RMI (Remote Method Invocation)

Java-spezifische Implementierung eines Systems für verteilte Objekte "Natürliche" Integration des verteilten Objekt-Modells in die Sprache Java

Von den an verteilte objektorientierte Systeme gestellten Forderungen werden erfüllt:

- Lokation unabhängige Programmierung und transparenter Methodenaufruf

Realisierung der grundlegenden Fähigkeiten eines verteilten Objektsystems

- Lokalisierung entfernter Objekte

  Zwei Mechanismen stehen zum Suchen von Referenzen auf entfernte Objekte zur Verfügung:

  1. Nachfragen bei einem RMI-eigenen Namensdienst (rmiregistry)
  2. Übergabe als Parameter und/oder Rückgabewerte von entfernten Funktionsaufrufen

- Kommunikation mit entfernten Objekten

  Die eigentliche Kommunikation wird vom RMI-Laufzeitsystem durchgeführt, Einzelheiten bleiben der eigentlichen Anwendung (und ihrem Programmierer) verborgen.

  Ein entfernter Methodenaufruf sieht wie ein normaler lokaler Methodenaufruf aus (transparenter entfernter Methodenaufruf)

- Übergabe von Objekten als Parameter und Rückgabewerte

  Das RMI-Laufzeitsystem stellt die notwendigen Mechanismen zur Übertragung von Objekt-Daten und -Code zur Verfügung (Kopieren von Objekten)

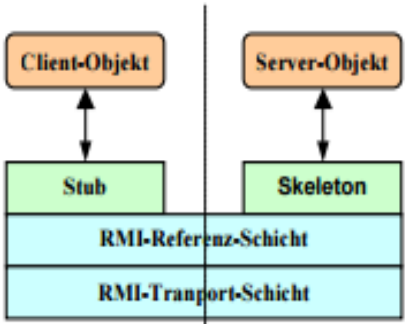
  Dynamischen Ladens von Code (dynamic code downloading) > Erzeugung von Objekten, deren Klassen-Implementierung lokal nicht verfügbar ist

Weitere wesentliche Eigenschaften von RMI

- Das dynamische Laden von Code ermöglicht die dynamische Änderung des Verhaltens von Server und/oder Client
- Unterstützung von Callbacks vom Server zum Client Übergabe von Referenzen auf – im Client vorhandene – entfernte Objekte als Parameter im entfernten Methoden- Aufruf
- Einbettung in die Sicherheitsmechanismen der Java-Plattform Verwendung von Security Manager, Class Loader und Policy-Files zur Überprüfung der Zulässigkeit des Ladens von entferntem Code und des Ausführens von Code (Ressourcenzugriff).

RMI -Architektur

- Das RMI-System besteht aus drei Schichten:
  - Stub-/Skeleton-Schicht
  - RMI-Referenz-Schicht
  - RMI-Transport-Schicht



- Die **Stub-/Skeleton-Schicht** bildet die Schnittstelle zwischen den Bestandteilen der verteilten Applikation (Client-Objekt und Server-Objekt) und dem restlichen RMI-System.

