První zápočtová úloha Java RMI

Úlohu je třeba odevzdat E-mailem cvičícímu. Termíny pro odevzdání úlohy jsou uvedené na webové stránce předmětu http://d3s.mff.cuni.cz/teaching/middleware/.

Cílem zadání je získat základní zkušenost s distribuovanými objekty a přestavu o tom, jak způsob práce s nimi může ovlivnit výkon aplikace.

Požadované znalosti

Jako příklad úlohy používá zadání jednoduché změření vzdálenosti mezi dvěma uzly grafu objektů, čímž napodobuje práci s dynamickými datovými strukturami. Pro pochopení úlohy není potřeba žádná zvláštní znalost.

Jako technologii vzdáleného volání používá zadání implementaci Java RMI. Pro implementaci úlohy je potřeba znalost následujících věcí:

- Způsob definice vzdáleně přístupného interface
  (interface `java.rmi.Remote`, výjimka `java.rmi.RemoteException`)
- Způsob implementace vzdáleně přístupného objektu
  (třída `java.rmi.server.UnicastRemoteObject`,
  dědění z této třídy a metoda `exportObject()`)
- Způsob spojení klienta a serveru pomocí registry
  (třída `java.rmi.Naming`, aplikace `rmiregistry`)
- Překlad implementace (aplikace `rmic`)

Jednoduchý příklad vzdáleného volání pro Java RMI je k dispoizici.

Zadání úlohy

http://d3s.mff.cuni.cz/teaching/middleware/files/as1.zip

Základem zadání je jednoduché změření vzdálenosti mezi dvěma uzly grafu objektů. Objekty implementují rozhraní `Node`:

```
public interface Node
{
    Set<Node> getNeighbors ();
    void addNeighbor (Node oNeighbor);
}
```

Metoda `addNeighbor()` přidá objekt do množiny sousedů volaného objektu a používá se při generování grafu. Metoda `getNeighbors()` vrátí množinu všech sousedů volaného objektu a používá se při měření vzdálenosti. Samotné měření vzdálenosti nabízí rozhraní `Searcher`:

```
public interface Searcher
{
    public static final int DISTANCE_INFINITE = -1;
    public int getDistance (Node oFrom, Node oTo);
}
```

Vaše úkoly jsou následující:

1. Seznamte se s poskytnutou implementací úlohy, která pracuje lokálně. Změřte rychlost provádění na několika typech náhodně generovaných řídkých a hustých grafů.

2. Vytvořte server, který bude poskytovat vzdáleně přístupný objekt s rozhraním `Searcher`. Upravte poskytnutou implementaci úlohy tak, aby vedle dosavadních funkcí umožnila také nalézt vzdálenost pomocí serverového rozhraní `Searcher`, kterému se jako graf předají lokální objekty s rozhraním `Node`.

   Změřte a porovnejte rychlost implementovaných variant. Jak přistupuje `Searcher` na serveru k lokálním objektům `Node`?

3. Upravte server tak, aby poskytoval vzdáleně přístupné objekty s rozhraním `Node`, které budou vytvářeny na žádost klienta. Upravte poskytnutou implementaci úlohy tak, aby vedle dosavadních funkcí umožnila také nalézt vzdálenost pomocí lokálního rozhraní `Searcher`, kterému se jako graf předají serverové objekty s rozhraním `Node`.

   Změřte a porovnejte rychlost implementovaných variant. Nezapomeňte, že je třeba provádět měření na stejných grafech (ať už vzdálených nebo lokálních) aby bylo možné výsledky porovnat. Jak přistupuje lokální `Searcher` k objektům `Node` na serveru?

4. Doplňte ještě nalezení vzdálenosti pomocí serverového rozhraní `Searcher`, kterému se (z klienta) jako graf předají serverové objekty s rozhraním `Node`.

   Změřte a porovnejte rychlost implementovaných variant. Jak přistupuje `Searcher` na serveru k objektům `Node` na serveru?

5. Porovnejte rychlosti všech variant v případě, kdy klient a server běží na stejném počítači, s případem, kdy klient a server běží na různých počítačích spojených sítí.

6. Výsledky předchozích měření pomáhají rozpoznat situace, kdy je rychlejší předávat dynamické struktury hodnotou a kdy naopak odkazem. Základní úloha však tyto situace demonstruce na extrémech, kdy se vše předává buď hodnotou nebo odkazem, přitom často je užitečná také kombinace obou přístupů.

