Адовая синхронизация

Как ходить на рандеву

В. Ю. Винник

Общеизвестные примитивы синхронизации

- Mutex, semaphore, critical section, condition variable, event.
- Известны давно: семафоры предложил Э. Дейкстра в 1965 г.
- Поддерживаются ядром ОС.
- Доступны через стандартные и сторонние библиотеки.
- Некоторые языки имеют поддержку на уровне синтаксиса.
  - Например, ключевое слово lock в языке C#.
- Atomic variable — значительный выигрыш в быстродействии
  - ценой усложнения алгоритмов и структур данных.
- Достаточны для реализации любых сколь угодно сложных, параллельных алгоритмов.

Производные средства синхронизации

- Примитивы синхронизации слишком низкоуровневы.
- Прямое применение для крупных прикладных задач не всегда удобно.
  - Как и проектирование ЭВМ из отдельных транзисторов.
- Удобнее — собирать из примитивов конструкции, пригодные к многократному использованию для широких классов задач.
  - Оформлять в виде библиотек.
  - Сложность внутреннего устройства скрыта за простым интерфейсом.
- Пример 1: очередь ограниченной ёмкости между потоками — производителями и потребителями.
- Пример 2: структура данных с множественным доступом на чтение и исключительным доступом на запись (single writer, multiple readers).

Несколько слов о языке Ада

- Задача разработки нового языка поставлена в конце 1970-х гг.
- Основной заказчик — Министерство обороны США.
- Новый язык должен был вобрать в себя весь лучший опыт, накопленный человечеством благодаря другим языкам.
  - Fortran, PL/1, Algol, Pascal, Modula, Simula, C, …
- К созданию языка были привлечены ведущие мировые специалисты.
- Язык Ада не стал по-настоящему массовым.
  - Ниша: сложные встроенные системы с высокими требованиями к надежности.
- В языке Ада на уровне синтаксических конструкций поддерживается тщательно продуманный механизм взаимодействия потоков (т. н. задач).

Задачи, входы и рандеву в языке Ада на примере

```
task Server is
  entry set_arg(x : in integer);
  entry get_result(y : out
integer);
end Server;
```

```
procedure Client is
  t : integer;

begin
    Server.set_arg(10);
    long_operation;
    Put("The server produced ");
    Server.get_result(t);
    Put(t);
end Client;
```

```
task body Server is
  arg, result : integer;
begin
  accept set_arg(x : in integer)
  do
    arg := x;
  end set_arg;

  result := long_computation(arg);

  accept get_result(y : out
integer)
  do
    y := result;
  end get_result;
end Server;
```

Задачи, входы и рандеву в языке Ада на примере

```
task Server is
  entry set_arg(x : in integer);
  entry get_result(y : out
integer);
end Server;
```

Интерфейс задачи

```
procedure Client is
  t : integer;

begin
    Server.set_arg(10);
    long_operation;
    Put("The server produced ");
    Server.get_result(t);
    Put(t);
end Client;
```

Использование задачи

```
task body Server is
  arg, result : integer;
begin
  accept set_arg(x : in integer)
  do
    arg := x;
  end set_arg;

  result := long_computation(arg);

  accept get_result(y : out
integer)
  do
    y := result;
  end get_result;
end Server;
```

Реализация задачи

Задачи, входы и рандеву в языке Ада на примере

```
task Server is
  entry set_arg(x : in integer);
  entry get_result(y : out
integer);
end Server;
```

```
procedure Client is
  t : integer;

begin
    Server.set_arg(10);
    long_operation;
    Put("The server produced ");
    Server.get_result(t);
    Put(t);
end Client;
```

