рис. 12. Статус заявки «Перезапущено»

## 4. Обработка заявок

Заявки обрабатываются в соответствии с:

- жизненным циклом;
- общим алгоритмом (workflow) обработки с учетом возможности автоматического переповтора.

### 4.1. Жизненный цикл заявки

Жизненный цикл заявки изображен на [Рис. 13](#).

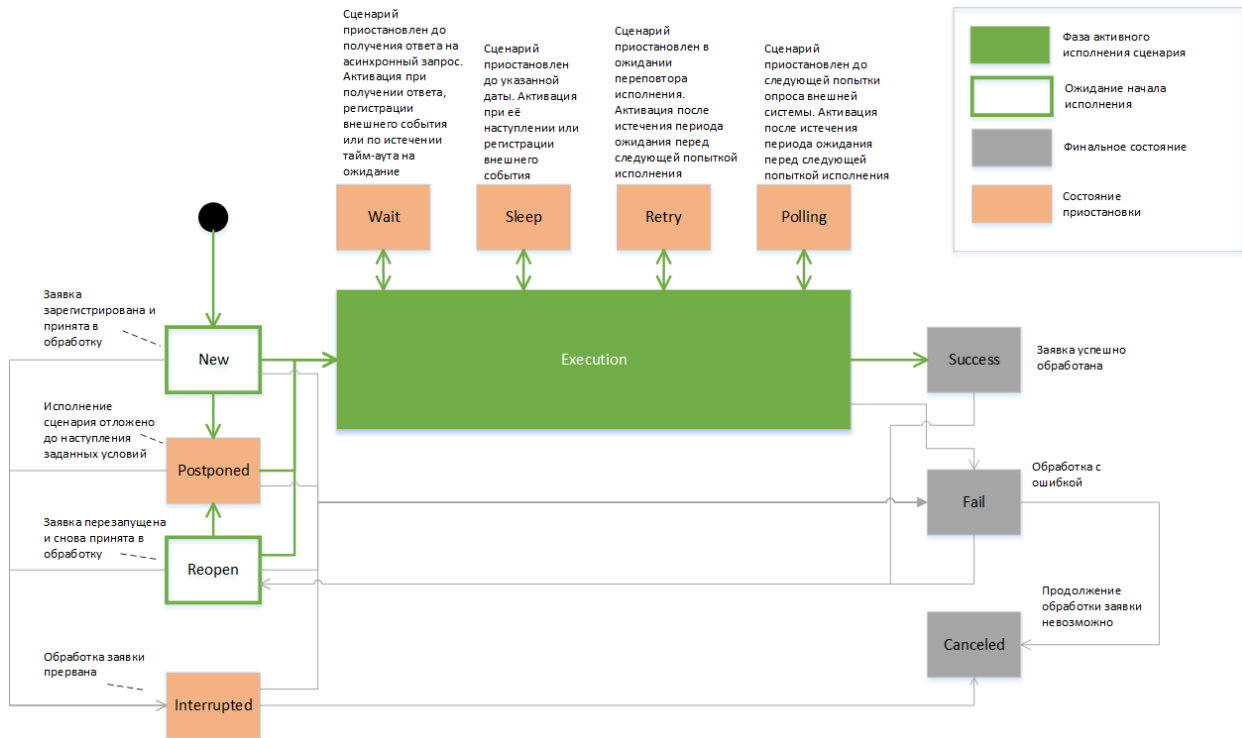


Рис. 13. Жизненный цикл заявки

Если в ходе обработки заявки возникли ошибки и заявке присвоен статус «Ошибка» (Fail), можно:

- отменить обработку заявки, если невозможно устранить причины ошибки и возобновлять обработку не требуется;
- устранить причины ошибки и перезапустить обработку заявки.

Новая или перезапущенная заявка может перейти в статус «Отложено» (Postponed) до наступления условий исполнения сценария, заданных в поле ЖЦ (lifecycle).

Переход одного или нескольких других заявок в определенное состояние по Workflow.

Идентификаторы заданных заявок и целевые состояния определяются полем lifecycle.

Посмотреть значение поля можно в теле ответа на запрос /orders/ {externalId} (доступно через консоль браузера при работе с UI).

### 4.2. Регистрация заявок

Заявки делятся на два типа:

- зависимые – используются для комплексных операций, когда важна последовательность выполнения бизнес-сценариев;
- независимые.

Пример комплексной операции – обработка заявки на смену тарифного плана и одновременное подключение опций для нового плана. Сценарий смены тарифа запускается первым, потом добавляются опции, поэтому в **CRAB** регистрируются зависимые заявки.

Зависимые заявки помещаются в именные очереди согласно телефонному номеру абонента (MSISDN) и выполняются строго в порядке регистрации в **CRAB**.

Независимые заявки помещаются в общую очередь и выполняются независимо от порядка регистрации в **CRAB**.

Заявка добавляется в нижней части рабочей области раздела заявки. Параметр `Тип обработки` задает тип обработки заявки:

- последовательная – для зависимых заявок;
- параллельная – для независимых заявок.

По умолчанию заявки выполняются в независимом режиме (параллельная обработка).

## 4.3. Контроль доступа к внешним ресурсам

Раздел содержит описания используемых терминов и последовательности обработки заявок.

### 4.3.1. Общие сведения

<i>Внешний ресурс</i>	Сущность внешней по отношению к <b>CRAB</b> системе, над которым интеграционные сценарии во время своего исполнения производят заданные операции. Примерами сущностей являются клиент, абонент, и т.п.
<i>Доступность внешнего ресурса</i>	Возможность выполнения операции над ресурсом в текущий момент
<i>Контроль доступа к внешнему ресурсу</i>	Определение доступности внешнего ресурса для конкретного сценария перед его исполнением
<i>Блокировка и разблокировка ресурса</i>	<ul style="list-style-type: none"> <li>• <i>блокировка ресурса</i> – захват внешнего ресурса сценарием в монопольное владение;</li> <li>• <i>разблокировка ресурса</i> – освобождение ресурса сценарием</li> </ul>

### 4.3.2. Последовательная обработка заявок

Для блокировки и разблокировки внешних ресурсов **CRAB** предоставляет механизм последовательной обработки заявок. Заявки, предполагающие монопольный доступ к какому-либо ресурсу, должны иметь последовательный тип обработки и одинаковое значение параметра `queue`. Ресурс, идентификатор которого указан в параметре `queue`, считается заблокированным заявкой, находящейся в начале очереди. Разблокировка ресурса происходит при завершении обработки заявки или её этапа.

Длительность блокировки ресурса определяется:

- длительностью исполнения синхронной части сценария. **CRAB** контролирует, чтобы длительность исполнения синхронной части не превышала 5 минут;

- временем ожидания ответа по асинхронному взаимодействию. В настоящий момент для последовательных заявок не может превышать 1 часа;
- количеством асинхронных взаимодействий. В настоящий момент не контролируется со стороны **CRAB**;
- временем нахождения заявки в состоянии повтора. Суммарное время не может превышать максимально допустимую длительность блокировки ресурса. При работе движка в режиме `ps.crab.crab_akka.execution.error_handling.retry_duration_control.mode`, с присвоенным значением `run_and_suspending_durations` суммарное время считается без учёта времени доставки заявки от момента её активации в состоянии `RETRY` до момента выполнения очередной попытки взаимодействия.

Максимальная длительность блокировки ресурса задаётся параметром `ps.crab.crab_akka.resource_control.max_acquisition_duration`.

Максимальная длительность блокировки ресурса в настоящий момент учитывается только в рамках синхронных и асинхронных повторов обработки заявок.

При смене типа обработки сценария с параллельного на последовательный условия проверки достижения максимальной длительности цикла могут быть ужесточены из-за приоритетного влияния параметра максимальной длительности блокировки ресурса.

## 4.4. Распределение заявок

Ядро **CRAB** позволяет распределять заявки:

- в соответствии с рейтами по сценариям;
- с автоматическим ограничением потоков заявок.

### 4.4.1. Управление очередностью исполнения заявок

При последовательной обработке заявка, перешедшая в состояние сна, удаляется из очереди и возвращается в неё после активации. Длительная приостановка одной заявки в этом случае рассматривается как изменение очередности заявок. Это позволяет избежать блокировки очереди.

### 4.4.2. Распределение заявок в соответствии с рейтами по сценариям

Ядро CRAB (CRAB\_AKKA) поддерживает режим работы с распределением заявок в соответствии с рейтами по сценариям ([Рис. 14](#)).

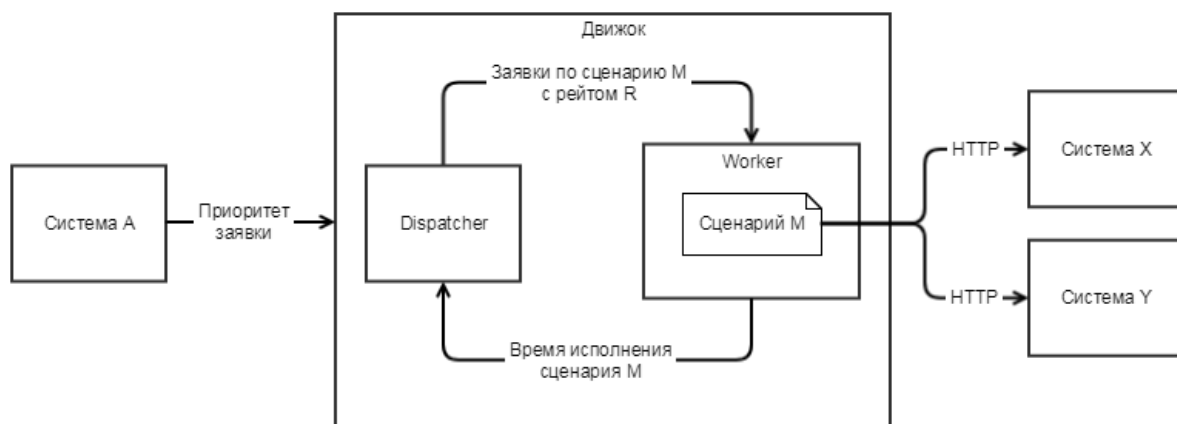


Рис. 14. Режим работы с распределением заявок в соответствии с рейтами по сценариям

Данный режим позволяет исключить взаимное влияние заявок по разным сценариям друг на друга при:

- вбросах больших пачек заявок по каким-либо сценариям;
- значительном увеличении длительности исполнения сценариев, которые обращаются к деградировавшим внешним системам.