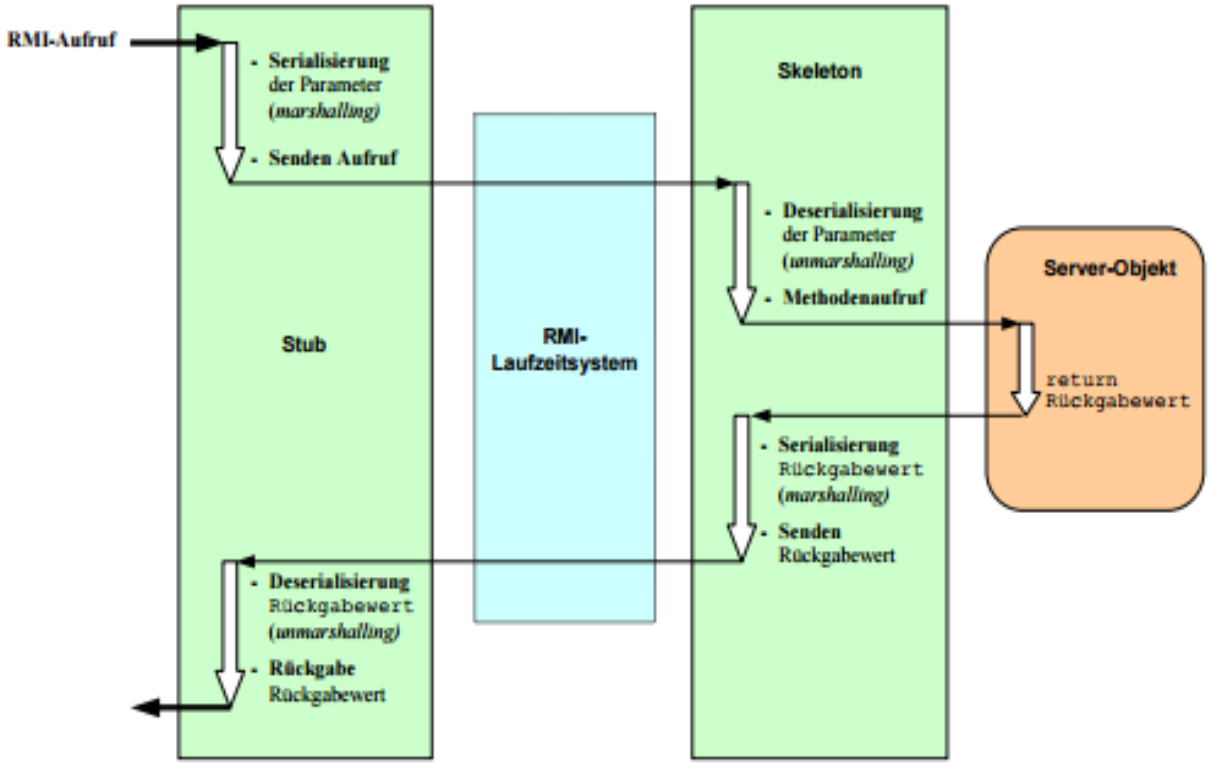
Der **Stub** ist ein Stellvertreter-Objekt des Server-Objekts auf der Client-Seite, der die gleiche Methoden-Aufruf- Schnittstelle wie das Server-Objekt anbietet. Er nimmt die RMI-Aufrufe des Clients entgegen und leitet sie an die RMI-Referenzschicht weiter. Umgekehrt erhält er von der RMI-Referenzschicht die Rückgabewerte eines RMI-Aufrufs und leitet diese an das aufrufende Client-Objekt weiter.

Das **Skeleton**-Objekt nimmt auf der Serverseite die RMI-Aufrufe von der RMI-Referenzschicht entgegen, ruft die entsprechenden im Server-Objekt implementierten Methoden auf, nimmt deren Rückgabewerte entgegen und leitet diese an die RMI-Referenzschicht weiter.

Die Funktionalität des Skeletons wird in neueren Java-Versionen durch einen allgemeinen Verteiler ersetzt. Es muss daher kein eigenes Skeleton-Objekt erzeugt werden. Der Stub muss dagegen an das jeweilige Server-Objekt angepasst sein. Seine Klasse wird aus der Server-Objekt-Klasse mittels eines RMI-Compilers (rmic) automatisch generiert.

- Die **RMI-Referenz-Schicht** ist für die Lokalisierung des Kommunikationspartners zuständig. Sie beinhaltet den RMI-Namensdienst (Registry) und verwaltet die Referenzen auf entfernte Objekte. Unter Berücksichtigung der jeweiligen Referenz-Semantik gibt sie die vom Stub erhaltenen RMI-Aufrufe einschließlich deren Parameter an die RMI-Transportschicht weiter.

