Modelos de Representação de Dados

Victória Tomé Oliveira

[1]Departamento de Computação – Universidade Federal do Ceará (UFC)
Fortaleza – CE – Brasil

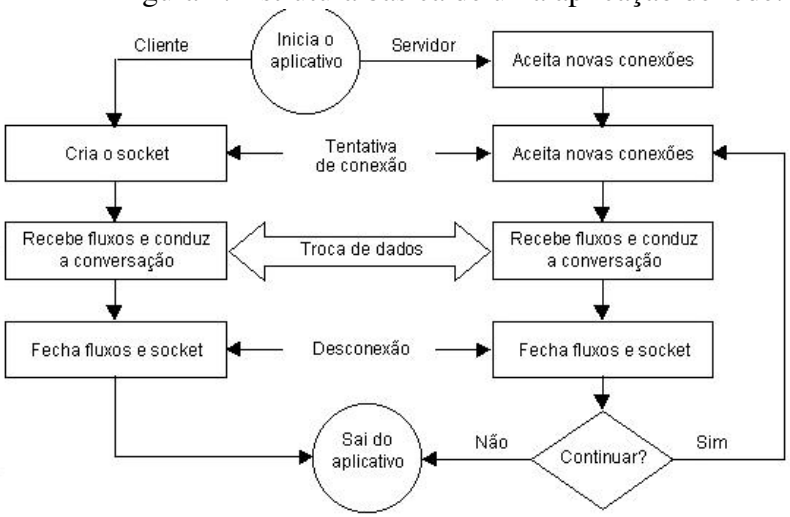
1. Sockets

Os *sockets* são compostos por um conjunto de primitivas do sistema operacional e foram originalmente desenvolvidos para o BSD Unix. Podem ser utilizados nos mais variados sistemas operacionais com recursos de comunicação em rede, sendo suportados pela maioria das linguagens de programação. *Sockets* são suportados em Java desde o JDK 1.0, para sua utilização devemos fazer uso das classes contidas no pacote java.net.

1.1. Estrutura básica de uma aplicação de rede

Uma aplicação que utiliza sockets normalmente é composta por uma parte servidora e diversos clientes. Um cliente solicita determinado serviço ao servidor, o servidor processa a solicitação e devolve a informação ao cliente. Muitos serviços podem ser disponibilizados numa mesma máquina, sendo então diferenciados não só pelo endereço IP, mas também por um número de porta. Porém, o mais comum é termos uma máquina dedicada oferecendo apenas um ou dois serviços, evitando assim a concorrência.

Figura 1. Estrutura básica de uma aplicação de rede.



1.1.1. Servidor

O serviço do computador que funciona como base deve, primeiro, abrir uma porta e ficar ouvindo até alguém tentar se conectar. Se o objeto for realmente criado, a porta é aberta. Após abrir a porta, precisamos esperar por um cliente fazer a conexão, assim que um cliente se conectar, o programa continuará. Por fim, basta ler todas as informações que o cliente nos enviar.

1.1.2. Cliente

O cliente abre uma conexão com o servidor, no IP 192.168.0.11 e porta 9090. O servidor responde perguntando a operação e os parâmetros. O cliente envia os parâmetros, quando recebe a resposta do servidor a conexão é encerrada. Usa-se os conceito de java.io para leitura do teclado e envio de mensagens para o servidor. Para a classe *Scanner*, tanto faz de onde que se lê ou escreve os dados, o importante é que esse *stream* seja um *InputStream*/*OutputStream*. É o poder das interfaces, do polimorfismo, aparecendo novamente.

2. Modelos de Representação de Dados

O processo de serialização de objetos é bastante utilizado em sistemas distribuídos (coleção de computadores independentes conectados por uma rede e equipados com um sistema de software distribuído) e na persistência de dados (manter dados além da duração da execução do programa). Com a transformação do objeto em bytes é possível enviar o objeto por uma rede, ou salvá-lo em um arquivo ou em um banco de dados.

Serialização é a técnica que permite transformar o estado de um objeto em uma sequência bytes. Depois que um objeto for serializado ele pode ser gravado (ou persistido) em um arquivo de dados e recuperado do arquivo e desserializado para recriar o objeto na memória.

