

07. Создание своего протокола + NIO

Лекции по информатике
для студентов второго курса Высшей школы ИТИС КФУ
2020

Ференец Александр Андреевич

старший преподаватель кафедры программной инженерии

С использованием материалов
к. т. н., доцента кафедры программной инженерии Абрамского М.М.

aferenets@it.kfu.ru

СОЗДАНИЕ ПРОТОКОЛА. Вопросы

1. Кто начинает “разговор”?
2. Как понять, что “разговор” окончен?
3. Когда перестать “слушать” и можно начать “говорить”?
4. Как компактнее уместить информацию?
5. Как лучше “объяснить”, о чём информация?

1. Инициализация сеанса/сессии
2. Ping/Timeout и объявление окончания сеанса/сессии
3. Указание конца пакета, длины сообщения
4. Дробление на пакеты, архивирование
5. Выделение технической части пакета

СОЗДАНИЕ ПРОТОКОЛА. Основные этапы

- ❑ Формат передачи и описания полезной нагрузки
- ❑ Начало и конец сеанса связи, формат stateless
- ❑ Выделение сообщений, этапов связи, синхронизация

Чего мы хотим от другой точки соединения?

Важно составить примерный список исходя из функционала приложения и организации синхронизации (см. другие слайды).

Endpoint может реагировать по модели Listener

Пример типов сообщения – методы HTTP (GET, PUT, OPTIONS и т.д.)

Можно использовать MIME-тип

Не путать тип Сообщения и тип данных!

Например

Типы Сообщения: Сообщение в чат, Изменение названия чата

Тип данных для сообщений выше: Просто текст (text/plain)


Пример из HTTP: заголовок *Content-type*

Content-Type: text/html; charset=UTF-8


Content-type: image/png

- ✓ Контрольная сумма
- ✓ Язык
- ✓ Используемая локаль
- ✓ Информация о генераторе содержимого

Желательно заменить на что-то более продвинутое
`new Connection(Socket s);`



```
List<Socket> sockets = new ArrayList<>(25);  
sockets.add(server.accept());
```



Индекс можно считать идентификатором
при некоторых условностях
`sockets.get(...)`

Конечно, ArrayList имеет множество особенностей, которые могут привести к путанице в сохраняемых сокетах. В идеальном случае необходимо создать свой пул сокетов – специальную структуру.

Сессия...

Если создаётся сокет, через который будет принят уникальный идентификатор, его можно считать “старым” сокетом, поместив пул под тем же идентификатором.

HTTP-сессия – общая практика в веб-приложениях. Вполне можно последовать тому же пути.

Стандартная работа с сокетом:

```
Socket socket = server.accept();  
InputStream in =  
s.getInputStream();  
int b;  
while( (b=in.read()) != -1 ) {  
    // process b  
}
```

К тому же есть `InputStream::available()`

А ещё есть `Socket::isConnected()`

Не работает или ненадёжно

Решение?

Отловить `SocketException` и экземпляров различных дочерних классов.

А как отследить вне активного периода записи или чтения?

Стандартная работа с сокетом:

```
Socket socket = server.accept();  
InputStream in =  
s.getInputStream();  
int b;  
while( (b=in.read()) != -1 ) {  
    // process b  
}
```

К тому же есть `InputStream::available()`

А ещё есть `Socket::isConnected()`

Не работает или ненадёжно

Решение?

Отловить `SocketException` и экземпляров различных дочерних классов.

А как отследить вне активного периода записи или чтения?

ping – pong

~~read() == 1, available, IOException~~

1. Фиксированная длина сообщения
2. Разделяющий символ (символ окончания)
3. Указание длины сообщения в фиксированной части сообщения

Как это сделано в HTTP?

~~read() == 1, available, IOException~~

1. Фиксированная длина сообщения
2. Разделяющий символ (символ окончания)
3. Указание длины сообщения в фиксированной части сообщения

Как это сделано в HTTP?

2 и 3!

