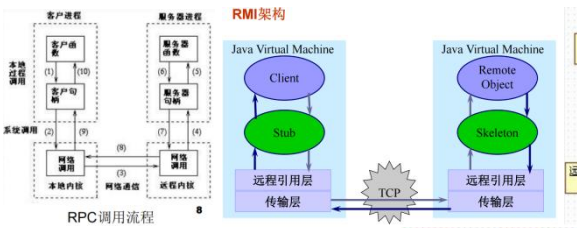
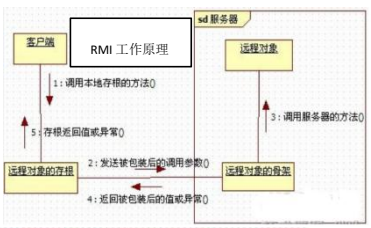
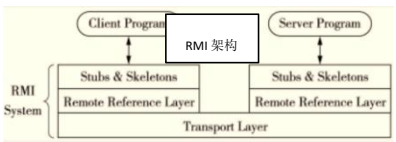
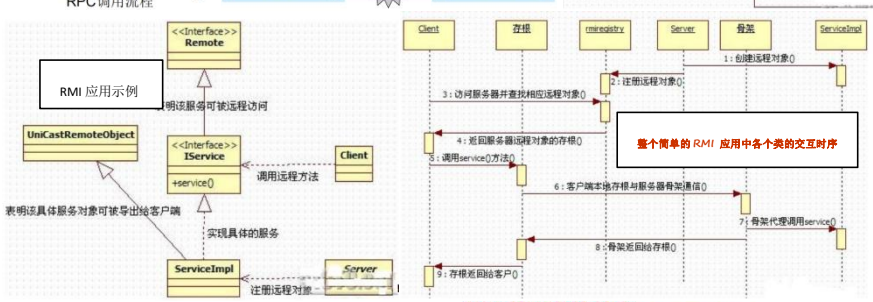
第一章 中间件概论1、**现代运行环境特点：**开放、分布、异构、多变 **2、现代应用系统特点**（性）？**/开发和部署开放系统**（分布式系统）面临的挑战有哪些？具体指的是什么？分布：应用系统中的任务由网络中多台计算机和移动终端中的相关应用共同协作完成，需考虑网络传输、数据安全、数据一致性、同步等诸多问题；异构：硬件、操作系统、网络协议、数据库系统以及开发工具种类繁多，需考虑数据表示、数据转换、调用接口、处理方式等诸多问题；动态协作：参与协作的应用允许位置透明、迁移透明、负载平衡等需求。**3、IBM 的客户信息控制系统 CICS**是最早具有中间件技术思想和功能的软件；联机事务处理平台软件，有效区分应用系统中的表述逻辑层、业务逻辑层和数据逻辑层。**4、三层结构**的客户/服务器系统包括：客户端服务程序、业务服务和其他“中间层”服务程序、数据服务(数据库)。**5、中间件概念：**是计算机网络环境中运行于操作系统于应用软件之间，提供应用互操作，简化应用软件的复杂性、克服网络环境中多种问题的一类系统软件；**中间件技术包含**运行环境、开发工具、应用程序三方面的内容。**6、中间件的组成：执行环境软件 EE 软件：**主体部分，使各节点的下层设备对应用软件透明化；**应用开发工具 AD 工具：**必备部分，帮助开发内含“透明动用对方”成分的应用软件/改造原有的无透明动用能力的应用软件。**7、中间件的基本功能框架结构**：1)传输协议层：TCP/IP、IPX/SPX、NetBIOS、OSI 等；2)应用平台层：Unix、NetWare、NT、VAX、OPENVMS 等；3)中间件服务层：通信、控制、计算、管理、输出、信息等；4)应用接口 API 层。**8、一个中间件应提供的软件运行环境服务：**DCE 服务，分布式计算环境服务；应提供两类：**基本服务**如线程服务、RPC 服务、目录服务等；**扩充性服务**如分布式文件服务和时间服务。**9、组件：**自下而上的视角来称，由组件复用块集成软件系统，是对数据和方法的简单封装；**构件：**自上而下的视角来称，软件由哪块构件复用块构造而成，是对软件构造成份的称呼。**10、中间件的基本功能：(1)通信支持(最基本的功能)**：中间件为其所支持的应用软件提供平台化的运行环境，该环境屏蔽底层通信之间的接口差异，实现互操作；**(2)应用支持**：服务上层应用，提供应用层不同服务之间的互操作机制。为上层应用开发提供统一的平台和运行环境，并封装不同操作系统提供 API 接口，向应用提供统一的标准接口，使应用的开发和运行与操作系统无关，实现其独立性。中间件耦合的结构，标准的封装服务和接口，有效的互操作机制，给应用结构化和开发方法提供有力的支持。**(3)公共服务**：是对应用软件中共性功能或约束的提取。将这些共性的功能或者约束分类实现，并支持复用，作为公共服务，提供给应用程序使用。通过提供标准、统一的公共服务，可减少上层应用的开发工作量，缩短应用的开发时间，并有助于提供应用软件的质量。**11、中间件的作用：**主要是用来支持网络环境中软件实体之间的有效交互；(1)支持软件实体的交互模式，起桥梁作用，实现异构应用资源之间的互联互通；(2)起通道和过滤网的作用，支持软件实体的交互质量(可靠性保障、安全…、高效…)；(3)共性代码库；(4)集成开发框架和部署平台.**12、中间件的局限性：**(1)中间件所应遵循的一些原则离实际还有很大距离 2)中间件之间难互操作 3)对可移植性的支持不足 4)技术发展风险 5)设计选择困难。**13、底层中间件**：用于支撑单个应用系统或解决一类问题。1)**基于对象组件模型的/面向对象中间件**：在分布、异构的网络计算环境中，将各种分布对象有机地结合在一起，完成系统的快速集成。实现对象重用，又称分布对象中间件。它向应用层提供分布对象的通信服务，使得上层应用在事务处理、分布式数据访问、对象处理等方面简单易行；使用对象请求代理 ORB 通信机制；2)**数据访问中间件**：数据访问中间件，是连接应用程序和数据库的软件，可以用来实现应用对未自不同厂商数据库的访问，数据访问中间件有直接访问和数据库复制两种形式：是随着数据库技术的发展而逐渐兴起的，也是其中发展较早的一类；ODBC、JDBC 属于该类中间件；3)**远程过程调用/过程式中间件**：使用 RPC 通信机制实现远程计算服务。从逻辑上分为客户(调用服务)和服务器(提供基于过程的服务)两部分。客户和服务器可以运行在同一计算机上，也可以运行在不同的计算机上，甚至客户和服务器底层的操作系统也可以不同；客户和服务器之间的通信可以使用同步通信，也可以采用线程式异步调用；程序员方便地编写客户端应用程序调用位于服务器上的过程，这种调用通过网络来实现；4)**交易/事务式中间件/TPM**：是一个联机事务处理平台软件，又称为事务处理管理程序，专门针对联机交易处理系统而设计的，交易中间件是一组程序模块，可减少开发联机交易处理系统所需的编程量，快速建立三层结构的联机事务处理应用；主要功能：提供联机事务处理所需要的通信、并发访问控制、事务控制、资源管理、安全管理、负载平衡、故障恢复和其他必要的服务；支持大量客户进程的并发访问，具有极强的扩展性，可靠性高等特点，主要应用于电信、金融、飞机订票系统、证券等拥有大量客户的领域；5)**消息/面向消息的中间件/MOM**：是一类以消息为载体进行通信的中间件，利用高校可靠的消息机制来实现不同应用间大量的数据交换；按其通信模型的不同，消息中间件的通信模型有两类：**消息队列和消息传递**。通过这两种消息模型，不同应用之间的通信和网络的复杂性脱离，摆脱对不同通信协议的依赖，可以在复杂的网络环境中高可靠、高效率的实现安全的异步通信。消息中间件的非直接连接，支持多种通信规程，达到多个系统之间的数据和共享和同步。6)**安全中间件**；7)**RFID 中间件**：拓展了基础中间件的核心设施和特性，将企业级中间件技术延伸到 RFID 领域，是 RFID 产业链的关键共性技术；屏蔽了 RFID 设备的多样性和复杂性，能够为后台业务系统提供强大的支撑，从而驱动更广泛丰富分 RFID 应用；RFID 中间件技术重点研究的内容包括：并发访问技术、目录服务及定位技术、数据及设备监控技术、远程数据访问、安全和集成技术、进程及会话管理技术等；8)**移动中间件**：用来连接不同的移动应用、程序和系统；重点研究内容：接入管理、多协议接入网关、认证服务、连接管理、同步/异步数据传递服务、安全管理、内容服务管理等；例如 MASTudio 移动中间件，基于其框架可快速开发手机办公软件；iMAG 移动中间件，基于其平台可快速开发移动应用。**14、高层中间件**：用于系统整合。1)**应用服务器中间件 WASM**：又称为“应用服务器软件平台”，是当代软件的基础设施，是最为重要的一类中间件。应用服务器能把不同的应用软件作为构件整合到一个协同工作的环境里，并为应用提供了名字、事务、安全、消息、数据访问等服务，还提供给用构件的研发、部署、运行及管理功能。**基于应用服务器的应用系统架构有三大流派**：基于 CORBA、基于 J2EE 和基于.Net。J2EE 架构是目前产业界基于应用服务器中间件的 web 应用系统架构的主流标准。Tomcat(Apache)、TomEE(Apache)、Jboss(RedHat)、Weblogic(BEA)、WebSphere(IBM)等。2)**企业应用集成中间件(EAIM) 3)工作流... 4)门户...**。15、**中间件技术应用**：1)中间件与电子商务整合；2)**中间件在 B/S 模式中的架构**：中间件在 B/S 模式下起到了功能层的作用。用户从 WEB 界面向服务器提交数据请求或应用请求时，功能层负责将这些请求分类为数据或应用请求，再向数据库发出数据交换申请。数据库对请求进行筛选处理之后，再将所需的数据通过功能层传递回到用户端。单一用户可以进行点对面的操作，无需通过其他软件进行数据转接。**15、主流中间件**：远程过程调用中间件 RPCM、面向对象中间件 OOM、消息中间件 MOM、数据库访问中间件 DAM、交易中间件 DTPM。

