

Middleware Architectures 1

Lecture 4: Application Server Architecture

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • <http://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <http://vitvar.com/courses/mdw>



Modified: Sat Sep 19 2020, 17:17:08
Humla v0.3

Overview

- **Application Server Architecture**
- Distribution of Objects

Application Server Overview

- An environment that runs an application logic
 - *A client communicates with the server using an application protocol*
- Application Server
 - *A modular environment*
 - *provides technology to realize enterprise systems*
 - *JEE containers – Java technology for AS components*
 - *Supports a variety of objects such as Servlets, JSPs, JMS*
 - *Provides services such as naming and directory, performance, failover*
 - *Provides Web server capabilities*
 - *Can be a single server or multiple servers*
- Web Tier – HTTP Server
 - *Web Server supports HTTP only*
 - *HTTP request/response, security, proxy, caching*

Application Server Layers



- Features
 - *AS instance appears as a single process in the OS*
 - *you can use standard OS commands to investigate its operation*
 - *AS listens on a single or multiple IPs (VIPs) and a tcp port*
 - *AS is a Java process*
 - *you can use Java tools to investigate its operation*
 - *Garbage collector stats, thread dumps, memory allocations, etc.*

Example Weblogic Infrastructure



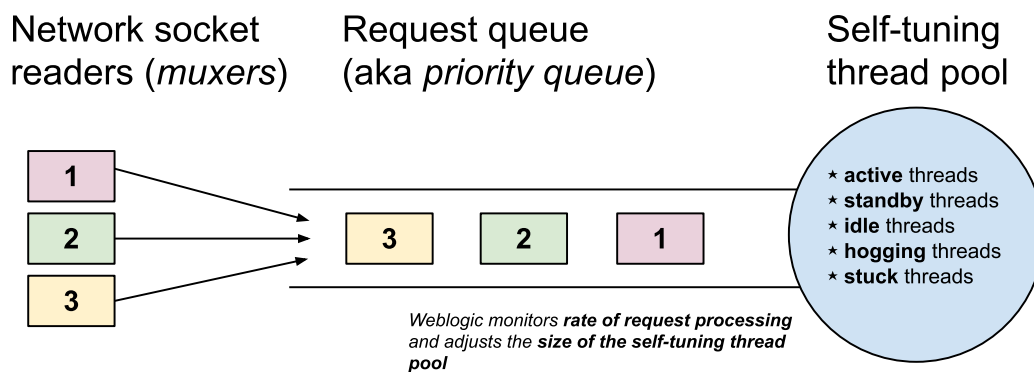
Terminology

- **Domain**
 - A group of servers with specific configuration of applications and objects
- **Administration Server**
 - An instance of application server that manages the domain
- **Managed Server**
 - An instance of application server running instances of applications and objects
- **Cluster**
 - A group of managed servers; they contain the same copy of applications and objects
- **Machine**
 - A physical machine and OS running one or more servers (Admin or Managed)
- **Node Manager**
 - A process that provides an access to admin and managed servers on the machine
- **Load Balancer**
 - A network element that distributes client requests to managed servers based on a specific algorithm

Servlet Technology

- Technology to extend application server functionalities
 - A Java class that can respond to any type of requests
 - A servlet defines an interface for a specific protocol
 - Your application implements the servlet's interface
- Commonly used to respond to HTTP requests
 - A basis for an application running on an application server
 - HTTP Servlet Java classes
 - **HttpServlet** – provides HTTP protocol interface
 - **HttpServletRequest** – represents HTTP request
 - **HttpServletResponse** – represents HTTP response

Handling Requests in Weblogic



- **Muxer** – component that handles communication via network sockets.
- **Request queue** – queue of requests to be processed.
- **Self-tuning thread pool** – a pool of threads in various states.
- **Work manager** – a configuration of **maximum threads** and a **capacity** that can be used to handle requests for a specific application/service.

