Chapter0. Introduction

2019年2月20日

8:01

Note Taking Area

**Key textbook**

* Distributed Systems: Concepts and Design: 分布式系统概念与设计
* Mastering Cloud Computing

**Course information**

* BOOKS: Using DSCD for about 10 weeks, and the last 2 or 3 weeks use MCC.
* Grade percentage:
  + Final exam 70% - let me die!
  + Lab and Assignment 30%.
* Don't miss labs!

Cue Column

Summaries

1. Key textbook and related books.
2. Course information.

Chapter1. Characterization of Distributed Systems

2019年2月20日

8:19

Note Taking Area

**A typical portion of the Internet**

rant 
desktop computer: 
server: 
network link: 
ackbone 
satellite link 

* Different kinds of servers: email, print, router/firewall, file, web, and other.
* the annals of distributed system:

计算机生成了可选文字:
1970: DARPA's TCP/IP 
1984: IEEE 802.3 
Ethernet & LAN 
1999: 
Grid Computing 
1997: 
802.11 (WieFi) 
2010: Microsoft 
Azure 
2008: google 
Apppng!ne 
19 1: UNIVAC l, 
Fir t Mainframe 
1966: Flynn's Texönomy 
*SD, SIMD, MISO, MIMD 
1969: A*PANET 
19/5: Xerox PARC 
Invented Ethernet 
1989: TCP/IP 
IETF RFC 1122 
1984: DEC's 
VMS cluster 
• Lee-Calliau 
1990. 
WWW, HITP, HTML 
1990 
2007: Manjrasoff Aneka 
?OD5: Ampzon 
(EC2', S3) 
2004: web 2.0 
Clouds 
Grids 
Clusters 
Mainframes 
1950 
1960: Cray's First 
Supercomputer 
1960 
1970 
1980 
2000 
2010 

* Map-reduce: remove invalid data.
* SISD, SIMD, MISD, MIMD: Single/multi instruction flow, and single/multi data flow.
  + 单/多指令流、单/多数据流。

**Hardware introduction**

* The GPU isn't like CPU, which is better at float computing.
* The cost of CPU to access different region in CPU is different.

**Trend in computing**

Logon d 
Multi-scale, Multi-Model 
Optimization 
Implicit Solvers 
Adaptive F.&sh Refinement 
Data analysis 
Linear and nonlinear solvers 
Graph algorithms 
Agent-based modeling 
High-precision arithmetic 
High-dim. parameter spaces 
Data assimilation 

* Many areas uses cloud machine, but white blocks mean there doesn't need cloud machine.
* The limit factor of super computer is power.

**Grid standards and middleware**

* HPC Systems: high-performance computing.
  + Homogeneous nodes -> high speed -> clusters or MPPs(centralized control) -> disparate clusters -> computational and data grids -> web2.0 service / internet clouds(virtualization) / internet of things(RFID and sensors)
* HTC Systems: high-throughput computing.
  + Disparate nodes -> P2P network(distributed control) -> geographically sparse -> computational and data grids -> web2.0 service / internet clouds(virtualization) / internet of things(RFID and sensors)
* P2P: peer to peer.
* MPP: massively parallel processors.

**Distributed systems**

* Definition: a system in which components located in networked computers communicate and coordinate their actions only by passing messages.
  + Motivation: sharing of resources (example web servers and web browsers).
* Computer network: a collection of interconnected autonomous computers.
  + Generality: built from general purpose hardware, instead of any particular application.
* Relationship between computer network and distributed system:
  + Transparency, and DS is a software runs on the top of CN.

**CN layer architecture**

Host I 
Layer 5 
Layer 4/5 interface 
Layer 4 
Layer 3/4 interface 
Layer 3 
Layer 2/3 interface 
Layer 2 
Layer 1/2 interface 
Layer I 
Layer 5 protocol 
Layer 4 protocol 
Layer 3 protocol 
Layer 2 protocol 
Layer 1 protocol 
Physical medium 
Host 2 
Layer 5 
Layer 4 
Layer 3 
Layer 2 
Layer I 

* Purpose: offer a communication services to higher level layers.
* Two interfaces in each layer:
  + Peer-to-peer: defines the form and types of messages exchanges between peers.
  + Services: defines the primitives (operations) that a layer provides to the layer above it.
  + Layering is non-linear.

**Internet architecture**

* ISO OSI seven layers:
  1. Physical: transmission of raw bits onto the communications medium.
  2. Data link: reliable transmission of frames, flow control, arbitration.
  3. Network: packet switching, routing congestion control.
  4. Transport: process-to-process channel, node-to-node connection, provides user services, flow control, multiplexing.
  5. Session: Protocols necessary to establish and maintain a connection or session between 2 end-users, orderly communication (dialog management, synchronization points, activities).
  6. Presentation: communication of information rather than data, code and number conversion, transmission of sophisticated data structure, and data compression.
  7. Application: electronic mail, ftp, telnet, distributed system, and client-server.
* TCP/IP four layers:
  1. Host-to-network layer: OSI physical and data link layers.
  2. Internet layer: OSI network layer - Internet Protocol / IP.
  3. Transport layer: transmission control protocol / TCP & user datagram protocol / UDP.
  4. Application layer.

