Architecture of Enterprise Applications I Overview

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

Roadmap

- Database Access
  - JDBC reading
  - O/R Mapping
  - Caching
  - Transaction
  - Non-relation DB
- Messaging and Asynchronous Communication
- Security
- MVC
  - Struts and Spring

- Web Services
- SOA
- Web 2.0
- Architectural Patterns

Roadmap

- Introduction to architecture of enterprise apps 2hrs
  - Definition and content of architecture
  - Qualification of architects
  - Features of Arch. Of Enterprise Apps.

- Key technique of Arch. Of Enterprise Apps 10hrs
  - Access control: SSO, cross domain auth., CAS
  - Performance: Memcached, Cluster, Reverse proxy
  - Searching: Lucene, Solr and Mongo db
  - SaaS: Software as a Service
  - Multi-tenant arch.
    - Configurable data, functionality, UI, content, layout and workflow
  - Parallel processing
    - MapReduce and Percolator

Roadmap

- Web Services 2hrs
  - Java EE Web Service
  - .NET Web Service
  - RESTful Web Services
- Service Oriented Service 4hrs
  - Basics of SOA
  - ESB
  - SOAD
  - SCA&SDO, JBI, WCF&BizTalk

Roadmap

- Web2.0 & RIA 2hrs
  - Basics of Web2.0 &RIA
  - Typical technique for Web2.0, including Ajax & Flex
  - Web2.0 & SOA

- Arch. Patterns for Enterprise Apps 10hrs
  - Common patterns
  - Performance tuning patterns
  - Offline concurrency patterns
  - Data source patterns
  - Session management patterns

Exams and Scores

- Exams
  - Final exam 40%
  - Quizzes 10%
- Assignments
  - 6*5=30%
  - Essay questions
- Project
  - Refine the design solution of last semester 20%
  - Oral defense in the 9th week

Schedule & Textbook

- Schedule
  - Wednesday and Friday 08:00-09:40 东中院1-200
  - Office hour: Wednesday 14:00-16:00 Software Building 1111

- Textbook
  - No textbook
  - References will be given in classes

Architecture of Enterprise Applications I
Analysis of Final Exam

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

Final Exam

- 在上学期，同学们通过面向对象分析与设计，对供大学学生学习交流的自主学习系统做出了设计方案
  - 整个系统按照课程分频道进行管理
  - 所有同学都可以注册为该系统的用户
  - 注册用户可以选择课程频道
  - 并在该课程频道发布自己的学习资料供他人下载，也可以对他人发布的学习资料进行评论

- 本学期同学们将要完善该系统的设计，并采用企业级信息系统设计与开发技术实现该系统
  - 在设计和构建系统的过程中，大家碰到了很多问题，在解决问题的方法上也产生了很多分歧
  - 请你根据自己的分析，帮助他们处理以下问题

Final Exam

- 在设计数据库结构以及如何访问数据库时，大家产生了意见分歧，请你回答下面的问题，来帮助他们做出设计决策：（15分，每题5分）
  - 当我们决定不使用O/R映射工具而是通过JDBC直接访问数据库时，有同学主张通过使用DataSource而不是DriverManager来获取数据库连接，请问使用DataSource这种方式能够与数据库的哪些方面解耦？
  - DataSource方式移除掉的耦合包括
    - 代码与数据库类型
    - 数据库URL
    - 所使用的Driver类等之间的耦合

Establishing Connection

- To obtain a connection, the application may interact with either:
  - the DriverManager class working with one or more Driver implementations

    OR
  - a DataSource implementation

Establishing Connection

- DriverManager
  - registerDriver and getConnection

```
// Load the driver. This creates an instance of the driver
// and calls the registerDriver method to make acme.db.Driver available to clients.
Class.forName("acme.db.Driver");

// Set up arguments for the call to the getConnection method.
// The sub-protocol "odbc" in the driver URL indicates the
// use of the JDBC-ODBC bridge.
String url = "jdbc:odbc:DSN";
String user = "SomeUser";
String passwd = "SomePwd";

// Get a connection from the first driver in the DriverManager
// list that recognizes the URL "jdbc:odbc:DSN".
Connection con = DriverManager.getConnection(url, user, passwd);
```

Establishing Connection

- DataSource
  - A logical name is mapped to a DataSource object via a naming service that uses the Java Naming and Directory Interface ™ (JNDI).

Property Name	Type	Description
databaseName	String	name of a particular database on a server
dataSourceName	String	a data source name
description	String	description of this data source
networkProtocol	String	network protocol used to communicate with the server
password	String	a database password
portNumber	int	port number where a server is listening for requests
roleName	String	the initial SQL rolename
serverName	String	database server name
user	String	user's account name

Establishing Connection

- DataSource
  - A logical name is mapped to a DataSource object via a naming service that uses the Java Naming and Directory Interface ™ (JNDI).

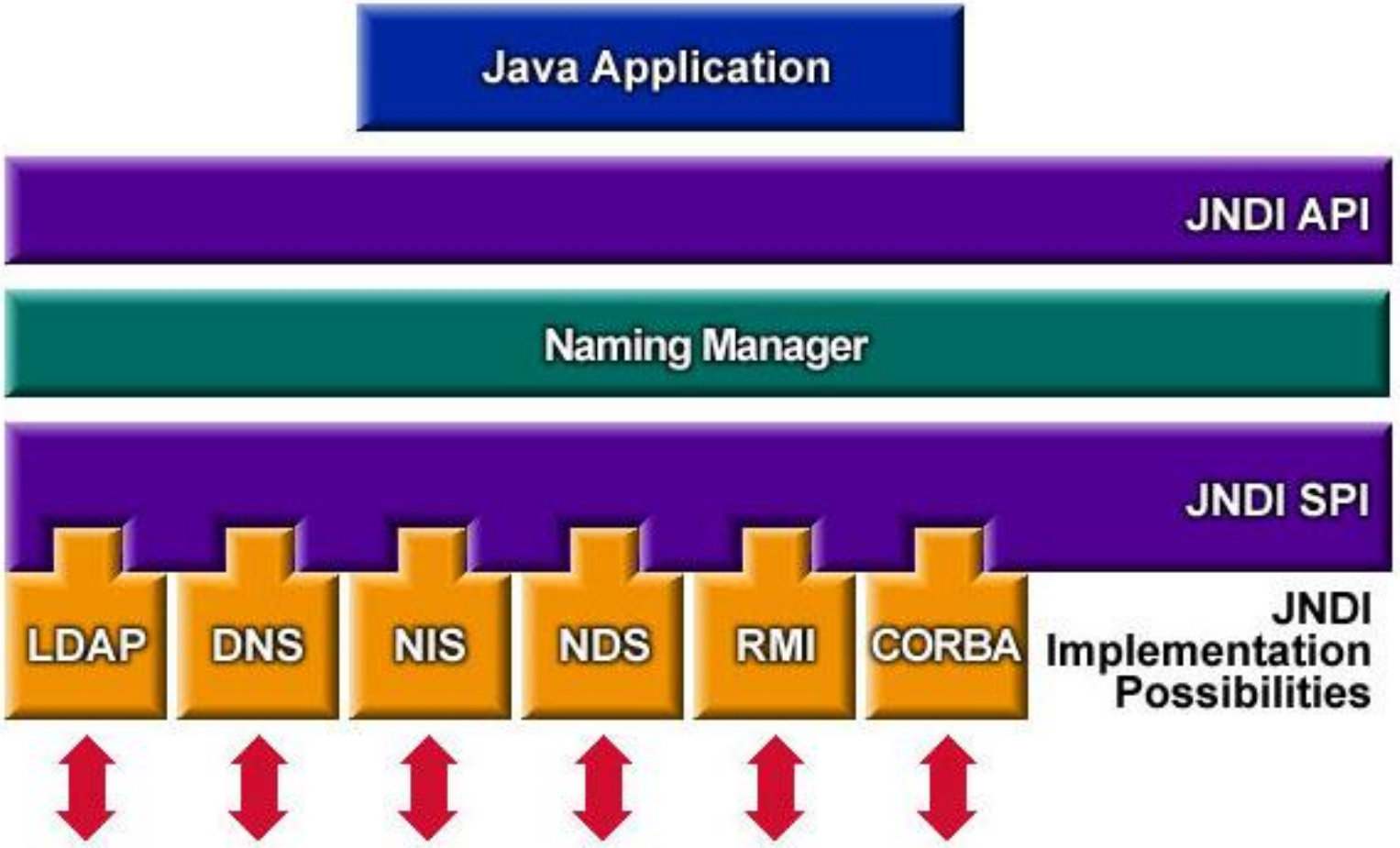
```
// Create a VendorDataSource object and set some properties
VendorDataSource vds = new VendorDataSource();
vds.setServerName("my_database_server");
vds.setDatabaseName("my_database");
vds.setDescription("data source for inventory and personnel");

// Use the JNDI API to register the new VendorDataSource object.
// Reference the root JNDI naming context and then bind the
// logical name "jdbc/AcmeDB" to the new VendorDataSource object.
Context ctx = new InitialContext();
ctx.bind("jdbc/AcmeDB", vds);
```