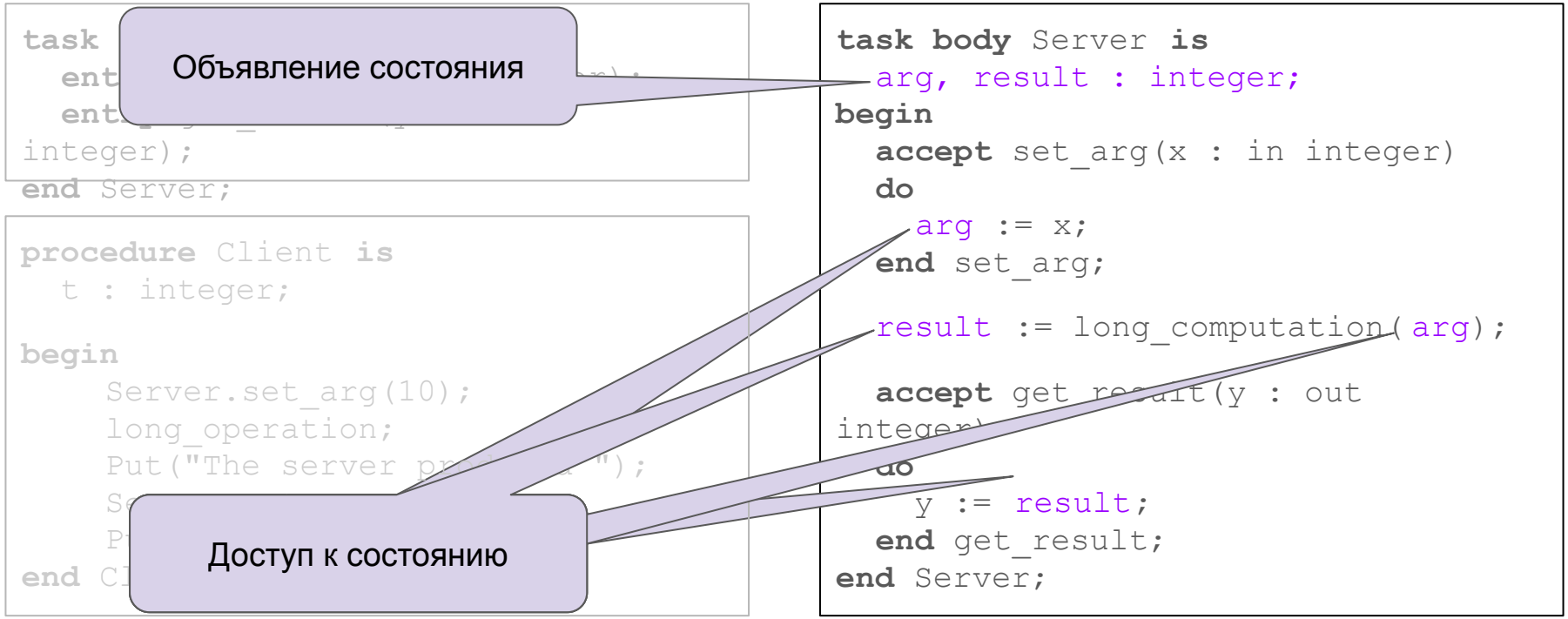
Входы

```
task body Server is
  arg, result : integer;
begin
  accept set_arg(x : in integer)
  do
    arg := x;
  end set_arg;

  result := long_computation(arg);

  accept get_result(y : out
integer)
  do
    y := result;
  end get_result;
end Server;
```

Задачи, входы и рандеву в языке Ада на примере

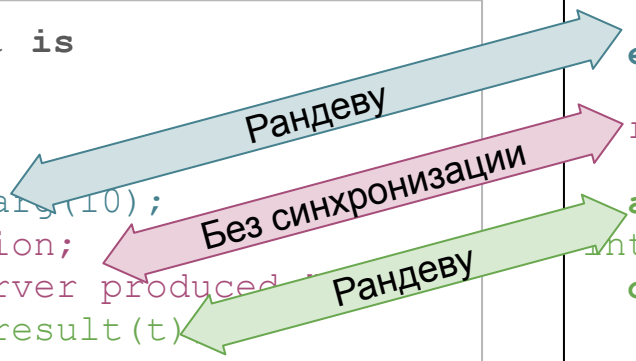


Задачи, входы и рандеву в языке Ада на примере

```
task Server is
  entry set_arg(x : in integer);
  entry get_result(y : out
integer);
end Server;
```

```
procedure Client is
  t : integer;

begin
    Server.set_arg(10);
    long_operation;
    Put("The server produced
    Server.get_result(t)
    Put(t);
end Client;
```




```
task body Server is
  arg, result : integer;
begin
  accept set_arg(x : in integer)
  do
    arg := x;
  end set_arg;

  result := long_computation(arg);

  accept get_result(y : out
integer)
  do
    y := result;
  end get_result;
end Server;
```

