

Relatório de Projeto: Sistema Distribuído com Sockets Java

Este relatório detalha a arquitetura, o funcionamento e a implementação de um sistema distribuído de busca de dados, desenvolvido como Prova de Conceito (PoC). O sistema utiliza Sockets Java para a comunicação entre um cliente e múltiplos servidores, demonstrando um fluxo de orquestração de serviços e manipulação de dados em formatos JSON e CSV.

1. Fundamentação Teórica

A arquitetura do projeto se baseia em conceitos fundamentais de redes de computadores e sistemas distribuídos.

- **Arquitetura Cliente-Servidor:** Este é um modelo de computação distribuída que distingue dois tipos de participantes: **clientes**, que solicitam serviços, e **servidores**, que fornecem esses serviços. No contexto deste projeto, temos um cliente que inicia uma busca e três servidores que colaboram para atender a essa solicitação. É um modelo central para a operação da internet e de redes locais¹.
- **Sistemas Distribuídos:** Um sistema distribuído é uma coleção de computadores autônomos interconectados por uma rede e equipados com um software que permite a coordenação de suas atividades e o compartilhamento dos recursos do sistema. O sistema apresentado é um exemplo prático, onde o **Servidor A** atua como um orquestrador, distribuindo a tarefa de busca para os **Servidores B e C**, que operam de forma independente ².
- **Sockets Java:** Um socket é um ponto final (endpoint) de um canal de comunicação bidirecional entre dois programas em uma rede. Em Java, a API de Sockets (`java.net.Socket` e `java.net.ServerSocket`) abstrai os detalhes do protocolo de rede subjacente (geralmente TCP/IP), permitindo que os desenvolvedores enviem e recebam fluxos de dados de forma simples e eficaz. O `ServerSocket` aguarda por conexões de clientes, e ao aceitar uma, cria um objeto `Socket` para a comunicação direta com aquele cliente ³.
- **Formatos de Dados (JSON e CSV):**
 - **JSON (JavaScript Object Notation):** É um formato leve e de fácil leitura para a troca de dados. Sua estrutura baseada em pares chave-valor é ideal para representar objetos de dados complexos, sendo amplamente utilizado em APIs web e arquivos de configuração.
 - **CSV (Comma-Separated Values):** É um formato de texto que representa dados tabulares, onde cada linha corresponde a um registro e as colunas são

separadas por um delimitador (geralmente uma vírgula). É frequentemente usado para importação e exportação de dados em planilhas e bancos de dados.

2. Visão Geral do Projeto

O projeto implementa um sistema onde um cliente solicita uma busca de texto a um servidor principal (Servidor A). Este servidor, por sua vez, atua como um orquestrador, repassando a consulta para outros dois servidores de dados (Servidor B e Servidor C). Cada servidor de dados realiza a busca em seu próprio arquivo JSON local. Os resultados são centralizados no Servidor A, convertidos para o formato CSV e enviados de volta ao cliente, que os salva em um arquivo.

3. Estrutura do Projeto

A organização dos arquivos segue uma estrutura modular, separando as responsabilidades de cada componente do sistema:

src/

└─ com/

└─ java/

└─ client/

| └─ Client.java

└─ server_a/

| └─ Main.java

| └─ utils/

| └─ ServerBConnector.java

| └─ ServerCConnector.java

└─ server_b/

| └─ Main.java

| └─ utils/

```
|   └─ JsonSearchUtil.java
|
|   └─ data/
|
|   └─ dados_servidor_b.json
|
└─ server_c/
   └─ Main.java
   └─ utils/
   └─ ServerCSearchUtil.java
   └─ data/
   └─ dados_servidor_c.json
```

pom.xml

README.md

- **client:** Contém a lógica do cliente que inicia a comunicação.
- **server_a:** O orquestrador que gerencia a comunicação entre o cliente e os outros servidores.
- **server_b, server_c:** Servidores de dados, cada um com sua lógica de busca e seu próprio arquivo de dados JSON.
- **pom.xml:** Arquivo de configuração do Maven para gerenciamento de dependências e compilação do projeto.

4. Pré-requisitos e Compilação

Para compilar e executar o projeto, os seguintes componentes são necessários:

- **Java Development Kit (JDK):** Versão 17 ou superior.
- **Apache Maven:** Versão 3.6 ou superior.

Compilação: Para compilar o projeto, navegue até o diretório raiz (onde se encontra o `pom.xml`) e execute o comando:

Bash

```
mvn clean compile
```

5. Execução do Sistema

Cada componente (cliente e servidores) deve ser executado em um terminal separado para permitir a comunicação em rede.

1. Iniciar Servidor A (Orquestrador)

- **Comando:** `mvn exec:java -Dexec.mainClass="server_a.Main"`
- **Porta padrão:** `3001`

2. Iniciar Servidor B (Dados)

- **Comando:** `mvn exec:java -Dexec.mainClass="server_b.Main"`
- **Porta padrão:** `4002`

3. Iniciar Servidor C (Dados)

- **Comando:** `mvn exec:java -Dexec.mainClass="server_c.Main"`
- **Porta padrão:** `4003`

4. Executar Cliente

Para busca padrão:

Bash

```
mvn exec:java -Dexec.mainClass="client.Client"
```

○

Para busca personalizada (substitua "`seu_termo`" pelo texto desejado):

Bash

```
mvn exec:java -Dexec.mainClass="client.Client" -Dexec.arguments="seu_termo"
```

○

6. Fluxo de Dados e Funcionamento

O fluxo de operações do sistema é sequencial e coordenado, conforme descrito abaixo:

1. **Requisição do Cliente:** O **Cliente** estabelece uma conexão via socket com o **Servidor A** e envia uma string de busca.

2. **Orquestração:** O **Servidor A** recebe a string. Em seguida, ele abre duas conexões de socket separadas, uma para o **Servidor B** e outra para o **Servidor C**, e repassa a mesma string de busca para ambos.
3. **Busca de Dados:**
 - Tanto o **Servidor B** quanto o **Servidor C** recebem a string.
 - Cada um realiza uma busca de **força bruta** (naive search) em seu respectivo arquivo JSON (`dados_servidor_b.json` e `dados_servidor_c.json`), procurando pela string nos campos `"title"` e `"abstract"` de cada objeto.
 - Os servidores retornam uma lista de objetos JSON que contêm o termo buscado.
4. **Agregação e Conversão:**
 - O **Servidor A** aguarda e recebe as respostas (listas de JSONs) dos Servidores B e C.
 - Ele agrega os resultados e os converte para um único texto no formato **CSV**.
 - Neste CSV, o separador de colunas é a vírgula (,) e a quebra de linha é representada pelo marcador customizado `##NL##`.
5. **Resposta e Finalização:**
 - O **Servidor A** envia a string CSV resultante de volta para o **Cliente**.
 - O **Cliente** recebe os dados, substitui o marcador `##NL##` por quebras de linha reais (`\n`) e salva o conteúdo final em um arquivo `.csv` no diretório local.

Referências

¹ Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer Networks* (5th ed.). Prentice Hall.

² Coulouris, G., Dollimore, J., & Kindberg, T. (2011). *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley.

³ Oracle. (2023). *Lesson: All About Sockets*. The Java™ Tutorials.