Day 09 - Piscine Java

Sockets

Резюме: Сегодня вы реализуете базовый механизм клиент-серверного приложения на Java - Sockets API

Contents

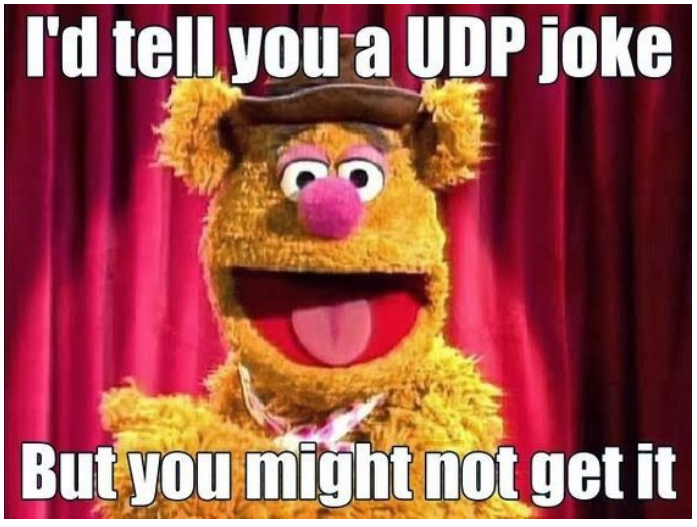
Chapter I
Preamble

Клиент-серверное взаимодействие является основой современных систем. Сервер берет на себя выполнение большого объема бизнес-логики и хранения информации. Это значительно снижает нагрузку с клиентских приложений.

Также разделение логики на серверную и клиентскую части позволяет гибко выстроить общую архитектуру системы, когда реализации сервера и клиента максимально независимы друг от друга.

Общение клиента и сервера происходит посредством большого количества протоколов, описанных в сетевой модели OSI в виде различных уровней:

Уровень	Пример
7. Прикладной	HTTP
6. Представления	ASCII
5. Сеансовый	RPC
4. Транспортный	TCP, UDP
3. Сетевой	IPv4
2. Канальный	Ethernet, DSL
1. Физический	USB, “витая пара”



Chapter II
General Rules

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Сейчас для вас существует только одна версия Java - 1.8. Убедитесь, что на вашем компьютере установлен компилятор и интерпретатор данной версии.
- Не запрещено использовать IDE для написания исходного кода и его отладки.
- Код чаще читается, чем пишется. Внимательно изучите представленный документ с правилами оформления кода. В каждом задании обязательно придерживайтесь общепринятых стандартов Oracle - https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html
- Комментарии в исходном коде вашего решения запрещены. Они мешают восприятию.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. И еще, для любых ваших вопросов существует ответ на Stackoverflow. Научитесь правильно их задавать.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!
- Не откладывайте на завтра то, что можно было сделать вчера ;)

Chapter III
Exercise 00 - Registration

Exercise 00: Registration	
Turn-in directory	ex00
Files to turn-in	Chat-folder

Перед тем, как приступить к созданию полноценного многопользовательского чата, следует реализовать базовую функциональность, а также заложить основную архитектуру системы.

Сейчас вам необходимо создать два приложения - `socket-server` и `socket-client`. Сервер должен поддерживать подключение ровно одного клиента и представлять собой отдельный maven-проект. JAR-файл сервера запускается следующим образом:

```
$ java socket-server --port=8081
```

Клиент также является отдельным проектом:

```
$ java socket-client --server-port=8081
```

В рамках текущего задания необходимо обеспечить функциональность регистрации. Пример работы клиента:

```
Hello from Server!
> signUp
Enter username:
> Marsel
Enter password:
> qwerty007
Successful!
```

После вывода сообщения `Successful!` соединение должно быть сброшено.