   Využijte poskytnutou implementaci metody `getTransitiveNeighbors()` rozhraní `Node`, která místo bezprostředních sousedů uzlu vrací všechny uzly do vzdálenosti dané parametrem metody. Využijte také rozšířenou implementaci rozhraní `Searcher`, která v každém kroku metody `getDistance()` nezjišťuje pouze bezprostřední sousedy uzlu, ale množinu sousedů uzlu do vzdálenosti dané parametrem metody.

   Pomocí měření zjistěte, při jaké hodnotě zmíněného parametru metody `getNeighbors()` se pro náhodně generované řídké a husté grafy přiblíží rychlost nalezení vzdáleností situacím v bodech 2 a 3. Testujte v prostředí s větší latencí sítě (například kolej–lab).

First assignment — Java RMI

Send the finished task by e-mail to your teaching assistant. Deadlines for the submission are at the subject web page http://d3s.mff.cuni.cz/teaching/middleware/.

The purpose of the task is to get basic understanding of distributed objects and an idea how accessing them and working with them can affect performance of the application.

Prerequisities

The task is based on computing distance between two nodes in a graph, simulating work with dynamic data structures. No other special knowledge is required.

Java RMI is the technology used for the remote procedure invocation. For implementing the assignment, understanding of the following is required:

- Definition of a remotely accessible interface
  (interface `java.rmi.Remote`, exception `java.rmi.RemoteException`)
- Implementation of remotely accessible object
  (class `java.rmi.server.UnicastRemoteObject`,
  inheriting from this class, method `exportObject()`)
- Connecting client and server using RMI registry
  (class `java.rmi.Naming`, `rmiregistry` application)
- Compilation
  (`rmic` application)

A simple example of a remote function call for Java RMI is available.

Assignment details

`http://d3s.mff.cuni.cz/teaching/middleware/files/as1.zip`

The core of the assignment is a simple distance measuring between two node objects in a graph. Objects implement the interface `Node`:

```
public interface Node
{
    Set<Node> getNeighbors ();
    void addNeighbor (Node oNeighbor);
}
```

The `addNeighbor()` method adds the object to a set of neighbors of the called object and is used when generating a graph. The `getNeighbors()` method return a set of all neighbors of an object and is used for distance computation. The actual distance computation is done through a `Searcher` interface:

```
public interface Searcher
{
    public static final int DISTANCE_INFINITE = -1;
    public int getDistance (Node oFrom, Node oTo);
}
```

Your tasks are:

1. Explore the provided implementation of the task that works locally only.

   Measure the speed of execution on several types of randomly generated graphs, both sparse and dense ones.

2. Create a server that would provide a remotely accessible object with `Searcher` interface. Update the provided implementation to allow search through the remote interface along the local one. The server `Searcher` interface would accept local objects with the `Node` interface.

   Measure and compare the speed of the implemented variants. How does the server `Searcher` accesses the local `Node` objects?

3. Update the server to provide remotely accessible objects with the `Node` interface that would be created upon client request. Update the provided implementation to allow computation of the graph distance using local `Searcher` working with the server `Node` objects along existing functionality.

   Measure and compare the speed again. Do not forget that it is necessary to measure the same graphs (local and remote ones) to get a relevant comparison. How does the local `Searcher` access the server `Node` objects?

4. Add another variant: compute the distance using the server `Searcher` interface to which you pass (from the client) the graph as a server `Node` objects.

   Measure and compare the speed again. How does the server `Searcher` access the server `Node` objects?

5. Compare the speed of all variants when both client and server are running on the same machine vs. situation when client and server and on different physical computers connected by a network.

6. Results of the previous measurements help to distinguish when it is faster to pass dynamic data structures by value and when it is faster to pass them by reference. The assignment demonstrates this on extreme all-or-nothing cases when either everything is passed by value or everything is passed by reference. But often a combination of both is the right choice.

   Use the provided implementation of the `getTransitiveNeighbors()` method of the `Node` interface that returns all neighbours that are close enough – i.e. that are accessible by at most n edges where n is an argument to that method. Also use the extended implementation of the `Searcher` interface that in each step of `getDistance()` does not retrieve only direct neighbors but a whole set of neighbors that are at most as far from the node as specified by the argument.

   Find out for which values of the argument to the `getNeighbors()` method are the measured times comparable to those obtained from settings described in 2 and 3. Use randomly generated sparse and dense graphs. Test in a higher latency environment, e.g. between networks in the campus and in the local lab.