Длительно работающий сервер и выбор входа

```
task body Server is
  running : boolean;
  arg, result : integer;
begin
  running := true;

  loop
    select
      accept set_arg(x : in
integer)
      do
        arg := x;
      end set_arg;
    or
      accept stop
      do
        running := false;
      end stop;
    end select;
    exit when not running;
    result := long_computation(arg);
    accept get_result(y : out integer)
    do
      y := result;
    end get_result;
  end loop;
end Server;
```

Задачи, входы и рандеву в языке Ада в общем виде

- Параллельная задача обладает *интерфейсом* и *состоянием*.
- Состояние скрыто от внешнего мира, *инкапсулировано*.
- Интерфейс задачи состоит из т. н. *входов* (entry).
- Входы имеют *имена*, способны *принимать* и *возвращать* значения.
- Входы имеют доступ к внутреннему состоянию задачи.
- Входы похожи на подпрограммы — процедуры или функции.
- Перечень входов с их сигнатурами — часть *типа* задачи.
- Таким образом, задача подобна *объекту* в ООП.
- Для клиента общение с задачей выглядит как вызов методов объекта.
- Задача (поток) — разновидность объекта с состоянием и поведением!

Задачи, входы и рандеву в языке Ада в общем виде

- Синхронный обмен вызовами-возвратами между двумя задачами называется *рандеву* (фр. *rendez-vous* — свидание).
- Задачи на рандеву находятся в несимметричных ролях: клиент и сервер.
- Участки клиента и сервера вне рандеву работают без синхронизации.
- Сервер *иногда* готов принять обращение к своему *входу* (accept).
- Готовность приёма может охватывать несколько разных входов (select).
- Клиент *иногда* желает обратиться к входу задачи-сервера.
- Задача, пришедшая на рандеву первой, ждёт прихода второй.
- Когда обе задачи пришли на рандеву, вход срабатывает синхронно.
- По окончании рандеву задачи продолжают работу без синхронизации.

Общая характеристика рандеву

- Обмен данными между потоками можно реализовать через общую область памяти с синхронизацией и оповещением.
- Подробности представления данных видны обоим потокам.
- В обоих потоках нужно явное управление семафорами и оповещениями.
- Логика потоков отягощена сразу двумя видами деталей реализации.
- Рандеву — высокоуровневый механизм обмена данными.
- Код клиента содержит лишь обращение к входам сервера.
- Код сервера отвечает за обмен данными с внутренним состоянием.
- Сложность синхронизации, ожидания и пробуждения потоков — скрыта.
- Сложность внутреннего представления данных — скрыта.

Поддержка рандеву в различных языках

- В языке Ада рандеву присутствуют с 1980 года.
- За пределами Ады данная концепция практически неизвестна.
- Luis Mateu, Jose M. Piquer and Juan Leo, Universidad de Chile, “Resurrecting Ada's Rendez-Vous in Java”, 1998.
- Неполный аналог: `java.util.concurrent.Exchanger`.
  - Двунаправленный канал обмена данными.
  - В поток-сервер и из него передаётся только один объект одного и того же типа.
- Есть некоторые общие черты с асинхронными задачами (C++ 11).
- Отдалённое сходство с сопрограммами (C++ 20).
- Не удалось найти реализацию рандеву на C++ с открытым кодом.

Основные черты реализации рандеву на C++

- Приложение создаёт объект, отвечающий за рандеву.
- Объект-рандеву отвечает за синхронизацию желания клиента и готовности сервера к взаимодействию.
- Сервер и клиент получают доступ к разным аспектам объекта-рандеву.
  - Сервер получает подобъект, реализующий отправку данных.
  - Клиент получает объект для приёма данных.
- Класс рандеву — шаблон, параметризованный интерфейсом `ISrv`.
- Поток-сервер через канал отправки получает (ссылку на) объект, реализующий интерфейс `ISrv`, и обращается к его функциям-членам.
- Объект с реализацией `ISrv` предоставляется потоком-сервером.

