Practical 1(rmi)

1. What is RMI?
   RMI stands for Remote Method Invocation. It is a Java API used for implementing distributed applications, allowing Java objects to invoke methods on remote Java objects running on different Java Virtual Machines (JVMs). RMI enables communication between different JVMs over a network, providing a way to build distributed applications in Java.

2. What is the method that is used by RMI client to connect to remote RMI servers?
   The method used by RMI clients to connect to remote RMI servers is `java.rmi.Naming.lookup()` or `java.rmi.registry.LocateRegistry.lookup()`. These methods are used to look up and obtain a reference to a remote object registered with the RMI registry, allowing the client to establish a connection and invoke methods on the remote object.

3. How distributed garbage collector does manages the disconnections detected on the client side?
   In a distributed system with a distributed garbage collector (DGC), when a client node is detected as disconnected, the DGC marks the objects associated with that node as unreachable. It then initiates a garbage collection process to reclaim memory occupied by these unreachable objects. The DGC communicates with other nodes to ensure consistency in garbage collection across the distributed system. After garbage collection, resources associated with the disconnected client node are released, and system state is updated for recovery. This process helps maintain memory efficiency and system stability in distributed environments.

4. What is the relationship between the RMI and CORBA?
   RMI (Remote Method Invocation) and CORBA (Common Object Request Broker Architecture) are both technologies used for implementing distributed computing in different ways. The relationship between RMI and CORBA is that they are alternative approaches to achieving similar goals of distributed object communication and interoperability.

   RMI is specific to Java and allows Java objects to invoke methods on remote Java objects using Java Virtual Machine (JVM) capabilities. It simplifies distributed computing within the Java ecosystem.

   CORBA, on the other hand, is a platform-independent standard developed by the Object Management Group (OMG) for distributed computing across different programming languages and platforms. It provides a language-neutral way for objects to communicate and interoperate in a distributed environment.

   In summary, RMI is Java-centric and suitable for Java-based distributed systems, while CORBA is language-independent and supports interoperability across heterogeneous systems and languages. Both technologies aim to facilitate communication between distributed objects but differ in their scope and approach.

5. What is Remote object?
   A remote object is an object that can be accessed and used by other programs or systems running in a different address space, typically over a network. Remote objects are designed to be accessed remotely, meaning they can receive method invocations and return results over a communication channel like TCP/IP or RMI (Remote Method Invocation). In distributed computing, remote objects play a key role in enabling communication and interaction between different components or applications running on separate systems. Examples include objects accessed via RMI in Java or objects accessed through web services in a distributed system.

6. What is Server object?
   A server object refers to an object that provides services or functionality to clients in a client-server architecture. It typically runs on a server machine and listens for client requests. The server object handles these requests by executing the necessary operations and returning results to the clients. Server objects encapsulate the server-side logic and functionality required to fulfill client requests, such as processing data, accessing databases, or performing computations. In distributed systems, server objects play a central role in providing services that can be accessed remotely by clients over a network. Examples include web servers, application servers, or specific service-oriented components in distributed applications.

7. What is rmiregistry?
   `rmiregistry` is a simple remote object registry provided by Java for use with RMI (Remote Method Invocation). It acts as a lookup service where RMI server objects can be registered and accessed by RMI clients. The `rmiregistry` runs on a specified port (default is 1099) and allows RMI clients to obtain references to remote objects by looking them up using their registered names. This registry is essential for facilitating communication between RMI clients and servers, enabling clients to locate and invoke methods on remote objects across distributed Java applications.

8. What are different types of classes that are used in RMI.
   In RMI (Remote Method Invocation), there are primarily three types of classes used:

   1. **Remote Interface:** An interface that extends the `java.rmi.Remote` interface. This interface declares the methods that can be invoked remotely by clients. Clients interact with remote objects through these remote interfaces.

   2. **Remote Object:** A class that implements a remote interface and extends `java.rmi.server.UnicastRemoteObject` or `java.rmi.server.RemoteObject`. Instances of remote objects are the actual objects that clients can access and invoke methods on remotely.

   3. **Stub and Skeleton:** In older versions of RMI, stubs and skeletons were used for client-side and server-side communication, respectively. Stubs act as proxies for remote objects on the client side, while skeletons handle incoming remote method invocations on the server side. However, with modern RMI implementations, stubs and skeletons are generated automatically and are transparent to the developer.