第二章 中间件通用技术原理与开发1、**远程过程调用**：过程调用(包括本地过程调用和远程过程调用)是进程间通信使用到的一种通信技术；进程间调用 IPC 是再多任务操作系统或联网的计算机之间运行的两个程序和进程间传送数据或信号所用的一些技术或方法；本地过程调用 LPC 用在多任务操作系统中，使得同时运行的任务能互相会话，这些任务共享内存空间使任务同步和互相发送信息。2、**远程过程调用 RPC 协议**：一种通过网络从远程计算机程序上请求服务，不需要了解底层网络技术的协议。RPC 类似 LPC，只在网上工作。RPC 协议规定允许互联网中一台主机程序调用另一台主机程序，而程序员无需对这个交互过程进行编程. RPC 协议中强调当 A 程序调用 B 程序中功能或方法时，A 是不知道 B 中方法的具体实现的。使得开发包括网络分布式多程序在内的应用程序更加容易。3、**RPC 框架**：4 个核心组件 Client、Server、Client Stub(存根)、Server Stub；1)Client：远程过程服务的调用方；2)Server：真正的远程过程服务提供者；3)客户端存根：存放服务端的地址消息，再将客户端的请求参数打包成网络消息，然后通过网络远程发送给服务方；4)服务端存根：接收客户端发送过来的消息，将消息解包并调用本地的过程(方法)。4、**RPC 工作原理**：RPC 协议采用客户机/服务器工作模式，RPC 服务请求程序是一个客户机，服务提供程序就是一个服务器。客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，最后客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。5、**一次客户机对服务器的 RPC 调用**，其内部操作大致有十步：1)调用客户端句柄，执行传送参数；2)调用本地系统内核发送网络消息；3)消息传送到远程主机；4)服务器句柄得到消息并取得参数；5)执行远程过程；6)执行的过程将结果返回服务器句柄；7)服务器句柄返回结果，调用远程系统内核；8)消息传回本地主机；9)客户句柄由内核接收消息；10)客户接收句柄返回的数据。6、**远程方法激活**：RMI 是在分布式程序中，其远程对象的方法能够被远行在不同主机上的其他 Java 虚拟机的方法调用的方法。是用 Java 实现的 RPC，RMI 是 Java 支撑分布式系统的基石。RMI 是远程过程调用(RPC)的一种面向对象实现，RMI 底层是通过 socket 通信和对象序列化技术来实现的。RMI 的目的就是要使运行在不同的计算机中的对象之间的调用表现得像本地调用一样。7、**RMI 架构**：RMI 应用程序通常包括两个独立的程序：服务器程序和客户机程序。客户端 Stub 和 Skeleton、服务器端 Stub 和 Skeleton、远程调用接口。分为三个抽象层：传输层、引用层、调用代理层，高层震盖低层。传输层负责 Socket 通信，依赖于 TCP/IP 协议实现客户机与服务器的互联。引用层被用来寻找各自的通信伙伴，在这一层还有一个提供名字服务的部分，称为注册表(registry)。调用代理层负责调用、参数和结果的序列化和反序列化等工作。8、**RMI 需要将行为的定义与行为的实现分别定义**，并允许将行为定义代码与行为实现代码存放并运行在不同的 JVM 上。在 RMI 中，**远程服务的定义**是存放在继承了 Remote 的接口中。**远程服务的实现代码存放**在实现该定义接口的类中。**RMI 支持两个类实现一个相同的远程服务接口**：一个类实现行为并运行在服务器上，另一个类作为一个远程服务的代理运行在客户机上。客户程序发出关于代理对象的调用方法，RMI 将该调用请求发送到远程 JVM 上，并且进一步发送到实现的方法中。实现方法将结果发送给代理，再通过代理将结果返回给客户端。9、**Remote 接口**用于标识其方法可以从非本地虚拟机上调用的接口。任何远程对象都必须直接或间接实现此接口。只有在“远程接口”(扩展 java.rmi.Remote 的接口)中指定的这些方法才可远程使用。也就是说需要远程调用的方法必须在扩展 Remote 接口的接口中声名并且要抛出 RemoteException 异常才能被远程调用。远程对象必须实现 java.rmi.server.UniCastRemoteObject 类，这样才能保证客户端访问获得远程对象时，该远程对象将会把自身的一个拷贝序列化后以 Socket 的形式传输给客户端，远程对象的接口和实现必须在客户端和服务器端同时存在并且保持一致才行。10、**客户端所获得的这个拷贝称为“存根”**，而服务器端本身已存在的远程对象则称之为“骨架”。此时存根是客户端的一个代理，用于与服务器端的通信。而骨架也可认为是服务器端的一个代理，用于接收客户端的请求之后调用远程方法来响应客户端的请求。11、**RMI 工作原理**：RMI 框架的 java 实现应用了代理模式来封装本地存根与真实的远程对象进行通信的细节。1)当客户端调用远程对象方法时 2)**存根**负责把要调用的远程对象方法的方法名及其参数编组打包，并将该包向下经远程引用层、传输层转发给远程对象所在的服务器。该包到达服务器后，向上经远程引用层，被远程对象的 Skeleton 接收，此 Skeleton 解析客户包中的方法名及编组的参数后 3)在服务器端执行客户要调用的远程对象方法 4)然后将该方法的返回值(或产生的异常)打包后通过相反路线返回给客户端 5)客户端的 Stub 将返回结果解析后传递给客户程序。事实上，不仅客户远程程序可以通过存根调用服务器端的远程对象的方法，而服务器端的程序亦可通过向客户端传递的远程接口回调客户端的远程对象方法。在分布式系统中，所有的计算机可以是服务器，同时又可以是客户机。12、**RMI 应用示例**：1)**远程接口 IService 接口**：用于声明服务器端必须提供的服务(即 service()方法)远程接口实现类。2)**ServiceImpl 类**：是具体的服务实现类服务端。3)**Server 类**:负责注册服务器远程对象，以便在服务器端存在常驻代理对象来对客户端的请求提供处理和响应。4)**客户端调用 Client 类**:在 RMI 注册表中查找指定对象，调用远程对象方法。










```
<?xml version="1.0" encoding="UTF-8" ?>
<students>
    <student studentNo="2022150278">
        <name>He GaoJie</name>
        <gender>male</gender>
        <specialty>Software Engineering</specialty>
    </student>
</students>
```