Establishing Connection

- DataSource
  - A logical name is mapped to a DataSource object via a naming service that uses the Java Naming and Directory Interface ™ (JNDI).

```
// Get the initial JNDI naming context
Context ctx = new InitialContext();
// Get the DataSource object associated with the logical name
// "jdbc/AcmeDB" and use it to obtain a database connection
DataSource ds = (DataSource)ctx.lookup("jdbc/AcmeDB");
Connection con = ds.getConnection("user", "pwd");
```

JNDI


Java Application
JNDI API
Naming Manager
JNDI SPI
LDAP
DNS
NIS
NDS
RMI
CORBA
JNDI
Implementation
Possibilities

Concepts

- Name
  - A people-friendly identifier for identifying an object or a reference to an object.
- Binding
  - The association of an atomic name with an object.
- Context
  - An object whose state is a set of bindings that have distinct atomic names.
- InitialContext
  - The starting point for resolution of names for naming and directory operations.
- ContextFactory
  - A specialization of an object factory. It accepts information about how to create a context, such as a reference, and returns an instance of the context.

Get InitialContext

- Hardcode - Weblogic

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
env.put(Context.PROVIDER_URL,"t3://localhost:7001");
InitialContext ctx = new InitialContext(env);
```

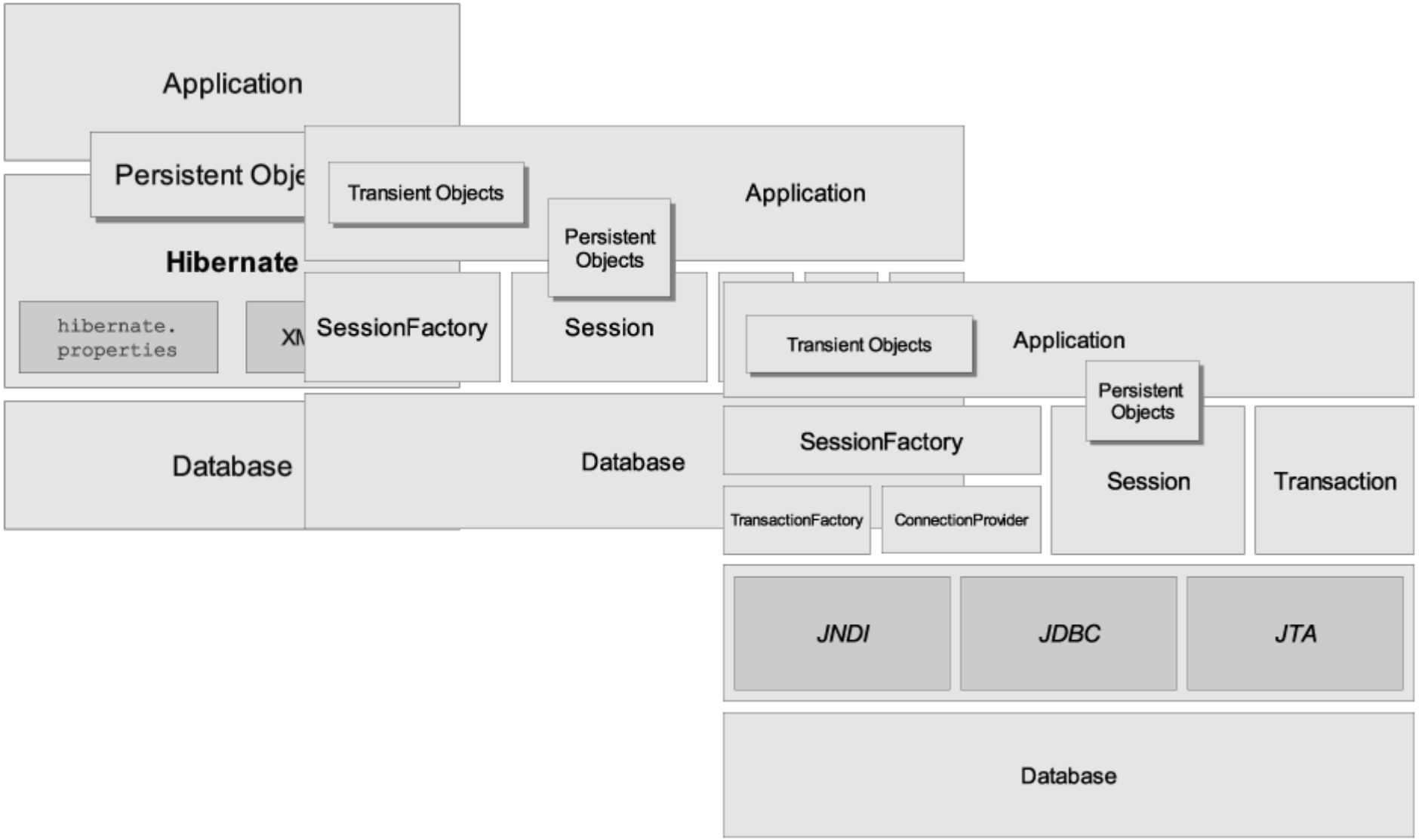
- jndi.properties - JBoss

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
```

Final Exam

- 在设计数据库结构以及如何访问数据库时，大家产生了意见分歧，请你回答下面的问题，来帮助他们做出设计决策：（15分，每题5分）
  - 经过反复论证，我们还是决定使用O/R映射工具来访问数据库，请说明在使用O/R映射工具时，造成应用程序中的实体和数据库中的数据会出现数据不同步问题的本质原因是什么？
  - 因为对实体的操作都是通过EntityManager来实现的，而EntityManager会将对实体的修改都保存在内存中，只有在flush时才会将其存入数据库，在这段时间中，数据就是不同步的。
  - 本质说来，主要是因为实体存在于应用服务器的内存中，而数据库中的数据写在持久性介质上，当对内存中的实体进行操作时，它和持久性介质上的数据就有可能存在不同步的问题。

Architecture



The second level cache

- specify the name of a class that implements org.hibernate.cache.spi.CacheProvider using the property hibernate.cache.provider_class.