Cue Column

**Instruction flow and data flow**

* SISD: old serial computer, and in one clock period, one data flow is operated.
  + Von Neumann machine, IBM PC, and 8 bit computer.
* SIMD: one instruction flow to operate multi data flow.
  + Digital signal processing, photo processing, and multimedia information processing.
* MISD: just a concept and never show up.
* MIMD: carry out multi instruction flow and process multi data flow at same time.
  + Multi cores computer.

Summaries

1. A typical portion of Internet.
2. Hardware introduction.
3. Trend in computing.
4. Grid standards and middleware.
5. Distributed systems.
6. CN layer architecture.
7. ISO OSI architecture and TCP/IP architecture.

Chapter2. System Models

2019年3月5日

11:24

**NOTE TAKING AREA**

e-Business application

* Architecture: structure in terms of separate specified component.
  + User: client <-> presentation layer <-> application logic layer <-> data layer.
* Presentation layer:
  + Interaction with user.
  + User submits operations and get responses.
  + The boundary of presentation layer and client can be thin, like Java Applet.
* Application logic layer (business process, business logic):
  + Process data before delivery of the result (implementation).
  + All those programs and module involved in processing the operation.
  + Single tier: all three into a single component.

Presentation 
Application 
Logic 
Data 
webserver 
Dumb Terminal 
•Mostly in Mainframes from the days that: 
•640K ought to be enough for anybody. 
•"I think there is a world market for 
maybe five computers". 

* 2-tier architecture: birth of PC.

user: client 
Presentation Layer 
Application Logic Layer 
Data Layer 

* Remote procedure call, fat client and thin client.
* Three tier: infrastructure that supports the development of ALL is called a middleware.
  + ALL: application logic layer.

ми«оррди • 
.I>RWI эВо-ј 
[1011Pluasajd 
јиачэ •.DST 

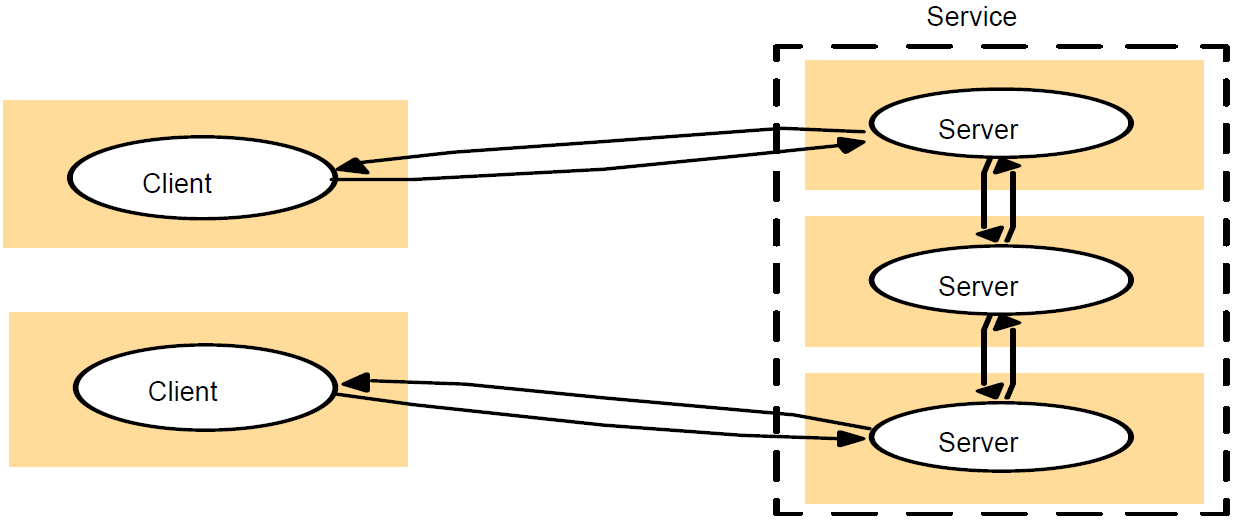
* Data layer resulted in better interfaces.
* N tier: many distributed systems (3-tier) interacting.
* Service layers in a distributed system
  + Hardware + network + OS = platform.
  + Middleware:
    - Mask heterogeneity.
    - Programming model.
    - Provides building blocks.

Architectural models

* Architectural models of distributed systems
  + Functions of components.
  + The placement of components across a network.
  + Inter-relationships between components.
* The client-server model:
  + Server: *process* that accepts to perform a service and *responds* accordingly.
  + Client: invokes services (*remote invocation*).
  + Client **object** invoke a method upon a server object.
  + **Server may be a client of other servers.**



* Multiple servers:
  + Partition services (web servers).
  + Replication for performance and fault tolerant.



* Proxy servers and caches:
  + Cache: a store of recently used data objects that is closer than the objects themselves.
  + Proxy server: a shared cache of web resources for the client machine at a site or across sites.
    - Access remote web servers through a firewall.

Client 
proxy 
Server 
Web 
server 
Web 
Server 

* Peer processes:
  + All processors play similar roles, and cooperate as peers to perform a distributed activity.
  + Reduce server bottleneck, but has consistency and synchronization issues.