例1，使用date属性：

```
<note date="08/08/2008">
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

例2，使用date元素：

```
<note>
<date>08/08/2008</date>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

例3，使用了扩展的date元素（最好的方式）：

```
<note>
<date>
  <day>08</day>
  <month>08</month>
  <year>2008</year>
</date>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

XML数据交换技术---XML文档树例子

根元素是？ bookstore

```
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

3 XML验证---内部的 DOCTYPE 声明

➢带有 DTD 的 XML 文档实例


```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to      (#PCDATA)>
<!ELEMENT from    (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>George</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Don't forget the meeting!</body>
</note>
```

在浏览器中打开此 XML 文件，并选择“查看源代码”命令，即可查看到内嵌的DTD声明。

枚举约束：把 XML 元素的内容限制为一组可接受的值
模式约束：限定为一系列可使用的数字或字母
whiteSpace 限定：需规定对空白字符的处理方式
使用 length、maxlength、minlength 限制元素中值的长度

XML 验证--XSD 限定/Facets

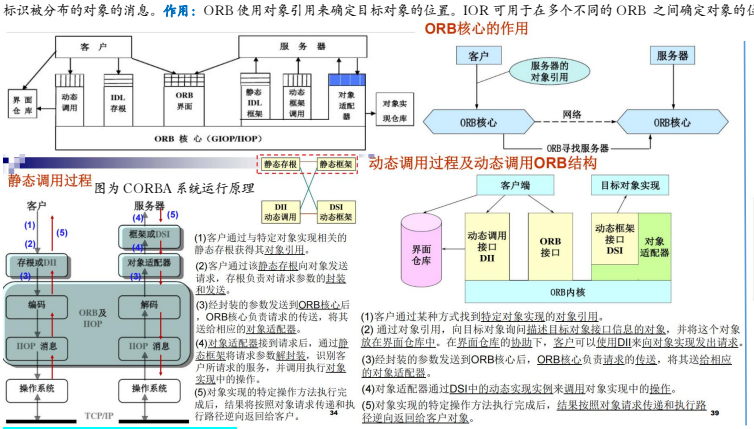
下面的例子定义了带有一个限定的名为“password”的元素。可接受的值是由 8 个字符组成的一行字符，这些字符必须是大写或小写字母 a~z 亦或数字 0~9 组成：

```
<xs : element name="password">
<xs : simpleType>
  <xs : restriction base="xs : string">
    <xs : pattern value="[a-zA-Z0-9]{8}"/>
  </xs : restriction>
</xs : simpleType>
</xs : element>
```