- 1.
2. Символ переноса или два символа переноса
3. Content-length для тела сообщения

Для активного взаимодействия необходимо поддерживать диалог или хотя бы отвечать статусным кодом на запрос.

Endpoint может обрабатывать запрос долго и ответы могут перемешаться!

- Запрос может содержать уникальный идентификатор, который будет и в ответе.
- Можно привязываться ко времени.
- ...

java.nio.Buffer

ByteBuffer, CharBuffer, DoubleBuffer, FloatBuffer, IntBuffer, LongBuffer, ShortBuffer

Методы (Вместо точек Int, Boolean, Float,...)

- Buffer.allocate(int capacity)
- put...(...)
- as...Buffer()
- get...()
- get...(int i)
- array()
- clear()
- rewind()

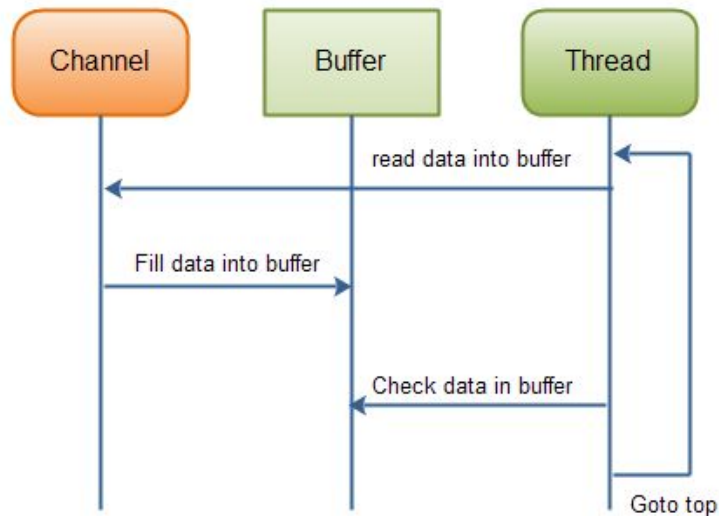
<CODE>

Рассмотрение примера с передачей цвета + примера с простым универсальным протоколом на основе класса Message

New IO (Non-blocking IO)

```
RandomAccessFile aFile = new RandomAccessFile("data.txt", "rw");  
FileChannel inChannel = aFile.getChannel();  
// создаем буфер размера 48 байтов (!)  
ByteBuffer buf = ByteBuffer.allocate(48);  
// читаем из канала в буфер, возвращается реальное количество считанных байтов  
int bytesRead = inChannel.read(buf);  
while (bytesRead != -1) {  
  
    // режим чтения полученных данных из буфера  
    buf.flip();  
    while(buf.hasRemaining()){  
        System.out.print(buf.get());  
    }  
    // режим записи новых данных в буфер  
    buf.clear();  
    bytesRead = inChannel.read(buf);  
}  
aFile.close();
```

New IO (Non-blocking IO)



3 колонки

Одновременно могут заправляться 3 автомобиля.
Это очевидно клиенты

3 сотрудника

Общаются с водителями, вставляют пистолеты в баки, берут деньги и т.д. (обслуживают клиентов)
Очевидно, это якобы потоки сервера для обработки клиентов.

3 колонки

1 сотрудник, ожидающий клиентов.

Начинает обслуживание, когда подъезжает новый клиент

- Реагирует на событие «подъехать»!

Может обслуживать нескольких одновременно

- Ждет, пока заливается бензин (ждет события «бензин залит»)
- Берет деньги у другого,
- Спрашивает, какой бензин третьему и т.п.