Application 
ble 
peers 5 N 
peer 4 
Application 

Variations of the client-server model

* Variations: use mobile code, for low cost computers with limited hardware resources, and need to add or remove mobile devices.
* Applets: mobile code example:

a) client request results in the downloading of applet code 
Client 
b) client interacts with the applet 
Client 
Applet code 
Applet 
Web 
Web 

* Thin clients: limited hardware resources:

Network computer or PC 
.hi 
network 
Compute server 
Ap lication 

Design requirements for DSs

* Judging how good the architecture is: performance, quality of service, dependability.
* Performance: responsiveness, throughput, load balancing.
* Quality of Service (QoS):
  + Non-functional properties experienced by users:
    - Deadline properties: hard / soft deadlines.
  + Adaptability: adapt to changing system configuration.
* Dependability: correctness, fault-tolerance, security.

Fundamental models

* Questions:
  + What's the main entity in the system?
  + How do they interact?
  + What are the characteristics that affect their individual and collective behavior?
* Purpose: specify assumptions, make generalizations.
* Interaction model: distributed algorithms - including communication, important factors, synchronous.
* Event ordering: considering a mail system:

计算机生成了可选文字:
send 
x 
1 
ml 
2 
receive 
z 
send 
3 
receive 
m 
2 
receive 
receive 
4 
send 
m 
3 
receive 
receive 
physical 
time 
m 
2 
receive 

* Failures: omission, arbitrary, and timing failures.

计算机生成了可选文字:
process p 
send 
m 
Communication channel 
Outgoing message buffer 
process q 
receive 
Incoming message buffer 

* Omission and arbitrary failures:

计算机生成了可选文字:
Class of.failure 
Fail-stop 
Crash 
Omission 
Send-omission 
Affects 
Process 
Process 
Channel 
Process 
Receive-omission Process 
Description 
Process halts and remains halted. Other processes may 
detect this state. 
Process halts and remains halted. Other processes may 
not be able to detect this state. 
A message inserted in an outgoing message buffer never 
arrives at the other end's incoming message buffer. 
A process completes asend, but the message is not put 
in its outgoing message buffer. 
A message is put in a process's incoming message 
buffer, but that process does not receive it. 
Arbitrary 
(Byzantine) 
Process or Process/channel exhibits arbitrary behaviour: it may 
channel 
send/transmit arbitrary messages at arbitrary times, 
commit omissions; a process may stop or take an 
incorrect step. 

* Timing failures:

计算机生成了可选文字:
Class of Failure 
Clock 
Performance 
Performance 
Affects 
Process 
Process 
Channel 
Description 
Process's local clock exceeds the bounds on its 
rate of drift from real time. 
Process exceeds the bounds on the interval 
between two steps. 
A message's transmission takes longer than the 
stated bound. 

**CUE COLUMN**

**SUMMARIES**

1. An example: e-Business application.
2. Architectural models.
3. Variations of the client-server model.
4. Design requirements for DSs.
5. Fundamental models.

Chapter3. Interprocess Communication

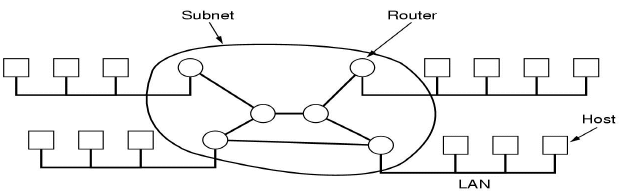
2019年3月13日

8:14

**NOTE TAKING AREA**

Network architecture

* Definition of network architecture: framework for designing and implementing networks.
* Components: **software** - protocols and services, **hardware** - transmission technology, media and devices, scale of LANs, MANs, and WANs, topology.
* Subnet: a part of the whole network,



Layered architecture:

* *The purpose of each layer is to* ***offer a communication services to higher level layers****.*
* Interfaces in each layer:
  + **peer-to-peer interface** - defines the form and types of messages exchanged between peers (indirect communication).
  + **service interface** (defines the primitives (operations) that a layer provides to the layer above it.
  + Layering is non-linear.
* Protocols: the functionality encapsulated within each layer.
  + Refers to **interfaces** and the **objects that implement those interfaces**.
  + Protocol and service: service is the set of **primitives** provided to the higher layer, while protocol **implementation these primitives**.
* Internet architecture includes ISO OSI architecture, TCP/IP architecture. *I'll escape this part that you can find the knowledge here from computer network lesson.*

Interprocess communication

* Message passing model: send - receive.
  + Synchronous communications: block and asynchronous communications: blocking (receive) and non-blocking (send, receive).
* Reliability: validity (guarantee messages delivered and doesn't lost), integrity (message arrives uncorrupted and without duplication).
* Sockets and ports:
  + *Sockets provide an abstract endpoint for inter-process communication (UDP, TCP).*
  + Destination: internet address (names rather than numbers), port (within a computer).
  + Socket must be bound to a **local port** and an **internet address**.
  + A process may **use multiple ports** but **cannot share ports**.

agreed ров 
апу рой 
message 
R---- 
other ports 
зетчет 
р = 138.37.38.249 
р 138.37.94248 

Remote object reference

* A valid identifier for an object throughout the distributed system should be **unique**, and be **passed as argument**, with **external representation**.