Основные понятия библиотеки rendezvouscxx

- Ворота (gate). Объект, обеспечивающий рандеву между клиентом и сервером. К нему могут подключаться клиент и сервер.
- Клиентский жетон (client token). RAII-объект, возвращаемый потоку-клиенту при подключении к воротам.
- Поток-клиент, запросивший подключение, блокируется до прибытия потока-сервера, затем получает жетон.
- Пока жетон не уничтожен, поток-сервер не может покинуть рандеву.
- Наличие жетона у клиента означает готовность сервера к общению.
- Жетон содержит ссылку на объект, реализующий интерфейс `ISrv`.
- Поток-клиент через жетон вызывает функции `ISrv`.

Основные понятия библиотеки rendezvouscxx

- Поток-сервер заявляет о готовности к общению, подключаясь к воротам.
- Для подключения к воротам поток-сервер предоставляет (ссылку на) объект, реализующий интерфейс `ISrv`.
- Поток-сервер, запросивший подключение к воротам, блокируется до прихода и последующего ухода клиента.
- Серверу аналог жетона не нужен. Деструкцией жетона клиент управляет окончанием рандеву. Сервер на рандеву остаётся, пока нужен клиенту.
- Клиентские ворота (client gate). Аспект объекта gate, содержащий только функцию подключения потока-клиента.
- Серверные ворота. Аспект объекта gate для потока-сервера.

Пример

- Поток-сервер умеет по данному URL выполнить запрос и вернуть ответ.
- Поток-клиент для своей работы иногда нуждается в HTTP-запросах.
- Поток-клиент поручает запрос потоку-серверу, продолжает свою работу и спустя некоторое время возвращается к потоку-серверу за ответом.
- Поочерёдно происходят два типа рандеву:
  - Сообщить потоку-серверу URL и инициировать запрос;
  - Получить ответ на запрос.
- Дополнительно поток-клиент может сообщить потоку-серверу, чтобы тот прекратил работу.

Интерфейсы взаимодействия

```
class i_http_request : public rendezvouscxx::i_server {
public:
    virtual void set_url(std::string const&) = 0;
    virtual void stop() = 0;
};

class i_http_response : public rendezvouscxx::i_server {
public:
    virtual int status() const = 0;
    virtual std::string text() const = 0;
};
```

Инициализация ворот и запуск потоков

```
gate_t<i_http_request> request_gate;
gate_t<i_http_response> response_gate;

std::thread client_thread {
    client_fn,
    request_gate.client(),
    response_gate.client() };

std::thread server_thread {
    server_fn,
    request_gate.server(),
    response_gate.server() };
```

Клиентский поток

```
void client_fn(
  gate_t<i_http_request>::client_t request_gate,
  gate_t<i_http_response>::client_t response_gate)
{
  {
    auto request_token = request_gate.connect();
    request_token->server().request("http://data.source.srv/page");
  }
  other_long_operations();
  {
    auto response_token = response_gate.connect();
    std::cout << response_token->server().text();
  }
}
```

Клиентский поток

```
void client_fn(
  gate_t<i_http_request>::client_t request_gate,
  gate_t<i_http_response>::client_t response_gate)
{
  {
    auto request_token = request_gate.connect();
    request_token->server().request("http://data.source.srv/page");
  }
  other_long_operations();
  {
    auto response_token = response_gate.connect();
    std::cout << response_token->server().text();
  }
}
```

Клиентский поток

```
void client_fn(
  gate_t<i_http_reques
  gate_t<i_http_respons
{
  {
    auto request_token = request_gate.connect();
    request_token->server().request("http://data.source.srv/page");
  }
  other_long_operations();
  {
    auto response_token = response_gate.connect();
    std::cout << response_token->server().text();
  }
}
```

- Клиентский поток блокируется до прихода сервера на рандеву.
- Поток-сервер блокируется.
- Клиент получает жетон.

Клиентский поток

```
void client_fn(
  gate_t<i_http_request>::client_t request_gate,
  gate_t<i_http_response>::client_t response_gate)
{
  {
    auto request_token = request_gate.connect();
    request_token->server().request("http://data.source.srv/page");
  }
  other_long_operations
  {
    auto response_token
    std::cout << response_token->server().text();
  }
}
```

- Через жетон клиент вызывает методы объекта-сервера.
- Объект-сервер имеет доступ к состоянию потока-сервера.

