Интеграционная шина Help-Pro: Руководство разработчика

Введение

Термины

Примечание: *Термины написаны так, как они употребляются в тексте. Некоторые пишутся с большой буквы, как имена, некоторые -- с маленькой, как обычные слова.*

- **очередь** -- AMQP queue.
- **app_name**, **node_name** -- параметры адресации Паттернов REQ-REP. Определяют мнемо-имя приложения и мнемо-имя Узла.
- **Клиент** -- см. Паттерны.
- **Сервер** -- см. Паттерны.

Общая информация

Шина использует в качестве транспорта протокол AMQP 0.9.1 (0-9-1), реализуемый брокером RabbitMQ.

Подтверждено, что с Шиной успешно работают следующие версии RabbitMQ:

- 2.5.0
- 2.8.7

Типы сообщений

На сегодня Шина поддерживает такие типы сообщений:

- **запрос** (request) -- сообщение от Клиента Серверу в Паттерне REQ-REP.
- **ответ** (reply) -- ответ Сервера на запрос Клиента.
- **уведомление** (notice) -- широковещательное уведомление в Паттерне PUB-SUB.

Формат сообщения

Тело (payload)

На 2013-03-17 тело сообщений Шины это **значения атрибутов (полей) запроса** (ответа, уведомления) в формате **JSON**.

Примеры:

- Запрос `Ping`:
  `{"seq":188887}`
- Запрос `ProLog::GetIdhash`:
  `{"session_id":"7ac8a2e9c0c9208450289925dbdbe09b"}`

Технические соглашения

"Технические соглашения" -- это процедуры и методики для формирования одних строк на основе других.

Имя очереди Сервера

Общий формат:

`bus.server.<app_name>.<node_name>`

Пример: Если app_name это "`pro_log`", а node_name это "`one`", имя очереди будет `"bus.server.pro_log.one".`

Мнемонический тип сообщения (message_type)

Общий формат:

`<request|reply|notice>.[app_name.]<mnemo>`

Примеры:

- Запрос `Ping`: `request.ping`
- Ответ `Ping`: `reply.ping`
- Запрос `ProLog::GetUser`: `request.pro_log.get_user`
- Ответ `ProLog::GetUser`: `reply.pro_log.get_user`

node_name по умолчанию (default_node_name)

node_name по умолчанию это `"one"`.

Схема взаимодействия при REQ-REP

Адресация

- У Сервера **обязательно** задан app_name.
- У Сервера может быть задан node_name. Если он не задан, присваивается

node_name по умолчанию, согласно соглашению.
- У Клиента не заданы ни app_name, ни node_name.

Параметры сущностей AMQP

- У Сервера и у Клиента тип exchange: `direct`, имя по умолчанию (`""`).
- У Сервера имя очереди по соглашению. Параметры: `auto_delete: true`.
- У Клиента имя очереди автоматическое (`""`). Параметры: `auto_delete: true`, `exclusive: true`.

Процедура взаимодействия

Участники: Клиент, Сервер.

Предположения-допущения:

- app_name, node_name Сервера это `"pro_log"`, `"one"`.
- Клиент делает запрос `ProLog::GetUser`.

Процедура:

1. Имя очереди Клиента присвоено автоматически брокером. Клиент знает это имя.
2. Зная app_name и node_name сервера, Клиент формирует название его очереди по соглашению.
3. При отправке запроса Клиент устанавливает такие `publish_options`:

```
publish_options = {
  expiration: timeout*1000[1],
  headers: {
    remote_time: "2013-03-16T21:15:08Z",
  },
  #immediate: true, # В RabbitMQ >= 3.0 эту опцию убрали.
  key: "bus.server.pro_log.one"[2],
  message_id: "1363468731.000902:138809"[3],
  reply_to: <имя_моей_очереди>[4],
  type: "request.pro_log.get_user"[5],
}
```

---

[1] В том случае, если для запроса установлен `timeout`. Если нет, `expiration` тоже не указывается и TTL сообщения в очереди определяется настройками или умолчаниями RabbitMQ.
[2] Это имя очереди Сервера.
[3] Секунды, ".", микросекунды, ":", случайное число [0.. 999999].
[4] Имя, которое присвоил брокер. Например, "amq.gen-AdNTGefhvTjZ5sljC9m8nL".
[5] Мнемонический message_type, генерируется по соглашению.