Cache	Provider class	Type	Cluster Safe	Query Cache Supported
ConcurrentHashMap (only for testing purpose, in hibernate-testing module)	org.hibernate.testing.cache.CachingRegionFactory	memory		yes
EHCache	org.hibernate.cache.ehcache.EhCacheRegionFactory	memory, disk, transactional, clustered	yes	yes
Infinispan	org.hibernate.cache.infinispan.InfinispanRegionFactory	clustered (ip multicast), transactional	yes (replication or invalidation)	yes (clock sync req.)

Cache mappings

- shared-cache-mode element in your persistence.xmlfile or the javax.persistence.sharedCache.mode property in your configuration:
  - ENABLE_SELECTIVE (Default and recommended value)
  - DISABLE_SELECTIVE
  - ALL
  - NONE

- hibernate.cache.default_cache_concurrency_strategy:
  - read-only
  - read-write
  - nonstrict-read-write
  - transactional

Cache mappings

- **Definition of cache concurrency strategy via @Cache**

```
@Entity
@Cacheable
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public class Forest { ... }
```

- **Caching collections using annotations**

```
@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
@JoinColumn(name="CUST_ID")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public SortedSet<Ticket> getTickets() {
   return tickets;
}
```

Cache mappings

- **Cache Concurrency Strategy Support**

Cache	read-only	nonstrict-read-write	read-write	transactional
ConcurrentHashMap (not intended for production use)	yes	yes	yes	
EHCache	yes	yes	yes	yes
Infinispan	yes			yes

Managing the caches

- **Explcitly evicting a cached instance from the first level cache using Session.evict()**

```
ScrollableResult cats = sess.createQuery("from Cat as cat").scroll();
//a huge result set
while ( cats.next() ) {
  Cat cat = (Cat) cats.get(0);
  doSomethingWithACat(cat);
  sess.evict(cat);
}
```

- **Second-level cache eviction via SessionFactoty.evict() andSessionFacyory.evictCollection()**

```
sessionFactory.evict(Cat.class, catId); //evict a particular Cat
sessionFactory.evict(Cat.class); //evict all Cats
sessionFactory.evictCollection("Cat.kittens", catId);
//evict a particular collection of kittens
sessionFactory.evictCollection("Cat.kittens"); //evict all kitten collections
```

Managing the caches

- **Browsing the second-level cache entries via the Statistics API**

```
Map cacheEntries = sessionFactory.getStatistics()
    .getSecondLevelCacheStatistics(regionName)
    .getEntries();
```

- **Enabling Hibernate statistics**

```
hibernate.generate_statistics true
hibernate.cache.use_structured_entries true
```

Pros and Cons

- JDBC/ODBC reading
  - Advantages:
    - Good performance, especially for accessing massive data
    - Take advantage of various functions provided by DBMS
    - Use stored procedures to implement complex logics
  - Disadvantages:
    - Coupling with DBMS
    - Coupling with data structure
    - Programming is complicate

Pros and Cons

- ORM
  - Advantages:
    - Independent of DBMS
    - Independent of data structure
    - OOP
  - Disadvantages:
    - Impact on performance
    - Can NOT utilize the extra functions in addition to standard functions defined in specification.
    - Mapping between O and R maybe complex.
    - Can NOT invoke stored procedures.

Pros and Cons

- JDBC/ODBC reading or ORM?
  - Requirements Driven
  - Requirement 1:
    - To list all records about courses selection of a certain semester.
    - It could be a big records set with over 150,000 records selected from millions of records
  - Requirement 2:
    - To get the average age of all undergraduate students
  - Requirement3:
    - To ensure that the database could be moved from ORACLE to DB2

Final Exam

- 在设计数据库结构以及如何访问数据库时，大家产生了意见分歧，请你回答下面的问题，来帮助他们做出设计决策：（15分，每题5分）
  - 系统中提供了一个功能，可以对全站的评论信息进行搜索。同学们认为：为了支持这个功能，这些评论信息应该用非关系型数据库存储，并最终选择了Mongo DB。请问，Mongo DB相对于关系型数据库，其优点和缺点分别有哪些？

  - Mongo DB不需要像关系型数据库那样具有Schema，因此对于非结构化数据的存储具有良好的支持
  - 同时由于Mongo DB是线性存储的，所以通过shard操作可以进行分布式存储
  - 但是，Mongo DB对事务的支持不如关系型数据库，所以不适合复杂事务处理

MongoDB specification

- Mongo DB is a document-oriented database, not a relational one ,which makes scaling out easier
- It can balance data and load across a cluster, redistributing documents automatically
- It supports MapReduce and other aggregation tools, and supports generic secondary indexes
- MongoDB could do some administration by itself, if a master server goes down ,MongoDB can automatically failover to a backup slave and promote the slave to the master

Basic concept of MongoDB

- A document is the basic unit of data for MongoDB
- A collection can be thought of as the schema-free equivalent of a table, the documents in the same collection could have different shapes or types.
- Every document has a special key "_id", it is unique across the document's collection
- Mongo DB groups collections into database and each database has its own permission and be stored in separate disks

Document

- Simple document

A document is roughly equivalent to a row in a relational database, which contain one or multiple key-value pairs

e.g. {"greeting" : "Hello, world!", "foo" : 3}

- Embedded document

embedded documents are entire Mongo DB documents that are used as the values for a key in another document,

```
{
    "name" : "John Doe",
   "address" : {
                    "street" : "123 Park Street",
                    "city" : "Anytown",
                    "state" : "NY"
   }
}
```

Overview

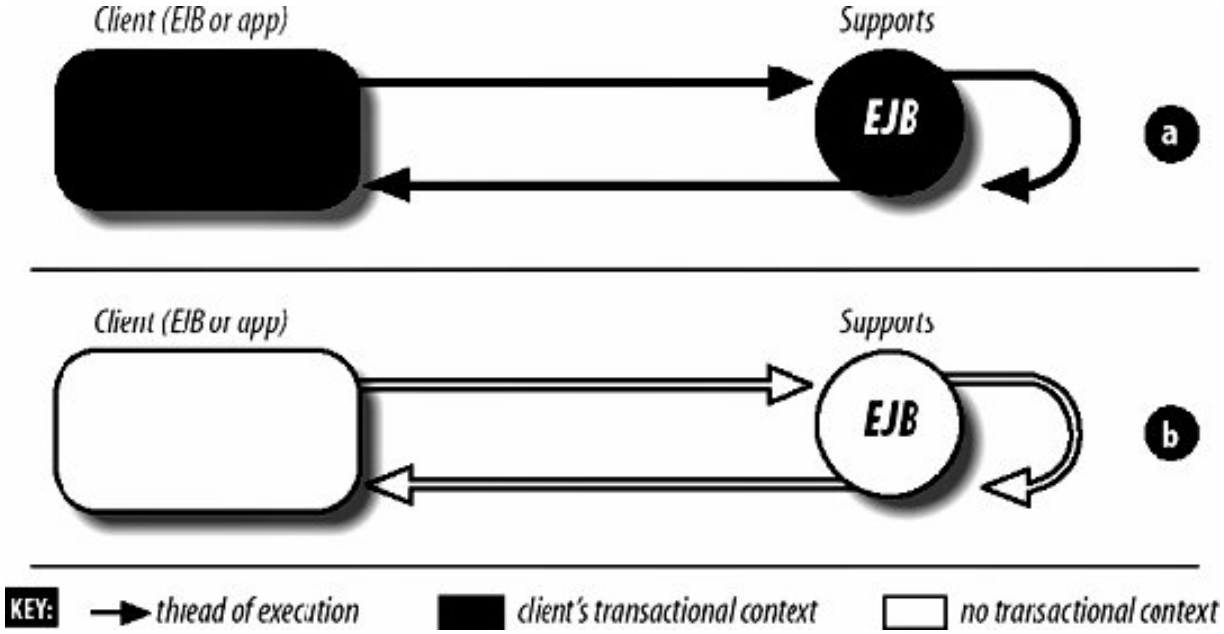
- 在实现具体的业务逻辑时，针对各种需求，大家有了不同的意见，请你根据自己的分析来回答下面的问题。（15分，每题5分）
  - 为了增加系统的可维护性，同学们定义了一个log()方法，凡是对数据库进行写入操作，都必须调用这个方法记录日志，而日志将被记录到与数据库管理系统不同的另一台服务器上的文件系统中。但是log()方法本身实现有些问题，经常会调用不成功。因此，同学们希望对数据库的写入操作都要是事务中执行，但是log()失败不能导致成功的数据库写操作回滚。请问，在这种情况下，log()的事务属性可以设置成什么？为什么？
  - log()方法的事务属性可以设置成RequiresNew，NotSupported或者Never，因为这3种属性都会使得log()将现有事务挂起，然后在新事务或非事务状态下执行log()，之后将挂起的事务恢复，从而保证log()和数据库写操作不属于同一个事务，也就不会导致成功的数据库写操作回滚。