Клиентский поток

```
void client_fn(
  gate_t<i_http_request>::client_t request_gate,
  gate_t<i_http_response>::client_t response_gate)
{
  {
    auto request_token = request_gate.connect();
    request_token->server().request("http://data.source.srv/page");
  }
  other_long_operations();
  {
    auto response_token = resp                    nnect();
    std::cout << respon
  }
}
```

- Деструкция жетона заканчивает рандеву.
- Поток-сервер разблокируется и продолжает свою работу.

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::server_t request_gate,
  gate_t<i_http_response>::server_t response_gate)
{
  class http_request_impl_t : public i_http_request { /* . . . */ };
  class http_response_impl_t : public i_http_response { /* . . . */ };
  while (true) {
    http_request_impl_t http_request_impl;
    request_gate.connect(http_request_impl);
    if (!http_request_impl.is_running()) break;
    auto const result = http_lib::get(http_request_impl.url());
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::server_t request_gate,
  gate_t<i_http_response>::server_t response_gate)
{
  class http_request_impl_t : public i_http_request { /* . . . */ };
  class http_response_impl_t : public i_http_response { /* . . . */ };
  while (true) {
    http_request_impl_t http_request_impl;
    request_gate.connect(http_request_impl);
    if (!http_request_impl.is_running() ) break;
    auto const result = http_lib::get( http_request_impl.url() );
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::
  gate_t<i_http_response>::
{
  class http_request_impl
  class http_respons
  while (true)
    http_request_impl_t http_request_impl;
    request_gate.connect(http_request_impl);
    if (!http_request_impl.is_running() ) break;
    auto const result = http_lib::get( http_request_impl.url() );
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

- Содержит состояние потока-сервера.
- Предоставляет интерфейс для управления состоянием.

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::
  gate_t<i_http_response>::
{
  class http_request_impl_t
  class http_response_i
  while (true) {
    http_request_impl_t http_request_impl;
    request_gate.connect(http_request_impl);
    if (!http_request_impl.is_running() ) break;
    auto const result = http_lib::get( http_request_impl.url() );
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

```
i_http_request { /* . . . */ };
public i_http_response { /* . . . */ };
```

- Поток-сервер блокируется, пока клиент не покинет рандеву.
- Клиент вызывает функции объекта `http_request_impl`.

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::
  gate_t<i_http_response>::
{
  class http_request_impl_t                  i_http_request { /* . . . */ };
  class http_response_im          public i_http_response { /* . . . */ };
  while (true) {
    http_request_impl_t http_request_impl;
    request_gate.connect(http_request_impl);
    if (!http_request_impl.is_running() ) break;
    auto const result = http_lib::get( http_request_impl.url() );
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

- Поток-сервер возобновляет свою работу.
- Видны изменения в состоянии, сделанные клиентом.

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::
  gate_t<i_http_response>::
{
  class http_request_impl_t :                 http_request { /* . . . */ };
  class http_response_impl                    ic i_http_response { /* . . . */ };
  while (true) {
    http_request_impl                         p_request_impl;
    request_gate.c                            ct(http_request_impl);
    if (!http_request_impl.is_running() ) break;
    auto cons  result = http_lib::get( http_request_impl.url() );
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

- Сформировать объект с состоянием для второго рандеву.
- Его состояние неизменяемо: клиент только читает данные.

Поток-сервер

```
void server_fn(
  gate_t<i_http_request>::
  gate_t<i_http_response>::
{
  class http_request_impl_t :                    ttp_request { /* . . . */ };
  class http_response_impl_t                     i_http_response { /* . . . */ };
  while (true) {
    http_request_impl_t                          request_impl;
    request_gate.con                            (http_request_impl);
    if (!http_req                               impl.is_running() ) break;
    auto const                                  sult = http_lib::get( http_request_impl.url() );
    http_response_impl_t http_response_impl(result.code, result.text);
    response_gate.connect(http_response_impl);
}
```

- Предоставить клиенту возможность прочитать данные.
- Приостановить поток, пока клиент не закончит.