These classes together enable the implementation of distributed applications using RMI, allowing Java objects to communicate and invoke methods across different JVMs over a network.

9. What is the main purpose of Distributed object applications in RMI?
   The main purpose of distributed object applications in RMI (Remote Method Invocation) is to enable communication and interaction between Java objects running on different Java Virtual Machines (JVMs) across a network. RMI allows developers to create distributed systems where objects located on one JVM can invoke methods on objects located on another JVM. This enables the building of distributed applications that leverage the modularity, reusability, and encapsulation of object-oriented programming while benefiting from the flexibility and scalability of distributed computing. The goal is to seamlessly extend the capabilities of Java applications across networked environments, facilitating distributed computing tasks such as remote method invocation, data sharing, and distributed processing.

10. How does the communication with remote objects occur in RMI
    In RMI, communication with remote objects occurs through method stubs and a process called marshalling and unmarshalling. The client obtains a reference to the remote object, and when invoking a method on this reference, the method call and parameters are serialized (marshalled) and sent over the network. The server receives this data, deserializes (unmarshals) it to reconstruct the method call, executes the method on the remote object, and then sends back any result in a serialized format. The client then deserializes the result to obtain the method's return value. This process allows Java objects to interact across networked JVMs as if they were local objects.

11. What are the steps that are involved in RMI distributed applications?
    1. **Define Remote Interface**: Create a Java interface that extends `java.rmi.Remote` with methods to be invoked remotely.
    2. **Implement Remote Object**: Create a class that implements the remote interface and extends `java.rmi.server.UnicastRemoteObject`.
    3. **Start `rmiregistry`**: Launch the RMI registry (`rmiregistry`) on a specified port.
    4. **Register Remote Object**: Bind the remote object instance to a name in the RMI registry using `Naming.rebind()`.
    5. **Write Server Application**: Implement a server application that creates and exports an instance of the remote object.
    6. **Write Client Application**: Implement a client application that looks up the remote object in the RMI registry using `Naming.lookup()`.
    7. **Invoke Remote Methods**: Use the remote object reference obtained from the registry to invoke remote methods as if they were local.

12. What is the use of java.rmi.Remote Interface in RMI?
    The `java.rmi.Remote` interface in RMI is used to mark interfaces whose methods can be invoked remotely. It defines remote interfaces whose methods can be accessed over a network using RMI. Objects implementing `java.rmi.Remote` can be registered with an RMI registry for remote access by clients. Clients receive proxy objects (stubs) that implement the remote interface to communicate with remote objects. This interface also helps in standardizing exception handling for remote method invocations using `java.rmi.RemoteException`.

13. Why are stubs used in RMI?
    Stubs are used in RMI to act as proxies for remote objects, handling network communication and method invocations on behalf of the client. They encapsulate RMI logic, making remote method calls appear as local method calls to the client. Stubs simplify the client-side interaction with remote objects by abstracting away network complexities and serialization details.

14. Why is the function or role of skeleton in RMI?
    The role of the skeleton in older RMI implementations was to receive incoming remote method calls on the server side, demarshal method parameters, and dispatch the method invocations to the actual remote object implementation. However, modern RMI implementations in Java no longer require explicit skeleton classes as stubs and skeletons are dynamically generated at runtime, simplifying the development process and abstracting away skeleton implementation details. Therefore, the role of the skeleton is now handled automatically by the RMI runtime, and developers primarily work with remote interfaces, implementations, and client-side stubs.

15. How dynamic class loading does happen in RMI
    In RMI, dynamic class loading occurs when the client looks up a remote object. The RMI runtime automatically downloads and loads the stub (proxy) class for the remote object from the server over the network. This stub class implements the remote interface and acts as a local proxy for the remote object. The client then invokes methods on the stub, which handles serialization and network communication to interact with the remote object on the server. This process enables transparent remote method invocation without the need for explicit class deployment on the client side.

