Абстракция блокирующего канала передачи данных

Темиргалеев Е. Э.
(Т.Е.Э. 11 окт 2016 г. 10:08:20)

Постановка задачи. Задача оформилась в контексте программирования протокола взаимодействия с MongoDB [1]. Цель — сделать его реализацию в рамках одного модуля (*ipuiK84*) независимой от конкретных сетевых библиотек (*ipuiK64*, *ipuiK79*).

Абстракция каркаса ББ *CommStreams.Stream* — неблокирующих соединений — не устраивала по следующим причинам:
1) Связь между абонентами (подключение клиента к серверу) устанавливается при создании объекта *CommStreams.Stream* через вызов *CommStreams.NewStream*

PROCEDURE **NewStream** (protocol, localAdr, remoteAdr: ARRAY OF CHAR; OUT s: Stream; OUT res: INTEGER)

и сохраняется до отключения одной из сторон. После этого потребуется снова создавать объект. При этом MongoDB-драйверу рекомендуется [2] обладать свойством возобновления соединения. Т. е. для поддержания этого свойства "драйвер" должен хранить параметры соединения. В нашем случае — минимальная реализация "драйвера" как реализация протокола — хотелось избежать хранения параметров, которые к самому протоколу никакого отношения не имеют.
2) Неблокирующие соединения не требуются. Применение блокирующих соединений упрощает программирование "драйвера".

Канал передачи данных. (Термин выбран исходя из аналогии с номером 157 из ГОСТ 17657-70 "Передача данных. Термины и определения".) Решение "проблемы" пункта (1) выбрано следующее:

- конкретная реализация канала хранит выставляемые клиентом в момент создания и/или настройки объекта специфические для неё параметры подключения;
- "драйвер" получает от клиента только ссылку на этот объект, используя его методы для передачи данных;
- соединение выполняется неявно перед передачей данных.

Эволюция модели (набор операций):

- Сначала предполагалось, что канал при ошибках будет восстанавливать соединение и повторять передачу.
- Поскольку с другой стороны такой же канал не применяется, повторная передача порции набора данных после переустановления соединения будет воспринята принимающей стороной как начало нового набора. Были введены явные операции начала/завершения сеанса передачи и соответствующие им коды результата операций передачи данных. Чтобы клиент мог необходимым способом организовать процесс и принимать решение: повторять передачу порции данных или начинать всё заново.
- Не было учтено, что каналом могут пользоваться несколько клиентов. В этом случае отслеживание состояние сеанса связи для каждого из них уже не простая задача. Операции начала/завершения сеанса были исключены.
- Введена операция разрыва соединения. Обеспечивает клиенту возможность гарантированной передачи набора данных с начала — через новое соединение.

Итоговый набор операций:

* Приём/передача *n* байт данных (с установлением соединения при необходимости). Результат:

- 0 — данные приняты/переданы полностью;
- 1 — данные приняты/переданы частично (от 1 до *n*-1 байт);
- 2 — ошибка приёма/передачи при сохранении соединения;
- 3 — обрыв соединения;
- 4 — нет соединения. Выставляется, если установка соединения не удалась.

* Разрыв (закрытие) соединения.

Реализация канала передачи данных (*ipuiK82*). Интерфейс реализации *ipuiK82.Channel*:

```
TYPE Channel = POINTER TO ABSTRACT RECORD
   (c: Channel) Open-, NEW, ABSTRACT; — реализация установки соединения
   (c: Channel) Close-, NEW, ABSTRACT; — реализация закрытия соединения
   (c: Channel) Receive- (VAR x: ARRAY OF BYTE; beg, len: INTEGER; OUT read: INTEGER), NEW, ABSTRACT; — реализация приёма
   (c: Channel) Send- (IN x: ARRAY OF BYTE; beg, len: INTEGER; OUT written: INTEGER), NEW, ABSTRACT; — реализация отправки
END;
```

c.Open вызывается, когда нет соединения и должен выставить результат *c.res* из {0, 4}. *c.Receive* и *c.Send* вызываются при наличии соединения и выставляют результат *c.res* из {0..3}. Примеры:

- *ipuiK83* реализует TCP-канал, используя библиотеку *ipuiK79*;
- *ipuiK102* реализует TCP/Unix-каналы, используя библиотеку *ipuiK64*.

Клиентский интерфейс *ipuiK82.Channel*:

```
TYPE Channel = POINTER TO ABSTRACT RECORD
      res: INTEGER; — результат
      (c: Channel) ReadBytes (VAR x: ARRAY OF BYTE; beg, len: INTEGER; OUT read: INTEGER), NEW; — приём данных
      (c: Channel) WriteBytes (IN x: ARRAY OF BYTE; beg, len: INTEGER; OUT written: INTEGER), NEW; — передача данных
      (c: Channel) Reset, NEW — разрыв соединения
   END;
```