  Auf der Serverseite nimmt sie die RMI-Aufrufe und deren Parameter von der RMI-Transportschicht entgegen, aktiviert gegebenenfalls das Server-Objekt und leitet die Aufrufe an das Skeleton-Objekt weiter Analog ist sie an der Übermittlung der Rückgabewerte beteiligt.
- In der **RMI-Transport-Schicht** werden die Kommunikationsverbindungen verwaltet und die eigentliche Kommunikation zwischen verschiedenen Adressräumen (JVMs) abgewickelt. Üblicherweise werden hierfür Sockets eingesetzt.

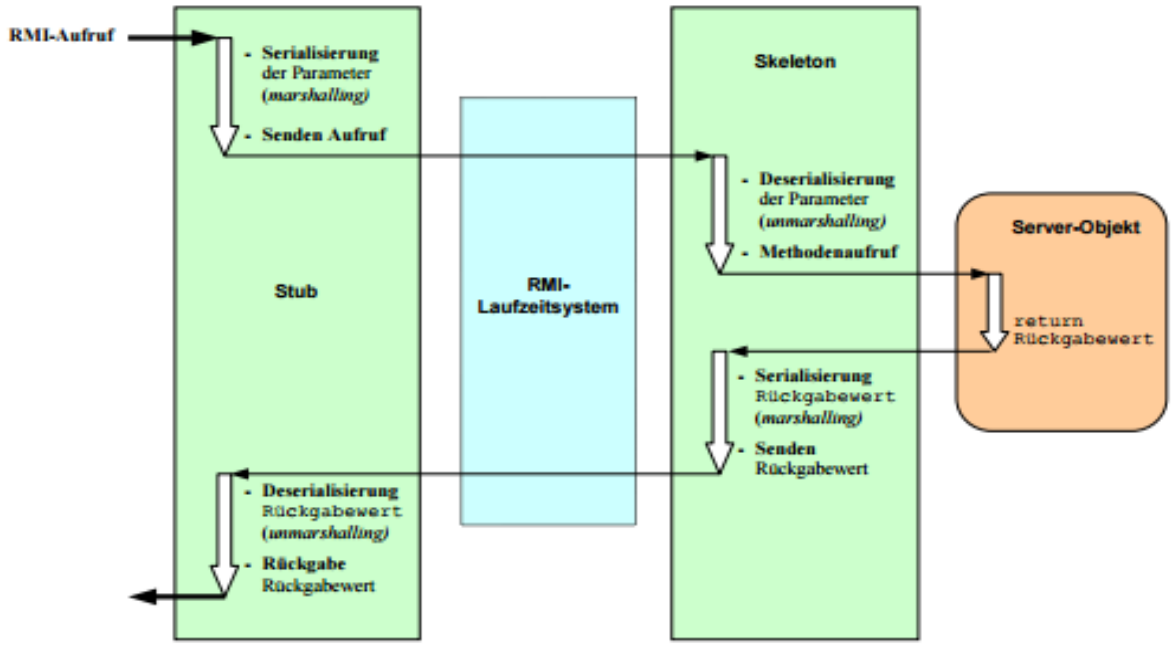


Zusammenspiel zwischen Stub und Skeleton

- Der Stub wandelt die Methoden-Aufruf-Parameter mittels Objekt-Serialisierung in einen seriellen Byte-Stream um (marshalling). Dadurch können die Parameter zusammen mit den für den Aufruf der Serverfunktion benötigten Informationen (Ziel-Objekt und der aufzurufenden Methode) über das RMI-Laufzeitsystem zum Server-Skeleton übertragen werden.
- Das – durch einen allgemeinen Verteiler realisierte – Skeleton deserialisiert den empfangenen Byte-Stream und gewinnt dadurch die Parameter zurück (unmarshalling). Anschließend ruft es die durch den übertragenen Methoden-Identifikator ausgewählte Methode des Server-Objekts mit den deserialisierten Parametern auf.
- Die Übertragung des Funktionsrückgabewertes erfolgt analog vom Server (Serialisierung) zum Client (Deserialisierung)