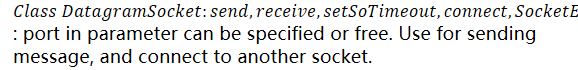
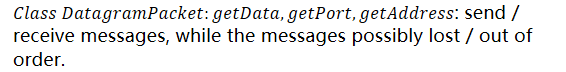
blts 
Intemet address 
number 
blts 
time 
32 bits 
object number 
interface of 
remote object 

* Request-reply protocol

Cllent 
doOperation 
(wait) 
tinuation) 
Request 
message 
message 
Server 
getRequest 
select object 
execute 
method 
send Reply 

* Issues of this protocol:
  + Message identifiers (sequence and sender id).
  + Failure model such as timeouts, lost reply messages.
  + Duplicate request messages.
  + History for retransmissions (log).
* UDP sockets: receive method returns the IP address + port of sender so that a reply can be sent.
  + Message size: specifies a byte-array to receive the message.
  + Non-blocking send & blocking receive (timeouts and threads for deadlocks)
* TCP sockets: *provides an abstraction of a stream of bytes to which data write and read. It hides the following:*
  + Message sizes, lost messages, flow control, message duplication and ordering, message destinations.
  + Connection: client-server model.
    - Client: creates a stream socket bound to a port and asks for a connection to a server port.
    - Server: creates a listening socket bound a port and waits to accept connect requests.
  + Each socket is **both for input and output**.

Java API for network transmission

* Java API for Internet address:
  + C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image022.png
  + C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image023.png
  + C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image024.png
* Java API for UDP sockets:
  + C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image025.png
  + 
* Java API for Datagrams
  + 

C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image028.png

* Java API for TCP socket:
  + C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image029.png
  + C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image030.png

Group communication

* Multicast: sends a single message **from one process to each of the members of a group of processes**.
  + Fault tolerance based on **replicated services**.
  + Finding discovery servers in spontaneous networking.
  + Propagation of event notifications.
* IP multicast: multicast group is specified by a class D Internet address.
  + Membership is **dynamic**, to join make a socket.
  + Programs using multicast use **UDP** and send datagrams to multicast addresses and (ordinary) port.

**CUE COLUMN**

TCP Java API example

* TCP client connection, sending request and receiving reply:

蝕 と 能 cl の 」 瓦 0 ″ ー 
” 房 な 能 1 ℃ , イ 物 を 鰓 
〃 0 鵬 ー 礪 な 舅 p 川 」 0 を 00W イ 0 」 ma 00 / イ “ ′ 用 0 わ 0 〃 
S な = 第 
ー - 宿 ewS 卩 1 第 、 
ハ 1 0 リ や 店 000E0 = 
Ⅲ 禰 し ” 朝 g 可 の え 
物 gda 第 掫 尾 0 北 丁 を 0 
〃 し TF な 0 」 0 川 を “ rod 物 g 舅 0 43 
引 em 。 " ゆ 回 “ 朝 : 、 朝 切 - 
em の ゆ , 物 ′ 煽 に 「 を 、 ge Ⅳ き 0 え 
お 。 朝 店 OF “ 加 n の ′ em 。 第 物 「 を 0 を 物 + e.g リ “ 。 - 
お 。 朝 な 0 を に 挈 ″ 。 ” の 向 。 川 。 第 物 ⅱ 第 ツ 0 二 g 。 き 衂 メ 
新 聞 イ 産 ! = ” 砌 れ 1 イ “ 求 。 40 「 挈 加 叫 em 。 設 ゆ 「 第 、 に を け M 町 。 の 

* TCP server make connections and echoes requests:

import java. net. g; 
importjava. to. 
public class TCPSener { 
public stanc void main (Swing argsÜ) { 
int serverport 7800; 
SenerSocket listenSocket neu• Saversocket(serverport); 
while (true) { 
Socket clientSocket = listensocker.accepr(); 
Connection c = new Connection(clienrsocker); 
} catch(IOExceprion e) : "+e.getMessage());} 

* Continued:

class Connection extends Thread 
DaraInpurSnvam in; 
Dara OurpurSrream out; 
Socket clien 'Socket; 
public Connection (Socket aCIienrSocker) 
try { 
out —new DataOutputStream( clientSocketgetOutputStreamO); 
this.startO; 
catch(IOExceprion e) "+egetMessageO);} 
public void 
// an echo server 
catch(EOFExcephon e) {System. 
finally{ try (IOException 

**SUMMARIES**

1. Network architecture.
2. Layered architecture, protocols.
3. Interprocess communication, sockets.
4. Remote object reference.
5. Java API for network transmission.
6. Group communication.

Chapter4. Distributed Objects and Remote Invocation

2019年3月20日

8:06

**NOTE TAKING AREA**

Distributed applications programming

* Distributed objects model:
  + Middle layers: **RMI**, **RPC**, and **events**. As well as **request reply protocol**, and **external data representation**.
  + Objects: data (attributes) + operations (method).