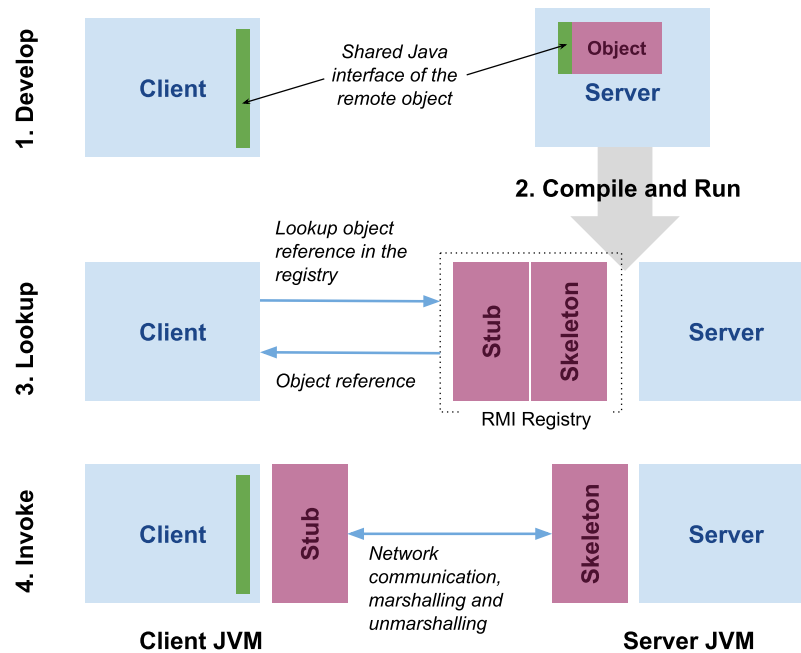
Overview

- Application Server Architecture
- Distribution of Objects

Remote Method Invocation

- Communication among Java-based applications
 - *Methods of a Java class can be invoked by other Java class remotely*
 - *Uses Java Remote Method Protocol (JRMP)*
 - *Java-specific application protocol over TCP/IP*
 - *Basis for JEE technologies, such as JMS*
- Terminology
 - **Client** – *a program that invokes a remote method*
 - **Server** – *a program that exports a remote object*
 - **Stub** – *a representation of the client-side object for communication*
 - **Skeleton** – *a representation of the server-side object for communication*
 - **Registry** – *a component that holds a stub*
 - **Marshalling/Unmarshalling** – *a process of transforming memory representation of the object to a form suitable for network transmission and vice-versa*

RMI Stubs and Skeletons



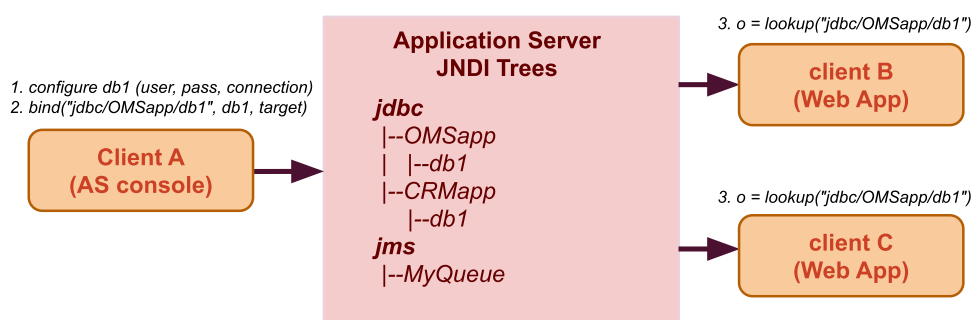
Java Naming and Directory Interface

- **Objectives**
 - Allows to access objects by names in various directory systems and their attributes
 - Independent of any specific directory service implementation
 - Enables to distribute Java objects across various systems in the environment
- **Terminology**
 - *Binding* – association between a name and a object
 - *Context* – a set of bindings
- **JNDI Provides:**
 - a mechanism to bind an object to a name.
 - a directory lookup interface
 - a pluggable service provider interface (SPI) – any directory service implmentation can be plugged in

