A CASE STUDY ON JAVA RMI

By:
Ankit B.K (THA076BCT006)
Ayush Batala (THA076BCT010)
Mishan Thapa Kshetri (THA076BCT019)
Nishant Uprety (THA076BCT023)

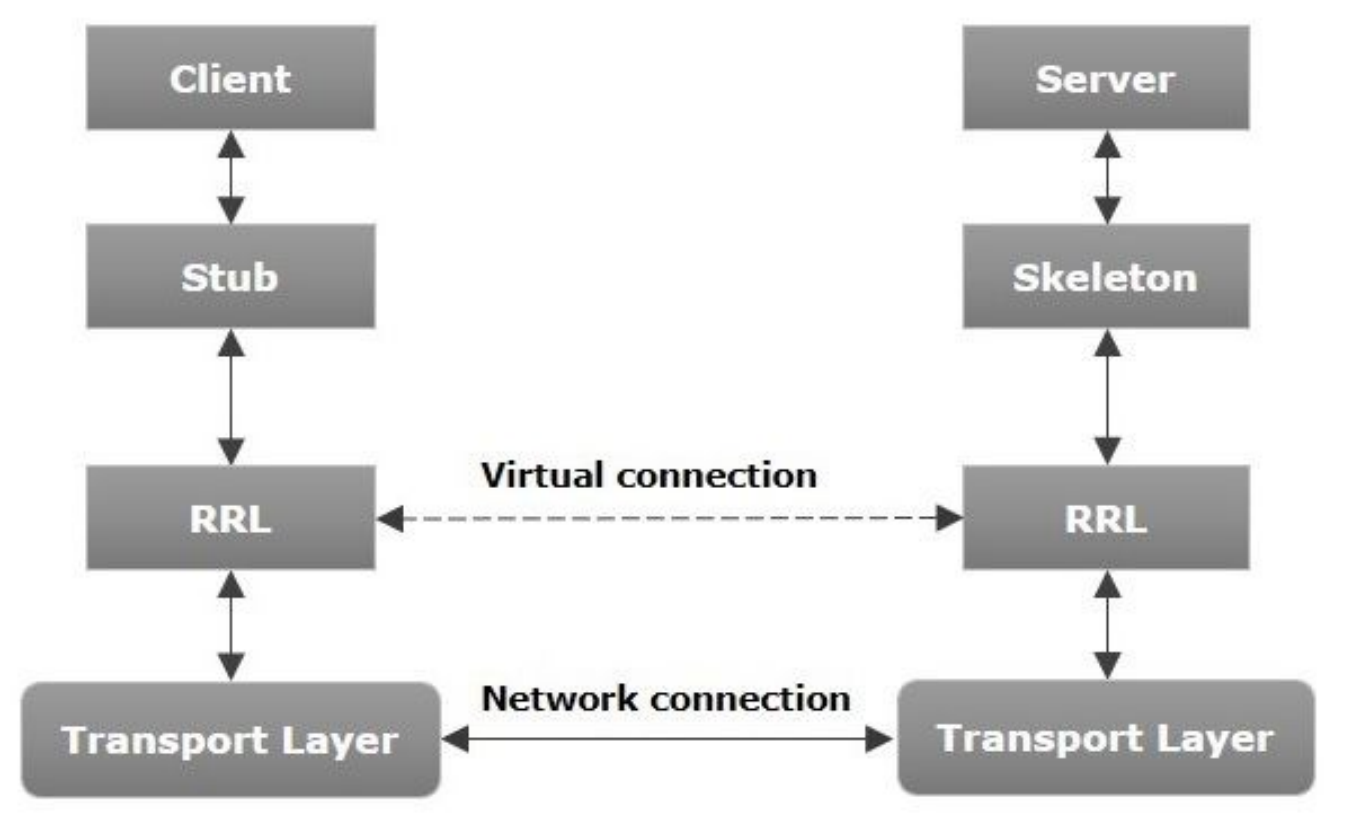
Introduction

- Distributed systems require communication between entities across different machines.
- Remote Procedure Call (RPC) simplifies remote communication, but lacks seamless support for distributed object systems.
- Distributed object systems demand Remote Method Invocation for object-oriented communication. RMI matches object invocation semantics, allowing programmers to work with distant objects as if they were local.
- Java RMI leverages the Java Virtual Machine (JVM) environment, embracing the Java object model. It facilitates seamless remote method invocation, providing an effective solution for distributed systems

Java Object Model

- Java's platform-independent nature ensures consistent program behavior across diverse hardware and operating system platforms.
- Java distinguishes between interfaces (method definitions without implementation) and classes (method implementations along with data fields).
- Interface-Driven Access: Access to object data is limited to invoking public methods of interfaces. This enforces a clean separation between client and server, making it essential in distributed systems.
- Significance in Distribution: Interfaces can reside on the client side, while the actual object implementation is located on the server side. This decoupling enables efficient and secure remote communication.