Implementauon 
of methods 
interface 
rn2 
0b -ect 
data 
mplementatJ0n 
of methods 

* State of objects: value of its attributes. (Encapsulating)
* Interact via interfaces: define types of arguments and exceptions of methods.
* The object (local) model:
  + OO programs: a collection of interacting objects.
  + Distributed object model:

remote 
invocation 
c 
invocation 
invocat On 
local 
invocation 
remote 
I nvocauon 

* Objects extend with remote objects and methods.
* Remote object reference: in distributed system, must be unique in **space** and **time**, error returned if accessing a deleted object, and can allow relocation.
* Remote interfaces: **Implements the methods** of its remote interface.
  + Interfaces specify externally accessed.
  + Parameters contain input, output, or both, instead of call by value, call by reference.
  + No pointer and constructors.

remote 
Data 
Object 
remote 
interface 
ml 
m3 
local 
rn4 
I mplementation 
of methods 

* CORBA: interface definition language (IDL).
* Java RMI: as other interfaces, keyword Remote.
* RMI, invocation semantics:
  + Handling remote objects: **exceptions** and **garbage collection**.
  + Design issue for RMI: **invocation semantics**, maybe **invocation not guaranteed**, at least once **result or exception Sun RPC**, At most once **Java and CORBA**.

Fault tolerance measures 
Retransmit request 
message 
No 
Yes 
Yes 
Du licate 
{ering 
Not applicable 
No 
Yes 
Re-execute pmcedumv 
or retransnjit reply 
Not applicable 
Re-execute procedure 
Retransmit reply 
Invocation 
semantics 
Maybe 
At-least-once 
A t-mos t-once 

* Implementation of RMI:

client 
object A proxy for 
u est 
Reply 
server 
off/t 
S e etorv 
& dispatcher 
for class 
sevvant 
Remote Communication Communication Remote r f rence 
reference module module 
module 

* Application-level object A invokes a remote method in application-level object B.
* Communication model: implements the **request-reply protocol**.
* Proxies: transparent to clients by behaving like a local object to the invoker. It sends a message.
* Remote reference module: creates remote object references and proxies.
  + Remote object table: entries for all **remote objects held by the process** and **entries for all local proxies**.
* Servants: provides the body of a remote object.
* Dispatcher: receives the request from communication module, and use the method ID to select the appropriate method in the skeleton.
* Skeleton: Implements the methods in the remote interface, and unmarshalls the arguments in the request and invokes the corresponding method in the servant.
* Binding and activation:
  + Binder: mapping from name to remote object references, lookup.
  + Activation: Objects active and passive, create new instance of class and initialize from stored state.
  + Activator: records location of passive and active objects, and starts server processes and activates objects within them.
* Object location issues: persistent object stores (cf Persistent Java), object migration (use remote object reference and address), and location service (assists in locating objects, maps remote object references to probable locations).
* Remote Procedure Call (RPC):

client 
program 
client process 
Re uest 
Reply 
client stub 
rocedure 
Communication 
module 
Communication 
module 
server process 
server stub 
procedure 
dispatcher 
service 
procedure 

* Events and notifications:

Event service 
Object Of interest 
Object Of interest 
observer 
2. 
object of interest 
3. 
notification 
observer 
subscriber 
notification 
subscriber 
notification 
subscriber 
notification 

* Java RMI: extends the java object model to provide support for distributed objects.

Products

* Java RMI, CORBA, DCOM
* Sun RPC

**CUE COLUMN**

Example: Shared whiteboard

* C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image041.png

import java. rmi. 
import java. util. Vector; 
public interface Shape extends Remote { 
int getVersion() throws RemoteException; 
GraphicalObject getAllState() throws RemoteException; 
public interface ShapeList extends Remote { 
Shape newShape(GraphicalObject g) throws RemoteException; 
2 
Vector allShapesd) throws RemoteException; 
int getVersion() throws RemoteException; 

* C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image043.png

void rebind (String name, Remote obj) 
This method is used by a server to register the identifier of a remote object by name, 
as shown in Figure 1 , line 2. 
void bind (String name, Remote obj) 
This method can altematively be used by a selver to register a remote object by name, 
but if the name is already bound to a remote object reference an exception is thrown. 
void unbind (String name, Remote obj) 
This method removes a binding. 
Remote lookup(String name) 
This method is used by clients to look up a remote object by name, as shown in Figure 
3 line l. A remote object reference is retumed. 
String [1 list() 
This method retums an anay of Stlings containing the names bound in the regisüy. 

* C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image045.png

import java.rmi. , 
public class ShapeLis tServer{ 
public static void main(String 
System. setSecurityManager(new RMISecurityManager()) ; 
Shapelist aShapeIist — nav ShapeListServant(); 
Naming. rebind/ "Shape List", aShapeList ); 
System. out.println("ShapeList server ready 
System. sener main " + e.getMessage());} 
1 

* C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image047.png

inportjava. rmi. * 
import java. rmi. server. UnicastRemoteObject; 
in port java. util. Vector; 
public class ShapeListServant extends UnicastRemoteObject implements ShapeList { 
private Vector theList; 
// contains the list of Shapes 
private int version; 
public ShapeListServant()throws RemoteException{...} 
public Shape newShape(GraphicalObject g) throws RemoteException { 
version + +,. 
Shape s = net" ShapeServant( g, version); 
theList.addElement(s); 
return s; 
public Vector RemoteException{...} 
public int getVersion() throws RemoteException { } 
3 