Practical 4 (Berkeley)

1. What is Berkeley Algorithm?
   The Berkeley Algorithm is a time synchronization algorithm used in distributed systems to coordinate the clocks of multiple computers or nodes. It was developed at the University of California, Berkeley.
   In short, the Berkeley Algorithm operates by electing a master timekeeper node (typically the server or a designated node) that periodically polls other nodes for their local clock times. The master node calculates the average time among all nodes, determines clock adjustments needed for each node to synchronize with this average time, and then broadcasts these adjustments back to the nodes.
   Each node adjusts its clock based on the received correction from the master, thus achieving a level of synchronization across the distributed system. The Berkeley Algorithm helps in reducing clock skew and ensuring coordinated timekeeping among distributed nodes, which is crucial for various distributed applications and protocols.

2. What is importance of Berkeley Algorithm?
   The Berkeley Algorithm is important because it synchronizes clocks across distributed nodes, ensuring consistent timestamps and coordinated actions. This synchronization is essential for maintaining data integrity, supporting fault tolerance, optimizing performance, and enabling reliable operation of distributed systems.

3. How Berkeley Algorithm is useful?
   The Berkeley Algorithm is useful for synchronizing clocks in distributed systems, enabling coordinated actions, maintaining data integrity with consistent timestamps, improving fault tolerance by reducing clock skew, and optimizing performance of distributed algorithms. It enhances overall system coordination, reliability, and efficiency by establishing a common notion of time across distributed nodes.

4. How clock synchronization is achieved through Berkeley algorithm?
   Clock synchronization through the Berkeley Algorithm is achieved as follows:
   1. **Master Node Selection**: One node is elected as the master timekeeper, typically the server or a designated node.
   2. **Polling**: The master node periodically polls other nodes for their local clock times.
   3. **Calculation of Average Time**: The master node calculates the average time among all nodes based on the received clock readings.
   4. **Determining Adjustments**: The master node determines the clock adjustments needed for each node to synchronize with the average time.
   5. **Broadcasting Adjustments**: The master node broadcasts these adjustments to all nodes in the network.
   6. **Clock Adjustment**: Each node adjusts its clock based on the received correction from the master, aligning its time with the average time computed by the master.
   This process ensures that the clocks of distributed nodes converge towards a common time, achieving synchronization across the distributed system.

5. Which algorithm is used for clock synchronization?
   The algorithm commonly used for clock synchronization in distributed systems is the Network Time Protocol (NTP). NTP is a widely adopted protocol that allows computers to synchronize their clocks over a network. It uses a hierarchical system of time servers and employs algorithms to adjust the local clock of a computer based on time information received from reference time sources. NTP is designed to achieve high accuracy and reliability in clock synchronization across distributed systems and the Internet.

6. How does the Berkeley algorithm achieve fault tolerant average and give better synchronization of time?
   The Berkeley algorithm achieves fault-tolerant and better time synchronization by averaging clock readings from multiple nodes, dynamically adjusting clock values based on this average, and regularly updating to correct discrepancies. It incorporates redundancy to tolerate faulty readings from individual nodes and adapts to node failures by excluding them from the averaging process, ensuring accurate and resilient clock synchronization in distributed systems.

7. What is the difference between Berkeley and Cristian's algorithm?
   The main difference is in their architecture and approach:
   - **Berkeley Algorithm**: Uses a master-slave model where a master node polls slave nodes for their times, computes an average, and adjusts clocks dynamically. It's fault-tolerant and suited for distributed environments.
   - **Cristian's Algorithm**: Relies on a client-server model where a client requests time from a single time server. It's simple but less fault-tolerant and requires a reliable time server.

8. What is the function of clock synchronization?
   The function of clock synchronization is to ensure that clocks across multiple devices or systems maintain accurate and consistent time. This is important for:
   - **Coordination**: Enabling accurate sequencing of events and actions across distributed systems.
   - **Data Integrity**: Ensuring consistent timestamps for data transactions and operations.
   - **Communication**: Facilitating reliable communication and coordination between systems.
   - **Security**: Supporting accurate logging and timestamping for security-related activities.
   - **Performance**: Optimizing performance by coordinating tasks based on synchronized time references.
   Overall, clock synchronization is essential for the reliable and efficient operation of distributed systems and applications.

9. What is the accuracy of clock synchronization?
   The accuracy of clock synchronization refers to how closely clocks in a system align with each other and with a reference time source. It depends on factors like network latency, clock drift, synchronization protocol quality, and hardware/software precision. Synchronization can achieve accuracies ranging from microseconds to milliseconds or better, based on system requirements and synchronization methods used.