Java RMI Architecture



Layers of Java RMI System

- Stub/Skeleton Layer:
  - *Stub: Client-side proxy object representing a remote object. It facilitates method invocation for clients.*
  - *Skeleton: Server-side component receiving method calls from client stubs and forwarding them to the remote object.*
- Remote Reference Layer:
  - *Manages references to remote objects.*
  - *References act as handles enabling client interaction with remote objects.*
- Transport Layer:
  - *Handles network communication between client and server.*
  - *Encompasses protocols and mechanisms for data transmission.*

Benefits of Layered Architecture

- Clear separation of concerns and responsibilities.
- Facilitates code organization and maintenance.
- Promotes modularity and scalability.
- Allows for easier troubleshooting and optimization.

Distributed Application using RMI

- Define Remote Interface:
  - *Create a Java interface with methods to be invoked remotely on the server-side.*
- Implement Remote Object:
  - *Implement the remote interface in a server-side class.*
  - *This class represents the actual object with remotely accessible methods.*
- Start RMI Registry:
  - *Launch the RMI registry on the server machine.*
  - *The registry maintains a list of available remote objects and references.*
- Register Remote Object:
  - *Instantiate the server-side remote object class.*
  - *Use the RMI registry's bind method to register the remote object with a specific name.*

Distributed Application using RMI(Cont....)

- Server Code:
  - *The server-side application, hosting the remote object, must be running to handle client requests.*
- Client Code:
  - *In the client application, use the RMI registry's lookup method to find the remote object's stub.*
  - *Utilize the stub reference to invoke remote methods on the server-side object as if it were local.*
- Compile and Run:
  - *Compile both client and server code.*
  - *Run the server application on the server machine.*
  - *Run the client application on the client machine to interact with the remote object.*

Benefits of RMI for Distributed Apps:

- Simplified remote method invocation.
- Encapsulation of network communication complexities.
- Enables object-oriented communication in distributed systems.

Real-World Applications of RMI

- Online Games:
  - *RMI enables real-time interactions between game clients and a central server, facilitating multiplayer gaming experiences.*
- File Sharing Applications:
  - *RMI can be employed to create efficient and secure file sharing systems where clients access files hosted on remote servers.*
- Scientific Computing Applications:
  - *RMI aids in distributing complex scientific computations across multiple nodes, enhancing processing power and speed.*
- Business Applications:
  - *RMI supports business software that demands seamless communication between various components, such as inventory management and order processing.*

Real World Example: JavaSpaces in Jini

- Jini Technology:
  - *Jini, a Java-based technology, facilitates dynamic and distributed system creation.*
  - *JavaSpaces, a core component of Jini, offers a distributed shared memory service.*
- JavaSpaces and RMI:
  - *JavaSpaces: Utilizes Java RMI to enable object sharing among networked clients and servers.*
  - *Provides a distributed and persistent shared memory service.*

Real World Example: JavaSpaces in Jini

- Features of JavaSpaces:
  - *Asynchronous Object Sharing: Enables sharing and exchange of objects among networked clients and services asynchronously.*
  - *Location Transparency: Clients and services can interact without prior knowledge of each other's locations.*
  - *Collaboration: Supports seamless communication and collaboration between distributed components.*
- JavaSpaces demonstrates how Java RMI is effectively integrated into Jini to enable efficient communication and collaboration between distributed components.

Distributed Job Processing with JavaSpaces

- Job Submission:
  - *Clients submit data-intensive tasks (jobs) to JavaSpaces as encapsulated Java objects.*
  - *Java RMI is used for client-JavaSpace communication to submit job objects.*
- Task Processing:
  - *Multiple servers act as workers within the network.*
  - *Workers continuously monitor JavaSpaces for available jobs.*
  - *Java RMI is utilized to invoke methods on the job object, executing the required processing.*
- Result Aggregation:
  - *After processing, workers generate result objects.*
  - *Result objects are written back to JavaSpaces.*
  - *Clients interested in results can query JavaSpaces for completed results.*

Conclusion

- Simplified Distributed Communication: Java RMI exemplifies the power of remote method invocation for distributed applications.
- Abstraction of Network Complexity: RMI abstracts the intricacies of network communication, simplifying the creation of distributed systems.
- Streamlined Interaction: RMI offers a straightforward approach to interact with remote objects, regardless of their physical locations.