* C:\AE79CE85\EF778A76-A16D-4094-A219-213C20E5F223.files\image049.png

import java. rmi. , 
import java. rmi .server * , 
import java. util. Vector; 
public class ShapeListClient{ 
public static void main(String 
System. setSecurityManager(new RWSecurityManager()) ; 
ShapeList aShapeList — null; 
try { 
a.ShapeList = (ShapeList) Naming. ; 
Vector sList = aShapeList. allShapes(); 
} catch(RemoteException e) {System. 
}catch(Exception e) {System.out.println("Client: " + e.getMessage());} 

* The Classes supporting Java RMI:

RemoteObject 
RemoteServer 
Activatable 
UnicastRemoteObject 
<servant class> 

**SUMMARIES**

1. Distributed application programming, RMI and RPC.
2. Products of remote invoke.

Chapter5. Indirect Communication

2019年3月24日

9:43

**NOTE TAKING AREA**

Space and time coupling

Time-coupled 
Properties: Communication directed 
towards a given receiver or receivers; 
receiver(s) must exist at that moment in 
Space coupling 
Examples: Message passing, remote 
invocation (see Chapters 4 and 5) 
Properties: Sender does not need to 
know the identity of the receiver(s); 
receiver(s) must exist at that moment in 
Space uncoupling 
Examples: IP multicast (see Chapter 4) 
Time-uncoupled 
Properties: Communication 
towards a given receiver or receivers; 
sender(s) and receiver(s) can have 
independent lifetimes 
Examples: See Exercise 15.3 
Properties: Sender does not need to know 
the identity of the receiver(s); sender(s) 
and receiver(s) can have independent 
life times 
Examples: Most indirect communication 
paradigms covered in this chapter 

* Space: whether sender knows receiver.
* Time: whether receiver exist at that moment or not.
* Open and closed groups: external sender can reach the members of group or not.
* Reliability and ordering:
  + Integrity, validity, and agreement.
  + Ordering methods: FIFO, causal, and total.
  + Group management: interface and notifying for membership changes, failure detection, and performing group address expansion.
    - Role of group membership management:

Group 
address 
expansion 
Grou 
send 
M ulticast 
communication 
process group 
Leave 
Fail 
Join 
Group membership 
management 

Design ideas of architecture

* The message queue paradigm:

Producers 
Message queue system 
Consumers 

* Message queue as a mediate between producers and consumers.
* The distributed shared memory abstraction:

Process 
accessing DSM 
Physical 
memory 
Distributed shared memory 
Mapping' 
Physical 
memory 
DSM appears as 
memory in address 
space of process 
Physical 
memory 

* The tuple space abstraction:

加 。 砿 朝 の ノ 
4 ( ア 施 心 W 艙 5 ・ 2 り み ノ 
( 物 グ な ・ ル を を 、 朝 ゝ 
( グ 翩 : 〃 始 宿 の 立 ゝ 
く 0 な が : 房 な : 引 石 8 獺 の 
《 房 び ん : 6 れ ク の 

**CUE COLUMN**

Example: architecture of Jgroups

* The flow chart as follow:

Applications 
Building 
blocks 
Channel 
CAUSA 
GMS 
F RAG 
UDP 
Protocol stack 

* Class of FireAlarmJG:

import org.jgroups.JChannel; 
public class FireAlarmJG { 
public void raised) { 
try { 
JChannel channel = new JChannel(); 
channel. 
Message msg new Message(null, null, 'Eire!"); 
channel. send(msg); 
catch(Exception e) { 

* Class of FireAlarmConsumerJG:

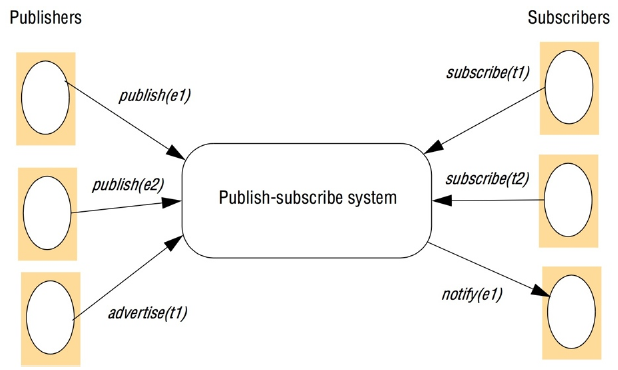
import org.jgmups.JChannel; 
public class Fire„41armConsumerJG { 
public String await() { 
fry { 
JChannel channel = new JChannel(); 
Message msg — (Message) channel. receive"); 
meturn (String) msg. GetObject(); 
} catch(Exception e) { 
mturn null; 

Some other examples

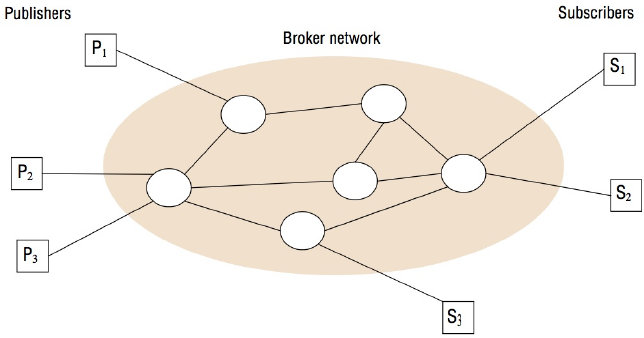
* Dealing room system:

Dealer s computer 
computer 
Dealer 
External 
source 
Intorma n 
provo 
Information 
provider 
Dealers computer 
Dealers computer 
Dea er 

* The publish-subscribe paradigm:



* Implementation issues:



* Flow chart of architecture of publish-subscribe systems:

— Publish-subscribe architecture — — — 
Matching 
Event routing 
Overlay networks 
Network protocols 

* Filtering-based routing:

upon receive publish(event e) from node xl 
matchlist match(e, subscriptions)? 
send notify(e) to matchlist; 
3 
.fivdlist : = match(e, routing); 4 
send publish(e) to fwdlist - x; 5 
upon receive subscribe(subscription s) from node x6 
if x is client then 
7 
add x to subscriptions; 8 
else add(x, s) to routing; 9 
send subscribe(s) to neighbours - x; 10 

* Rendezvous-based routing:

upon receive publish (event e) from node x at node i 
rvlist := EN(e); 
ili in n'list then begin 
matchlist : =match(e, subscriptions); 
send notify(e) to matchlist; 
end 
send publish(e) to rvlist - i; 
upon receive subscribe(subscription s) from node x at node i 
rvlist SN(s); 
ili in n'list then 
add s to subscriptions; 
send subscribe(s) to rvlist - i; 

* Example publish-subscribe system:

System (and further reading) 
CORBA Event Service (Chapter 8) 
TIB Rendezvouz [Oki et al. 19931 
Scribe [Castro et al. 2002b] 
TERA [Baldoni et al. 2007] 
Siena [Car-zaniga et al. 2001] 
Gryphon 
Hermes [Pietzuch and Bacon 2002] 
MEDYM [Cao and Singh 20051 
Meghdoot [Gupta et al. 2004] 
Structure-less CBR [Baldoni et al. 2005] 
Subscription 
Channel-based 
T opic-based 
Topic-based 
Topic-based 
Content-based 
Content-based 
Topic- and 
content-based 
Content-based 
Content-based 
Content-based 
Distribution 
model 
Centralized 
Distributed 
Peer-to-peer 
(DHT) 
Peer-to-peer 
Distributed 
Distributed 
Distributed 
Peer-to-peer 
Peer-to-peer 
Event routing 
Ffiltering 
Rendezvous 
Inforrned gossip 
Filtering 
Filtering 
Rendezvous and 
filtering 
Flooding 
Rendezvous 
Informed gossip 

Example of message queue

* The networked topology in WebSphere MQ:

Queue rnanager 
Client 
Proxy 
Stub 
Client channel 
Services 

* Programming model by JMS:

Message 
producer 
Sends to 
Destination 
Topic 
Queue 
Connection factory 
Connection 
Message 
Message 
consumer 
Receives from 
Destination 
Topic 
Queue 

* Class of FireAlarmJMS:

tmporrjavax.jms. *; 
import javax. nam ins. 
public class Fire-4/armJMS 
public void raise() { 
Context cm = new InitialConrexr(); 2 
Topic ConnectionFactory ropicFacrory — 3 
(TopicConnecrionFacrory)cmlookup ("TopicConnectionFactory"),• 4 
Topic topic 5 
TopicConnection topic-Conn — 
ropicConnecrionFacropy. creareTopicConnecrion(); 7 
TopicSession topicSess — topicConn. createTopicSessionfalse, 8 
Sessior,AUTO ACKNOWLEDGE); 
TopicPublisher ropicPub = roptcSess. crearepublisher(topic); 10; 
TerrMessage msg = ropicSess.oeateTerrMessageO; 11 
msg. M); 12 
topicpub.publish(message) ; 13 
} catch (Exception e) { 14 
} 15 

* Class of FireAlarmConsumerJMS:

import javax.jms.*; import javax.naming. 
public class FireÅlannConsumerJMS 
public String await() { 
Context ctx = new InitialContett(),• 
2 
TopicConnectionFactory topicFactory = 3 
(TopicConnectionFactory)ctx. 4 
Topic topic = 5 
Topic Connection topicConn — 
6 
topic ConnectionFac teTopicConnection O; 
7 
TopicSession topicSess = topicConn. cwateTopicSession(false, 
Session -4 UTO_ACKVOWLEDGE); 
9 
Topicsubscriber topicSub = topicsess.createsubscriber(topic); 10 
topicSub.start(); 11 
TextMessage msg = (TextMessage) topicSub.receive();12 
return msg.getText(); 13 
} catch (Exception e) { 14 
return null; 15 

API of JavaSpaces

Operation 
Lease write(Entry e, Transaction txn, long leave) 
Entry read(Entry Impl, Transaction t.rn, long timeout) 
Entry readIfExists(EnDy tmpl, Transaction txn, long timeout) 
Entry take(Entry tmpl, Transaction txn, long timeout) 
Entry takeUErists(Entry tmpl, Transaction txn, long timeout) 
EventRegistration notify(Entry tmpl, Transaction txn, 
RemoteEventListener listen, long lease, 
MarshalledObject handback) 
Effect 
Places an entry into a particular 
JavaSpace 
Returns a copy of an entry matching 
a specified template 
As above, but not blocking 
Retrieves (and removes) an entry 
matching a specified template 
As above, but not blocking 
Notifies a process if a tuple matching 
a template is written to a 
JavaSpace 

