Introduction to Java Enterprise Development

- Supporting concepts
- Enterprise technologies
- Creating an enterprise application with Java

Supporting concepts

- HTTP
- Threads
- Java
- Specialized servers
- Basic enterprise design
- Enterprise technologies
- Security
Basic enterprise design

- Two-tier
- Three-tier
- N-tier

Two-tier applications

- Connect client to Web server or database
Three-tier applications

- Allow programmer to leverage other resources
- DB for storing info
- Application server to process info

Three-tier applications: some designs
N-tier applications

- Enterprise app’s residing on client & several servers

Technology choices

- Previous examples: basic designs
- Before creating enterprise app’s
  - Need to decide
    - Technologies used
    - Computers running them
Security

- Java security model
- Identify people: UserID / password
- Groups / roles
- Role: e.g., manager ==> same permissions
- Permissions concept
- Access Control List (ACL)

Enterprise technologies

- JDBC (Java Database Connectivity)
- JNDI (Java Naming and Directory)
- Servlets
- Server-Side Scripting
- JavaServer Pages
- Distributed Objects and RMI (Remote Message Invocation)
- Enterprise JavaBeans
- Java Messaging Service
JDBC (Java Database Connectivity)

- Libraries (enabling technology)
- Used to connect Java program to data source
- JDBC 2.0 extends model from relational databases to other types

JNDI (Java Naming and Directory)

- Enabling tech
- Provide access to objects
  - Enterprise beans
  - JMS destinations
- Other technologies rely on JNDI
Servlets

- Extend Web servers
  - 1. Replace CGI for server-side processing of HTTP requests
  - 2. Extend dynamic Web pages using SSI or JSPs
  - 3. Form a gateway between clients and other services
  - 4. Generic interface for services

JavaServer Pages

- Create dynamic Web pages
- Java as scripting language
- Mostly where Web page changes with each request
Distributed Objects and RMI (Remote Message Invocation)

- Enabling tech
- App’s & lib’s
  - Use RMI to form network connections
- Provide transparent link between Java app’s

Enterprise JavaBeans (EJB)

- Standard building blocks for corp. server app’s
- Tested services from server provider
- ==> Focus resources on app-specific programming
- Component design ==> good to encapsulate business rules and processes
- Foundation for standard services
Java Messaging Service

- Act as front end to messaging providers
- ==> Objectify communications between applications

Creating enterprise application w/Java

- Define project & goal
- Analyze requirements
- Design set of components & objects to fulfill requirements
- Determine communication mechanism components need to interact
- Determine components to implement
- Write (and test) client app’s & comp’s
Define project & goal

- Functionality across multiple computers / domains
- No particular platform at this point
- Focus on what entire app needs to do
- Separate functionality in later steps
- \(\Rightarrow\) List of requirements

Analyze

- Design set of comp’s & obj’s to fulfill req’s
- Thinking: “application boundaries”
- Main goal: determine set of client app’s & components that support them
- Focus: lib’s client might use to comm. w/ supporting components
  - No design of client app’s or user interface
  - Exception: Web pages as interface to app on server
Determine comm. mech. between components

- e.g.,
  - HTTP
  - RMI
  - CORBA
  - Sockets
  - JMS

HTTP

- For app’s using Web as front end
- Also useful where components are separated by firewall
RMI, CORBA, sockets

- Sockets: only to support existing protocols or app’s
- RMI, CORBA: distributed objects concept enable interapp comm.
- RMI: most native distributed object choice for app’s written in pure Java
- CORBA (not covered here): connect programs written in other languages

JMS

- Allows programs to communicate via encapsulated messages
- Messages can be sent
  - Between Java prog’s
  - To non-Java prog’s (depending on provider)
Determine what type components to implement

- JSP
- Servlets
- Enterprise JavaBeans
- RMI
- JMS

JSP & Servlets:

- Drive Web page user interface
- Servlets
  - Also provide generic services over HTTP / other protocol
Enterprise JavaBeans: define comp’s that live on server of some kind

• E.g., application server
  • Use EJB ==> leverage services provided by server
    • E.g., transaction management & resource optimization

RMI & JMS

• Can be used w/ EJBs or servlets & applets or app’s
• E.g., can create app providing services by receiving JMS messages and sending JMS messages for reply
Final step

• Write client app’s and components
• Test them

Why use Java for enterprise development?

• Portable
• Standards def’d by enterprise Java tech reduce education costs
• Tech’s like EJB & servlets split app into components and centralize code
• RMI, JNDI, & JMS ==> easy to manage relationships between components
• More ...
Why Java (cont.)

• Leverage technologies to break code into manageable components
  • During development
  • Once running
• Java creating “safety zone” around each enterprise app to protect it from “bad neighbors”