Хранитель состояния рандеву

```
class http_request_impl_t : public i_http_request {
public:
  void request(std::string const& url) override { m_url = url; }
  void stop() override { m_is_running = false; }

  std::string url() const { return m_url; }
  bool m_is_running() const { return is_running; }

private:
  std::string m_url;
  bool m_is_running = true;
};
```

Хранитель состояния рандеву

```
class http_request_impl_t : public i_http_request {
public:
  void request(std::string const& url) override { m_url = url; }
  void stop() override { m_is_running = false; }

  std::string url() const { return m_url; }
  bool m_is_running() const { return is_running; }

private:
  std::string m_url;
  bool m_is_running = true;
};
```

Хранитель состояния рандеву

```
class http_request_impl_t : public i_http_request {
public:
  void request(std::string const& url) override { m_url = url; }
  void stop() override { m_is_running = false; }

  std::string url() const { return m_url; }
  bool m_is_running() const { return is_running; }

private:
  std::string m_url;
  bool m_is_running = true;
};
```

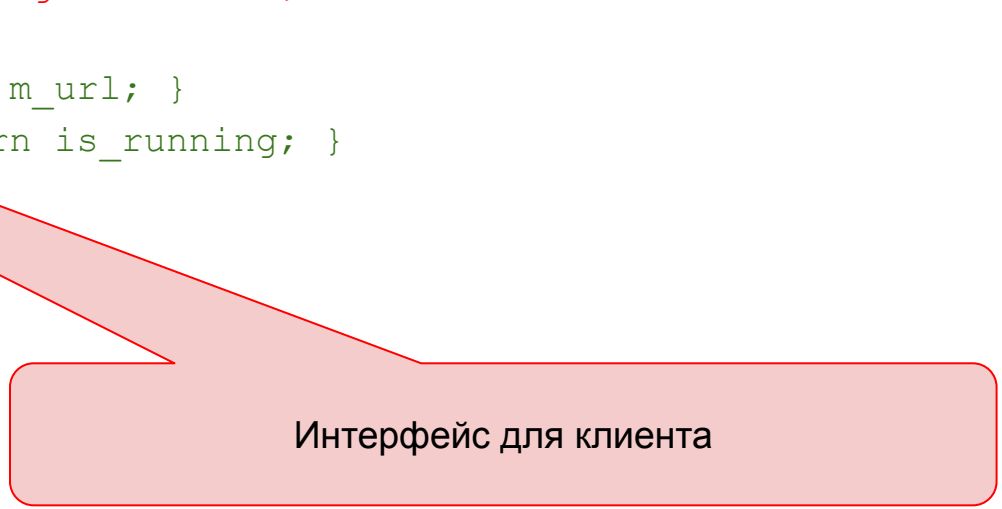
Данные состояния

Хранитель состояния рандеву

```
class http_request_impl_t : public i_http_request {
public:
  void request(std::string const& url) override { m_url = url; }
  void stop() override { m_is_running = false; }

  std::string url() const { return m_url; }
  bool m_is_running() const { return is_running; }

private:
  std::string m_url;
  bool m_is_running = true;
};
```



Хранитель состояния рандеву

```
class http_request_impl_t : public i_http_request {
public:
  void request(std::string const& url) override { m_url = url; }
  void stop() override { m_is_running = false; }

  std::string url() const { return m_url; }
  bool m_is_running() const { return is_running; }

private:
  std::string m_url;
  bool m_is_running = true;
};
```

Устройство ворот: состояние

```
std::mutex m_mx_client; // клиент пытается подключиться
std::mutex m_mx_server; // сервер пытается подключиться
std::mutex m_mx_common; // взаимосвязь клиента и сервера

std::condition_variable m_cv_client_chaned;
std::condition_variable m_cv_server_chaned;
std::condition_variable m_cv_disconnected;

bool m_is_client_connected;
ISrv* m_server;
bool m_is_mutual_connection;
```

Устройство ворот: подключение клиента

```
std::unique_ptr<client_token_t> connect_client() {
    std::unique_lock client_lock(m_mx_client);
    std::unique_lock common_lock(m_mx_common);
    m_is_client_connected = true;
    m_cv_client_chaned.notify_all();
    m_cv_server_chaned.wait(
        common_lock,
        [this]() { return m_server != nullptr; });
    m_is_mutual_connection = true;
    static_cast<i_server*>(m_server)->on_client_connected();
    return std::make_unique<client_token_t>(
        *this, std::move(client_lock));
}
```