4. Сервер получает запрос, делает работу, генерирует и отправляет ответ. При отправке ответа Сервер устанавливает такие `publish_options`:

   ```
   publish_options = {
     correlation_id: <message_id_запроса>,
     key: <reply_to_запроса>,
     type: "reply.pro_log.get_user",
   }
   ```

5. **ВАЖНО:** В случае ошибки Сервером может быть отправлен bounce-ответ и `type` у него будет соответствующий. См. Bounce-ответы Сервера.

Bounce-ответы Сервера

Поскольку мы имеем дело с коммуникациями, нам необходимо **учитывать нештатные ситуации**, например, когда пришло повреждённое или некорректно сформированное сообщение. Стороны, участвующие в коммуникации, должны делать всё возможное, чтобы помочь разработчику **скорее установить причину сбоя**.

Одно из средств помощи при отладке -- bounce-ответы ("отлупы"). Bounce-ответ **это сформированный по всем правилам** ответ, который отправляется Сервером в том случае, если полученный им запрос содержит ошибки.

Очень важно, чтобы Клиент был готов получить bounce-ответ и мог адекватно отработать эту ситуацию.

Bounce-ответы имеют тип `reply.internal.server_error`.

Тело (payload) bounce-ответа состоит из полей `class_name` и `message`.

Примеры bounce-ответов:

- ```
  {
    class_name: "Bus::ServerError::InvalidRequest",
    message: "Message type is not set",
  }
  ```
- ```
  {
    class_name: "Bus::ServerError::InvalidRequest",
    message: "Error parsing remote time",
  }
  ```
- ```
  {
    class_name: "Bus::ServerError::ApplicationError",
    message: "Error connecting to DB",
  }
  ```
  ```

Механизм симуляции (simulate)

Хорошо разработанный Клиент должен уверенно работать не только когда "всё хорошо", но и учитывать **каждую** из возможных сбойных ситуаций и выходить из неё с минимальными потерями, не теряя контроля и не впадая в неадекват.

Для того, чтобы имитировать сбойные ситуации, придуман и местами[6] поддерживается механизм **симуляции**.

Симуляция -- это возможность сказать Серверу в запросе: *"Сервер, сделай вид, что на твоей стороне случилась такая-то штатная/нештатная ситуация"*.

То есть, в процессе отладки Клиента разработчик может "заказывать" ситуацию на Сервере, и имеет возможность отладить на своей стороне механизмы её обработки.

Например, можно попросить Сервер сделать вид, что обработка заняла продолжительное время ("заказать паузу"). Это параметр `pause`.

Для запросов типа `GetUser` можно "заказать" ситуации "пользователь не найден" и прочие.

Протоколы приложений

ПроЛог

Request::ProLog::GetIdhash

Узнать idhash пользователя по идентификатору сессии.

Ответ это Reply::ProLog::GetIdhash.

Атрибуты

- `session_id` `String`
  Идентификатор сессии (значение сессионной cookie браузера).
- `simulate` `Hash` (optional)
  Параметры симуляции.
  - `flow` `String` (optional)
    Мнемо-обозначение заказываемой ситуации.
    Допустимые значения: `"anonymous_session"`, `"session_not_found"`, `"user_not_found"`.
  - `pause` `Float` (optional)

[6] В ПроЛоге поддерживается в полном объёме. Во всех будущих "настоящих" (не прототипно-опытных) протоколах будет поддерживаться в полном объёме.

Просим сделать паузу в секундах перед отправкой ответа.

Примеры