10. What are the two methods used for time synchronization?
    The two main methods used for time synchronization are:

    1. **Network Time Protocol (NTP)**:
       - Uses a hierarchical system of time servers to synchronize computer clocks over a network.
       - Adjusts local clocks based on time information obtained from reference time sources.
       - Provides accurate time synchronization and is widely used for internet-connected systems.

    2. **Precision Time Protocol (PTP)**:
       - Provides more precise time synchronization than NTP, suitable for high-performance and real-time applications.
       - Uses hardware timestamps and precise synchronization algorithms to achieve microsecond-level accuracy.
       - Commonly used in industrial automation, telecommunications, and financial trading systems.

    These methods are designed to synchronize clocks across distributed systems, ensuring accurate and coordinated timekeeping for various applications and environments.

11. What is an example of clock synchronization?
    An example of clock synchronization is the Network Time Protocol (NTP), which is widely used to synchronize clocks over the internet and computer networks. NTP allows computers to maintain accurate time by periodically adjusting their clocks based on time information obtained from reference time servers. This ensures that devices across the network have synchronized clocks, enabling coordinated actions, accurate event sequencing, and reliable communication in distributed systems.

12. What is Berkeley algorithm used for?
    The Berkeley algorithm is used for clock synchronization in distributed systems. It helps coordinate and synchronize the clocks of multiple nodes by computing an average time and adjusting clock values dynamically based on this average. This synchronization is crucial for maintaining consistency, coordination, and reliability across distributed systems.

13. Is Berkeley algorithm active or passive?
    The Berkeley algorithm is an active algorithm because it involves active participation of nodes in the distributed system. In this algorithm, a designated master node actively polls other nodes, computes an average time, and broadcasts clock adjustments to synchronize the clocks of all participating nodes. Each node in the system actively adjusts its clock based on the information received from the master node. Therefore, the Berkeley algorithm requires proactive involvement and communication among nodes to achieve clock synchronization.

14. What is the formula for propagation time for Berkeley algorithm?
    The formula for propagation time (\(T_p\)) in the context of the Berkeley algorithm is calculated as follows:

$T_p = 21 \times (t_r - t_s)$

Where:

- tr t_r is the time received from a slave node by the master node.
- ts t_s is the time sent by the master node to the slave node.

The propagation time (Tp T_p) represents half of the round-trip time (RTT) between the master node and a slave node. It is used in the Berkeley algorithm to estimate clock adjustments based on the difference in clock readings between nodes.

.

15. What are the benefits of clock synchronization
    The benefits of clock synchronization include:
    1. **Coordinated Actions**: Ensuring accurate sequencing and coordination of events and actions across distributed systems.
    2. **Data Integrity**: Maintaining consistent timestamps for data transactions and operations, supporting data integrity and reliability.
    3. **Communication**: Facilitating reliable communication and coordination between systems, enabling effective distributed computing.
    4. **Security**: Supporting accurate logging and timestamping for security-related activities such as audit trails and access control
    5. **Performance Optimization**: Improving system performance by synchronizing tasks based on accurate and synchronized time references.
    Overall, clock synchronization enhances the reliability, efficiency, and performance of distributed systems by aligning clocks and maintaining consistent time across networked devices and applications.

Practical 5(token Ring)

1. What is Mutual Exclusion?
   Mutual exclusion refers to the property that ensures only one process or thread accesses a shared resource or critical section at a time, preventing concurrent or simultaneous access by multiple processes. This ensures that conflicting operations do not interfere with each other, maintaining data integrity and preventing race conditions in concurrent systems.

2. What are the different mutual exclusion algorithms available in distributed systems?
   In distributed systems, different mutual exclusion algorithms include:
   1. **Token Ring Algorithm**: Nodes pass a token to control access to a critical section. Only the node holding the token can enter the critical section.
   2. **Ricart-Agrawala Algorithm**: Nodes request access and grant permission based on timestamps, ensuring mutual exclusion without relying on a central coordinator.
   3. **Maekawa's Algorithm**: Divides nodes into clusters and requires permission from specific clusters to access critical sections, reducing message complexity compared to other algorithms.
   4. **Raymond's Algorithm**: Uses a tree structure to manage access requests and permissions, allowing efficient distributed mutual exclusion.
   Each algorithm provides a different approach to achieving mutual exclusion in distributed environments, considering factors like message complexity, coordination overhead, and fault tolerance.