Transaction attributes defined

- NotSupported
  - Invoking a method on an EJB with this transaction attribute suspends the transaction until the method is completed.

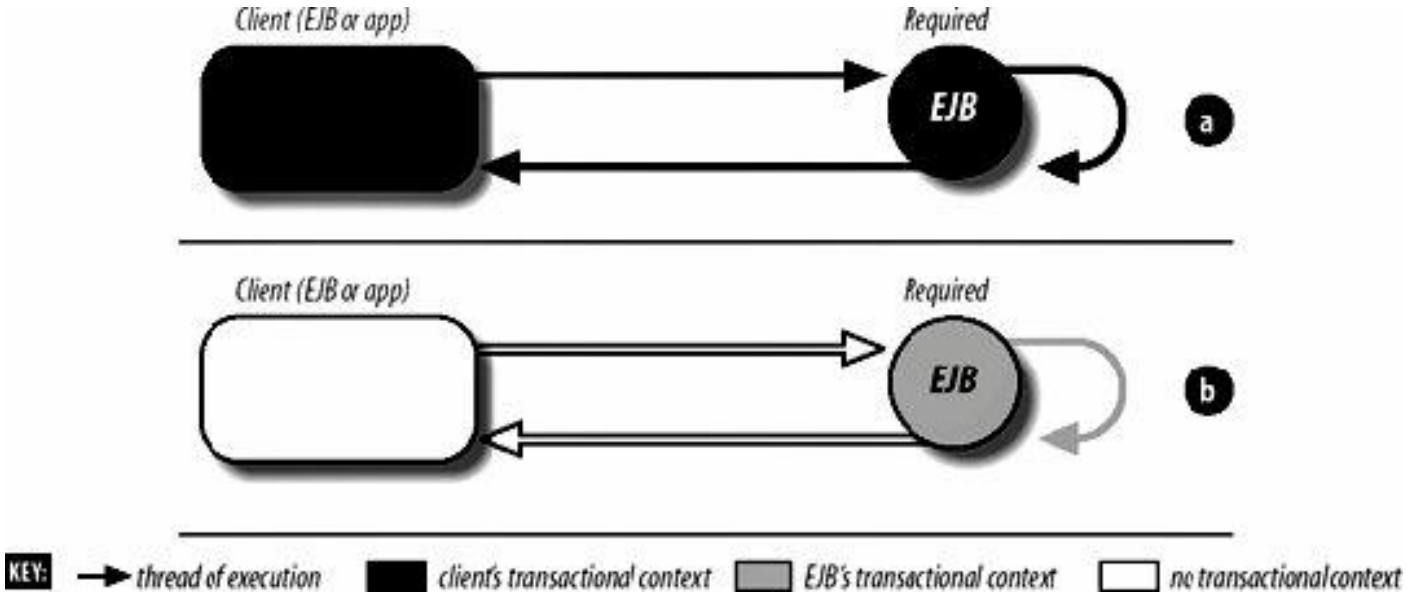
Transaction attributes defined

- Supports
  - This attribute means that the enterprise bean method will be included in the transaction scope if it is invoked within a transaction.



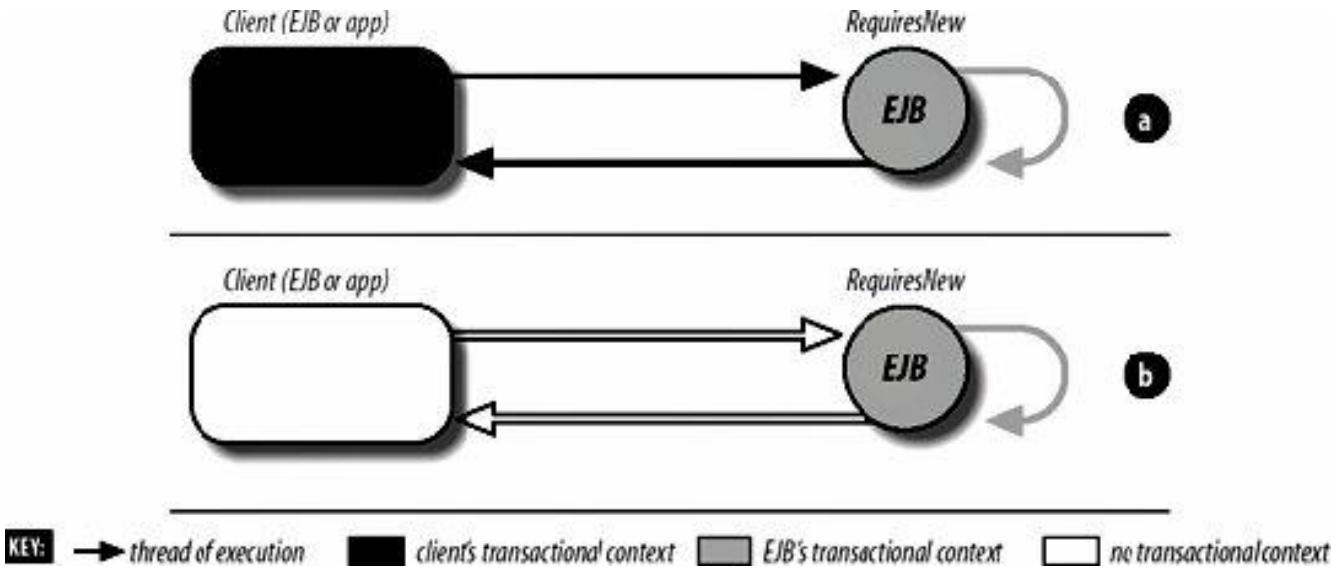
Transaction attributes defined

- Required
  - This attribute means that the enterprise bean method must be invoked within the scope of a transaction.