XML：1、XML：可扩展标记语言 W3C。被设计用来传输和存储数据，其设计宗旨是传输数据，而非显示数据。XML 作为一种用来描述数据的标记语言，具有对数据进行统一描述的强大功能。具有可扩展性、结构化语义、平台无关性特点；充分满足互联网分布式异构环境的需求；已成为网络数据传输和交换的主要标准和载体。一个完整的 XML 文档主要包括三个部分：文档声明、处理指令(可省略)、元素和属性定义。2、XML 与 HTML 的主要差异：XML 不是 HTML 的替代；为不同的目的而设计；XML 被设计为传输和存储数据，焦点是数据的内容；HTML 被设计用来显示数据，焦点是数据的外观；HTML 旨在显示信息，XML 旨在传输信息。3、XML 用途：把数据从 HTML 分离；简化数据共享和传输。4、XML 文档结构/注意点：形成一种树结构；必须包含根元素。XML 优势之一就是可以经常在不中断应用程序的情况进行扩展。XML 属性提供关于元素的额外信息，通常提供不属于数据组成部分的信息。XML 标签大小写敏感。属性值需加引号；如果属性值有双引号(")，使用单引号包围。另外，xml 必须有关闭标签、必须被正确的嵌套。5、XML 实体引用：<message>if a<10 then</message>会发生错误，使用实体引用：1)<－ 2)>－ 3)&－& 4)'－' 5)"－" ；<message>if a < 10 then</message>。6、文档空格会被保留；命名规则：元素名称不能以数字、标点符号、xml 开始，不能包含空格；尽量避免使用属性(原因 1)无法包含多重的值 2)无法描述树结构 3)不易扩展 4)难以阅读和维护)。7、XML DTD(文档定义类型)：作用：定义 XML 文档的构建模块。它使用一系列合法的元素来定义文档结构。DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用。8、**内部的 DOCTYPE 声明**：<!DOCTYPE 根元素 [元素声明]>9、**外部文档声明**：<!DOCTYPE 根元素 SYSTEM “文件名”>。10、**所有 XML 文档均由以下简单的构建模块构成**：1)元素:主要构建模块，可包含文本、其他元素或空；2)属性：提供有关元素的额外信息，置于某元素的开始标签中，以名称/值的形式成对出现；3)**实体**:定义普通文本变量，实体引用是对实体的引用；4)PCDATA:被解析的字符数据；5)CDATA。11、DTD 中 **XML 元素声明使用语法**：<!ELEMENT 元素名称 类别/(元素内容)>。12、**空元素声明**：<!ELEMENT 元素名称 EMPTY>。13、**只有 PCDATA 元素声明**：<!ELEMENT 元素名称 (#PCDATA)>。14、**特殊声明元素**：+声明最少出现一次；*0 次或多次；?0/1 次；x|y 非 x 既 y。15、DTD 中，**属性声明**通过 ATTLIST：语法：<!ATTLIST 元素名称 属性名称 属性类型 默认值>；DTD 实例：<!ATTLIST payment type CDATA “check” >XML 实例：<payment type=” check” >。16、**DTD 属性**：1)**规定默认属性值**：<!ELEMENT square EMPTY>/<!ATTLIST square width CDATA “0” >；2)**不强制包含属性且没有默认值选项**，使用#IMPLIED，语法：<!ATTLIST 元素名称 属性名称 属性类型 #IMPLIED>；3)**没有默认值选项但强制提交属性**：<!ATTLIST 元素名称 属性名称 属性类型 #REQUIRED>；4)**属性拥有固定值且不允许改变**：<!ATTLIST 元素名称 属性名称 属性类型 #FIXED “value” >；5)**属性值为一系列固定值之一**：<!ATTLIST 元素名称 属性名称(en1|en2|..) 默认值>。17、**DTD 实体**：用于定义引用普通文本或特殊字符的快捷方式的变量；由三部分组成：&、实体名称和；；声明：<!ENTITY 实体名称 “实体的值” >；XML 例子：<author>&writer;&copyright;</anthor>。18、**外部实体声明**：<!ENTITY 实体名称 SYSTEM “URI/URL” >。19、通过把 XML 解析器的 validateOnParse 设置为 false 就可以关闭验证。20、XMLSchema：类似且将取代 DTD。1、java 解析 XML 文件的 4 种方式：1)**DOM**：是用与平台和语言无关的方式表示 xml 文档的官方 W3C 标准，是以层次结构组织的节点或信息片断的集合；这个层次结构允许开发人员在树中寻找特定信息；分析该结构通常需要加载整个文档和构造层次结构，才能做任何工作；由于它是基于信息层次的，因而 DOM 被认为是基于树或基于对象的。**DOM 以及广义的基于树的处理的优点**：树在内存中是持久的，因此可以修改它以便应用程序能对数据和结构作出更改；可以在任何时候在树中上下导航；使用简单。2)**SAX**：处理有够类似于流媒体优点，分析能够立即开始而不是等待所有数据被处理；由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中；应用程序甚至不必解析整个文档，它可以在某个条件得到满足时停止解析；比 DOM 快许多。3)**JDOM**：目的是成为 java 特定文档模型，它简化与 xml 的交互并且比使用 DOM 实现更快；**JDOM 与 DOM 主要有两方面不同**，一是 JDOM 仅使用具体类而不使用接口，这在某些方面简化了 API，但是也限制了灵活性；二是 API 使用了 Collections 类，简化了那些已经熟悉这些类的 java 开发者的使用。**JDOM 文档声明其目的**是“使用 20%成更少的精力解决 80%成更多 java/xml 问题”；JDOM 对于大多数 java/xml 应用程序来说当然是有用的，并且大多数开发者发现 API 比 DOM 容易理解得多；JDOM 还包括对程序行为的相当广泛检查以防止用户做任何在 xml 中无意义的事。JDOM 本身不包含解析器，它通常使用 SAX2 解析器来解析和验证输入 xml 文档。它包含一些转换器以将 JDOM 表示输出成 SAX2 事件流、DOM 模型或 XML 文本文档。JDOM 是在 Apache 许可证变体下发布的开放源码。4)**DOM4J**：代表了完全独立的开发结果；它合并了许多超出基本 XML 文档表示的功能，包括集成的 XPath 支持、xml Scheme 支持以及用于大文档或流化文档的基于事件的处理；它还提供了构建文档表示的选项，通过 DOM4J API 和标准 DOM 接口具有并行访问功能，为支持这些功能 DOM4J 使用接口和抽象基本类方法。DOM4J 大量使用了 API 中的 Collections 类，但是在许多情况下还提供了一些替代方法以允许更好的性能或更直接的编码方法。是一个非常优秀的 java XML API，具有**性能优异**、功能强大和极端易用使用的特点，是一个开放源代码的软件。5)**DOMorSAX?**
DOM 采用建立树形结构的方式访问 xml 文档，SAX 采用事件模型；DOM 解析器把 xml 文档转化为一个包含其内容的树，并可以对树进行遍历；用 DOM 解析模型的优点是编程容易，开发人员只需要调用建树的指令，然后利用 navigationAPIs 访问所需的树节点来完