В этих целях в сервисе распределения реализовано динамическое регулирование рейтов при обработке заявок с учетом времени их исполнения. Для этого:

- используется статистика длительности исполнения заявок, полученная от сервиса исполнения. Накопленная за долгосрочный период статистика длительности исполнения заявок хранится персистентно с возможностью ее полного восстановления при перезагрузках и переключениях на резервные узлы сервиса распределения;
- зарегистрированные заявки разбиваются на потоки (классы) по задаваемым наборам атрибутов. Потоки заявок маркируются путем добавления маркера потока в каждую заявку при ее создании;
- предусмотрена возможность настройки емкости диспетчера с помощью параметров:
  - `ps.crab.crab_akka.distribution.capacity.parallel_count`;
  - `ps.crab.crab_akka.distribution.capacity.sequential_count`.

По умолчанию режим распределения заявок по рейтам отключен. Включается с помощью настроечного параметра `ps.crab.crab_akka.distribution.mode` (режим распределения) с присвоенным значением `rated_flows`.

### 4.4.3. Автоматическое ограничение потоков заявок

#### Ограничение потока заявок

*Поток заявок* — множество заявок на выполнение однотипной работы. *Ограничение потока заявок* — контроль за количеством одновременно исполняемых этапов сценариев по заявкам потока.

Ограничение потоков заявок позволяет:

- обеспечивать максимальную пропускную способность **CRAB** за счёт работы на пределе пропускной способности внешних систем;
- минимизировать количество неуспешных сценариев при деградации/недоступности внешних систем;
- предотвращать избыточную нагрузку на внешние системы.

Настройка механизма ограничения потоков включает:

- активацию режима распределения потоков;
- настройку политик ограничения (лимитирования) потоков.

#### Режимы распределения

Ядро **CRAB (CRAB\_AKKA)** поддерживает следующие режимы распределения с ограничением потоков:

- `stopped_flows` — режим распределения с автоматической приостановкой потоков с последующим их возобновлением;
- `limited_flows` — режим распределения с автоматическим замедлением потоков с последующим их ускорением.

Под *приостановкой потока* в рамках настоящего документа понимается снижение максимально допустимого количества этапов в прогрессе до нуля, под *замедлением потока* — снижение максимально допустимого количества этапов в прогрессе, под *ускорением* —

увеличение максимально допустимого количества этапов в прогрессе.

Режим `limited_flows` включает возможности режима `stopped_flows` и обладает более широкими возможностями управления потоками заявок.

Распределитель (`dispatcher`) в режиме:

<code>stopped_flows</code>	<code>limited_flows</code>
<ul style="list-style-type: none"> <li>• обнаруживает недоступность внешней системы;</li> <li>• автоматически приостанавливает потоки заявок, для обработки которых требуется обращение к недоступной внешней системе;</li> <li>• обнаруживает доступность внешней системы;</li> <li>• автоматически возобновляет обработку заявок</li> </ul>	<ul style="list-style-type: none"> <li>• обнаруживает деградацию пропускной способности потока;</li> <li>• автоматически замедляет поток;</li> <li>• обнаруживает восстановление пропускной способности потока;</li> <li>• автоматически ускоряет поток.</li> </ul>

Распределение заявок с учётом ограничений потоков осуществляется только если потребление ёмкости распределителя не превышает заданного значения параметра `ps.crab.crab_akka.distribution.capacity.overload_threshold_percentage`. При его превышении распределение происходит в соответствии с правилами режима `rated_flows`. При возвращении потребления ёмкости к норме вновь вступают в силу правила распределения режима с ограничением потоков.

Для мониторинга состояния потока используются атрибуты `FlowLimits` и `StagesCountByFlowWithinActivityPeriod` JMX метрики `FlowsProcessing`.

Для контроля размера очереди по потоку используется атрибут `ReadyCountByPriorityAndFlow` JMX метрики `OrderDistributionCounts`.

## Активация режимов распределения

### Предварительная настройка режимов распределения с ограничением потоков заявок

Для обнаружения недоступности/деградации внешней системы при синхронном взаимодействии с ней должен быть настроен режим повторов и настроены политики лимитирования.

Для правильного функционирования `limited_flows` и `stopped_flows` требуется корректное задание целевых шагов сценариев при их разработке.



Если по неизвестной причине остановилось исполнение одного потока, следует:

- перегрузить *активный* узел распределителя;
- если это не дало положительного результата, — присвоить параметру `ps.crab.crab_akka.distribution.mode` значение `rated_flows` с последующей перезагрузкой всех узлов распределителя.

### Активация режима `stopped_flows`

Для активации режима необходимо параметру `ps.crab.crab_akka.distribution.mode` присвоить значение `stopped_flows` с последующей перезагрузкой всех узлов распределителя.

### Активация режима `limited_flows`

Для активации режима необходимо параметру `ps.crab.crab_akka.distribution.mode` присвоить значение `limited_flows` с последующей перезагрузкой всех узлов распределителя.

Перед включением режима рекомендуется настроить оповещение (алерт) в системе мониторинга, который будет сигнализировать о превышении 99-м перцентилем длительности успешных этапов со значением 2 часа.

В случае первого обнаружения срабатывания алерта необходимо сообщить об этом команде разработки, т.к. лимитирование для потоков с длинными этапами может работать с особенностями.

### Режим распределения с точностью до сценария

Режим распределения заявок, задаваемый в параметре `ps.crab.crab_akka.distribution.mode`, распространяется на все потоки заявок. Если заявки определённых сценариев требуется распределять в ином режиме, следует использовать политики лимитирования, задаваемые в параметре `ps.crab.crab_akka.distribution.flows.limits.policies` в Inventory продукта.

Применение политик лимитирования возможно только для режимов распределения `stopped_flows` и `limited_flows`.

Возможным значением режима распределения для определённых сценариев, задаваемым в параметре

`ps.crab.crab_akka.distribution.flows.limits.policies.<name>.params.mode`, является:

- `stopped_flows`;
- `limited_flows`.

```
distribution:
  mode: stopped_flows
  flows:
    limits:
      policies:
        name.strategy.1:
          scenarios:
            - name: "ccmCustomerOrder"
            - name: "ufmLifeCycleUpdate"
          params:
            mode: limited_flows
```

### Политики учёта ошибок при лимитировании потоков

*Лимитирующий класс ошибок* — класс ошибок, возникновение которых должно приводить к ужесточению ограничения потока вплоть до полной его остановки. Политики лимитирования задают сценарии, по которым должно осуществляться ужесточение ограничения по лимитирующим классам ошибок.

Политики задаются с помощью параметра

`ps.crab.crab_akka.distribution.flows.limits.error_handling.policies` с последующей перезагрузкой всех узлов **worker**.

```
...
policies:
  errors.policies:
    error_classes:
      - "trotting"
      - "crab.async.timeout"
    scenarios:
      - name: "workflowWithError"
      - name: "unstableWorkflow"
  errors.policies2:
    error_classes:
      - "service_unavailable"
    scenarios:
      - name: "workflowWithError"
```

В качестве лимитирующих классов ошибок можно использовать:

- внутренний класс `crab.async.timeout` (ошибка, возникающая при истечении времени ожидания ответа при асинхронном взаимодействии);
- внутренний класс `crab.polling.timeout` (ошибка, возникающая при истечении максимального времени опроса);
- классы ошибок, определённые при настройке автоматических переповторов.

Задаваемые классы ошибок и сценарии не могут превышать 64 символа, быть пустыми либо отсутствовать. Пары «класс ошибок + сценарии» должны быть уникальными.

#### 4.4.4. Ручная остановка и возобновление потоков заявок

##### Остановленный поток заявок

Остановленный поток заявок – поток, по которому прекращено исполнение сценариев.

Поток может быть остановлен автоматически движком **CRAB** или вручную сотрудником эксплуатации.

Ручная остановка потоков необходима для минимизации негативных последствий выполнения плановых работ на внешних системах и оборудовании.

Для минимизации негативных последствий во время плановых работ предполагается следующая последовательность действий:

- остановка потоков, на которые могут оказать влияние плановые работы;
- ожидание завершения плановых работ;
- возобновление остановленных потоков после завершения плановых работ.

Для потока, остановленного вручную:

- прекращается выдача на исполнение заявок, предполагающих исполнение:
  - нового этапа сценария;
  - очередной попытки после ухода на повтор при ошибках взаимодействий;
- прекращается выдача заявок сверх лимита;
- продолжают исполняться этапы сценариев, начатые до выполнения операции остановки потока.



Выход потока из состояния ручной остановки возможен только с помощью ручной операции возобновления.

### Операция остановки потока

Для остановки потоков используется асинхронная JMX-операция `FlowsProcessing.disableFlowsByFlowNameKeyword`.

Для остановки потоков требуется:

1. Определить шаблон имени для одновременной остановки нескольких потоков (например, в качестве шаблона можно использовать регион – "200").
2. Выполнить указанную JMX-операцию, передав в качестве параметра определённый шаблон имени потока.
3. Сверить возвращённый список имён поток с ожидаемым.
4. Убедиться, что в файлах журнала активного узла **DISPATCHER** появилась запись об остановке указанных потоков.
5. Убедиться, что значение метрики `FlowsProcessing.FlowLimits.flowState` для указанных потоков равно `DISABLED_MANUALLY (1)`.



При применении операции по остановке потока на время работ необходимо учитывать, что заявки в состоянии `WAIT` по этому потоку будут массово уходить в состояние `RETRY`. Чтобы избежать переполнения ёмкости активатора в режиме гарантированной активации (см. параметр `ps.crab.crab_akka.activation.mode` в `INVENTORY` продукта), необходимо вручную сдвинуть `activationTime` на время работ для заявок по остановленному потоку.