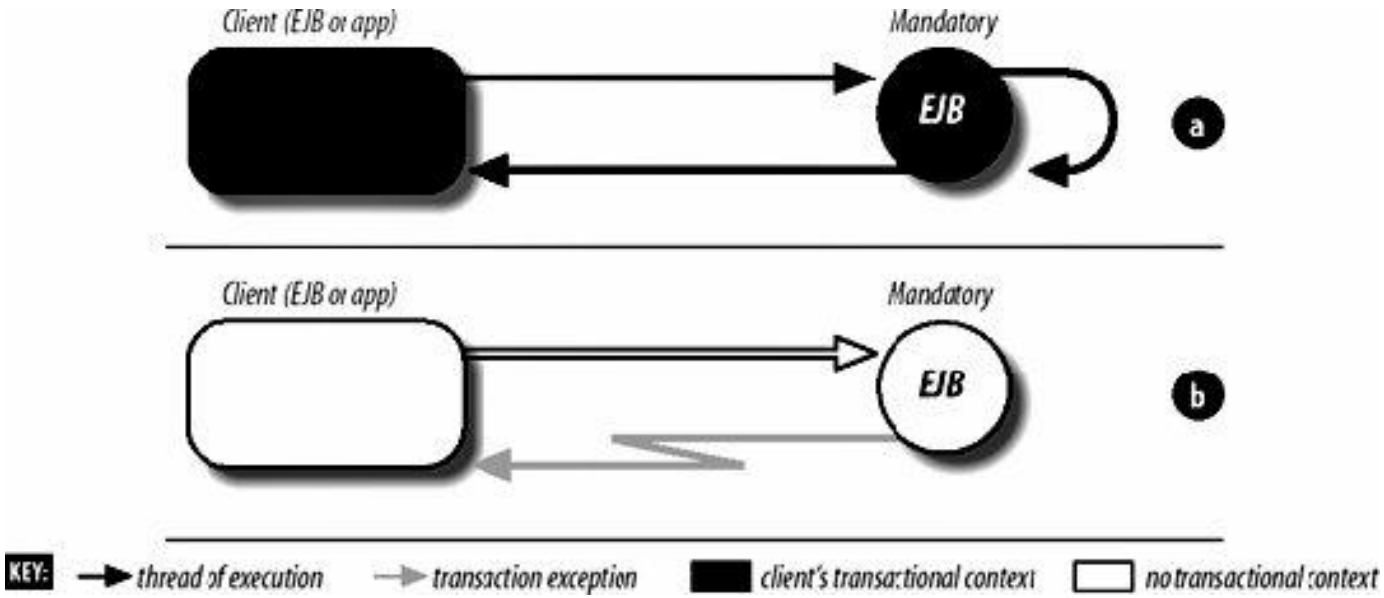
Transaction attributes defined

- RequiresNew
  - This attribute means that a new transaction is always started.



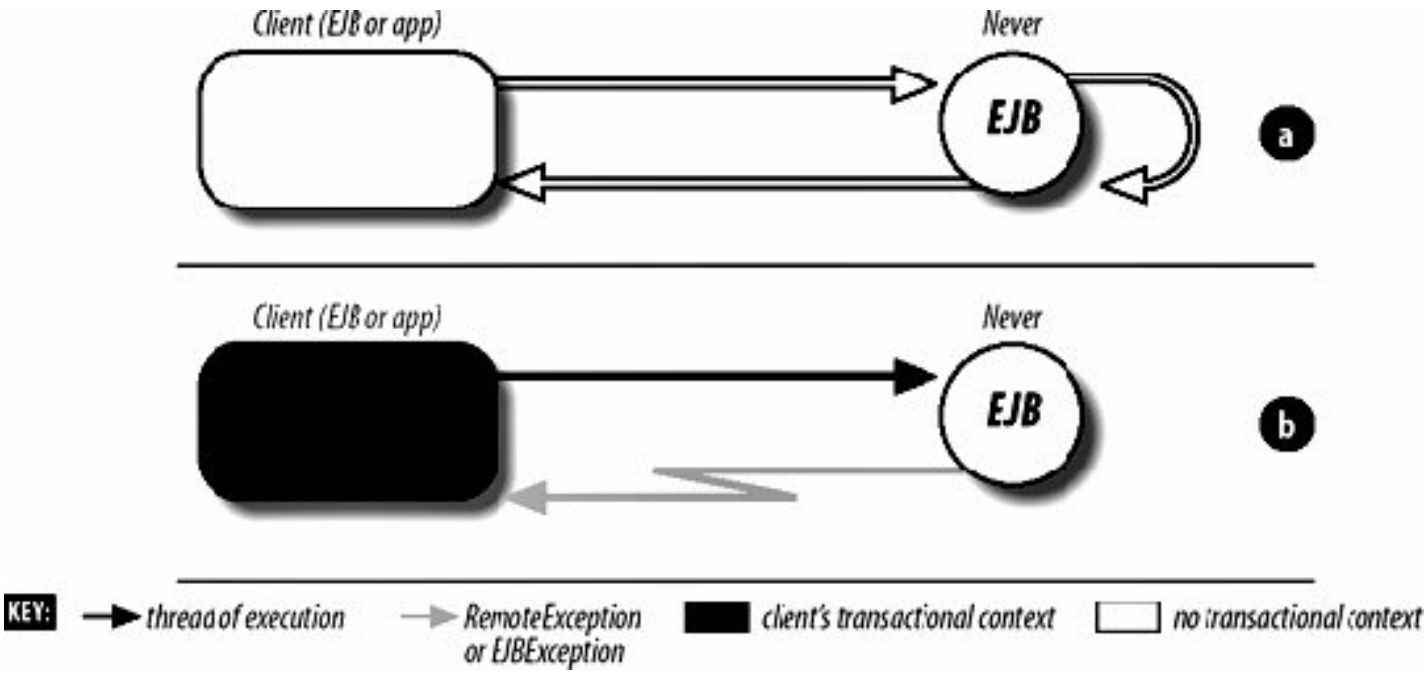
Transaction attributes defined

- Mandatory
  - This attribute means that the enterprise bean method must always be made part of the transaction scope of the calling client.



Transaction attributes defined

- Never
  - This attribute means that the enterprise bean method must not be invoked within the scope of a transaction.



Transaction attributes defined

TRANSACTION ATTRIBUTE	CLIENT'S TRANSACTION	BEAN'S TRANSACTION
Required	None	T2
	T1	T1
RequiresNew	None	T2
	T1	T2
Supports	None	None
	T1	T1
Mandatory	None	Error
	T1	T1
NotSupported	none	None
	T1	None
Never	none	None
	T1	Error

