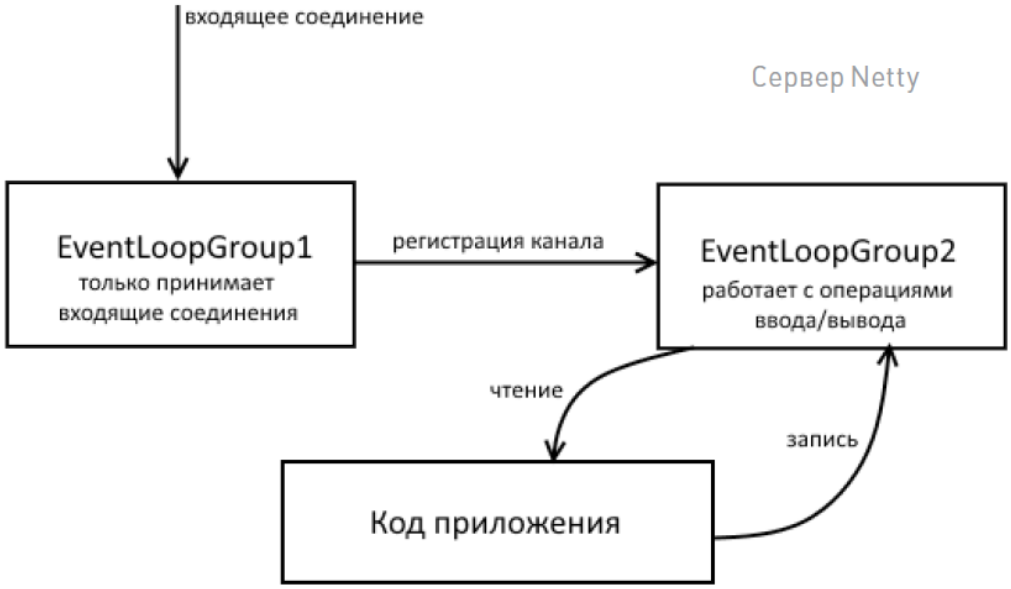
Весь проект состоит из 5 классов: Server, ServerInitializer, ServerHandler, Data, StatusData. Ниже приведены комментарии к каждому классу. Все тесты проводились в браузере Mozilla Firefox 43.0.1. Браузер Google Chrome не подходит, т.к. анализирует передаваемые ссылки. Например, если указать somedomain/redirect?url?.., первый раз переход состоится пройдя через сервер, в последующие разы Chrome сам будет перенаправлять по данной ссылке, не обращаясь к серверу. Так же желательно временно отключить в браузере cookies.

Server :

Все операции ввода-вывода для канала выполняются в цикле событий EventLoop. Несколько циклов событий объединяются в группу EventLoopGroup. ServerBootstrap для сервера создает два экземпляра EventLoopGroup. Один экземпляр только принимает соединения от клиентов, второй обрабатывает остальные события: чтение/запись данных и так далее. Это помогает избежать тайм-аута при подключении к серверу новых клиентов.



Каналы похожи на потоки ввода/вывода (stream) с небольшими отличиями. Они обеспечивают двухстороннюю передачу данных, в один канал можно и писать, и читать. Чтение из канала и запись данных происходит асинхронно.

ServerInitializer:

Когда Netty подключается к серверу или принимает соединение от клиента, он должен знать, как обрабатывать данные, которые принимаются и отсылаются. Инициализацией и конфигурацией обработчиков данных занимается ChannelInitializer. Он добавляет реализации ChannelHandler к ChannelPipeline. ChannelPipeline передает данные на обработку всем обработчикам в порядке, в котором они были добавлены. Каждому последующему обработчику передаются данные, уже обработанные в предыдущем. В данном случае, для каждого канала создаются свои обработчики.

ServerHandler:

ChannelHandlerContext позволяет взаимодействовать с ChannelPipeline и другими обработчиками. За обработку данных отвечают экземпляры ChannelHandler. Обработчики переводят объекты в бинарные данные и наоборот, а также предоставляют метод обработки ошибок, которые возникают в процессе. Таким образом, вся бизнес-логика происходит в обработчике. Данный обработчик прокрывает весь требуемый функционал. При инициализации:
Каждый канал проходит методы в таком порядке:

1. channelActive

- Определяется ip компьютера с которого шел запрос, записывается в счетчик последних 16 соединений (задание 5) и в счетчик ip (задание 3: ip и соответствующее ему количество запросов со временем последнего)
- Инкрементируются значения общих и активных запросов

2. channelRead0

- Метод channelRead0 позволяет обработать полученное сообщение указанного типа и возвращает ответ. В зависимости от входящего URI происходит переадресация, возврат html страницы или текста, запись ответа и подсчет входящего и исходящего потока байт для любого варианта. Метод write записывает данные в буфер, но не отправляет клиенту.
- Для задания №1 – возврат hello world через 10 секунд реализация немного изменена: обе команды write, flash объеденены в одну, реализована задержка и вывод через 10 секунд. запрос считается сразу отработавшим (не ожидая 10 секунд), поэтому становится неактивным. Для того, чтобы проследить 10 секундную активность запроса можно использовать другую реализацию (Thread.sleep(time)).

3. channelReadComplete

- Декрементирует счетчик активных запросов
- Вызывает метод для запроса, который рассчитывает скорость обмена данными
- Метод flush отправляет данные клиенту

Data:

Один из двух классов, хранящих данные. Реализует паттерн Singleton. Хранит на протяжении всего жизненного цикла информацию, указанную в заданиях. Имеет внутренний класс IpCounter, который нужен для подсчета переходов с каждого IP адреса.

StatusData:

Класс, экземпляры которого хранят в себе информацию о соединениях. В классе Data существует List<StatusData>, который хранит информацию о последних 16 соединениях.