Application Server and JNDI

- **Distribution of objects**
 - *Application Server provides central directory for various kinds of objects*
 - *Datasources, JMS queues and topics, etc.*
 - *Clients store objects in the central directory*
 - *Administrator configures objects using Application Server Console or via AS API*
 - *Clients retrieve objects from the central directory*
- **Benefits**
 - *replication of objects across clients*
 - *central configuration of objects' parameters*
 - *scalability – allowing/disabling connections as required*

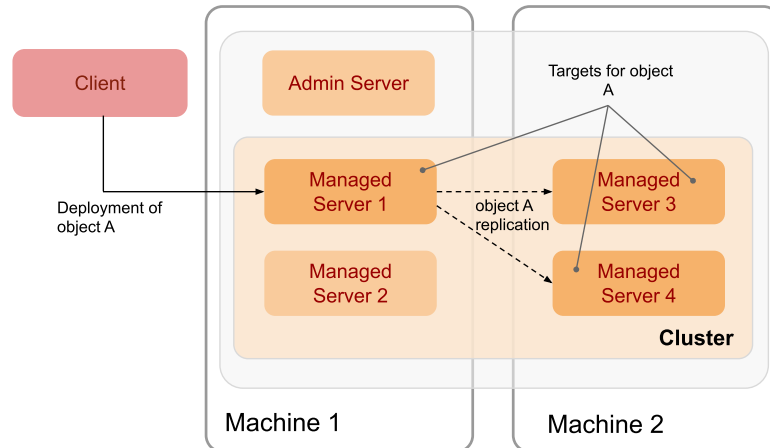
JNDI Example



- **Example Scenario**
 - *Client A creates a datasource, configures it and registers it in the JNDI tree*
 - *Client A is a Admin server console app*
 - *Client B and C lookup the object under specific JNDI name and retrieves the object from the tree*
 - *They get the object from the tree and use it to connect to the DB*
 - *They do not need to know any DB specific details*
 - *The object is pre-configured from the server*

Deployment to Cluster

- Deployment of an object
 - Client deploys to one managed server in the cluster
 - Object gets replicated to its targets
 - Targets can be configured for the object, usually all servers but can be selected servers



Cluster-wide JNDI Tree

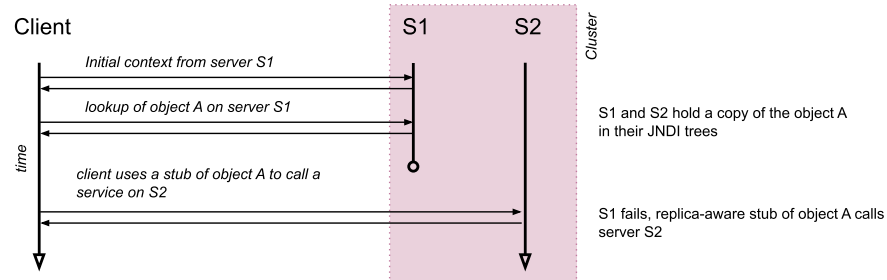
- Cluster
 - Every managed server has its own JNDI tree
 - Servers in a cluster sync up their JNDI trees as per the target configuration
 - A stub of the object appears in every managed server's JNDI tree
 - They use JNDI replication service
- When a client retrieves an object from the tree
 1. Client connects to the cluster using the cluster address
 2. Client creates an initial context (represents a naming service)
 3. Client uses the initial context to lookup objects
 4. Client uses the stub of the object to call the service

Object Failover

- Failover

- *Failover = ability to locate an object on another server that holds a copy of the object without impact on the performance and configuration*

Replica-aware stub of object A, failover in cluster



- *A client gets a stub of the object by calling **lookup** on the context*
- *A client uses the stub of the object to access the object on the server*
- *When a server fails, **replica-aware stub** calls the next server that holds the object copy*