Overview

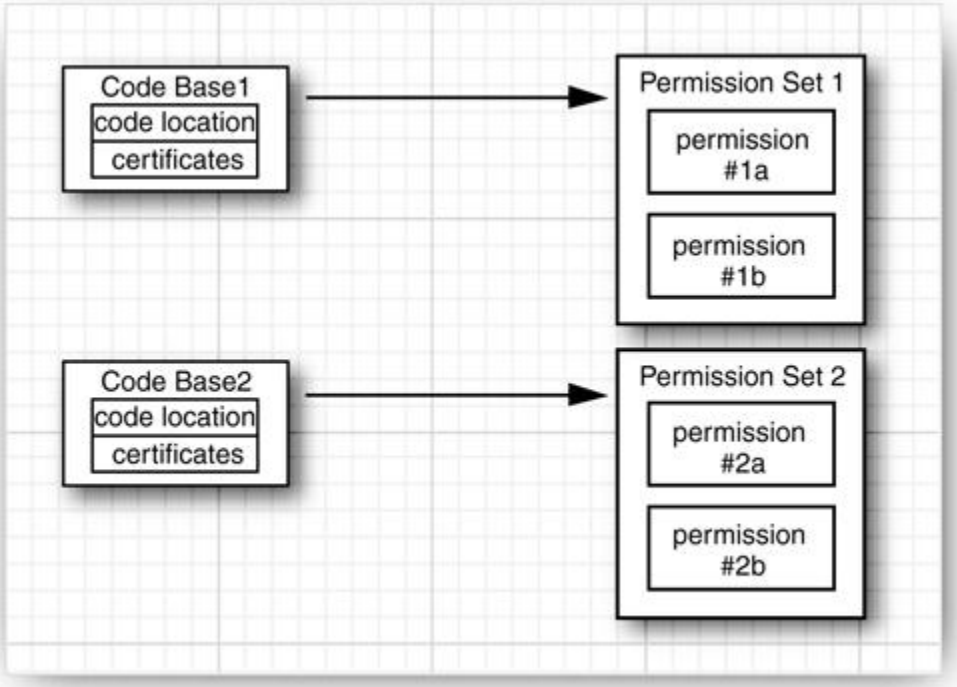
- 在实现具体的业务逻辑时，针对各种需求，大家有了不同的意见，请你根据自己的分析来回答下面的问题。（15分，每题5分）
  - 同学们用JAAS体系进行安全管理，请叙述Policy、domain和permission之间的关系。
  - 我们将代码群组成若干个域，每个域就是一个domain，针对不同的domain，可以赋予不同的permission集合，而同一个domain中的代码具有相同的permission集合，所有关于domain和permission集合的定义就构成了一个policy。

- Code:

```
FilePermission p = new FilePermission("/tmp/*", "read,write");
```

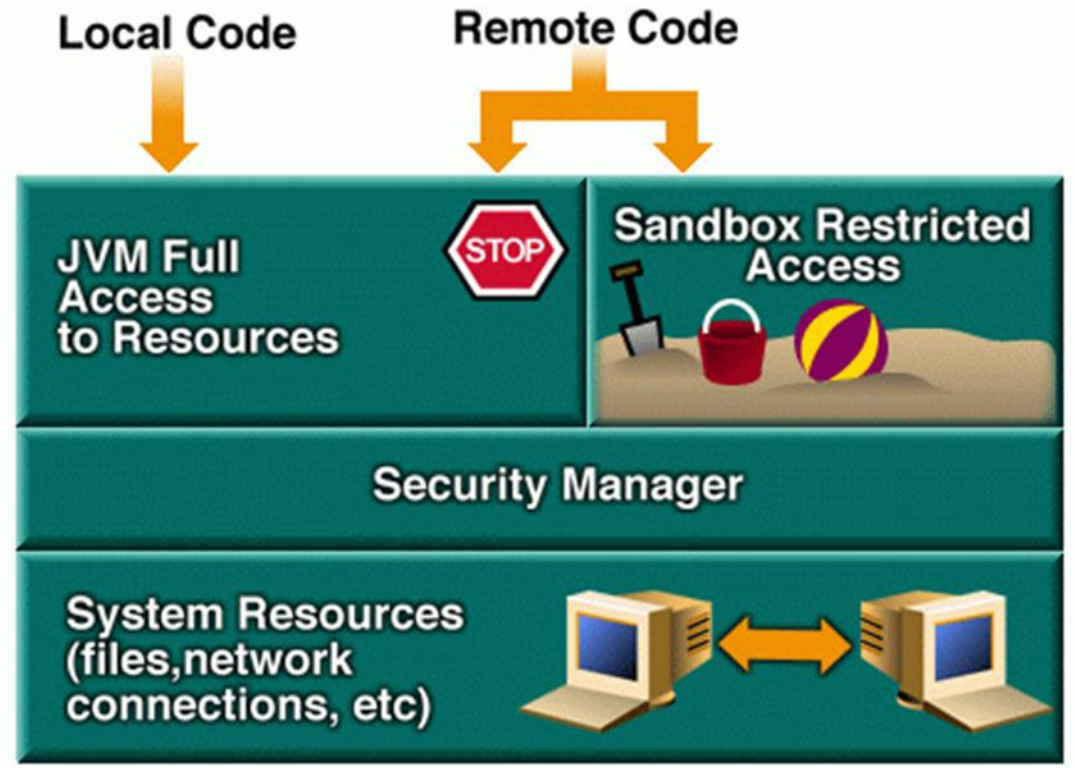
- Permission file:

```
java.io.FilePermission "/tmp/*", "read,write";
```