Пример запроса смещения `activationTime` из-за работ, продолжающихся ближайшие 3 часа:

```
UPDATE orders
SET activation_time = activation_time + INTERVAL '3' HOUR
WHERE activation_time < sysdate + INTERVAL '3' HOUR
  AND state_id = 10
  AND nvl(workflow_name, operation_name) in ('dummy',
'dummyWorkflow');
```

### Операция возобновления потока

Для возобновления потока, остановленного вручную, используется асинхронная JMX-операция `FlowsProcessing.enableFlowsByFlowNameKeyword`.

При возобновлении потока максимальное количество одновременно исполняемых этапов сценариев по заявкам потока (лимит по потоку) задаётся равным `ps.crab.crab_akka.distribution.flows.limits.recovery.initial_limit`. После этого поток переходит в режим автоматического управления потоком.

Операция возобновления может быть использована только для потоков, остановленных вручную.

Для возобновления потоков требуется:

1. Определить шаблон имени для одновременного возобновления нескольких потоков

- (например, в качестве шаблона можно использовать регион - "200").
2. Выполнить указанную JMX-операцию, передав в качестве параметра определённый шаблон имени потока.
  3. Сверить возвращённый список имён потоков с ожидаемым.
  4. Убедиться, что в файлах журнала активного узла **DISPATCHER** появилась запись о возобновлении указанных потоков.
  5. Убедиться, что значение метрики `FlowsProcessing.FlowLimits.flowState` для указанных потоков равно `UNDEFINED` (42).

## 4.5. Исполнение сценариев

Раздел включает подразделы с общими сведениями и детальным описанием процессов исполнения CRAB-сценариев.

### 4.5.1. Общие сведения

#### Сценарий

*Сценарии* — средство автоматизации процессов какой-либо предметной области. Процесс представляет собой набор определённых шагов и порядок их прохождения.

В зависимости от формы описания шагов процесса, порядка их прохождения и языка реализации сценарии делятся на:

- *operation-сценарии*, в которых шаги, их порядок и логика описываются с помощью [Apache Camel](#) Groovy DSL;
- *workflow-сценарии*, в которых шаги и их порядок описываются workflow-спецификацией, а логика – средствами [Apache Camel](#) Groovy DSL.

#### Шаги сценария

*Шаг сценария* — логически законченное действие в рамках заданного процесса. Шаги сценария по назначению представляют собой синхронное взаимодействие, асинхронные запросы, обработку асинхронных ответов и внешних событий интеграционного процесса. Исполнение сценария завершается при достижении терминальных шагов, которые подразделяются на *целевые* и *нецелевые*.

Целевые шаги задаются при разработке сценария в спецификации `Workflow` атрибутом `targetStates`.

Для сценариев с неопределёнными целевыми шагами будет применяться классификация, задаваемая настроечным параметром

`ps.crab.crab_akka.execution.state_classification`.

Например:

```
target:
- step: DONE
- step: ROLLED_BACK
```

Оставшиеся терминальные шаги будут считаться нецелевыми.

#### Этапы сценария

*Этап сценария* (`stage`) — один или более шагов сценария. В сценариях без приостановки

(sleep) всегда один этап. Каждая приостановка (sleep) добавляет новый этап.

## Статусы сценария

Статус сценария определяется достижимостью целевых терминальных шагов из текущего шага.

Возможные значения статуса:

undefined	Неопределенный статус, в случае если сценарий еще не исполнялся
target	Исполнение сценария завершено достижением целевого шага
on_target	Сценарий исполняется, целевой шаг достижим по workflow
off_target	Сценарий исполняется, но ни один целевой шаг уже недостижим по workflow
failed	Исполнение сценария завершено без достижения целевого шага

### 4.5.2. Начало исполнения сценария

В распределённой информационной среде при взаимодействии по асинхронным каналам передача данных между информационными системами может происходить с потерями, задержками и т.п. сбоями. При отсутствии ответа на запрос в установленное время система-клиент может запрос повторить либо сообщить о необходимости ручного вмешательства. При невозможности переповтора/вмешательства система-клиент будет продолжать исполнять сценарий, игнорируя ошибку или выполняя откат. Результатом может стать постоянная несогласованность («рассинхрон») данных между системами.

Для минимизации вероятности рассинхрона **CRAB\_AKKA** позволяет при обработке заявок задавать предельный срок начала исполнения сценария. Предельный срок вычисляется на основе параметров:

periodToStart	Период времени (в секундах), отведённый на начало исполнения сценария. Задаётся при создании заявки и не может быть изменён	Подробнее см. описание сущности Lifecycle
Crab-Request-Sent-Time	Время отправки клиентского запроса. Задаётся при выполнении запроса на регистрацию или перезапуск заявки	Подробнее см. описание реестра интерфейсов <b>CRAB</b>

По истечении предельного срока **CRAB\_AKKA** завершит обработку заявки ошибкой без исполнения сценария и отправит нотификацию с результатом (если она была заказана).

Предельный срок лишь минимизирует вероятность рассинхрона, для полной гарантии его отсутствия система-клиент должна дожидаться ответа.

Время начала исполнения сценария системой-клиентом определено как 50 минут при времени ожидания ответа сценарием – в один час. 10 минут – это запас времени на исполнение сценария, определенный сотрудником эксплуатации исходя из известных пиковых нагрузок в течение месяца.

Для мониторинга начала исполнения сценария используется атрибут StartStatsByScenario JMX метрики ExecutionStartControl. Ориентиром для выбора значения предельного срока

начала исполнения сценария может служить отношение метрик `totalStartPeriodSec / totalStartCount`.

### 4.5.3. Исполнение workflow-сценариев

Подраздел содержит сведения о workflow с приостановкой и активацией исполнения заявок по тайм-ауту, workflow с асинхронными взаимодействиями с ожиданием ответа, опросах состояния внешней системы и исполнением сценариев при перезапуске.

#### 4.5.3.1. Общие сведения

Согласно концепции [UML State Machine](#) процесс (Workflow) исполнения заявок в статусе Execution можно описать набором состояний (states) и переходов (transitions) между состояниями в результате определенных событий (events) ([Рис. 15](#)).

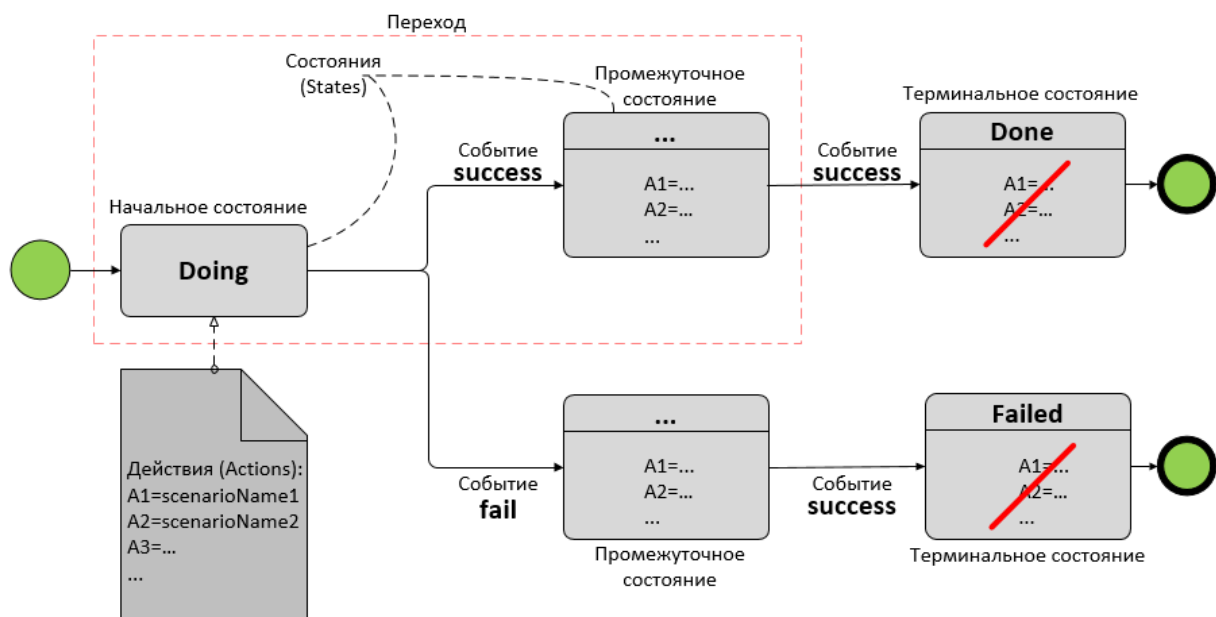


Рис. 15. Алгоритм (workflow) обработки заявок



Подробнее Workflow описан в *руководстве по эксплуатации* продукта.

Workflow имеет начальное (`initialState`) и конечное состояния. Конечное состояние определяется флагом `terminal`.

Состояния содержат наборы действий (actions), где действие – это предустановленный сценарий **CRAB**. В терминальном состоянии actions невозможны.

Максимальное количество состояний равно 30. Факт перехода из одного состояния в другое сохраняется в базе данных.

Для любого состояния (Done/Failed) заявка считается обработанной.

Правила переходов между состояниями:

- по событию `success` осуществляется переход в состояние `Done`;
- по событию `fail` – в состояние `Failed`.