Пример — приём данных *contentLen* байт в файл:

```
IMPORT Channels := ipuiK82;
…
VAR c: Channels.Channel; f: Files.File; w: Files.Writer;
   contentLen, read, len: INTEGER; buf: ARRAY 4096 OF BYTE;
…
c.res := 0;
WHILE (0 < contentLen) & (c.res IN {0, 1}) DO
   len := MIN(contentLen, LEN(buf)); c.ReadBytes(buf, 0, len, read);
   IF c.res IN {0, 1} THEN DEC(contentLen, read); w.WriteBytes(buf, 0, read) END
END;
ASSERT((0 = contentLen) & (c.res = 0))
```

Пример — отправка файла:

```
VAR c: Channels.Channel; f: Files.File; r: Files.Reader;
   contentLen, len, i, written: INTEGER; buf: ARRAY 4096 OF BYTE;
…
r := f.NewReader(…); r.SetPos(0); contentLen := f.Length(); c.res := 0;
WHILE (0 < contentLen) & (c.res = 0) DO
   len:= MIN(contentLen, LEN(buf));
   r.ReadBytes(buf, 0, len); ASSERT(~r.eof, 100); DEC(contentLen, len);
   i := 0; WHILE (i < len) & (c.res IN {0, 1}) DO
      c.WriteBytes(buf, i, len - i, written);
      INC(i, written)
   END
END;
ASSERT((0 = contentLen) & (c.res = 0))
```

Отказ от централизованного создания каналов в *ipuiK82*. *CommStreams* средствами *Meta*

вызывает процедуры-конструкторы из переданного в параметре *protocol* модуля реализации:

PROCEDURE **NewStream** (protocol, localAdr, remoteAdr: ARRAY OF CHAR; OUT s: Stream; OUT res: INTEGER)

Таким образом уничтожается зависимость между клиентскими модулями и модулями реализаций. Цена этому: передача параметров через литерные цепочки в универсальном — текстовом — представлении, которое вынуждены декодировать модули реализации.

Ради упрощения модулей реализации от этой схемы было решено отказаться. Плата: зависимость между клиентским модулем и используемыми модулями реализации:

```
MODULE Client;
   IMPORT WinChannels := ipuiK83, MongoConnections := ipuiK85, …;
   …
   VAR connection: MongoConnections.Connection;
   …
   PROCEDURE WinSetup* (IN addr: ARRAY OF CHAR; port: INTEGER);
   BEGIN
      MongoConnections.InitConnection(connection);
      MongoConnections.SetChannel(connection, WinChannels.NewChannel(addr, port));
      …
   END WinSetup;
…
END Client.
```

Глобальная переменная-результат как средство устранения зависимости на уровне команд. Введена глобальная переменная *ipuiK82.result* в которую командные модули, сопутствующие модулям реализаций каналов (*ipuiK103* для *ipuiK83*, *ipuiK104* для *ipuiK102*) помещают новый объект канала:

```
MODULE ipuiK103;
   IMPORT ipuiK82, ipuiK83;

   PROCEDURE NewTCPChannel* (IN remoteAddr: ARRAY OF CHAR; port: INTEGER);
   BEGIN
      ipuiK82.result := ipuiK83.NewChannel(remoteAddr, port)
   END NewTCPChannel;

END ipuiK103.
```

Команды клиентских модулей забирают новый канал из *ipuiK82.result*:

```
MODULE Client;
   IMPORT Channels := ipuiK82, MongoConnections := ipuiK85, …;
   …
   VAR connection: MongoConnections.Connection;
   …
   PROCEDURE Setup*;
      VAR   c: Channels.Channel;
   BEGIN
      MongoConnections.InitConnection(connection);
      ASSERT(Channels.result # NIL, 20); c := Channels.result; Channels.result := NIL;
      MongoConnections.SetChannel(connection, c);
      …
   END Setup;
…
END Client.
```

Благодаря применению такого соглашения зависимость клиентского модуля от модуля

релизации устраняется. Конкретные детали настройки уходят на уровень выше и могут быть вынесены в командную строку. Как в тексте,

```
Win !"ipuiK103.NewTCPChannel('127.0.0.1', 27017)"
Lin !"ipuiK104.NewTCPChannel('127.0.0.1', 27017)"
!Client.Setup
```

так и программно:

```
Dialog.Call("ipuiK103.NewTCPChannel('127.0.0.1', 27017); Client.Setup", " ", res)
```

Способ передачи параметров команды через глобальную переменную "идёт" с системы Оберон (глобальная переменная *Oberon.Par* [3, 3.3.1]). В Блэкбокс такая схема используется коммандерами и ссылками (*DevCommanders.par* и *StdLinks.par*).

Список источников

1. MongoDB Wire Protocol. // URL: http://docs.mongodb.org/meta-driver/latest/legacy/mongodb-wire-protocol/ (дата обращения 20.08.2014)
2. Feature Checklist for MongoDB Drivers. // URL: http://docs.mongodb.org/meta-driver/latest/legacy/feature-checklist-for-mongodb-drivers/ (дата обращения 24.09.2014)
3. Wirth N., Gutknecht J. Project Oberon: The Design of an Operating System, a Compiler, and a Computer // URL: http://oberoncore.ru/library/wirth_gutknecht_project_oberon_the_design_of_an_operating_system_and_compiler (дата обращения: 11.10.2016)