* Class of AlarmTupleJS:

import net.jini. core. entry. 
public class Alarm TupleJS implements Entry { 
public String alarm Type; 
public Alarm TupleJS() { } 
public Alarm TupleJS(String alarm Type) { 
this.alarmType = alarmType;} 

* Class of FireAlarmJS:

import net.jini.space.JavaSpace; 
public class FireAlarmJS { 
public void raised) { 
JavaSpace space = "9; 
Alarm TupleJS tuple — Alarm 
space.MTite(tuple, null, 
catch (Exception e) 

* Class of FireAlarmReceiverJS:

计算机生成了可选文字:
import net.jini.space.JavaSpace; 
public class FireAlannConsumerJS { 
public Strmg await() { 
fry { 
JavaSpace space SpaceAccessor.findSpace(); 
AlannTupleJS template new "); 
AlannTupleJS recvd (Alarm TupleJS) space.read(template, null, 
Long.M4X VALUE); 
return recvd.alarmType; 
catch (Exception e) { 
return null; 

Summary of indirect communication styles

计算机生成了可选文字:
Space- 
uncoupled 
Time- uncoupled 
Style of service 
Communication 
Main intent 
Sca lability 
Associative 
Groups 
Yes 
possible 
Communicatim- 
1 -to-many 
Reliable 
dis tributed 
computing 
Limited 
No 
publish- 
subscribe systems 
Yes 
Possible 
Commtmicatim- 
I -to-many 
Information 
dissemination or 
EAT; mobile and 
ubiquitous 
systems 
Possible 
Content-based 
publish-subscrü'e 
only 
Message queues 
Yes 
Yes 
C«nmunicatim- 
dissemination or 
EAI; 
commercial 
processmg 
Possible 
No 
DSM 
Yes 
State-based 
1 -to-many 
Parallel and 
distributed 
computation 
No 
Tuple spaces 
State-based 
1-1 or I-to-many 
Parallel and 
distributed 
computation; 
mobile and 
ubiquitous 
systems 
Yes 

**SUMMARIES**

1. Space and time coupling, reliability and ordering.
2. Design ideas of architecture: message queue and distributed spaces.

Chapter6. Operating System Support

2019年4月2日

10:22

**NOTE TAKING AREA**

Middleware layers and system layers

* Middleware: RMI, RPC, and events. Request reply protocol, and external data representation.
  + On the bottom of applications and on the top of operating system.
* System layers: OS of kernel, libraries and servers, with **several nodes** and platform.
  + I/O management, memory management, processes, multiuser, file ownership, security, communication, and protection of processes (kernel).
* Core OS functionality:
  + Process manager: creates processes = thread + space.
  + Communication manager: sockets.
  + Thread manager: creates threads, synchronisation, and scheduling.
  + Memory manager: RAM, disk allocation.
  + Supervisor: hardware abstraction (interrupts, exceptions, caches).

Program, process, thread

* Process: a program that is currently executing, **program ≠ process**.
* Thread (lightweight process): OS abstraction of an **activity**.
* Process: **execution environment** + one or more **thread**.
  + Execution environment: threads in the same process share the same execution environment.
  + Consists of an address space, thread synchronisation mechanism, communication interface (socket), and high level resources (file and window).
* Address space: unit of virtual memory, contains one or more **regions**, text, stack, heap.

计算机生成了可选文字:
N=32 or 64 
2N 
Auxiliary 
regions 
Stack 
Heap 
opposite 
direction 

* Processes and threads:
  + Processes: historically first abstraction of single thread of activity, and can run concurrently, CPU sharing if single CPU need own execution environment.
  + Threads (lightweight processes): can share execution environment, and can be created / destroyed dynamically.

Threads in client server model

* Distributed system: using threads reduce waiting time, for remote invocations (blocking of invoker), for disk access (unless caching), and obtain better speed up with threads.

seuejouob 
Induno-lndul 
speenu N 
la,uas OJ s 
Septu Z peaxu 

* Threads with clients: separate (data production, RMI calls to server), pass data via buffer, run concurrently, and improved speed, throughput.

Thread I 
Item 2 & 3 
Thread 2 
blocked 

Multi-threaded server architectures

* Worker pool: fixed pool of worker threads, size does not change, can accommodate priorities but inflexible.
* Other architecture: thread-per-request, thread-per-connection, and thread-per-object.
  + Thread-per-request:

workers 
remote 

* Spawns new worker for each request, worker destroys when finished.
* Allows max throughput: no queuing and no I/O delays.
* But overhead of creation and destruction high.
* Thread-per-connection:

s 丨 GO 
; 0 山 a 亅 
LIO 刂 0 .10 〕 - d 

* Create new thread for each connection for multiple requests, destroy thread on close.
* Lower o/heads.
* But unbalanced load.
* Thread-per-object:

spe(qo 
0 山 亅 
speaJLß pa(qo-Jad 

* As per-connection, but new thread created for each object.
* Physical parallelism: multi-processor machines (cf casper, SoCS file server).

**CUE COLUMN**

**SUMMARIES**

1. Middleware layers and system layers.
2. Program, process, thread.
3. Threads in client-server model.
4. Multi-threaded server architecture.