Это все еще один сотрудник (один Thread)

Рабочий ждет, когда произойдут события (может проверять, произошло ли одно из них):

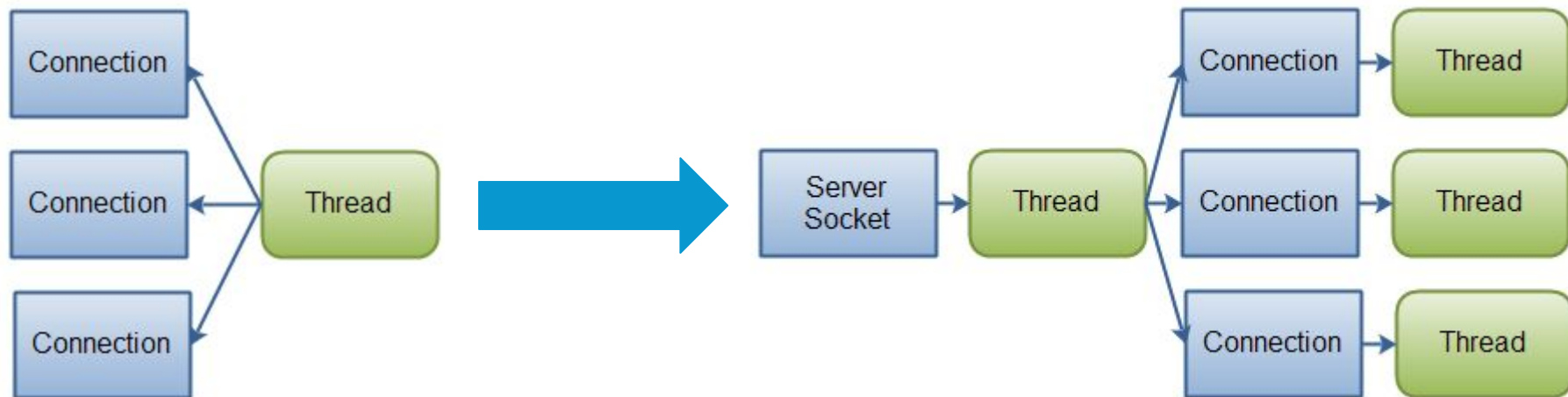
- Подъезжает новый клиент,
- Заканчивает заливаться бензин.

Действует в зависимости от события

При этом он может выполнять параллельные задачи по событиям одного и того же клиента (например, брать деньги, не дожидаясь окончания закачивания бензина)

Как это связано с NIO?

NIO. Мысленный эксперимент даёт плоды



```
java.nio.channels.Selector
```

- Регистрируют каналы, связывая их с событиями, которые могут произойти.
- Могут проверять, произошло ли что-нибудь, и действовать в зависимости от события.
- В мысленном эксперименте с АЗС наш рабочий – селектор!

Константа, связанная с событием
Один бит 1, остальные 0.

```
OP_ACCEPT = 1 << 4 // (10000)
```

Сработали события – возвращается число (ops), где на битах сработанных событий будут 1.

Как проверить, что произошло именно OP_ACCEPT?

`readyOps & OP_ACCEPT`

Будет не равно нулю, если у аргументов есть общие ненулевые биты.

Но у `OP_ACCEPT` только один бит не 0.

Значит будет не равно нулю, если на позиции для `ACCEPT` в `readyOps` будет 1.

А это и означает, что произошло событие `ACCEPT`

```
Selector selector = Selector.open();
ssc.register(selector, SelectionKey.OP_ACCEPT);
while (true) {
    int num = selector.select();
    if (num == 0) continue;
    Set<SelectionKey> keys = selector.selectedKeys();
    for(SelectionKey key : keys) {

        if ((key.readyOps() & SelectionKey.OP_ACCEPT) != 0) {
            Socket s = ss.accept();
            // Для работы делаем channel
            SocketChannel sc = s.getChannel();
            sc.configureBlocking(false);
            // Регистрируем на чтение
            sc.register(selector, SelectionKey.OP_READ);

        } else if ((key.readyOps() & SelectionKey.OP_READ) != 0) {
            // Пришли данные. Узнаем, от кого и дальше как обычно:
            SocketChannel sc = (SocketChannel)key.channel();
            process(sc);
        }

    }
}
```

<CODE>

Рассмотрение примеров из репозитория <https://github.com/DV8FromTheWorld/JDA> и кода интерфейса слушатель, абстрактного класса слушатель.