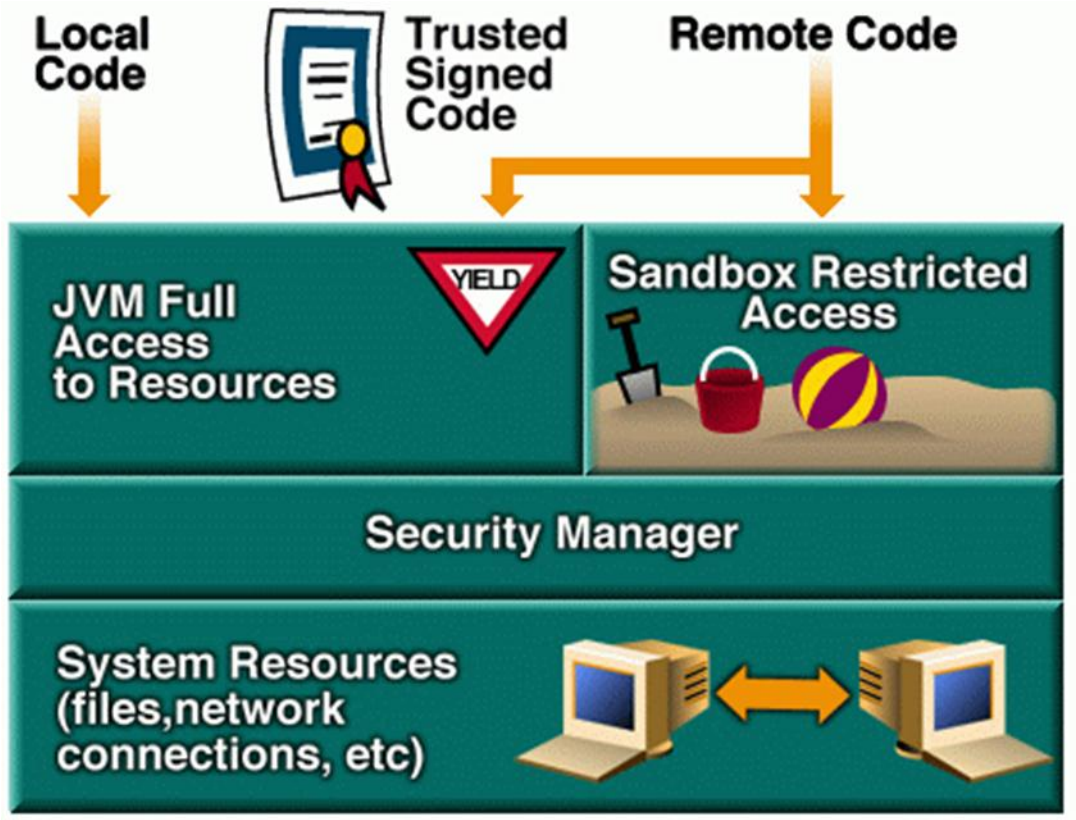
Java Platform Security

- JDK1.0



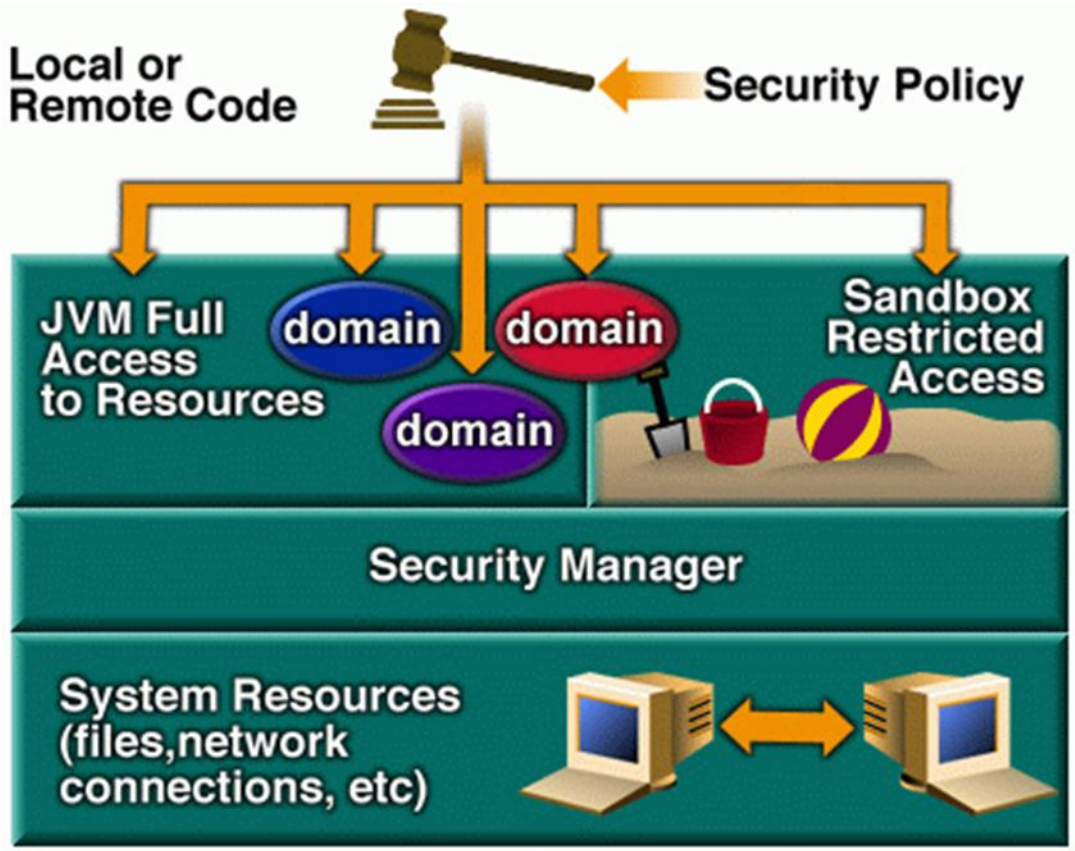
Java Platform Security

- JDK1.1



Java Platform Security

- JDK1.2+

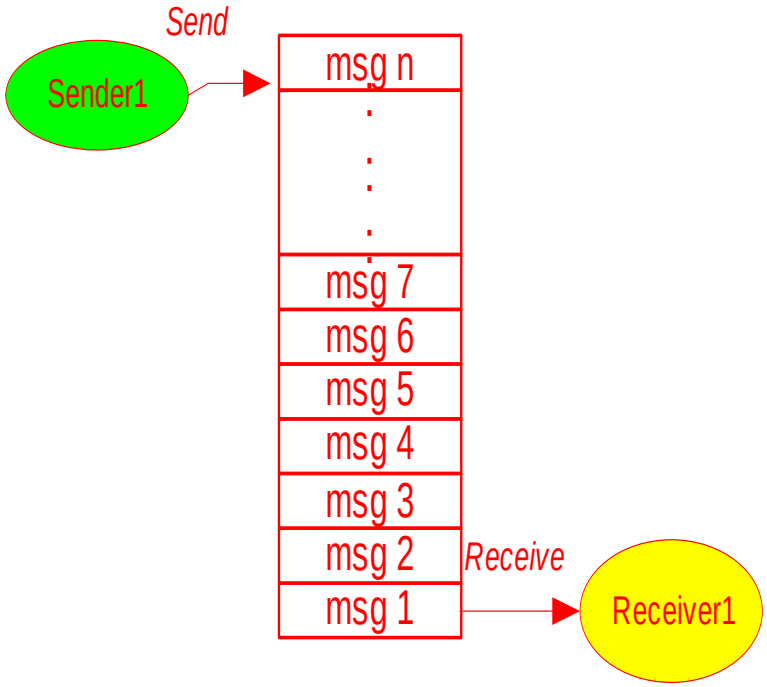


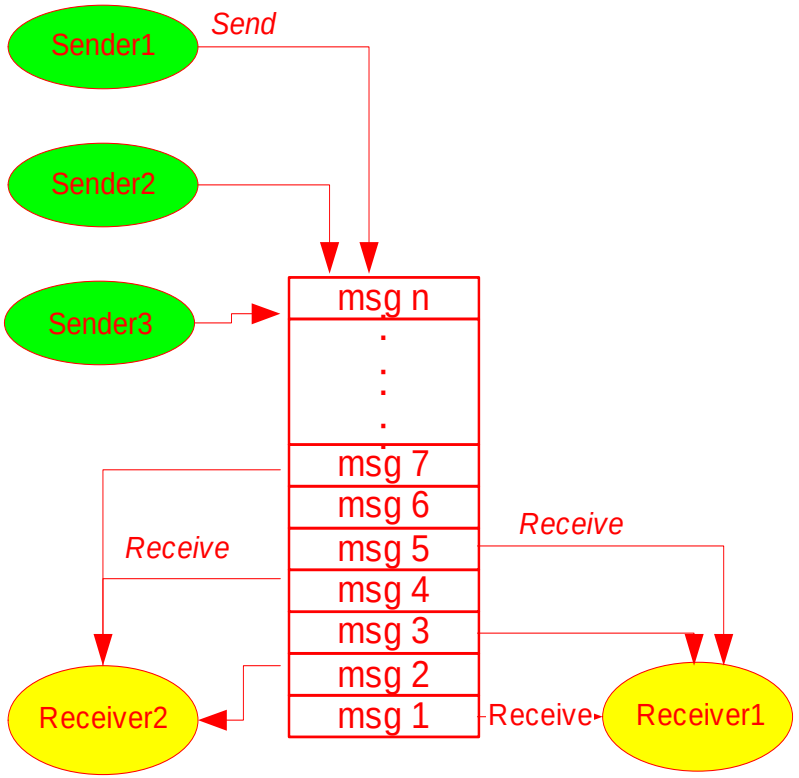
Overview

- 在实现具体的业务逻辑时，针对各种需求，大家有了不同的意见，请你根据自己的分析来回答下面的问题。（15分，每题5分）
  - 同学们在系统中使用消息机制来处理用户的请求，使得响应时间可以缩短，但是对究竟是使用发布/预定模型还是点对点模型产生了分歧。请问，这两种模型分别适用于什么场景？
  - 发布预定模型中每条消息可以被多个消息接收者接收到，而在点对点模式中，每条消息只能被一个消息接收者接收到。

