DS Tutorial - Spring 2019

> Wrote by Bill Junru ZHONG
> Last modified on April 9, 2019

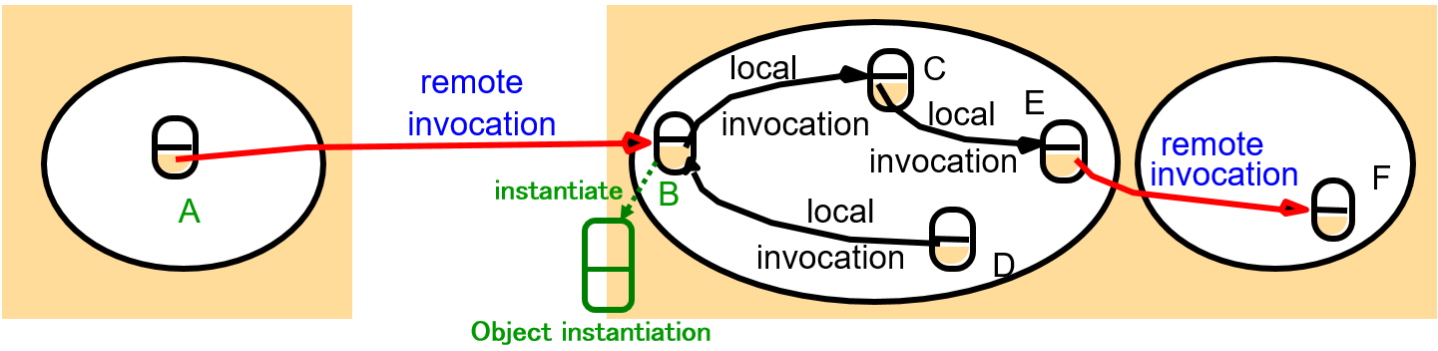
Java RMI Review

Reading

- Slide: Topic 5: Remote Object Invocation
- Book: Section 5.5: Case Study - Java RMI

Tips

As for me, I treat the RPC (RMI) methods (protocols) as a *request-reply* models, that I can always keep my mind clean and put it into a client-server model (sort of).



As the name of this method writes, Java RMI is a technology that can invocate methods on remote computers. Like the example `timeServer` we have, the client machine asks the server for a system time by calling a remote function.

RMI is fairly easy since you just need to feed your own function implementations to the template code.

Python Simple XML RPC Server

Reading

- Python XML RPC
- Sample code

Basic XML RPC

Compare with Java RMI, Python XML RPC is even more simple. The basic code (grabbed from the official example) is shown below.

- Server

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler) as server:
    server.register_introspection_functions()

    # Register pow() function; this will use the value of
    # pow.__name__ as the name, which is just 'pow'.
    server.register_function(pow)

    # Register a function under a different name
    def adder_function(x, y):
        return x + y
    server.register_function(adder_function, 'add')

    # Register an instance; all the methods of the instance are
    # published as XML-RPC methods (in this case, just 'mul').
    class MyFuncs:
        def mul(self, x, y):
            return x * y

    server.register_instance(MyFuncs())

    # Run the server's main loop
    server.serve_forever()
```

- Client

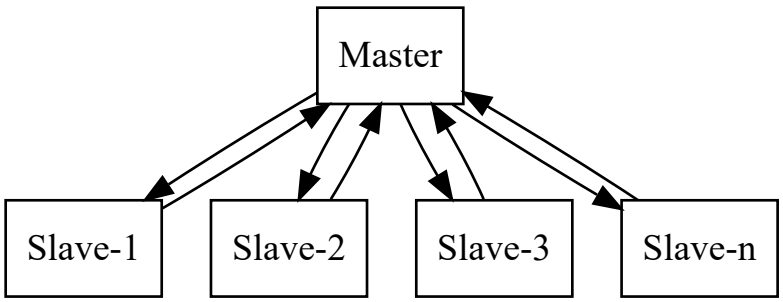
```
import xmlrpc.client

s = xmlrpc.client.ServerProxy('http://localhost:8000')
print(s.pow(2,3))  # Returns 2**3 = 8
print(s.add(2,3))  # Returns 5
print(s.mul(5,2))  # Returns 5*2 = 10

# Print list of available methods
print(s.system.listMethods())
```

Sample Structure of the Project

For your reference, I designed a sample structure for the project. You can take it or design one by yourself.



In this structure, the **server** will have to transfer the message correctly from the source to the destination, and also handle broadcasting messages. The information transfers with some RPC protocol.

Implementation Details (not finished)

- Your message class should include sender and receiver information, and a flag for broadcasting.
- You can implement a `send()` method on server class, then the client can call this method to send messages.
- Obviously, you should maintain a list of current users on your server.
- It is easy to see you need to implement **multi-threading** on your server class.