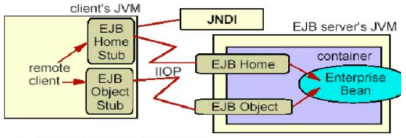
成任务，可以很容易的添加和修改树中元素。然而由于使用 DOM 解析器的时候需要处理整个 xml 文档，所以对性能和内存的要求比较高，尤其是遇到很大的 xml 文件的时候；由于它的遍历能力，DOM 解析器常用于 xml 文档需要频繁的改变的服务中；SAX 解析器采用了基于事件的模型，它在解析 xml 文档的时候可以触发一系列的事件；当发现给定 tag 时，它可以激活一个回调方法，告诉该方法制定的标签已经找到。SAX 对内存的要求通常会比较低，因为它让开发人员自己来决定所要处理的 tag，特别是开发人员只需要处理文档中所包含的部分数据时，SAX 这种扩展能力得到了更好的体现；但使用 SAX 解析器时编码工作会比较困难，而且很难同时访问同一个文档中的多处不同数据。1、**XSD** 简易元素：指那些指包含文本的元素，不会包含任何其他的元素或属性。可拥有指定的默认值或固定值，当没有其他的值被规定时，默认值会自动分配给元素(同属性)。文本类型可以是 xml scheme 定义中包括的类型中的一种(布尔、字符串、数据等)，也可以是自行定义的定制类型。定义简易元素语法<xs : element name=" xxx(元素名称)" type=" yyy(数据类型)" />。2、XMLScheme(实现元素替换)内建的数据类型：string/decimal/integer/boolean/date/time。3、XSD 属性定义语法：<xs:attribute name=" xxx" type=" yyy" />，在缺省情况下属性是可选的，如需规定为必选使用 use 属性：<xs : attribute name=" lang" type=" xs:string" use=" required" />。4、XSD 复合类型指示器：1)Order 指示器用于定义元素的顺序。<all>指示器：子元素可以按照任意顺序出现，且必须只出现一次；<choice>：可出现某个子元素或另一个子元素(非此即彼)，可出现任意次；<sequence>：子元素必须按照特定顺序出现。2)Occurrence 用于定义某个元素在出现的频率。<maxOccurs>规定出现最大次数，<minOccurs>。3)Group 用于定义相关的数社元素。<Group name>&<attributeGroup name>。5、<any>实现通过未被 scheme 规定的元素拓展 xml 文档。6、block 属性防止其他元素替换某个指定元素。7、XSD 日期定义格式：“YYYY-MM-DD”，时间“hh:mm:ss”，YYYY-MM-DDThh:mm:ss。8、**JSON** 技术：一种主流的数据交换格式。用于数据交互服务器返回给客户端的数据，一般都是 JSON 格式或 XML 格式。JSON 是存储和交换文本信息的语法，类似 xml。特点：1)一种轻量级的数据交换格式。与 xml 这种数据交换格式相比，速度更快，文档更小。Xml 文件会包含很多对象的层次关系，解析和转换极不方便，字符更多导致文件过大。而这些弊端 JSON 都已经克服，简单的符号使得解析更快。2)一种与平台无关的数据交换。Json 作为数据交换的格式与平台种类无关。不管服务器使用 java、C#或其他语言都能从这种字符串格式转换为具体语言的对象。同样，任何语言也都能将自己的对象转换为 json 字符串。9、json 结构：1)使用名称-值对儿的形式来表示一个对象；2)值的有序列表，也就是数组。10、json 表示对象：如{‘id’:1,’ name’:’ 王小贱’ ,’ age’:24}；访问某个属性值：对象.属性名。Json 表示数组：[{},{}]。11、json 实现数据交换：1)S→C：客户端请求数据，服务器将 java 对象先转换成字符串，经响应把字符串传到客户端后，客户端就会接收到这个转换结果，但 JavaScript 要求把这个字符串变成对象格式才更方便访问，在客户端的 JavaScript 代码中又需要将这个 json 字符串变成 JavaScript 能够识别的对象，这样就完成了从服务器端的对象到客户端对象的整个转变过程。2)C→S：如果在客户端填写了数据后想提交给服务器，首先是将客户端数据构造成一个 json 对象字符串，经网络传递给服务器端，服务器端再依据转换规则将 json 字符串变成 java 识别的对象。12、json 字符串转换成 js 对象：var jsonObj=eval(“(“+json+”)”); eval 添加(“(“+json+”)”)的目的是迫使 eval 函数在处理 JavaScript 代码的时候强制将括号内的表达式 expression 转化为对象。13、1)Java 对象转换成 json 字符串：JSONObject obj=JSONObject.fromObject(emp); String jsonstr=obj.toString(); 2)Java 数组转 json 字符串：JSONArray jsonArr=JsonArray.fromObject(fs); String jsonStr=jsonArr.toString(); 3)json 字符串转 java 对象：JSONObject jsonObj=JSONObject.fromObject(jsonStr); A a=(A)JSONObject.toBean(jsonObj, A.class); 4)json 字符串转 java 数组：JSONArray jsonArr=JSONArray.fromObject(jsonStr); List<A>aa=(List<A>) JSONArray.toCollection(jsonArr, A.class); 14、缓存问题的解决：xhr.open(‘get’,’ getNumber.do?’ +Math.random(),true); 不用 GET 请求方式，改用 POST 提交，浏览器不进行地址判断；同步问题：xhr.open(‘get’,’ check..’ ,false); false 代表同步类型的请求，为 true 代表异步类型请求。