Для того, чтобы обеспечить безопасное хранение паролей, следует использовать механизм хеширования с помощью `PasswordEncoder` и `BCryptPasswordEncoder` (см. компоненты `Spring Security`). Бин для данного компонента необходимо описать в классе конфигурации `SocketsApplicationConfig` и использовать в `UsersService`.

Основная логика клиент-серверного взаимодействия, а также использование `UsersService` через Spring Context должны быть реализованы в классе `Server`.

Дополнительные требования:

- Используйте только один DataSource - HikariCP
- Работа репозитория должна быть реализована через JdbcTemplate
- Сервисы, репозитории, утилитные классы должны быть бинами контекста.

Архитектура серверного приложения (клиент-приложение на усмотрение разработчика):

- Char
  - SocketServer
    - src
      - main
        - java
          - edu.school21.sockets
            - server
              - Server
            - models
              - User
            - services
              - UsersService
              - UsersServiceImpl
            - repositories
              - CrudRepository
              - UsersRepository
              - UsersRepositoryImpl
            - app
              - Main
            - config
              - SocketsApplicationConfig
        - resources
          - db.properties
    - pom.xml

Chapter IV
Exercise 01 - Messaging

Exercise 01: Messaging	
Turn-in directory	ex01
Files to turn-in	Chat-folder

После того, как вы реализовали скелет приложения, следует обеспечить многопользовательский обмен сообщениями.

Необходимо модифицировать приложение таким образом, чтобы оно поддерживало следующий жизненный цикл пользователя в чате:

1. Регистрация
2. Вход (в случае, если пользователь не обнаружен - обрывать соединение).
3. Отправка сообщения (каждый пользователь, подключенный к серверу, должен получить данное сообщение)
4. Выход

Пример работы приложения со стороны клиента:

```
Hello from Server!
> signIn
Enter username:
> Marsel
Enter password:
> qwerty007
Start messaging
> Hello!
Marsel: Hello!
NotMarsel: Bye!
> Exit
You have left the chat.
```

Также каждое сообщение должно быть сохранено в базе данных и иметь следующую информацию:

- Отправитель
- Текст сообщения
- Время отправки

Примечание:

- Для полноценного тестирования необходимо запустить несколько jar-файлов клиентского приложения.

Chapter V
Exercise 02 - Rooms

Exercise 02: Rooms	
Turn-in directory	ex02
Files to turn-in	Chat-folder

Для того, чтобы приложение было полноценным, добавим в него концепцию “комнат”. В каждой комнате может быть определенный набор пользователей. Комната содержит набор сообщений от пользователей-участников.

Каждый пользователь может:

1. Создать комнату
2. Выбрать комнату
3. Написать сообщение в комнату
4. Выйти из комнаты

Если пользователь повторно входит в приложение, ему должно быть показано 30 последних сообщений, содержащихся внутри комнаты, в которой он был в последний раз.

Пример работы приложения со стороны клиента:

```
Hello from Server!
   1. signIn
   2. SignUp
   3. Exit
> 1
Enter username:
> Marsel
Enter password:
> qwerty007
   1. Create room
   2. Choose room
   3. Exit
> 2
Rooms:
   1. First Room
   2. SimpleRoom
   3. JavaRoom
   4. Exit
> 3
Java Room ---
JavaMan: Hello!
> Hello!
Marsel: Hello!
> Exit
You have left the chat.
```

Отдельной задачей для вас станет применение формата JSON для обмена сообщениями. Таким образом, каждая команда или сообщение от пользователя должны быть переданы на сервер (и получены от сервера) в виде JSON-строки.

Например, команда по отправке сообщения может выглядеть следующим образом (конкретное содержание сообщений на усмотрение разработчика):

```
{
	“message” : “Hello!”,
	“fromId” : 4,
	“roomId”: 10
}
```

CHECKLIST

https://docs.google.com/document/d/1toPjpq1kHG0Aj6CXlTkAkmUnalBlY2RmzYSqIFogzDw/edit?usp=sharing