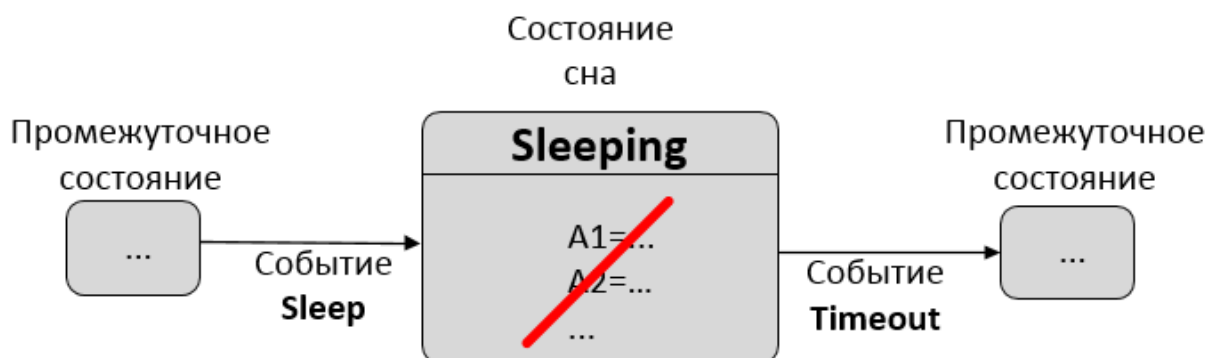
В зависимости от значения обязательного атрибута `operationName` (подробнее об атрибутах см. в *Описании программного интерфейса*, раздел "Модели данных") обработает заявку следующим образом:

- для `operationName` со значением "workflow" - по алгоритму `Workflow`;
- для `operationName` со значением идентификатора сценария - по алгоритму интеграционного сценария, сохранив полную обратную совместимость.

#### 4.5.3.2. Workflow с приостановкой и активацией исполнения заявок по тайм-ауту

Для описания приостановки и активации исполнения заявки в `Workflow` появляются дополнительные сущности:

- состояние сна;
- событие `Sleep`;
- событие `Timeout`.



Описание `Workflow`:

1. Сценарий генерирует событие `Sleep` о приостановке `Workflow`.
2. Система сохраняет `workflow`-контекст и дату активации `Workflow` в базе данных. Формат даты - `YYYY-MM-DDThh:mm:ss[.sss]±hh:mm`.
3. `Workflow` переходит в состояние `Sleeping`.
4. При наступлении времени активации по событию `Timeout` `Workflow` переходит в следующее состояние.

Ограничения для состояния сна:

- в состоянии сна можно находиться не менее 20 секунд. Максимальное значение не ограничено, но при разработке сценариев нужно учитывать, что незавершенные заявки старше максимального периода хранения (`ps.crab.crab_akka.db.orders.storage_period_in_days.in_progress`, по умолчанию 180 дней) могут быть принудительно удалены из хранилища;
- Выйти из состояния можно по событию `Timeout`;
- Состояние не может содержать `actions` (сценарии).

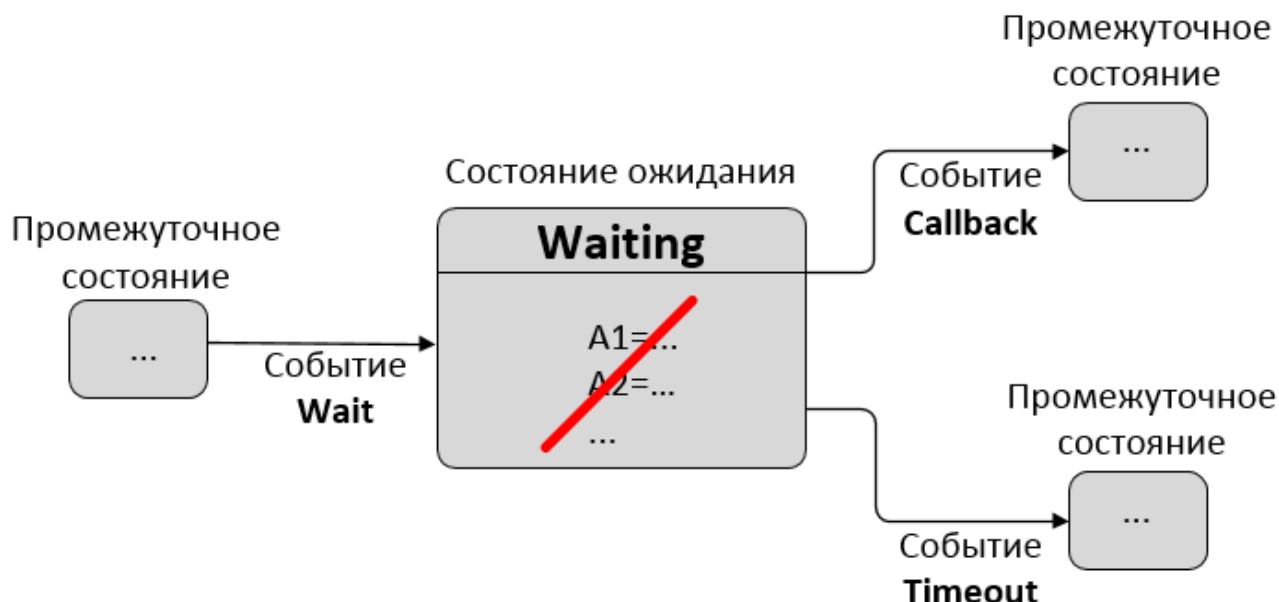
Об ограничениях состояний см. также в *Описании программного интерфейса* в разделе «`StatesObject` – описание состояний».

#### 4.5.3.3. Workflow с асинхронными взаимодействиями с ожиданием ответа

Для описания `workflow` с асинхронными взаимодействиями с внешними системами

используются сущности:

- событие ожидания результатов асинхронного взаимодействия — `Wait`;
- событие получения результатов взаимодействия — `Callback`;
- событие истечения времени ожидания результатов — `Timeout`;
- состояние ожидания результатов асинхронного взаимодействия — `Waiting`.



Ограничение для состояния `Waiting` — состояние не может содержать `actions` (сценарии).

`Workflow` реализует шаги сценария:

- приостановку сценария по событию `Wait` и перевод заявки в состояние `Waiting` на время, заданное сценарием;
- активацию заявки по событию `Callback`;
- активацию заявки по событию `Timeout`;
- передачу в сценарий результатов взаимодействия.

### Приостановка сценария и перевод заявки в состояние `Waiting`

Сценарий инициирует одно или несколько асинхронных взаимодействий, выполняя запросы к внешней системе, затем сообщает `CRAB_AKKA`:

- информацию о необходимости дождаться результатов взаимодействий (`Callback`);
- список выполненных запросов;
- максимальное время ожидания результатов (`Timeout`) по всем запросам.



На значение максимального времени ожидания результатов действует ограничение:

- для последовательных заявок оно не может превышать 3600 секунд;
- для параллельных — 8 035 200 секунд (3 месяца).

`CRAB_AKKA` по событию `Wait` переводит заявку в состояние `Waiting` и регистрирует все

запросы, по которым ожидается `Callback`.

#### Активация заявки по событию `Callback` по всем запросам

`CRAB_AKKA` регистрирует поступающие `Callback` от внешних систем. После регистрации всех `Callback` в текущем состоянии `Waiting` `CRAB_AKKA` активирует заявку.

#### Активация заявки по событию `Timeout`

При отсутствии `Callback` хотя бы для одного запроса в рамках текущего состояния `Waiting` `CRAB_AKKA` активирует заявку по зарегистрированному событию `Timeout`.

#### Передача результатов в сценарий

Перед возобновлением исполнения `CRAB_AKKA` передает сценарию все зарегистрированные результаты по запросам текущего состояния `Waiting`.

Логика обработки отсутствия результатов по всем или части запросов определяется разработчиком сценария.

#### 4.5.3.4. Опрос состояния внешней системы

В распределённой информационной среде асинхронное взаимодействие информационных систем может осуществляться с опросом состояния внешней системы ([Рис. 16](#)). В отличие от других типов асинхронного взаимодействия `workflow` с опросом применяется, если:

- внешняя система не способна присылать результат исполнения асинхронной операции;
- перед продолжением исполнения сценария нужно дождаться требуемого состояния одной системы после взаимодействия с другой системой.

Альтернативой механизма опроса, предоставляемого ядром `CRAB`, является самостоятельная реализация опроса разработчиком сценария. Но использование специального механизма опроса, предоставляемого `CRAB_AKKA`, обеспечивает следующие преимущества:

- упрощает `workflow` спецификации;
- предотвращает разработку избыточного кода в `workflow`-сценариях по реализации опроса состояния внешней системы;
- избавляет от хранения избыточных данных (`workflow`-переходов и т.д.).

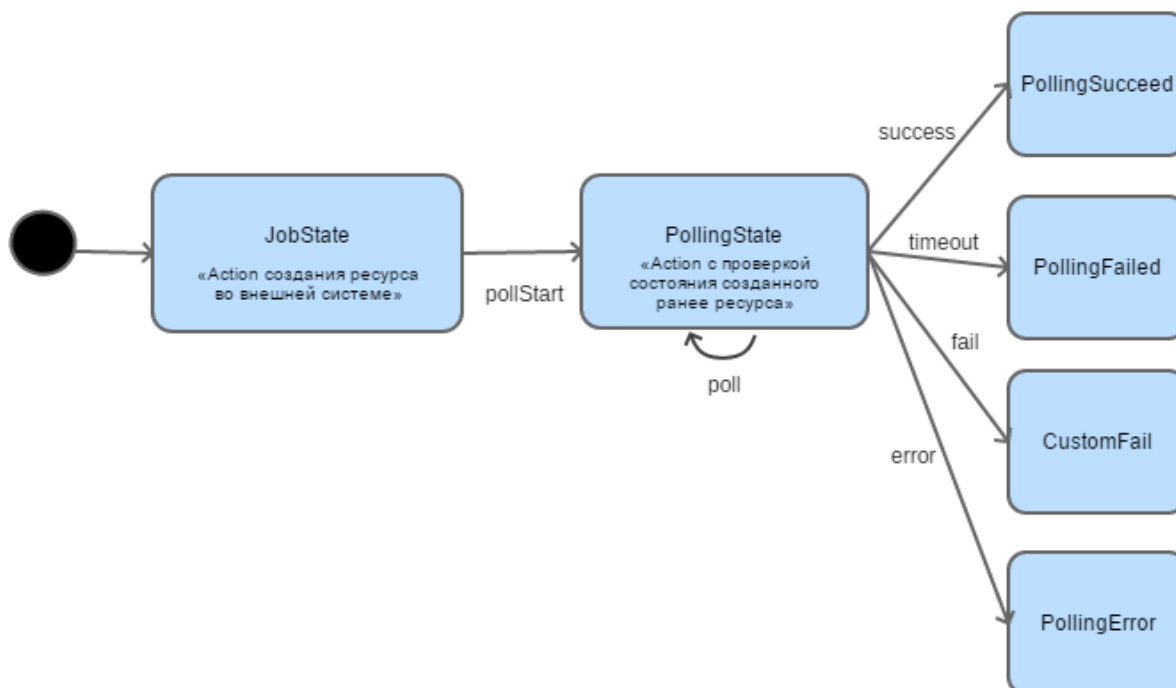


Рис. 16. Workflow опроса внешней системы

Состояние, в котором происходит опрос, называется *состоянием опроса* (*polling*). Состояние опроса может иметь несколько *actions*.