第三章 CORBA 面向对象应用程序体系规范 1、CORBA 公共对象请求代理体系结构是 OMG 组织指定的一个应用软件体系结构和对象技术规范。CORBA 的核心是一套标准的语言、接口和协议，以支持异构分布应用程序间的互操作性及独立于平台和编程语言的对象重用。是一种分布式应用程序通信技术。被设计用来对不同对象系统进行集成，提供灵活的对象调用与功能实现。客户通过 ORB 调用服务器对象。2、由 OMG 指定的最关键的规范：1)对象管理结构 2)CORBA 规范(OMA 核心)，OMA 提供一个完整的体系结构，并以足够的灵活性、丰富的形式适用各类分布式系统。OMA 包括两部分：1)对象模型定义如何描述分布式异质环境中的对象；2)参考模型描述对象之间的交互。OMA 描述了面向对象技术在分布式处理中的运用。3、ORB 是 CORBA 的基础，是在分布式环境下，CORBA 应用所使用的、基于对象模型的软件总线。ORB 负责对象在分布环境中透明地收发请求和响应，是构建分布对象应用、在异构或同构环境下实现应用间互操作的基础。ORB 通过标准接口使对象之间的连接独立于所使用的硬件和软件平台，从而保证了对平台以及对操作系统、网络协议和编程语言的透明性。4、对象服务独立于应用领域；公共设施通用领域内定义的对象，面向终端用户提供的一组共享服务接口；域界面专用领域内定义的对象，针对某一特殊应用领域提供的接口；应用界面针对某一具体应用而产生。5、ORB 核心的作用：Step1 1)当客户激活一个调用操作时，操作中指出的目标对象的对象引用经客户存根传递到 ORB 核心；2)ORB 核心代表客户自动寻找对应的服务器；3)找到服务器后，ORB 要确保调服务器做好接收请求的准备工作。Step2 1)客户端的 ORB 核心接收被调用操作的参数，并将它们编码为网络可传输的格式；2)服务器端的 ORB 核心将来自网络的操作参数进行编码，然后送给服务器，并启动服务器执行所调用的操作。Step3 服务器端执行完操作后，如果返回参数，ORB 核心将它编码传入网络；客户端的 ORB 核心对它进行解码并将操作结果返回给客户。6、IDL 语言：必须知道目标对象所能支持的服务；描述性语言，与编程语言无关，将界面与对象实现分开，支持分布式才处理的关键字 in、out、inout 等。7、IDL 文件：module 包名{interface 接口名{返回值 方法名();};};8、IDL 编译：idlj –fall xx.idl 生成 S/C 端代码，有 6 个 java 文件 1)client 需要的：_xxStub.java、xx.java、xxHelper.java、xxHolder.java、xxOperations.java 2)server：xx.java、xxOperations.java、xxPOA.java。9、存根：一段程序代码，为界面中操作提供虚实现。实际工作：接收客户程序的请求，对请求的参数进行封装和发送，对返回结果的接收和解封装。特点：自动生成，静态、一旦生成不再改变，与 ORB 的具体实现相关。与框架特点相同。10、框架：一段程序代码，为指定界面编写服务器实现代码的框架。对客户请求解码→定位所请求的对象方法→执行方法→把执行结果编码发送回客户。像一个位于服务器端的存根。同对象的实现必对象适配器打交道。11、静态调用：通过存根和框架的调用。存根为客户提供静态调用方式，框架提供静态实现方式，两者分别充当客户应用程序和服务器应用程序与 ORB 之间的粘合剂。12、动态调用：CORBA 运行客户在运行时选择目标对象，然后动态地调用目标对象上的指定操作。基本含义：构造并发送一个请求，请求中参数说明只有在运行时才知道。目标对象上的操作信息放在对象的接口中，动态调用的接口信息在界面仓库中。理论上将，只要服务器能够提供新的服务和接口，客户就能利用 CORBA 的动态调用机制，去发现服务的远程目标对象和接口，进而调用所需要的方法。CORBA 支持两种用于动态调用的界面：1)动态调用界面 DII–通用存根，支持客户方的动态请求调用，允许不通过存根就能调用请求，用来在运行时刻发现并调用目标对象中的操作；2)动态框架界面 DSI–通用框架，支持服务方的动态对象调用，允许在没有静态框架信息的条件下获得对象实现，允许服务器在运行时刻动态地解释并实现一个调用。13、ORB 提供两个用于存储有关对象信息的服务：1)界面仓库，存储各界面信息的模块；2)实现仓库：存储对象实现信息。14、对象适配器：OA 是联系对象实现与 ORB 本身的纽带。提供了服务器端对象实现与 ORB 核心之间的适配层，为对象实现提供了大部分所需的 ORB 功能。与 5 个组件打交道：ORB 核心、动态框架接口、静态框架、实现仓库、对象实现。对不同的对象实现需求会有不同的对象适配器；不同对象适配器支持不同对象类型，每种编程语言都需要一个相应的对象适配器。**作用：** 1)对象引用的生成和解释 2)对象实现的注册 3)根据对象引用找到它对应的对象实现(定位)4)服务器进程的激活 5)对象的激活与撤销 6)对象向上调用。15、互操作通信协议 GIOP，规定用于支持 ORB 之间互操作的对象引用格式 IOR，同时给出了一个通用的互操作体系结构：1)ORB 之间的直接互操作(相同域中)2)ORB 之间基于桥的互操作(不同域中)。域指一个范围，在其中的对象具有公共的特征，遵从公共的规则。16、GIOP：通用的 ORB 互通协议。客户端服务器通过 ORB 交互，客户放的 ORB 和服务器放的 ORB 通过 GIOP 通信。它规定了两个实体：客户和服务器 ORBs 之间的通信机制。GIOP 是互操作体系结构的基础，定义了用于 ORB 之间通信的一种标准传输语法和一组消息格式。设计尽可能简单，开销最小，同时又具有最广泛的适用性和可扩展性，以适应不同的网络。GIOP 由三部分组成：1)公共数据表示：包括简单数据类型和构造数据类型。2)GIOP 消息格式：规定 Client 和 Server 两个角色之间要传输的消息格式。主要包括 Request 和 Reply 两种消息。3)GIOP 消息传递：主要规定在任何面向连接的网络传输层上的一些操作规则。GIOP 为工作在具体传送协议上的实施协议规定了编码方式和消息格式的框架。在实现时必须映射到具体的传输层协议或者特定的传输机制之上。17、IOR 可操作对象引用：包括所有客户与服务器联系所需的各种信息(如 CORBA 服务器对象进程的 IP 地址和 TCP 端口等)。ORB 将通过 IOR 在网络上唯一标识被分布的对象的消息。**作用：** ORB 使用对象引用来确定目标对象的位置。IOR 可用于点多个不同的 ORB 之间确定对象的位置。



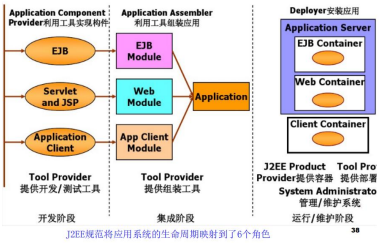
J2EE 的目标：①提供平台无关的、可移植的、支持并发访问和安全的；②在服务器端分布式应用的构造形式，包括了 Java Servlet、JSP、EJB 等多种形式；③编程思想是组件和容器；4~9：applets、servlets、JSP、JNDI、JDBC、Java IDL；⑩(J*T*A)Transaction 服务指事务(交易)管理，具有：原子性、一致性、隔离性和持久性；JAR 打包 EJBs、Application Clients；WAR 打包 JSPs，HTML，GIF，JPG，Servlet class；EAR 打包 JARS 和 WARs；打包需提供相应的 Deployment Descriptor(DD，XML 格式)

服务器端构件---EJB



1. 客户端利用JNDI查找EJB Home。
2. 客户端利用EJB Home的create方法创建一个 EJB。
3. 本地EJB Home通知容器创建一个远程 EJB构件。
4. 容器向客户端返回EJB Object stub。
5. 客户端调用EJB Object上的方法，调用被转发到远程的EJB完成。