Não existe um formato de serialização único - o melhor formato para os trabalhos depende de coisas como o tipo e quantidade de dados que estão sendo serializados e o software que os lerá. Será dada uma atenção especial aos formatos de serialização JSON e YAML, pois serão utilizados neste trabalho.

2.1. *JavaScript Object Notation* - JSON

JSON (*JavaScript Object Notation*) é um formato leve de intercâmbio de dados. É fácil para humanos ler e escrever. É fácil para as máquinas analisar e gerar. Ele é baseado em um subconjunto do *JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999*. O JSON é um formato de texto que é completamente independente da linguagem, mas usa convenções familiares aos programadores da família C de linguagens, incluindo C, C++, C#, Java, JavaScript, Perl, Python e muitos outros. Essas propriedades tornam o JSON uma linguagem ideal de intercâmbio de dados [1].

O JSON é construído em duas estruturas:

- Uma coleção de pares nome/valor. Em vários idiomas, isso é realizado como um objeto, registro, estrutura, dicionário, tabela de hash, lista de chaves ou matriz associativa.
- Uma lista ordenada de valores. Na maioria dos idiomas, isso é realizado como uma matriz , vetor, lista ou sequência.

Essas são estruturas de dados universais. Praticamente todas as linguagens de programação modernas as suportam de uma forma ou de outra. Faz sentido que um formato de dados intercambiável com linguagens de programação também seja baseado nessas estruturas [1].

2.2. YAML

YAML Ain't Markup Language (YAML) é uma linguagem de serialização de dados projetada para ser amigável ao ser humano e funcionar bem com linguagens de programação modernas

para tarefas diárias comuns. O YAML foi projetado desde o início para ser útil e amigável para as pessoas que trabalham com dados. Ele usa Unicodecaracteres imprimíveis, alguns dos quais fornecem informações estruturais e o restante contém os próprios dados. O YAML alcança uma limpeza exclusiva, minimizando a quantidade de caracteres estruturais e permitindo que os dados se mostrem de maneira natural e significativa [2].

O YAML é construído em três estruturas:

- Mapeamentos (hashes/dicionários)
- Sequências (matrizes/listas)
- Escalares (strings/números).

Existem centenas de idiomas diferentes para programação, mas apenas um punhado de idiomas para armazenar e transferir dados. Embora seu potencial seja praticamente ilimitado, o YAML foi criado especificamente para funcionar bem em casos de uso comuns, como: arquivos de configuração, arquivos de log, sistema de mensagens entre processos, compartilhamento de dados em vários idiomas, persistência de objetos e depuração de estruturas de dados complexas. Quando os dados são fáceis de visualizar e entender, a programação se torna uma tarefa mais simples [2].

Enquanto a maioria das linguagens de programação pode usar o YAML para serialização de dados, o YAML se destaca ao trabalhar com essas linguagens que são fundamentalmente construídas em torno das três primitivas básicas. Isso inclui linguagens ágeis, como Perl, Python, PHP, Ruby e Javascript [2].

3. Metodologia

Para a realização desta atividade foi criado um código em JAVA de Cliente/Servidor Socket. O servidor espera um cliente fazer a conexão, assim que o cliente conecta-se, o cliente solicita ao usuário uma equação. Faz o tratamento desta equação dividido a equação em dois vetores, um de operandos e outro de operadores. Logo após, é criado uma estrutura em JSON (codigo 1) e YAML (codigo 2), e enviado essa estrutura para o servidor.

O servidor ler a estrutua enviado pelo cliente, trata essa estrutura e envia a resposta da equação. Após a resposta do servidor a conexão é encerrada.

4. GitHub

`https://github.com/victoriatome/SistemasDistribuidos_Calculadora`

5. Considerações Finais

Ao termino deste trabalho, conclui-se que não existe um formato de serialização perfeito, todos tem vantagens e desvantagens. Sendo assim, o formato de serialização mais adequado para um determinado cenário depende de várias coisas, como o tipo e quantidade de dados que estão sendo serializados, a quantidade de acesso aos dados, software que os lerá etc.

Referências

[1] D. Crockford, "Introducing json." https://www.json.org/json-en.html. [Online; acesso em 01-Maio-2020].

[2] Y. Org, "Yaml: Yaml ain't markup language." https://yaml.org/. [Online; acesso em 01-Maio-2020].