Алгоритм *polling*'а представляет собой цикл из периодического синхронного запроса к внешней системе о состоянии некоторого ресурса и последующего анализа сценарием результата запроса. Если:

- результат опроса ожидаемый — результат опроса считается успешным (событие *success*) и исполнение сценария продолжается;
- ожидаемый результат еще не достигнут — уход на новую попытку (событие *poll*);
- ожидаемый результат уже не достижим — результат опроса считается неудачным (событие *fail*).

Цикл опроса не прерывает этап исполнения сценария и блокирует очередь последовательных заявок. Между попытками опроса заявка находится в отдельном состоянии — *опрос* (*POLLING*).

Если же истекло общее время, отведенное на опрос, **CRAB\_AKKA** после очередной попытки генерирует *workflow*-событие *timeout*.

Перед началом опроса сценарий передает ядру **CRAB** параметры опроса:

- значение начальной задержки перед первой попыткой опроса;
- интервал между попытками опроса;
- максимальную длительность опроса.

Конкретные значения параметров опроса в каждом сценарии задаются разработчиком сценария.

В каждой попытке опроса исполнение начинается с первого *action* в состоянии опроса.

Переповторы синхронных запросов в состоянии опроса работают как обычно:

- в соответствии с настроенными классами ошибок и политиками повтора;
- повтор происходит с *action*, в котором произошла ошибка.



Для мониторинга активации заявок при опросе используется атрибут `StatsByActivationFlow` JMX метрики `OrdersActivation`.

#### 4.5.3.5. Исполнение сценария при перезапуске

Пользователь/Клиент перезапускает заявку. Система определяет начальное состояние при перезапуске (по зафиксированной последовательности переходов как самое последнее посещенное состояние, являющееся одной из точек входа, – `EntryPoint`).

Система сбрасывает текущие результаты исполнения заявки, переводит заявку в статус «`Reopen`» и активирует её исполнение.

Существующие ограничения:

- перезапустить заявку возможно из состояний `Success` и `Fail`.
- `workflow` при перезапуске заявки возобновляется с последнего посещенного состояния, являющегося точкой входа (`EntryPoint`). При отсутствии точек входа `workflow` возобновляется с начального состояния (`initialState`).

#### 4.5.4. Автоматический переповтор заявок

Подраздел содержит сведения о взаимодействии с внешними системами, классификации ошибок синхронных и асинхронных взаимодействий, переповторах синхронных и асинхронных взаимодействий.

##### 4.5.4.1. Взаимодействие с внешними системами

Сценарии взаимодействуют с внешними системами, осуществляя запросы (`requests`) по заданным адресам (`URL`). Внешние системы согласно запросам выполняют соответствующие операции.

Тип обработки запроса-инициатора операции определяет тип взаимодействия систем:

- *синхронный*;
- *асинхронный*.

Тип взаимодействия	Краткое описание
<b>Синхронный</b>	Ответ внешней системы на запрос уже содержит результат выполнения операции. Осуществляется по HTTP протоколу
<b>Асинхронный</b>	<p>Ответ внешней системы на запрос является подтверждением получения запроса на исполнение. Непосредственно исполнение может быть начато с некоторой задержкой, а результат исполнения будет передан внешней системой самостоятельно отдельным запросом. Запрос осуществляется по HTTP и AMQP протоколам. Результатом асинхронного взаимодействия является:</p> <ul style="list-style-type: none"> <li>• ответ от внешней системы (<code>callback</code>);</li> <li>• прерывание (<code>external event</code>);</li> <li>• отсутствие ответа в заданное время (<code>timeout</code>)</li> </ul>

После завершения синхронного или асинхронного взаимодействия классифицируется его результат. Если результат взаимодействия отнесен к какому-либо классу ошибки, выполняется поиск политики повтора.

Если поиск политики завершился успехом, сценарий приостанавливается на заданное в

политике время, а по его истечении подается на повторное исполнение.

#### 4.5.4.2. Классификация ошибок синхронных взаимодействий

Синхронное взаимодействие сценария с внешней системой может быть успешным либо неуспешным. Для систематизации и обработки неуспешных взаимодействий требуется их классификация. Ошибки взаимодействия могут возвращаться:

- в статус-коде ответа внешней системы;
- в теле ответа внешней системы;
- в исключении (программной ошибке) на уровне соединения (например, истечении тайм-аута попытки соединения с сервером, закрытии соединения, истечении тайм-аута чтения и т.п.).

Чтобы классифицировать ошибки взаимодействия, необходимо задать правила классификации. При создании правил классификации:

- задаётся имя правила (*rule*) для группировки запросов (*requests*) по системе;
- указывается набор запросов для системы. Для группировки разных запросов может использоваться маска, например, `/ps/bis/*/yyyy` или `/ps/*`. При этом в первую очередь будут выбраны правила с более специфичной маской запроса;
- указываются классы (*classes*) ошибок для группировки ошибок взаимодействия с общей реакцией (политикой).

#### Классификация ответов

Каждый ответ внешней системы на HTTP запрос характеризуется атрибутами:

<b>Статус-код</b>	<p>Ответы со значением этого атрибута в диапазонах:</p> <ul style="list-style-type: none"> <li>• 1xx и 3xx считаются успешными;</li> <li>• 2xx могут содержать как успешный результат, так и ошибку;</li> <li>• 4xx–5xx говорят об ошибке при обработке запроса</li> </ul>
<b>Тело ответа</b>	<p>Тело ответа может содержать детали возникшей ошибки как на стороне внешней системы, так и на стороне следующих по цепочке обработки ответа систем</p>

Для классификации ответов задаются статус-коды (*status\_codes*) и при необходимости ключевые слова (*body\_keywords*), которые должны присутствовать в теле ответа.

Например, требуется выделить классы ошибок:

- ошибки при недоступности систем `urwin` и `cart` со статус-кодами ответов `500`, `502`, `503`;
- отдельные ошибки системы `urwin` с кодом ответа `500` и телом ответа с сообщениями о недоступности `SSO/ошибках подключения`;
- ошибки других систем с кодом ответа `502`, `503`.

Спецификация правил

`ps.crab.crab_akka.execution.error_handling.sync.classification.rules` для описанного примера будет следующая:

```
any.errors.rule:
  requests:
    - path: /**
  classes:
    all.unavailable:
      responses:
        - status_codes:
          - 502
          - 503

urwin_and_cart.unavailable.rule:
  requests:
    - path: /ps/v1/urwin/**
    - path: /ps/v1/cart/**
  classes:
    urwin.unavailable:
      responses:
        - status_codes:
          - 500
          - 502
          - 503

urwin.status_change.errors.rule:
  requests:
    - path: /ps/v1/urwin/tasks/*/items/*/state/*
  classes:
    urwin.others:
      responses:
        - status_codes:
          - 500
      body_keywords:
        - ["Service unavailable", "SSO"]
        - ["Connection reset"]
    urwin.unavailable:
      responses:
        - status_codes:
          - 500
          - 502
          - 503
```

Если внешняя система в случае возникновения логических ошибок может вернуть код 200, для классификации таких ошибок следует задать отдельное правило. Например:

```
hex.rule:
  requests:
    - path: /ps/hex/*
  classes:
    hex.errors:
      responses:
        - status_codes:
          - 200
      body_keywords:
        - ["Operation timed out"]
```

В этих правилах запрещается применять *multi segment wildcard*.

### Классификация исключений

Каждое исключение характеризуется атрибутами:

<b>Тип ошибки</b>	Тип ошибки представлен полным именем класса возникшего исключения
<b>Иерархия ошибки</b>	Иерархия ошибки представлена набором вложенных исключений
<b>Сообщение ошибки</b>	В сообщении ошибки указано дополнительное описание случившейся проблемы

Для классификации исключений применяются атрибуты:

- имя (`name_keyword`) класса любого исключения в иерархии;
- при необходимости ключевые слова (`message_keywords`), которые должны присутствовать в сообщении исключения.

Например, требуется выделить классы ошибок:

- исключения при взаимодействии с системами `urwin` и `cart`;
- ошибки соединения с сообщениями об отказе установки соединения.

Спецификация правил в

`ps.crab.crab_akka.execution.error_handling.sync.classification.rules` для описанного примера будет следующая:

```

urwin_and_cart.unavailable.rule:
  requests:
    - path: /ps/v1/urwin/**
    - path: /ps/v1/cart/**
  classes:
    urwin.exception:
      exceptions:
        - name_keyword: "Exception"

any.exception.rule:
  requests:
    - path: /**
  classes:
    all.connection:
      exceptions:
        - name_keyword: "java.net.ConnectException"
      message_keywords:
        - ["Connection refused"]
        - ["Connection failed"]

```

Для классификации ошибок в зависимости от возникшего исключения интерфейс классификации ошибок расширен возможностью указания типа возникающего исключения и ключевых слов его сообщения. Предыдущий интерфейс предусматривал классификацию только любого исключения (только `ANY_HTTP_EXCEPTION`). Например:

```

urwin_and_cart.unavailable.rule:
  requests:
    - path: /ps/v1/urwin/**
    - path: /ps/v1/cart/**
  classes:
    urwin.exception:
      exceptions:
        - "ANY_HTTP_EXCEPTION"

any.exception.rule:
  requests:
    - path: /**
  classes:
    all.connection:
      exceptions:
        - "ANY_HTTP_EXCEPTION"

```

Начиная с версии **5.7.0** данный интерфейс является устаревшим (`DEPRECATED`), в версии **6.1.0** и последующих его поддержка полностью прекращена.