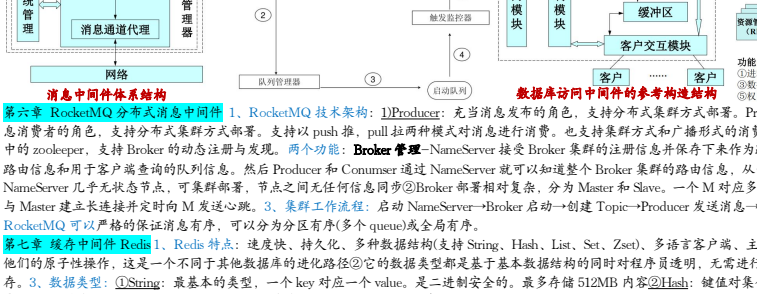
J2EE应用开发中的角色

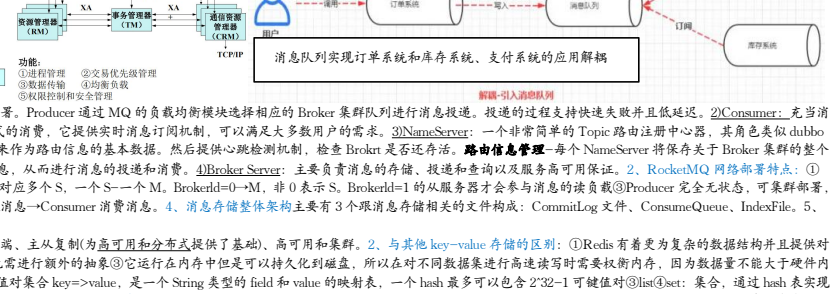


38 J2EE规范将应用系统的生命周期映射到了6个角色

第四章 J2EE 分布式应用程序平台和规范 1、J2EE 中的构件：1)客户端构件：Applets、Application Clients 2)服务器端构件：web 构件(Servlets 和 JSPs)、EJBs。2、J2EE 中的服务：服务是 J2EE 应用构件所使用的功能。由 J2EE 平台提供高实现，分为 Service API(开发时使用)、运行时服务。3、J2EE 中的通信：通信指支持协作构件之间的通信，由容器提供，容器由 J2EE 平台提供高提供。4、J2EE 应用中的构件：1)客户端构件–Applets：java 类，GUI 组件，一般运行在 Web 浏览器中，也可以运行在支持 applet 编程模型的应用中(如 Java SDK 中的 appletviewer)。在 J2EE 应用中一般用来提供用户界面。2)客户端构件–Application Clients：Java 程序，一般具有图形界面。可以直接访问 J2EE 中间层的所有构件，直接调用 EJB 提供的服务。一般基于浏览器的客户端只能直接访问 Web 构件的服务。3)服务器端构件–Servlets：Java 类，运行在服务器端。不需要图形界面接收 HTTP 请求，动态生成 HTTP 响应。4)服务器端构件 JSP：HTML 文档，但是潜入了 JSP 特定的 tag，在页面中加入了 Java 代码，动态生成页面的内容。在服务器端，JSP 页面被编译成 Java Servlet 执行。5、J2EE 中的服务：1)Service API–JNDI：JNDI 用于在网络中查找/定位构件或资源，支持在程序中存取带名字的 Java 对象。是一种通用的目录服务，目前有多种可用的目录服务，如果编写一个应用时使用了某种特定的目录服务，很难迁移到另一种目录服务上。JNDI 屏蔽了不同目录服务之间的差异，JNDI service API 和具体的目录服务无关。2)JDBC：提供与厂商无关的数据库连接。提供一种通用的方法用来查询、更新关系型数据库表，并把数据库操作的结果转换成 Java 数据类型。一个 JDBC 驱动支持完成 3 件事情(跨平台)：建立与数据库的连接；向数据源发送查询和更新的语句；处理结果。JDBC2.0 包含了一个内嵌的数据库连接池，增强了应用的数据库无关性。3)JTA：是支持分布式事务处理的标准 API，与实现无关。需要一种两阶段提交协议来实现，因为分布式事务可能会跨越多个组件、多台机器，并可能设计多个数据源。如 WebSpace 中。利用一种支持两阶段提交的数据库驱动来实现 JTA。4)Java Mail：定义了一组在程序中用来调用邮件功能的 API，这组 API 屏蔽了真正的邮件服务所使用的协议。5)Java IDL：是 Java 实现的 CORBA 规范，支持异质对象的连接与互操作性。

第五章 主流中间件工作原理及实现 1、消息中间件的产生：传统通信必备的条件：①发送和接收应用程序同时在线②通过网络能同时通信，发送者和接受者需知道相互间程序的调用接口。实际情况：①应用程序并不总是同时运行②网络并不总是可用和可靠的③在所有者域对应用程序的改变，要求在其他域也作相应改变，不切实际。2、消息中间件 MOM：依据消息传送或消息队列的原理来工作。能够简化应用之间数据的传输，提供可靠的、跨平台的消息传输手段，实现应用程序之间的协同。支持同步和异步两种通信模式，其中异步通信模式是基于消息队列转发机制的。缺点：不支持程序控制的传递。3、消息：实质上是一个由用户定义的数据结构，由头信息和体信息组成。头信息是对消息结构的描述，对整条信息起控制作用，含消息的属性及相关的系统消息，如消息标识、消息类型、目的队列名、日期时间等。体信息主要是消息的应用数据，是应用程序通信的数据，其具体语义是由通信双方事先约定。**消息种类：**①请求消息 request–除了发送数据，该消息要求对方一定要应答②应答消息 reply–用于请求消息的回应③通知消息 report–单向而不需要应答的消息。**MOM 的传递模型：**①点对点模型 PTP②发布–订阅模型 pub/sub。4、队列管理器：负责创建和删除队列并控制队列的属性。由消息路由和排队模块、消息通道代理模块和系统管理模块组成。消息路由和排队模块负责消息传递的方向。消息通道代理模块 MCA 负责消息的底层传递和网络通道的故障恢复。系统管理模块管理和维护中间件系统。5、触发机制：①一条消息到达需触发的应用消息队列②队列管理器根据触发类型决定是否构成一次事件触发条件③当触发条件满足时，队列管理器创建一条触发消息，并将它发送到启动队列④触发监控器从启动队列中读出触发消息⑤发一条激活命令给触发队列相关的应用程序⑥当应用程序被启动后，从消息队列取走消息并执行相应的操作。6、触发类型：每条消息触发、多条消息触发和特定优先级消息触发。7、消息中间件功能：无连接消息传递，消息优先化，有保障的消息传递，事务处理消息，动态队列创建，消息路由，不同硬件系统的集成，跨软件平台的支持。8、数据库访问中间件：指一切连接应用程序和数据库的软件。**功能：** 同时管理多个客户机连接的多线程，可以接收不同的厂家中立协议，可以用一组管理数据操作的业务规则进行编程，集中处理密集任务和将数据表达抽象到最高层、分开客户机应用程序与数据库管理系统，可以异步提供当前数据表或行的状态给客户机。**访问方式：**①直接访问方式–应用程序像访问本地数据库一样可以直接访问和更新位于远端的数据库②数据库复制方式–客户端缓存和服务器缓存。**常见的数据库访问中间件：** ODBC、OLE DB、JDBC、数据库引擎和数据库网关。**优点：**①移植性好–中间件封装了各种与平台有关的细节，使更换操作系统和通信协议等底层的配置无需改变应用程序代码②集成方便–能非常容易地集成到应用开发环境中，无需关的代码改动③易于扩充–与邮政速和整体升级只要保持对外接口不变就不会影响到系统的其他部分，在功能上对应用程序实现了透明性④使用简单–对各种数据源使用统一的访问方式，降低了用户参与程度。缺点：需要大量的数据通信。9、事务中间件 TPM：也称为交易中间件，是在开放系统环境下提供保证交易完整性和数据一致性的一种环境平台，是针对复杂环境下分布式应用的速度和可靠性要求而实现的。作用：提供高效的处理能力；简化应用系统开发；可以有效完成多服务进程和负载均衡；能保证分布式交易的完整性；提高系统的容错能力；减少系统的运行、维护工作量。