Устройство ворот: подключение клиента

```
std::unique_ptr<client_token_t> connect_client() {
    std::unique_lock client_lock(m_mx_client);
    std::unique_lock common_lock(
    m_is_client_connected = t
    m_cv_client_chaned.notify_
    m_cv_server_chaned.wait(
        common_lock,
        [this]() { return m_se
    m_is_mutual_connection =
    static_cast<i_server*>(m_server)->on_client_connected();
    return std::make_unique<client_token_t>(
        *this, std::move(client_lock));
}
```

Только один клиент может войти в данное рандеву

Другие клиенты ждут, пока жетон у этого клиента

Устройство ворот: подключение клиента

```
std::unique_ptr<client_token_t> connect_client() {
    std::unique_lock client_lock(m_mx_client);
    std::unique_lock common_lock(m_mx_common);
    m_is_client_connected = true;
    m_cv_client_chaned.notify_all();
    m_cv_server_chaned.wait(
        common_lock,
        [this]() { return m_server != nullptr; });
    m_is_mutual_connection = tr
    static_cast<i_server*>(m_server
    return std::make_unique<cl
        *this, std::move(clien
}
```

Если сервер уже пришёл на рандеву, оповестить

Устройство ворот: подключение клиента

```
std::unique_ptr<client_token_t
    std::unique_lock client_lo
    std::unique_lock common_lock(m_
    m_is_client_connected = tr
    m_cv_client_chaned.notify_all();
    m_cv_server_chaned.wait(
        common_lock,
        [this]() { return m_server != nullptr; });
    m_is_mutual_connection = true;
    static_cast<i_server*>(m_server)->on_client_connected();
    return std::make_unique<client_token_t>(
        *this, std::move(client_lock));
}
```

Убедиться, что сервер пришёл на рандеву

Устройство ворот: подключение сервера

```
bool connect_server(ISrv& server) {
    std::unique_lock lock(m_mx_server);
    std::unique_lock common_lock(m_mx_common);
    m_cv_client_chaned.wait(
      common_lock, [this]() { return m_is_client_connected; });
    m_server = &server;
    m_cv_server_chaned.notify_all();
    m_cv_client_chaned.wait(
      common_lock, [this]() { return !m_is_client_connected; });
    m_is_mutual_connection = false;
    m_cv_disconnected.notify_all();
}
```

Устройство ворот: подключение сервера

```
bool connect_server(ISrv& server) {
    std::unique_lock lock(m_mx_server);
    std::unique_lock common_lock(m_mx_common);
    m_cv_client_chaned.wait(
      common_lock, [this]() {
    m_server = &server;
    m_cv_server_chaned.notify_all();
    m_cv_client_chaned.wait(
      common_lock, [this]() { return !m_is_client_connected; });
    m_is_mutual_connection = false;
    m_cv_disconnected.notify_all();
}
```

Только один сервер может войти в данное рандеву

Устройство ворот: подключение сервера

```
bool connect_server(ISrv& server) {
    std::unique_lock lock(m_mx_server);
    std::unique_lock common_lock(m_mx_common);
    m_cv_client_chaned.wait(
      common_lock, [this]() { return m_is_client_connected; });
    m_server = &server;
    m_cv_server_chaned.notify_
    m_cv_client_chaned.wait(
      common_lock, [this]() {
    m_is_mutual_connection = false;
    m_cv_disconnected.notify_all();
}
```

Убедиться, что клиент пришёл на рандеву

Устройство ворот: подключение сервера

```
bool connect_server(ISrv& server) {
    std::unique_lock lock(m_mx
    std::unique_lock common_lo
    m_cv_client_chaned.wait(
      common_lock, [this]() { return m_is_client_connected; });
    m_server = &server;
    m_cv_server_chaned.notify_all();
    m_cv_client_chaned.wait(
      common_lock, [this]() { return !m_is_client_connected; });
    m_is_mutual_connection = false;
    m_cv_disconnected.notify_all();
}
```