### 4.5.4.3. Классификация ошибок асинхронных взаимодействий

Асинхронное взаимодействие сценария с внешней системой может быть успешным либо неуспешным. Для систематизации и обработки неуспешных взаимодействий требуется их классификация.

Индикатором неуспешного асинхронного взаимодействия может быть:

- отсутствие ответа внешней системы (тайм-аут);
- ответ, содержащий информацию об ошибке в поле:
  - `step_status`;
  - `body`.

Чтобы классифицировать ошибки взаимодействия, необходимо задать правила классификации. При создании правил классификации:

- задаётся имя правила (*rule*) для группировки запросов (*requests*) по системе;
- указывается набор HTTP и AMQP запросов для системы. Для группировки разных запросов может использоваться маска, например, `/ps/bis/*/yyyy` или `/ps/**` для HTTP и `/crab?exchange=* &key=*` для AMQP. При этом в первую очередь будут выбраны правила с более специфичной маской запроса.
- указываются классы (*classes*) ошибок для группировки ошибок взаимодействия с общей реакцией (политикой).

#### Классификация тайм-аутов

Для классификации тайм-аутов в правиле (*rule*) задается значение `timeout: true`.  
Например:

```
test-rule:
  requests:
    http:
      - path: /test
    amqp:
      - path: "/a?exchange=a&key=a" # virtual host 'a', exchange 'a',
routing key 'a'
      - path: "%2F?exchange=a&key=a" # пример с экранирования слеша в
виртуальном хосте, virtual host '/', exchange 'a', routing key 'a'
  classes:
    technical:
      timeout: true
```

Все завершенные по тайм-ауту взаимодействия, для которых отсутствуют правила классификации, классифицируются на встроенный класс `crab.async.timeout`.

#### Классификация ответов

Для классификации ответов в правиле задаётся секция `responses`. Например:

```

test-rule:
  requests:
    http:
      - path: /a
  classes:
    async.class1:
      responses:
        - step_status_keywords:
            - k2 # здесь только одиночный кейворд
            - k21 # это альтернативный кейворд
          body_keywords:
            - [ "k1" ]
            - [ "k2", "k3" ] # это список кейвордов, которые должны
            присутствовать одновременно

```

При классификации результата **CRAB**:

- на основании URL запроса определяет наиболее подходящее правило классификации;
- выбирает класс, если результат взаимодействия удовлетворяет его критериям;
- при отсутствии подходящего правила или класса ошибки все тайм-ауты классифицирует как `crab.async.timeout`.

URL запроса может быть:

- получен от сценария как один из атрибутов выполненного запроса в рамках асинхронного взаимодействия;
- определен автоматически в том случае, когда сценарий не передал URL и в рамках операции (*action*), вернувшей в качестве результата значение "wait", был выполнен только один HTTP или AMQP запрос. При двух и более запросах (вне зависимости от протокола) в одном *action* сценарий должен явно указать URL для каждого из взаимодействий.



Если URL не передан и не может быть однозначно определен, то такое взаимодействие фактически исключается из обработки ошибок. Такие случаи рекомендуется выявлять на этапе эксплуатации и выполнять доработки сценариев.

Правила классификации для асинхронных взаимодействий задаются в отдельной секции инсталлятора с параметрами обработки ошибок асинхронного взаимодействия.

#### 4.5.4.4. Переповторы синхронных взаимодействий

##### Назначение и алгоритм автоматического переповтора

Автоматический переповтор заявок обеспечивает успешное исполнение интеграционных сценариев при наличии инфраструктурных ошибок в процессе взаимодействия ядра **CRAB** с внешними системами по HTTP протоколу.

Алгоритм автоматического переповтора срабатывает при возникновении инфраструктурных ошибок взаимодействия сценария с внешними системами и включает:

1. Перехват инфраструктурной ошибки ядром **CRAB**.
2. Классификацию ошибки согласно заданным правилам.
3. Поиск политики переповтора для данного сценария с определённым классом ошибки.
4. Прерывание исполнения сценария на период, вычисляемый на основании параметров политики переповтора.
5. Освобождение исполнителя (**WORKER**) и выжидание периода переповтора.
6. Активация активатором (**ACTIVATOR**) приостановленной заявки по истечении времени ожидания переповтора.
7. Продолжение исполнения сценария с прерванного `action` (для `workflow` сценариев) или точки сохранения (для `operation` сценариев).

Если при повторном взаимодействии произошла ошибка того же класса, указанные в алгоритме действия будут повторены. Таким образом, продукт циклически повторяет взаимодействие до тех пор, пока:

- взаимодействие не завершится успехом;
- не возникнет новый класс ошибки, в результате которой будет начат новый цикл;
- не возникнет класс ошибки, для которого отсутствует политика повтора;
- не будет исчерпан ресурс, ограниченный значениями:
  - максимальной длительности цикла синхронного переповтора (`ps.crab.crab_akka.execution.error_handling.sync.retry.max_cycle_duration`). При работе движка в режиме `ps.crab.crab_akka.execution.error_handling.retry_duration_control.mode` с присвоенным значением **`run_and_suspending_durations`** максимальная длительность считается без учёта времени доставки заявки от момента её активации в состоянии `RETRY` до момента выполнения очередной попытки взаимодействия;
  - числа повторных попыток (`ps.crab.crab_akka.execution.error_handling.sync.retry.policies.parameters.max_attempts`).

Если класс ошибки не определен или для класса ошибки не определена политика переповтора, завершён период действия политики или исчерпано количество попыток — продолжается исполнение сценария.

### Включение режима переповторов

Для включения требуется в `INVENTORY CRAB_AKKA`:

1. Присвоить параметру `ps.crab.crab_akka.execution.error_handling.sync.mode` значение `retry`.
2. Задать хотя бы одну политику переповтора в параметре `ps.crab.crab_akka.execution.error_handling.sync.retry.policies`.
3. Настроить классификацию ошибок.
4. Запустить скрипт конфигурирования ядра **CRAB**.
5. Выполнить рестарт узлов исполнения сценариев (**WORKER**).

### Ограничения и рекомендации

Для предотвращения длительной блокировки очередей последовательных заявок суммарное время нахождения заявки в статусе `RETRY` ограничено значением параметра `ps.crab.crab_akka.resource_control.max_acquisition_duration`. Подробнее см. *«Последовательная обработка заявок»*.

Рекомендуется минимизировать количество переповторов, выбирая параметры политики с учетом вероятности возникновения ошибок и их массовости. Частые массовые переповторы могут вести к перегрузке ядра и существенной задержке в обслуживании для других заявок.



## Примеры конфигурации

Конфигурация автоматического переповтора заявок задаётся в **INVENTORY CRAB\_AKKA** параметрами:

- `ps.crab.crab_akka.execution.error_handling.sync.classification.rules` – YAML спецификация с правилами классификации ошибок при синхронном взаимодействии сценариев с внешними системами;
- `ps.crab.crab_akka.execution.error_handling.sync.retry.policies` – YAML спецификация с политиками переповтора для разных классов ошибок.

### 4.5.4.5. Переповторы асинхронных взаимодействий

#### Назначение и алгоритм автоматического переповтора

Автоматический переповтор заявок обеспечивает успешное выполнение интеграционных сценариев в процессе взаимодействия ядра **CRAB** с внешними системами по HTTP и AMQP протоколам.

Алгоритм автоматического переповтора срабатывает при отсутствии ответов от внешних систем и включает:

1. Определение ядром **CRAB** отсутствия ответа от внешней системы (тайм-аута).
2. Классификацию тайм-аута согласно заданным правилам.
3. Поиск политики переповтора для данного сценария с определённым классом ошибки.
4. Прерывание исполнения сценария на период, вычисляемый на основании параметров политики переповтора.
5. Освобождение исполнителя (**WORKER**) и выжидание периода переповтора.
6. Активация активатором (**ACTIVATOR**) приостановленной заявки по истечении времени ожидания переповтора.
7. Продолжение исполнения сценария с последней точки входа (для `workflow` сценариев) или точки сохранения (для `operation` сценариев).

Если при повторном взаимодействии произошла ошибка того же класса, указанные в алгоритме действия будут повторены. Таким образом, продукт циклически повторяет взаимодействие до тех пор, пока:

- взаимодействие не завершится успехом;
- не возникнет новый класс ошибки, в результате которой будет начат новый цикл;
- не возникнет класс ошибки, для которого отсутствует политика повтора;
- не будет исчерпан ресурс, ограниченный значениями:
  - максимальной длительности цикла асинхронного переповтора (`ps.crab.crab_akka.execution.error_handling.async.retry.max_cycle_duration`). При работе движка в режиме `ps.crab.crab_akka.execution.error_handling.retry_duration_control.mode` с присвоенным значением `run_and_suspending_durations` максимальная длительность считается без учёта времени доставки заявки от момента её активации в состоянии `RETRY` до момента выполнения очередной попытки взаимодействия;
  - числа повторных попыток (`ps.crab.crab_akka.execution.error_handling.async.retry.policies.parameters.max_attempts`).

Если класс ошибки не определен или для класса ошибки не определена политика переповтора, завершён период действия политики или исчерпано количество попыток — продолжается исполнение сценария.

## Включение режима переповторов

Для включения требуется в INVENTORY **CRAB\_AKKA**:

1. Присвоить параметру `ps.crab.crab_akka.execution.error_handling.async.mode` значение `retry`.
2. Задать хотя бы одну политику переповтора в параметре `ps.crab.crab_akka.execution.error_handling.async.retry.policies`.
3. Настроить классификацию ошибок.
4. Запустить скрипт конфигурирования ядра **CRAB**.
5. Выполнить рестарт узлов исполнения сценариев (**WORKER**).