订单系统：用户下单后，该系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功
库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息进行库存操作

第六章 RocketMQ 分布式消息中间件 1、RocketMQ 技术架构：1)Producer：充当消息发布的角色，支持分布式集群方式部署。Producer 通过 MQ 的负载均衡模块选择相应的 Broker 集群队列进行消息投递。投递的过程支持快速失败并且低延迟。2)Consumer：充当消息消费者的角色，支持分布式集群方式部署。支持以 push 推，pull 拉两种模式对消息进行消费。也支持集群方式和广播形式的消费，它提供实时消息订阅机制，可以满足大多数用户的需求。3)NameServer：一个非常简单的 Topic 路由注册中心器，其角色类似 dubbo 中的 zookeeper，支持 Broker 的动态注册与发现。两个功能：**Broker 管理**–NameServer 接受 Broker 集群的注册信息并保存下来作为路由信息的基本数据。然后提供心跳检测机制，检查 Brokrt 是否还存活。**路由信息管理**–每个 NameServer 将保存关于 Broker 集群的整个路由信息和用于客户端查询的队列信息。然后 Producer 和 Conumser 通过 NameServer 就可以知道整个 Broker 集群的路由信息，从而进行消息的投递和消费。4)Broker Server：主要负责消息的存储、投递和查询以及服务高可用保证。2、RocketMQ 网络部署特点：①NameServer 几乎无状态节点，可集群部署，节点之间无任何信息同步②Broker 部署相对复杂，分为 Master 和 Slave。一个 M 对应多个 S，一个 S←→一个 M。BrokerId=0→M，非 0 表示 S。BrokerId=1 的从服务器才会参与消息的读负载③Producer 完全无状态，可集群部署，与 Master 建立长连接并定时向 M 发送心跳。3、集群工作流程：启动 NameServer→Broker 启动→创建 Topic→Producer 发送消息→Consumer 消费消息。4、消息存储整体架构主要有 3 个跟消息存储相关的文件构成：CommitLog 文件、ConsumeQueue、IndexFile。5、RocketMQ 可以严格的保证消息有序，可以分为分区有序(多个 queue)或全局有序。

第七章 缓存中间件 Redis 1、Redis 特点：速度快、持久化、多种数据结构(支持 String、Hash、List、Set、Zset)、多语言客户端、主从复制(为高可用和分布式提供了基础)、高可用和集群。2、与其他 key–value 存储的区别：①Redis 有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径②它的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象③它运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。3、数据类型：①String：最基本的类型，一个 key 对应一个 value。是二进制安全的。最多存储 512MB 内容②Hash：键值对集合 key=>value，是一个 String 类型的 field 和 value 的映射表，一个 hash 最多可以包含 2^32−1 可键值对③list④set：集合，通过 hash 表实现④Zset：有序集合。4、Redis 键(key)：Redis 键命令用于管理 redis 的键。基本语法：redis 127.0.0.1:6379> COMMAND KEY_VALUE。5、Redis 发布订阅：是一种消息通信模式，发送者(pub)发送消息，订阅者(sub)接收消息。6、Redis 事务：可以一次执行多个命令，并且带有以下 3 个重要保证：①批量操作在发送 EXEC 命令前被放入队列缓存②收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行③在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中。一个事务从开始到执行会经历 3 个阶段：开始事务、命令入队、执行事务。单个 Redis 命令的执行是原子性的，但 Redis 没有在事务上增加任何维持原子性的机制，所以 Redis 事务的执行并不是原子性的。事务可以理解为一个打包的批量执行脚本，但批量指令并非原子化的操作，中间某条指令的失败不会导致前面已做指令的回滚，也不会造成后续的指令不做。7、Redis 应用场景：①缓存：现在几乎是所有中大型网站都在用的必杀技，合理的利用缓存不仅能够提升网站访问速度，还能大大降低数据库的压力。Redis 提供了键过期功能，也提供了灵活的键淘汰策略②排行榜：Redis 提供的有序集合数据类构能实现各种复杂的排行榜应用③计数器：为保护数据实时效，每次浏览都得+1，并发量高时如果每次都请求数据库操作无疑是种挑战和压力，Redis 提供的 incr 命令来实现计数器功能，内存操作，性能非常好，非常适用于这些计数场景④分布式会话：集群模式下，在应用不多的情况下一般使用容器自带的 session 复制功能就能满足，当应用增多相对复杂的系统中，一般都会搭建以 Redis 等内存数据库为中心的 session 服务，session 不再由容器管理，而是由 session 服务及内存数据库管理。⑤社交网络：点赞、踩、关注/被关注、共同好友等是社交网站的基本功能，社交网站的访问量通常来说比较大，而且传统的关系数据库类型不适合存储这种类型的数据，Redis 提供的哈希、集合等数据结构能很方便的的实现这些功能。⑥消息系统：消息队列是大型网站必用中间件，如 ActiveMQ、RabbitMQ、Kafka 等流行的消息队列中间件，主要用于业务解耦、流量削峰及异步处理实时性低的业务。Redis 提供了发布/订阅及阻塞队列功能，能实现一个简单的消息队列系统。另外，这个不能和专业的消息中间件相比。

补充 1、开发基于 CORBA 的应用：开发 idl 文件、idlj、开发 S 端、开发 C 端、启动 orbd 服务、启动 server 服务、启动 client