3. What are the requirements of mutual exclusion?
   The requirements of mutual exclusion in a distributed system are:
   1. **Safety**: Ensuring that only one process accesses a critical section at a time to prevent conflicting updates and maintain data integrity.
   2. **Liveness**: Guaranteeing that if a process requests access to a critical section, it eventually gains access, even in the presence of failures or delays.
   3. **Fairness**: Ensuring that processes waiting to enter the critical section are granted access in a fair and orderly manner, without starvation or indefinite postponement.
   These requirements collectively ensure correct and efficient synchronization of concurrent processes accessing shared resources in a distributed system.

4. Classify Distributed mutual exclusion algorithm.
   Distributed mutual exclusion algorithms can be classified into:
   1. **Token-Based Algorithms**:
      - Nodes pass a token to control access to critical sections.
      - Examples: Token Ring Algorithm, Ricart-Agrawala Algorithm.
   2. **Quorum-Based Algorithms**:
      - Nodes require permission from a subset (quorum) of nodes to enter critical sections.
      - Examples: Maekawa's Algorithm, ISIS Algorithm.
   3. **Centralized Algorithms**:
      - Use a central server or coordinator to manage a
   ccess to critical sections.

5. Differentiate between token based algorithms and non-token based algorithms.
   Token-Based Algorithms:
   - Use a token (or permission) that is passed among processes to control access to critical sections.
   - Examples: Token Ring Algorithm, Ricart-Agrawala Algorithm.
   - **Non-Token-Based Algorithms**:
   - Do not rely on passing tokens; instead, they use other mechanisms like requesting permission from a subset of nodes (quorum) or using centralized coordination.

- Examples: Quorum-Based Algorithms (e.g., Maekawa's Algorithm), Centralized Algorithms (e.g., Lamport's Bakery Algorithm).

6. What are different non-token based algorithm are there in distributed system?
   Non-token-based algorithms in distributed systems include:
   - **Quorum-Based Algorithms**:
     - Maekawa's Algorithm
     - ISIS Algorithm
   - **Centralized Algorithms**:
     - Lamport's Bakery Algorithm (adapted for distributed systems)

7. What are different token based algorithm are there in distributed system?
   Token-based algorithms in distributed systems include:

   - **Token Ring Algorithm**
   - **Ricart-Agrawala Algorithm**

8. What are the different performance measure of mutual exclusion algorithms?
   - **Message Complexity**: Number of messages exchanged between nodes.
   - **Synchronization Delay**: Time taken for a process to enter the critical section after requesting access.
   - **Fault Tolerance**: Ability to handle node failures or network partitions.
   - **Scalability**: Ability to perform efficiently as the system size increases.
   - **Fairness**: Ensuring that processes waiting for access eventually gain entry (avoiding starvation).

9. Give comparative performance analysis of mutual exclusion algorithms
   - **Token-Based vs. Non-Token-Based**:
     - Token-based algorithms can have lower message complexity but require token passing.
     - Non-token-based algorithms may have higher message complexity but offer more flexibility in terms of coordination.
   - **Quorum-Based vs. Centralized**:
     - Quorum-based algorithms distribute coordination but require agreement among quorums.
     - Centralized algorithms rely on a single point of coordination, which can become a bottleneck.

   Performance analysis considers factors such as scalability, fault tolerance, message overhead, and fairness to determine the suitability of mutual exclusion algorithms for specific distributed system architectures and requirements.

Practical 6 (Ring and Bully)

1. What is Election Algorithm?
   An Election Algorithm is a distributed computing algorithm used to elect a leader or coordinator among a group of nodes in a distributed system. The leader is typically responsible for coordinating activities, making decisions, or providing services to other nodes.

2. Name different election algorithms?
   Different election algorithms include:
   - Bully Algorithm
   - Ring Algorithm
   - Chang-Roberts Algorithm
   - LCR Algorithm (Lynch, Fischer, and Patterson)
   - HS Algorithm (Hirschberg and Sinclair)

3. What is Bully and Ring Algorithm?
   - **Bully Algorithm**: Nodes in the system elect the node with the highest priority (ID) as the leader. If a lower priority node detects the absence of a leader, it initiates an election by challenging higher priority nodes.

   - **Ring Algorithm**: Nodes arrange themselves in a logical ring topology. When a node detects the absence of a leader, it sends an election message around the ring. The node with the highest ID becomes the leader.