## Ограничения и рекомендации

1. Возможность повторов множественных асинхронных взаимодействий в рамках одного wait-состояния отсутствует.
2. Для асинхронных взаимодействий не поддерживается повтор на основании «сценарной» классификации.



Под «сценарной» классификацией понимается механизм определения класса ошибки внутри сценария, а не на основании правил классификации, заданных сотрудником эксплуатации.

3. Для предотвращения длительной блокировки очередей последовательных заявок суммарное время нахождения заявки в статусе `RETRY` ограничено значением параметра `ps.crab.crab_akka.resource_control.max_acquisition_duration`. Подробнее см. [Последовательная обработка заявок](#).



Поэтому в параметрах сценариев время ожидания ответа по асинхронным взаимодействиям рекомендуется настраивать таким образом, чтобы оставалась возможность для повторов взаимодействия. Для большинства взаимодействий будет достаточно 10 минут ожидания ответа. Для более точного определения следует воспользоваться статистикой длительности исполнения этапа сценариев (см. метрику `p99SucceedStageDurationMsForShortTermPeriod`)

4. Рекомендуется минимизировать количество переповторов, выбирая параметры политики с учетом вероятности возникновения ошибок и их массовости. Частые массовые переповторы могут вести к перегрузке ядра и существенной задержке в обслуживании для других заявок.

## Уход на переповтор синхронного запроса при повторе асинхронного взаимодействия

При возобновлении исполнения сценария после ухода на повтор асинхронного взаимодействия сценарий может быть прерван на переповтор синхронного взаимодействия. В этом случае период прерывания сценария вычисляется на основании параметров политики переповтора синхронного взаимодействия.

После завершения переповтора синхронного взаимодействия будет продолжен переповтор асинхронного взаимодействия с параметрами, с которыми он был начат. Суммарное время нахождения заявки в статусе `RETRY` также будет ограничено значением параметра `ps.crab.crab_akka.execution.error_handling.async.retry.max_cycle_duration`.

## Примеры конфигурации

Конфигурация автоматического переповтора заявок задаётся в INVENTORY CRAB\_AKKA параметрами:

- `ps.crab.crab_akka.execution.error_handling.async.classification.rules` – YAML спецификация с правилами классификации ошибок при асинхронном взаимодействии сценариев с внешними системами;
- `ps.crab.crab_akka.execution.error_handling.async.retry.policies` – YAML спецификация с политиками переповтора для разных классов ошибок.

## 4.6. Активация заявок

Активация заявок — это процесс возобновления исполнения приостановленных сценариев в случаях:

- наступления времени исполнения следующего этапа сценария;
- истечения времени ожидания ответа от внешней системы;
- истечения времени ожидания переповтора заявки.

Производительность активации зависит от количества приостановленных заявок, для которых наступило время активации.

Разные сценарии выполняют приостановку, как правило, на разный период времени. Например, получение результатов проверки персональных данных абонентов, может выполняться через 6-8 часов. Отключение услуг при истечении бесплатного периода использования выполняется через неделю или месяц. В исключительных ситуациях, отдельные этапы сценариев могут выполняться через полгода или даже год. При массовой активации большое количество заявок по одному сценарию будет замедлять активацию заявок по другому.

Для снижения взаимного влияния заявки по разным сценариям при активации разделяются на группы по их возрасту и активация каждой группы выполняется в своем потоке.

Разделение на группы возможно только для SLEEP-заявок. Количество приостановленных заявок в остальных состояниях невелико и инцидентов, связанных с недостаточной производительностью их активации, не наблюдается.

### 4.6.1. Изоляция SLEEP-заявок для активации по возрасту

Для изоляции SLEEP-заявок предусмотрены три группы:

- **youngest**;
- **middle**;
- **oldest**.

Для первых двух групп конфигурацией предусмотрен максимальный возраст заявки в группе, остальные заявки помещаются в группу **oldest**.

Первый (младший) поток активирует sleep-заявки возрастом от 0 до одного дня, второй (средний) — от одного до 30 дней. Заявки старше 30 дней активируются третьим (старшим) потоком.

Границы диапазонов задаются в INVENTORY CRAB\_AKKA параметрами **ps.crab.crab\_akka.activation.groups.sleep.youngest.age\_in\_days** и **ps.crab.crab\_akka.activation.groups.sleep.middle.age\_in\_days**.

Для мониторинга активации используется ресурс **OrdersActivation** без разбивки данных по потокам.

## 4.7. Прерывание обработки заявок

**CRAB\_AKKA** поддерживает возможность прерывания обработки заявок. Прерывание обработки позволяет службе эксплуатации по определённым причинам отменять исполнение сценариев до начала их исполнения, тем самым устраняя аварийные ситуации.

Для прерывания требуется обратиться к одному из узлов **WEBAPP** и выполнить REST API функцию прерывания обработки заявок с помощью команды cURL (или аналогичной). В результате:

- заявка будет проверена на возможность прервать её обработку, при положительном результате обработка будет прервана;
- заявке будет присвоен статус «Прервана» (**INTERRUPTED**) с отображением нового статуса в центре управления (UI) **CRAB**;
- исполнение **INTERRUPTED** заявок будет игнорироваться на исполнителе (**WORKER**).

Команда может быть выполнена, если заявка находится в одном из разрешённых для прерывания обработки статусах:

- «Новое» (**NEW**);
- «Перезапущено» (**REOPEN**);
- «Отложено» (**POSTPONED**).



Прерванную заявку при необходимости можно перевести в статус «Отменено» (**CANCELED**):

- отменой заявки в центре управления (UI);
- командой cURL и REST API функции отмены заявки.

## 4.8. Зависимая обработка заявок

### 4.8.1. Зависимости между заявками

При обработке бизнес-сценариев между взаимодействующими в них заявками выстраивается механизм зависимостей, при котором одни (условные) заявки зависят от других (предусловных) заявок.

**Предусловная заявка** – CRAB-заявка, от которой зависит запуск на обработку одной или нескольких заявок.

**Условная заявка** – CRAB-заявка, которая:

- зависит от одной или нескольких предусловных заявок;
- поступает в обработку после того, как все предусловные заявки достигли ожидаемого результата.

Ожидаемый результат задаётся перечислением (подробнее см. «Регистрация условного требования с заданным Lifecycle» в описании программного интерфейса **CRAB**) терминальных состояний сценария предусловной заявки, достижение любого из которых позволяет начать обработку условной заявки. Например, в случае обработки трёх заявок, из которых заявка № 1 *предусловная*, заявки №№ 2 и 3 — *условные*.

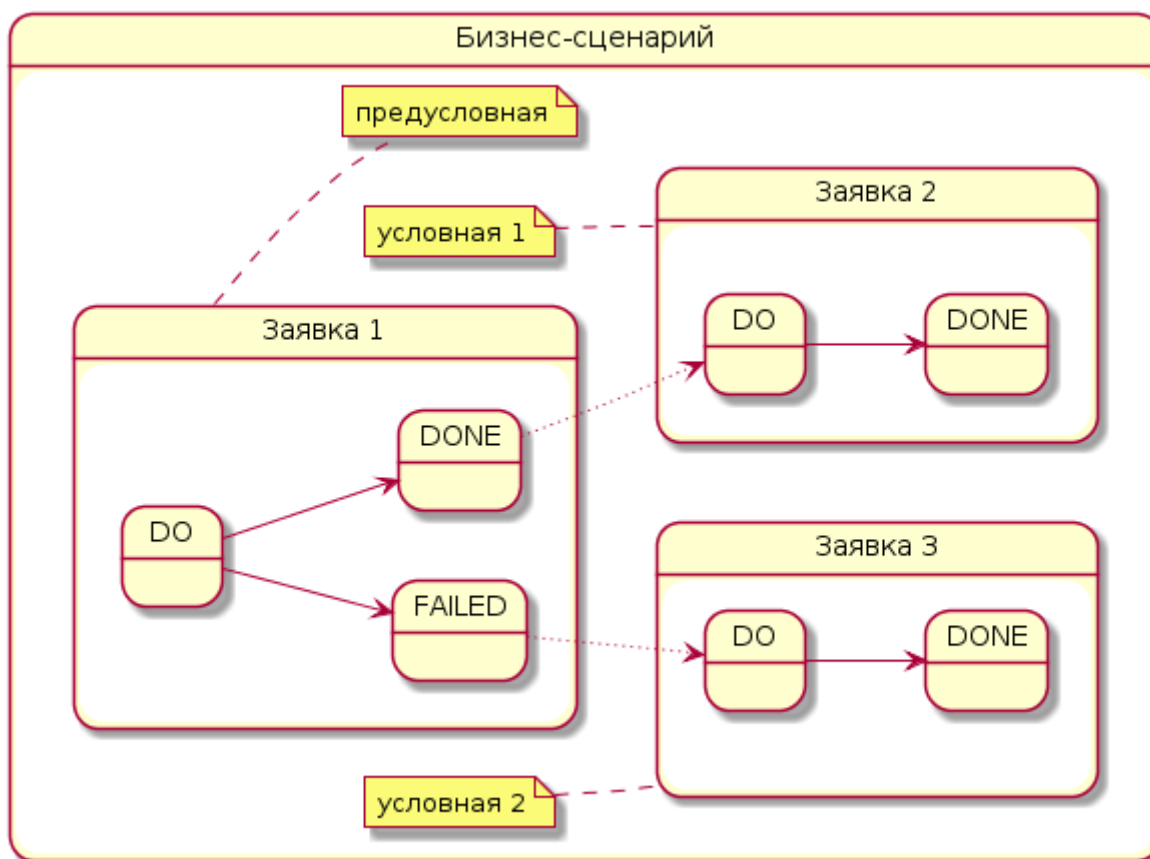


Рис. 17. Зависимая обработка заявок

Перечень терминальных состояний сценария предусловной заявки включает состояния **DONE** и **FAILED**. Достижение состояния **DONE** позволяет запустить обработку первой условной заявки (№ 2), состояния **FAILED** — второй условной заявки (№ 3).