- ```
  {
    session_id: "7ac8a2e9c0c9208450289925dbdbe09b",
  }
  ```
- ```
  {
    session_id: …,
    simulate: {
      pause: 1.5,
    },
  }
  ```
- ```
  {
    session_id: …,
    simulate: {
      flow: "session_not_found",
    },
  }
  ```

Request::ProLog::GetUser

Получить информацию о пользователе.

Ответ это Reply::ProLog::GetUser.

Атрибуты

- **`idhash`** **`String`**
  idhash искомого пользователя.
- **`simulate`** **`Hash`** (optional)
  Параметры симуляции.
  - **`flow`** **`String`** (optional)
    Мнемо-обозначение заказываемой ситуации.
    Допустимые значения: `"user_is_forged"`, `"user_not_found"`.
  - **`pause`** **`Float`** (optional)
    Просим сделать паузу в секундах перед отправкой ответа.

Примеры

- ```
  {
    # TODO: Формат `idhash` теперь другой, поправить при случае.
    idhash: "67bs9wxaawpiy5dn0q7w06mafaqhpv4",
  }
  ```
- ```
  {
    idhash: …,
  ```
  ```

```
  simulate: {
    pause: 1.5,
  },
}
```

- ```
  {
    idhash: …,
    simulate: {
      flow: "session_not_found",
    },
  }
  ```

Reply::ProLog::GetIdhash

Атрибуты

- `is_ok` **`Bool`**
  Флаг успеха.
- `message` **`String`** (optional)
  Если не `is_ok`, здесь кратко сообщается причина.
- `idhash` **`String`** (optional)
  Фактически найденный idhash.

Примеры

- ```
  {
    is_ok: true,
    idhash: "67bs9wxaawpiy5dn0q7w06mafaqhpv4",
  }
  ```
- ```
  {
    is_ok: false,
    message:  "Session not found: 202c~4b70",
  }
  ```

Reply::ProLog::GetUser

Атрибуты

- `is_ok` **`Bool`**
  Флаг успеха.
- `message` **`String`** (optional)
  Если не `is_ok`, здесь кратко сообщается причина.
- `id` **`Integer`**
  ID записи в базе.
- `email` **`String`**
- `first_name` **`String`**
- `gender` **`String`**
  ```

Пол. `"f"`, `"m"` или `""` если не задано.

- `idhash` **`String`**
- `last_name` **`String`**
- `middle_name` **`String`**
- `nickname` **`String`**
- `groups` **`Array`**
  Массив мнемо-названий групп, которым принадлежит пользователь. О группах написано в тезисах ПроЛога.
- `is_enabled` **`Bool`**
- `is_super` **`Bool`**
- `created_at` **`String`**
- `updated_at` **`String`**

Примеры

- ```
  {
    is_ok: true,
    id: 1,
    email: "alex.r@askit.org",
    first_name: "Алексей",
    gender: "m",
    idhash: "67bs9wxaawpiy5dn0q7w06mafaqhpv4",
    last_name: "Фортуна",
    middle_name: "Владимирович",
    nickname: "",
    is_enabled: true,
    is_super: true,
    created_at: "2013-02-13T16:33:22+04:00",
    updated_at: "2013-02-18T21:43:11+04:00",
    groups: [],
  }
  ```
- ```
  {
    is_ok: false,
    message: "User not found
  (idhash:\"67bs9wxaawpiy5dn0q7w06mafaqhpv4\")",
  }
  ```

Тестовая инсталляция Шины и ПроЛога

Для опытов допустимо использовать инсталляцию Шины и ПроЛога напрямую через Интернет со следующими параметрами (`connect_options`):

TODO: После 2013-11 уже не так, пусть Х-Про сооружают опытную площадку на своих ресурсах.

```
connect_options = {
  host: "dragon.askit.org",
  #port: 5672,    # По умолчанию.
  vhost: "d130227",
  username: "amqpuser",
  password: "kaka",
}
```

(Недописанное/размышления/мусор)

11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11