4. What election algorithm does?
   An election algorithm determines which node in a distributed system should act as the leader or coordinator by following a set of rules or protocols to elect a single node from a group of nodes.

5. Why election algorithms are normally needed in a distributed system?
   Election algorithms are needed in distributed systems to establish a single node (leader) responsible for managing system-wide tasks, ensuring coordination, and maintaining system stability. Leaders can facilitate decision-making, resource allocation, fault recovery, and overall system management.

6. What is leader election algorithm and why do we need this algorithm?
   A leader election algorithm is used to select a leader node from a group of nodes in a distributed system. We need this algorithm to ensure that there is a designated node responsible for coordinating activities, maintaining consistency, and providing centralized control in the system.

7. How many types of messages are there in election algorithm? What are the features required for election algorithms?
   Election algorithms typically involve different types of messages:
   - Election Messages: Used to announce the start of an election or propagate election information.
   - Response Messages: Used to respond to election messages and participate in the election process.

   Features required for election algorithms include:
   - Correctness: Ensuring that only one node is elected as the leader at any given time.
   - Fault Tolerance: Handling node failures or network partitions gracefully.
   - Efficiency: Minimizing message complexity and synchronization delays.
   - Scalability: Performing efficiently as the system size or complexity increases.

8. What is the purpose of election system?
   The purpose of an election system is to establish a leader or coordinator node dynamically in a distributed system to manage system-wide tasks, provide centralized control, and ensure efficient coordination among nodes.

9. What is the time complexity of leader election
   The time complexity of leader election algorithms varies depending on the specific algorithm and the underlying communication and coordination mechanisms. Some algorithms have linear time complexity $O(n)$, where n is the number of nodes, while others may have logarithmic time complexity $O(\log n)$ or constant time complexity $O(1)$ under certain assumptions or conditions. The complexity also considers message overhead, synchronization delays, and fault tolerance aspects of the algorithm.

Practical 7 (web service)

1. What is Web Service?
   A Web Service is a software system designed to allow interoperable communication between different applications over a network. It enables interaction and data exchange between software components using standard protocols like HTTP, XML, and JSON.

2. What are different types of Web Services?
   Different types of Web Services include:
   - SOAP (Simple Object Access Protocol) based services
   - REST (Representational State Transfer) based services
   - XML-RPC (XML Remote Procedure Call) services

3. What is SOAP?
   What is REST?
   - **SOAP (Simple Object Access Protocol)**: A protocol used for exchanging structured information in the form of XML-based messages over various transport protocols like HTTP, SMTP, etc.
   - **REST (Representational State Transfer)**: An architectural style that uses HTTP methods (GET, POST, PUT, DELETE) to access and manipulate resources. It emphasizes simplicity, scalability, and statelessness.

4. Explain Web Services Architecture?
   Web Services Architecture typically consists of:
   - **Service Provider**: Exposes services to be consumed by clients.
   - **Service Requestor**: Consumes services provided by the Service Provider.
   - **Service Registry**: Stores metadata and location information of available services.
   - **Message Format**: Defines the format (e.g., XML, JSON) for data exchange.
   - **Transport Protocol**: Specifies the underlying protocol (e.g., HTTP, SMTP) for message delivery.

5. Explain the concept of Service Provider.
   A Service Provider is a component or entity that exposes Web Services. It publishes services and makes them available to be accessed and consumed by Service Requestors over the network.

6. Explain the concept of Service Requestor.
   A Service Requestor is a client application or component that consumes Web Services provided by Service Providers. It sends requests to access and utilize the functionalities offered by the services.

7. Explain the concept of Service Registry.
   A Service Registry is a centralized repository that stores metadata and location information of available services in a distributed system. It helps Service Requestors discover and locate services dynamically.

8. Differentiate SOAP and REST
   - **SOAP**:
     -Protocol-based, using XML for message formatting.
     - Supports more complex operations and messaging patterns (e.g., security, transactions).
     - Requires predefined contracts (WSDL) for service description.
   - **REST**:
     - Architectural style using HTTP methods (GET, POST, PUT, DELETE) to access and manipulate resources.
     - Uses lightweight data formats like JSON or XML for message formatting.
     - Emphasizes simplicity, statelessness, and scalability without strict contracts like WSDL.