Реализация механизма зависимостей между заявками позволяет:

- обрабатывать бизнес-сценарии, шаги которых зависимы по результатам работы, но реализуются разными CRAB-заявками. Например, если одна заявка создаёт некоторую сущность, а другая её изменяет, то условные заявки позволяют начать изменение сущности только после её успешного создания;
- соблюдать строгий порядок обработки бизнес-событий. Например, если при изменении некоторой сущности регистрируется заявка по обработке данного события, то условные заявки позволяют гарантировать последовательность обработки событий без пропусков (даже в случае возникновения ошибок);
- динамически расширять бизнес-сценарии. Например, если в обработку была принята предусловная заявка в рамках некоторого бизнес-сценария, то условные заявки позволяют динамически расширить бизнес-сценарий дополнительными шагами в случае успешной или даже неуспешной обработки предусловной заявки.

#### 4.8.2. Режимы зависимой обработки заявок

Ядро **CRAB (CRAB\_AKKA)** поддерживает следующие режимы зависимой обработки заявок:

- **polling** (устаревший) — режим периодической активации условной заявки для опроса состояния предусловной заявки — включен по умолчанию;
- **reactive** — режим активации условной заявки по событию обработки предусловной заявки.

Если обработка предусловной заявки завершится неожиданным результатом, условная заявка:

- в режиме **polling** перейдёт в состояние **FAIL**;
- в режиме **reactive** — будет продолжать находится в состоянии **POSTPONED**, пока не будут выполнены условия активации либо она не будет удалена из СУБД **CRAB**.

Мониторинг и диагностика обработки условных заявок осуществляется действующими метриками **CRAB\_AKKA**.

### 4.8.3. Активация режима **reactive**

Для активации режима необходимо параметру **ps.crab.crab\_akka.execution.start\_control.conditional.mode** присвоить значение **reactive** с последующей перезагрузкой узлов **worker**, **webapp** и **consumer**.

## 5. Обеспечение отказоустойчивости CRAB\_AKKA

Глава включает описание механизмов обеспечения отказоустойчивости распределителей и целостности кластера, гарантированной активации заявок, а также существующие ограничения реализации.

### 5.1. Обеспечение отказоустойчивости распределителей

Отказоустойчивость распределителей обеспечивается схемой резервирования, которая предусматривает:

- наличие основных и резервных узлов распределителя;
- персистентность их состояния;
- автоматическое переключение между основным и резервным узлами кластера.

#### Основные и резервные узлы распределителя

В схеме развертывания ядра **CRAB** предусмотрены основные и резервные узлы, которые размещаются на отдельных серверах.

При остановке основного активного распределителя (**dispatcher**) резервный сразу становится активным.



**Важно!** Для автоматического переключения с основного на резервный распределитель требуется *персист* в СУБД.

При падении или потере соединения с активным распределителем резервный становится активным через 10 секунд (по умолчанию, см. параметр `crab_akka_cluster_singleton_handover_after_sec` инсталлятора продукта).

Порядок перехода на схему развертывания с двумя распределителями и автопереключением см. в *приложении С1* настоящего руководства.

#### Персистентное хранение состояния распределителя

Персистентное хранение состояния необходимо для предотвращения потери данных при:

- обновлении версии продукта;
- рестарте основного узла распределителя;
- переключении с основного на резервный узел.

Консистентность состояния обеспечивается сочетанием:

- снимков состояния (снапшотов) узла распределителя;
- журналов событий.

В качестве персистентного хранилища используются:

- локальная файловая система (по умолчанию);
- либо СУБД.

#### Персистентное хранение состояния в файловой системе распределителя

При локальном хранении состояния в файловой системе распределитель регулярно:



- сохраняет в указанный в параметре `ps.crab.crab_akka.persistence.filesystem.dir` каталог:
  - информацию о всех заявках, ожидающих или уже находящихся в обработке;
  - статистику исполнения заявок, используемую для регулирования рейтов сценариев;
- удаляет устаревшие данные.

Каталог файловой системы включает два вложенных каталога:

- *journal* – для хранения журналов событий по статистике и заявкам;
- *snapshot* – для хранения снимков. Когда количество событий достигает заданного значения параметра `eventCountBetweenSnapshots`, снимок создается и сохраняется в каталоге. При этом устаревшие события и снимки с файловой системы удаляются.

Рекомендуется выделять не менее:

- 500 Мб для хранения событий;
- 2 Гб для хранения снимков.



**Внимание!** Не удаляйте файлы с событиями вручную, это приведет к критическим ошибкам в работе распределителя.

## Персистентное хранение состояния в СУБД

При хранении состояния в СУБД распределитель регулярно записывает в схему персиста:

- информацию о всех заявках, ожидающих или уже находящихся в обработке;
- статистику исполнения заявок, используемую для регулирования рейтов сценариев;

Устаревшие данные удаляются утилитой **Persist Cleaner**.

Для настройки персиста в СУБД используются параметры, описание которых см. в секции «*Параметры персистентности узла с ролью Dispatcher*» раздела «*Настройка топологии и конфигурации (INVENTORY) CRAB\_AKKA*» руководства по установке **CRAB**.

## Переключение режима персистентности

Технологическая инструкция с контрольным списком требований на этапах подготовки, переключения режима и проверки работоспособности распределителя приведена в *приложении А* настоящего руководства.

## 5.2. Обеспечение целостности кластера

*Кластер* – группа совместно работающих узлов с заданными ролями, предназначенная для непрерывной обработки и выполнения принятых **CRAB\_AKKA** заявок. Целостность кластера характеризуется наличием:

- узлов с полным набором ролей, требуемых процессом работы с принятыми заявками;
- связей между узлами согласно схемам развертывания и резервирования.

Целостность кластера обеспечивается:

- узлами кластера с ролью *seed* (*seed-узлами*), отвечающими за формирование кластера;
- алгоритмом выбора его активной части на основе *кворума* для предотвращения расщепления (*Split Brain*).

Для *seed-узлов* задается кворум (*seedQuorum*) – минимальное необходимое количество узлов с этой ролью в кластере. Кворум определяется автоматически при развертывании кластера через инсталлятор продукта.



Ядро системы:

- не запустится, пока в кластере не соберется кворум для `seed`-узлов;
- остановится, если подключенных `seed`-узлов осталось меньше заданного кворума;
- остановится, если подключенных `seed`-узлов стало больше или равно  $seedQuorum * 2$ .

Для корректной работы алгоритма кворум должен рассчитываться по формуле:  $\langle \text{общее количество seed-узлов для площадки} \rangle / 2 + 1$ .

Формула гарантирует, что будет активна только часть кластера, содержащая большинство `seed`-узлов. Роль `seed` будет назначена существующим узлам распределителя и новому выделенному узлу.



**ВАЖНО!** Для обеспечения работоспособности ядра с двумя распределителями в разных ЦОД необходимо наличие минимум трех `seed`-узлов.

`Seed`-узлы также принимают запросы на подключение новых узлов с ролями `worker`, `activator`, `consumer` и `dispatcher`.

## 5.3. Ограничения реализации

### Полная остановка обработки заявок

Если состояние сети не позволяет ядру **CRAB** автоматически выбрать активный узел, будет выполняться полная остановка обработки заявок для предотвращения возможной аварии.

То есть для распределителя (**dispatcher**) требование целостности внутреннего состояния превалирует над доступностью.

При восстановлении сети обработка будет возобновлена автоматически.

При частичной потере связности между узлами (внутри и между ЦОД) в ситуации, когда система не может однозначно выбрать `seed`-узел для исключения из кластера, будут исключаться оба `seed`-узла, что приведет к рестарту всего кластера.

Для разрешения данной ситуации и стабилизации работы кластера необходимо:

- устранить нарушение связности – и кластер восстановится автоматически;
- либо вручную выбрать и остановить сбойный узел.


Процедура отката на схему развертывания с одним работающим распределителем описана в приложении [«Откат на схему развертывания с одним работающим распределителем»](#) настоящего руководства.

## 5.4. Гарантированная активация заявок

### Назначение и алгоритм гарантированной активации заявок

Гарантированная активация заявок позволяет решать проблему «зависания» заявок в промежуточных состояниях (`WAIT`, `SLEEP`, `RETRY` и `POSTPONED`) при потерях команд на активацию.

Для этого в `INVENTORY CRAB_AKKA` для настроечного параметра `ps.crab.crab_akka.activation.mode` предусмотрены следующие режимы:

<p>at_least_once</p>	<p>В этом режиме при отправке команды активатором (Activator) время активации в СУБД сдвигается вперед на время текущей задержки активации (см. метрики <code>currentProcessingLagMs</code> для <code>OrdersActivation</code>). Это позволяет повторно активировать заявку, если предыдущая команда по какой-то причине не попала к исполнителю (Worker). Количество повторных активаций не ограничено. При росте повторных активаций функция активации будет деградировать. Необходимо контролировать рост повторных активаций, так как их заметное количество может свидетельствовать о системной проблеме в инфраструктуре или ошибке в продукте, которые требуют устранения</p>
<p>at_least_once_inprogress</p>	<p>В этом режиме алгоритм активации аналогичен алгоритму <code>at_least_once</code>, но с ограничением его применения только заявками в статусах <code>WAIT</code> и <code>RETRY</code>. Как минимум, этот режим активации обязателен при использовании режимов распределения <code>limited_flows</code> или <code>stopped_flows</code></p>
<p>at_most_once</p>	<p>В этом режиме время активации в СУБД сбрасывается активатором сразу после успешной отправки команды. Активация при этом не гарантируется.</p> <div data-bbox="555 981 1471 1115" style="background-color: #e0e0e0; padding: 10px; border: 1px solid #ccc;"> <div style="display: flex; align-items: center;">  <p>Не рекомендуется для использования в промышленной эксплуатации</p> </div> </div>

### Включение и мониторинг режима активации

Для включения режима гарантированной активации требуется присвоить соответствующее значение параметру `ps.crab.crab_akka.activation.mode` и перезагрузить активаторы.

Для мониторинга функции используются поля `waitRepeats`, `sleepRepeats`, `postponedRepeats`, `retryRepeats` атрибута `ActivationByOrderState` ресурса `OrdersActivation`.