RMI-Registry

- Die RMI-Registry ist ein einfacher Namensdienst, der es Clients ermöglicht, entfernte Objekte über ihren Namen zu lokalisieren.
- Eine RMI-Registry muss lokal auf dem System, das Objekte für RMI-Aufrufe (entfernte Objekte) bereitstellt (RMI- Server) existieren.
- Üblicherweise wird die RMI-Registry in einer eigenen JVM als Hintergrundprozess gestartet. Default ist Port 1099, kann aber auch mit einer anderen Port-Nummer gestartet werden.
- Ein entferntes Objekt, das über die Registry zugänglich sein soll, muss vom Server-Prozess der Registry bekannt gemacht (registriert, "gebunden") werden.
- Ein Client, der einen RMI-Aufruf ausführen möchte, benötigt eine Referenz auf das entsprechende (entfernte) Server- Objekt. Diese erhält er durch eine Nachfrage ("lookup") bei der RMI-Registry des Server-Rechners. Falls für das – über einen Namen referierte – Objekt ein Eintrag vorhanden ist, liefert die Registry sein als Referenz dienendes Stub-Objekt zurück.
- Normalerweise wird die RMI-Registry nur zur Lokalisierung des ersten vom Client referierten entfernten Objekts verwendet. Weitere gegebenenfalls benötigte entfernte Objekte können dann über dieses Objekt ermittelt werden (als Parameter/Rückgabewerte von RMI-Aufrufen)
- Die Kommunikation mit der RMI-Registry erfolgt üblicherweise über das von der Java-Klasse java.rmi.Naming bereitgestellte Interface (statische Methoden)
- Tatsächlich stellt die RMI-Registry bereits selbst ein besonderes entferntes Objekt dar, zu dem auch mittels RMI, das durch die Naming-Funktionen gekapselt wird, zugegriffen wird.



java.rmi.server.codebase

- Diese Eigenschaft gibt die Orte von den Klassen an, die von der VM veröffentlicht werden (z.B. Stub-Klassen, benutzerdefinierte Klassen, die den deklarierten Rückgabetyp eines Remote-Methodenaufrufs implementieren). Der Wert dieser Eigenschaft ist ein String im URL-Format oder eine Leerzeichen-getrennte Liste von URLs.
- Hinweis: Diese Eigenschaft muss korrekt eingestellt sein, um Klassen und Schnittstellen mit Java RMI dynamisch herunterzuladen. Wenn diese Eigenschaft nicht richtig eingestellt ist, werden Sie Probleme haben, Server und Client zu starten.

java.rmi.server.hostname

- Der Wert der Eigenschaft ist ein Hostname als String, der mit Remote-Stubs für lokal erstellte Remote-Objekte verknüpft werden soll, damit Client-Methoden auf dem Remote-Objekt aufgerufen werden können. Default Wert ist die IP-Adresse des lokalen Hosts.

java.rmi.server.useCodebaseOnly

- Wenn dieser Wert true ist, ist das automatische Laden von Klassen außer dem lokalen CLASSPATH und von der java.rmi.server.codebase auf der VM verboten. Durch die Verwendung dieser Eigenschaft wird verhindert, dass Client-VMs Bytecodes aus anderen Codebases dynamisch herunterlädt (leider einige Bugs).

java.security.policy

- Diese Eigenschaft wird verwendet, um die Richtliniendatei anzugeben, die die Berechtigungen enthält, die Sie gewähren möchten.

RMISecurityManager

- Eine Unterklasse von SecurityManager, die von RMI-Anwendungen verwendet wird, die heruntergeladenen Code verwenden. Der RMI-Klassenloader lädt keine Klassen von entfernten Standorten herunter, wenn kein Sicherheitsmanager gesetzt wurde.