Известить клиента о своём приходе

Устройство ворот: подключение сервера

```
bool connect_server(ISrv& server) {
    std::unique_lock lock(m_mx
    std::unique_lock common_lo
    m_cv_client_chaned.wait(
      common_lock, [this]() { re       _is_client_connected; });
    m_server = &server;
    m_cv_server_chaned.n     ify_all();
    m_cv_client_chaned.wait(
      common_lock, [this]() { return !m_is_client_connected; });
    m_is_mutual_connection = false;
    m_cv_disconnected.notify_all();
}
```

Дождаться ухода клиента

Устройство ворот: подключение сервера

```
bool connect_server(ISrv& server) {
    std::unique_lock lock(m_mx
    std::unique_lock common_lo
    m_cv_client_chaned.wait(
      common_lock, [this]() { ret        _is_client_connected; });
    m_server = &server;
    m_cv_server_chaned.noti    _all();
    m_cv_client_chaned.wait(
      common_lock, [this]() { return !m_is_client_connected; });
    m_is_mutual_connection = false;
    m_cv_disconnected.notify_all();
}
```

Подтвердить свой уход

Устройство ворот: отключение клиента

```
void on_client_disconnecting()
{
    std::unique_lock lock(m_mx_common);
    m_server->on_client_disconnecting();
    m_is_client_connected = false;
    m_server = nullptr;
    m_cv_client_chaned.notify_all();
    m_cv_disconnected.wait(
        lock,
        [this]() { return !m_is_mutual_connection; });
}
```

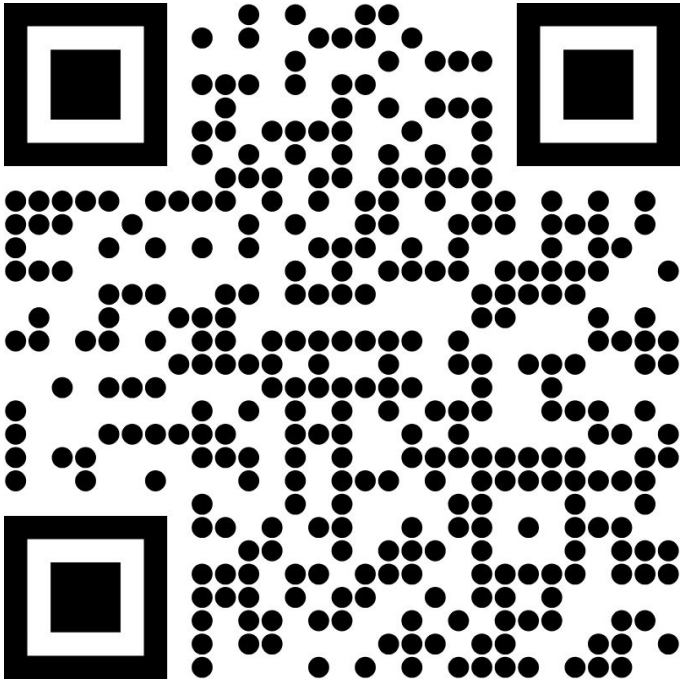
Вызывается в деструкторе клиентского жетона

Дополнительная функциональность

- Ворота можно закрыть, отключив возможность данного рандеву.
- К закрытым воротам не может подключиться ни клиент, ни сервер.
- Попытка подключения к закрытым воротам завершается сразу.
- Клиент при этом получает `nullptr` вместо жетона.
- Сервер получает `false`, если ворота закрыты.
- Ни клиент, ни сервер сами не могут закрыть ворота.
- Это может сделать тот код, который владеет объектом `gate`.
- Ещё одно полезное расширение — ограниченное время ожидания.
  - В репозиторий ещё не добавлено.

Заключительные замечания

- Реализованный в предлагаемой библиотеке механизм отличается от рандеву из языка Ада в в нескольких деталях.
- На Аде рандеву всегда включает вызов клиентом ровно одного входа на стороне сервера.
- В библиотеке `rendezvouscxx` клиент может вызывать методы сервера в любом количестве и порядке, пока обладает жетоном.
- Синтаксическое неудобство: нужно объявлять отдельную переменную для жетона и окружать её область видимости фигурными скобками.
- Оператор select в Аде позволяет комбинировать любые входы сервера.
- В `rendezvouscxx` все входы должны быть методами одного класса.



https://github.com/vadimvinnik/rendezvouscxx

```
    return thanks;
} // the talk
```