Middleware Labs: Java RMI

Petr Tůma Vojtěch Horký Antonín Steinhauser
Vladimír Matěna

March 6, 2019

General Information

- Labs
  - Every other week
  - 5 labs total, 5 middleware technologies
  - Wednesday, 12.20 (SU2)
  - See the calendar on the web
- Web
  - `http://d3s.mff.cuni.cz/teaching/middleware/`
- Mailing list
  - `nswi080@d3s.mff.cuni.cz`
  - `https://d3s.mff.cuni.cz/mailman/listinfo/nswi080`

Requirements for Getting the Credits

- Details on the web page
- At least 7 points for the semester
- Standard tasks
  - "Hello World" in various technologies ;-)
- Alternative tasks
  - Less boring than the "Hello World" stuff ;-)

Standard Tasks

- 2 points for solving the task correctly and in time
- 1 point for incorrect submission in time
  - Fixed submission presented at next labs
- 1 point for missed deadline (max before next labs)
  - Must show at the labs
- 5 tasks together, each for 2 points

Alternative Tasks

- Only after previous consultation
  - Contact Petr Tůma for details (task, points, deadlines)
- Preferred way to get credit
- 2 to 6 points (depending on the difficulty)
- Deadline: end of summer examination period
  - Later only in special cases

Topics

- Benchmark (2), set of benchmarks (4 – 6)
- Your very own topic (?)

Be Original – No Cheating!

- This is not a mandatory subject
- Go cheat somewhere else
- You do not want to learn how to modify someone else's code
- You want to learn middleware technologies
- If the tasks seems boring to you ...
- ... settle for an alternative.

Submission (generic notes)

- Working implementation
- Answer all the questions from the assignment
- By e-mail
  - Deadline is on the web
- Make sure it works in the lab downstairs

Submission (generic notes) II

- Documentation
  - README with key decisions overview
  - Notes on compiling/running
- The submission shall be easy to start
  - No need for Maven or Ant script
  - No need for packages etc.
  - Updated versions of the `run-*` scripts
  - Do not send stubs, compiled files or Eclipse `.projects`

Java RMI

Department of
Distributed and
Dependable
Systems

Task: Distance Between Graph Nodes

```
public interface Searcher {
  public int getDistance(Node from, Node to);
}

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

Node[] graph;
```

Local Implementation

- Interfaces `Node` and `Searcher`
  - Classes `NodeImpl` and `NodeSearcher`
- Java `main()` is in the `Main` class
  - Measures the speed on a random graph

Task

- Extend the provided implementation to search the graph remotely
  - local / remote nodes
  - local / remote searcher
- Compare speed
  - on sparse / dense graphs
  - on a single computer / over the network
  - different values of a parameter of the algorithm

Task (cont.)

- Read the task description on the web
- Read these slides
  - Avoid common problems
- Report problems
  - Well before submission
  - Use solely the mailing list for questions
  - The same problem might affect more people

Remote Searcher

- Extend the `Searcher` interface (see `Example`)
  - Interface `java.rmi.Remote`
  - Exception of type `java.rmi.RemoteException`
- Remotely accessible object (see `ExampleImpl`)
  - Must be exported – 2 ways
  - Derive from `java.rmi.server.UnicastRemoteObject`
    - Export ensured by parent constructor
  - Call `UnicastRemoteObject.exportObject(obj)` manually
    - Does not handle semantics of `hashCode()`, `equals()`, `toString()` — not a problem with `Searcher` (just one instance)

Remote Searcher (cont.)

- Executable server (see `ExampleServer`)
  - Create instance (and export) of the remote object
  - Register with `java.rmi.Naming.[re]bind()`
- Extend `Main` with RMI (see `ExampleClient`)
  - Get reference to a remote `Searcher`
    - `java.rmi.Naming.lookup(path)`
  - Add call to remote `Searcher.getDistance()` with local objects `NodeImpl` in `searchBenchmark()` method
    - How does the server access the neighbors of the passed nodes?

Remote Node Objects

- Extend interface `Node` with RMI (like `Searcher`)
- Class inherits from `UnicastRemoteObject` and implements `Node`
  - To allow for `hashCode()`, `equals()`, `toString()`
  - Copy/paste + edit is enough
    - We want that local `Nodes` from previous task still behaved locally

Remote Node Objects (cont.)

- How to create and return instances for client requests?
- Implement `NodeFactory` with method `createNode()`
  - Similar to remote `Searcher` – interface with RMI, implementing class, create and call `Naming.bind()` inside the existing server
  - Do not create a standalone server, we want just one for the 4th variant
- Client gets the reference using `lookup()` and also creates the remote `Node` objects together with the local graph
- How does the local `Searcher` access the remote `Nodes`?
- What exactly does the `NodeFactory` return to the client?

Remote Searcher on Remote Nodes

- Everything is ready, just add this variant to `searchBenchmark()` and compare the speed
- How does the `Searcher` on server access the `Node` objects on (the same) server?

Impact Of the Network

- So far, client and server were running on the same machine
  - Overhead of RMI communication, but no network latency
- Run on more machines
  - Server on the machine next to you, client on yours
  - Change paths in `[re]bind()` and `lookup()`
    - Remote machine name instead of `localhost`
    - Modify to use `args[0]`
  - Run `rmiregistry` and `Server` in SSH session on the remote machine
  - Run the client locally
  - Beware of `CLASSPATH`

Passing by Value vs. Passing by Reference

- Previous tasks solve "extreme" cases
- How about combining both approaches?
- Idea: "batch" transfer of bigger parts of the graph
- `getTransitiveNeighbors(int distance)`
  - Returns all neighbors up to some distance
- Use the `getDistanceTransitive` method of the `Searcher` interface
  - In each step, requests neighbors up to the specified distance
- Try different values for the distance parameter
  - Compare measured times with previous variants

Implementation Notes

Extend single project, do not create 4 separate ones.

- Interface hides different implementations
  - Even `Remote` interface can be used locally
    - Just catch exceptions that would never occur
  - E.g. remote graph is just another array `Node[]`
    - Easy to have the same (logically) local and remote one
    - Similarly with `Searcher`
- Measure everything in one run to ease comparison
  - Just add measuring and a column to results in `searchBenchmark()`

Building – the `make` Script

- `javac` (with Eclipse unnecessary)
- `rmic`
  - Deprecated in Java 8
  - Creates stubs for remote objects
  - Parameters are class names implementing the remote objects
  - `keep` does not remove the generated stub sources
  - Sometimes unnecessary
    - Client can access the classes (yes in our task)
    - Classes inherit from UnicastRemoteObject
  - Subtle differences for generated stubs and proxies
    - `equals()`

Launching

- Use launcher scripts from the Hello World example
  - Important parameters
  - Simple Run as. . . /Application in Eclipse is not enough!
    - But can be set-up to work as well

Launching (cont.)

- `rmiregistry` application – run in background
  - Port in use? – use different port number (> 1024)
    - Edit path in calls to `[re]bind()` and `lookup()`
    - `localhost` becomes `localhost:1234`
  - For simplicity – it has the same `CLASSPATH`
    - We want to avoid setting permissions for codebase etc.
- Starting the server – see `run-server` script
- Starting the client – see `run-client` script

Submission

- Working implementation
- Documentation
  - Answer all the questions from the assignment
  - Describe measurement results
- By e-mail (deadline is on the web)
- Make sure it works in the lab downstairs
- The submission shall be easy to start
  - Use the provided implementation
  - No need for Maven or Ant script
  - Do not add packages etc.
  - Updated versions of the run-server scripts