Projeto Sincronismo de Relógios

Projeto de Sincronismo de Relógios usando o algoritmo de Berkley para a disciplina de Sistemas Distribuídos.

Victor Domingos Santiago, TIA 41357205.

Projeto

Implementação

O projeto pode ser dividido em três partes principais, divididas pelos pacotes: **Controller**, **Model**, e **Utils**.

Controller

Estas são as classes principais do projeto.

Aqui estão as classes **Master** e **Slave**, que como o nome já diz, cuidam das lógicas do nó mestre e dos seus nós escravos.

A comunicação entre eles é feita através de mensagens utilizando o protocolo UDP. Um envia *Strings* para o outro com comandos e respostas. Por exemplo, o mestre poderia mandar a String "SEND_TIME" para o escravo, e assim o escravo saberia que ele tem que mandar a sua hora atual de volta.

Nota: Considerando que este programa seria rodado apenas localmente, e não em um servidor, foi optado por *mockar* as horas enviadas pelo escravo. Ou seja, é gerado uma hora aleatória: Um número de segundos entre -60 e +60 é escolhido, e ele é adicionado no tempo atual. A tolerância de Berkley está estabelecida em 45s.

Model

Aqui está a classe **Machine**, que nada mais faz do quê servir de *DTO* (Data Transfer Object) para os dados usados pelo **Master** (e.g.: IP, porta) sobre os seus **Slaves**.

Já a classe **Berkley** é onde está contida toda a lógica de cálculo de sincronismo de relógios.

Utils

Coleção de métodos e variáveis úteis para serem usadas pelo projeto.

Por exemplo, a classe **DateUtils** tem todos os métodos relacionados a data. Como gerar uma data aleatória em um intervalo de tempo, transformar de **String** para **Date** e vice-versa, etc.

Já a **SyncUtils** contém métodos para ajudar a trocar mensagens entre **Master** e **Slave**.

Já a **Command** e **Config** contém apenas variáveis utilizadas pelo projeto. Respectivamente, os comandos passados entre uma classe e outra para executar ações, e variáveis de configuração como porta da **Master**, intervalo entre uma sincronia e outra, etc.

Observação: Caso haja alguma dúvida sobre o funcionamento de algum método específico, tudo está documentado dentro do código-fonte usando o padrão do Javadoc.

Como rodar

1. Importe o projeto para a sua **IDE** de preferência (Testado no **Netbeans** e **IntelliJ**);

2. Execute a classe **Master**;
3. Execute a classe **Slave**;

As classes irão se comunicar entre si, e todas as mensagens serão impressas para que seja visto o quê está acontecendo. O horário que está sendo enviado, e o horário ajustado.

Para realizar testes com mais de um *Slave*, vá até a classe **Master**, e digite o número de slaves que ela terá:

```
public static void main(String args[]) throws Exception {
    Master master = new Master(3); //Insira aqui o número de slaves.
    master.execute();
}
```

Depois, basta executar a classe *Slave* o mesmo número de vezes, mas com a condição de que todos os *Slaves* tenham portas diferentes, e suas portas sejam diferentes da porta do **Master** (fixada em **9876**, mas pode ser alterada no arquivo Config).

Para alterar a porta antes de executar a classe, basta alterar o parâmetro no construtor:

```
public static void main(String args[]) throws Exception {
    Slave slave = new Slave(9800); //Altere a porta aqui.
    slave.initialize();
    slave.execute();
}
```

FAQ

Por quê o número de escravos é fixo?

R.: Uma das hipóteses especificadas no enunciado é a de quê "o conjunto de nós escravos (vivos ou falhos) não muda".

O quê está incluso na entrega?

R.: README em *Markdown* e *PDF*, screenshots da execução do projeto e código-fonte. O arquivo **"Projeto"** pedido não está incluído. Ao invés disso, decidir documentar tudo dentro deste *README*.