Point-to-point

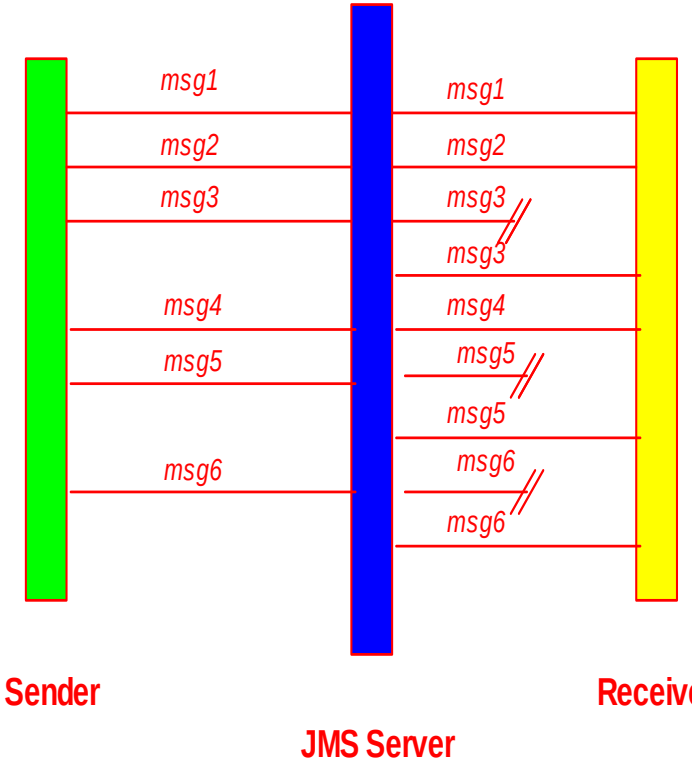
- Queue
  - A virtual channel for sending or receiving messages
  - A message is sent by one application and received by another application
  - The receiver of a message is unique

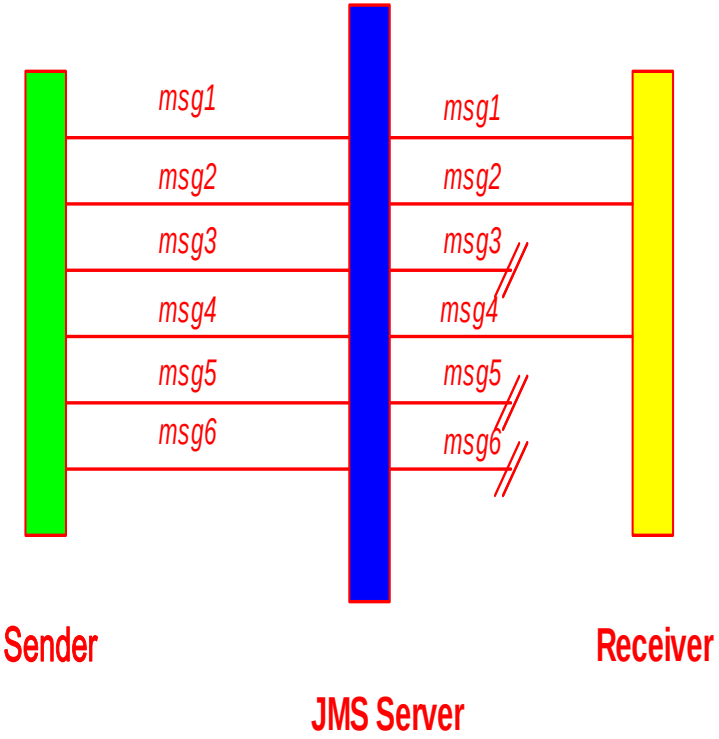




Point-to-point

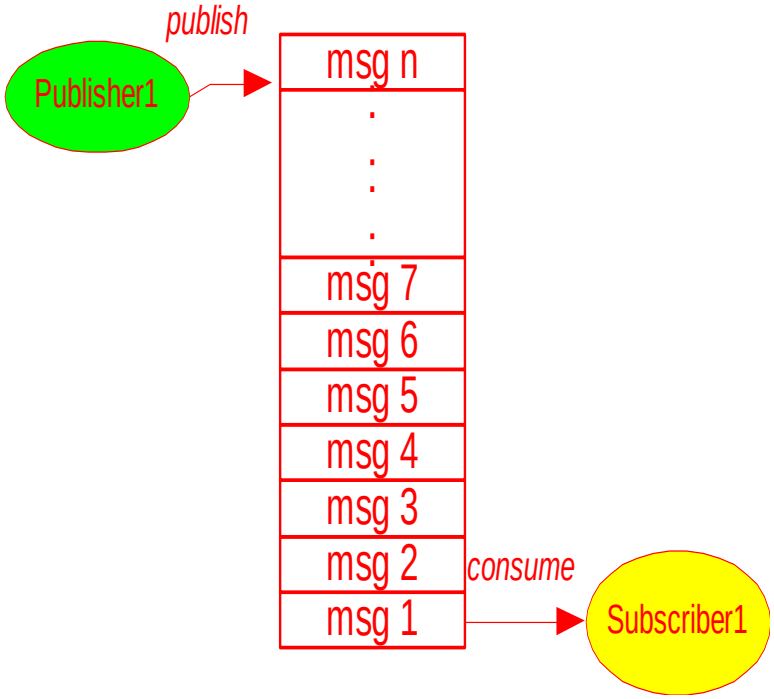
- Persistent messages vs. Non-persistent messages

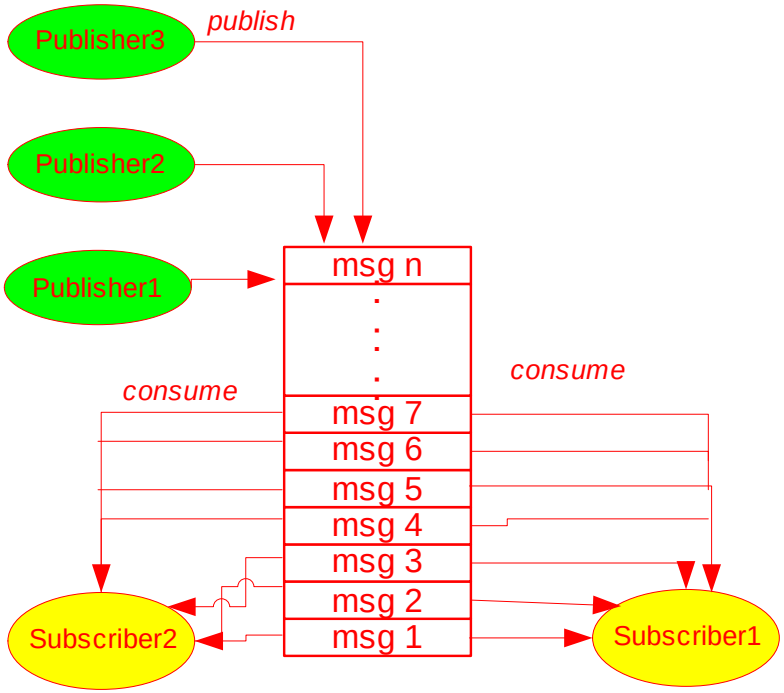




Publish/subscribe

- Topic
  - A virtual channel for publishing or consuming messages
  - A subscriber can subscribe messages by selecting topics
  - A message published on a topic will be forwarded to all subscribers of the topic





Thank You!