**第一章 软件体系结构概论**

**1. 软件危机的表现和原因**

表现：◎软件成本日益增长 ◎开发进度难以控制

◎软件质量差 ◎软件维护困难

原因：◎用户需求不明确 ◎缺乏正确的理论指导

◎软件规模越来越大 ◎软件复杂度越来越高

*如何克服软件危机*

*构件（定义、构件管理、构件重用）*

***构件****是指语义完整、语法正确和有可重用价值的单位软件，是软件重用过程中可以明确辨识的系统；结构上，它是语义描述、通讯接口和实现代码的复合体。*

***连接件****是软件体系结构的一个组成部分，它通过对构件之间的交互规则的建模来实现构件之间的连接。与构件不同，连接件不需要编译。*

*软件体系结构（定义、意义、应用现状）*

**第二章 软件体系结构建模**

*建模的种类[5]*

◎ 结构模型 ◎ 框架模型 ◎ 动态模型

◎ 过程模型 ◎ 功能模型

**1. “4+1”模型**

从5个不同的视角包括逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件体系结构。每一个视图只关心系统的一个侧面，5个视图结合在一起才能反映系统的软件体系结构的全部内容。

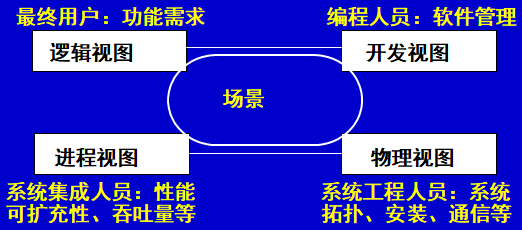
逻辑视图（ Logical View ），设计的对象模型（使用面向对象的设计方法时）。

过程视图（ Process View ），捕捉设计的并发和同步特征。

物理视图（ Physical View ），描述了软件到硬件的映射，反映了分布式特性。

开发视图（ Development View ），描述了在开发环境中软件的静态组织结构。

架构的描述，即所做的各种决定，可以围绕着这四个视图来组织，然后由一些用例（use cases）或场景(scenarios)来说明，从而形成了第五个视图。



*核心模型（图）*

*软件过程：需求分析、建立体系结构、设计、实现、测试*

**第三章 软件体系结构风格**

*风格定义*

*经典风格*

**1. 客户/服务器风格**

**2. 浏览器/服务器风格 （两者逻辑特点 优点 缺点）**

*公共对象请求代理体系结构（CORBA架构的技术规范、特点）*

**第四章 并发计算**

1. 高并发度对网站性能的影响(不考)

**第五章 分布式计算架构**

**1. 分布式系统的概念、优点和缺点**

定义：一个支持由多个独立的计算单元组成的硬件系统的计算平台。

多个自治节点

资源可能无法直接访问。

软件的并发

多个控制/多个失败

*中间件层*

*定义：中间件是可以实现进程间间通信的软件，它提供了一个API，将应用程序代码与底层网络通信格式和协议（FAPs）隔离开来。*

*三种中间件系统：面向事务的中间件、面向消息的中间件、面向对象的中间件*

*面向事务的中间件：支持分布式计算，涉及数据库应用程序*

*面向消息的中间件：支持分布式组件间可靠的异步通信*

*面向对象的中间件：基于面向对象的范例，主要支持分布式组件间的同步通信*

*最流行的面向对象的中间件：CORBA、DCOM、DotNET、EJB。*

*远程过程调用RPC：*

*RPC允许在不同的硬件和操作系统平台上调用操作。*

*+提供与本地过程调用相同的机制，但在进程间级别。*

*+支持通用接口定义语言(IDL)。*

*RPC机制是由RPC软件提供的，RPC软件可以透明地处理所涉及的所有步骤。RPC的功能软件包括:*

*+位置服务器的功能,并和并发请求。*

*+参数传递和数据表示。*

*+故障管理*

*+安全管理*

*RPC软件提供了会话和表示层的实现。注意，传输层(下面)是作为一个套接字实现的。*

**2. CORBA（通用对象请求代理架构）技术标准**（对象请求代理如何连接需求端和服务端、名录服务过程和原理）

**第六章 Web服务**

**1. Web Service的基本层次结构（概念、层次结构）**

**XML：**可扩展标记语言，标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。可扩展[标记语言](https://baike.baidu.com/item/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)，[标准通用标记语言](https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E9%80%9A%E7%94%A8%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)的子集，是一种用于标记电子文件使其具有结构性的[标记语言](https://baike.baidu.com/item/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)。可扩展[标记语言](https://baike.baidu.com/item/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)，[标准通用标记语言](https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E9%80%9A%E7%94%A8%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)的子集，是一种用于标记电子文件使其具有结构性的[标记语言](https://baike.baidu.com/item/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)。可扩展[标记语言](https://baike.baidu.com/item/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)，[标准通用标记语言](https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E9%80%9A%E7%94%A8%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)的子集，是一种用于标记电子文件使其具有结构性的[标记语言](https://baike.baidu.com/item/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)。

**2. SOAP**（消息格式、协议绑定、服务交换协议）

**3. WSDL**

三个基本属性、六个元素、服务接口定义、服务实现定义、元素与属性的映射关系、四种消息交换的方式-异同点-描述角度）

XFire—不涉及

**第七章 面向服务体系架构SOA**

1. **分布式系统的特征**（与SOA关系）

定义： 分布式系统属组件分布在网络计算机上且通过消息传递进行通信和动作协调的系统

特征：并发性、缺乏全局时钟、故障独立性

产生原因：构造和使用分布式系统的主要动力来源是资源共享

产生动机：(1) 信息交换；(2) 资源共享；(3) 通过重复提高可靠性；(4) 通过并行化提高性能；(5) 通过专门化简化设计；(6) 问题本身的特点决定。

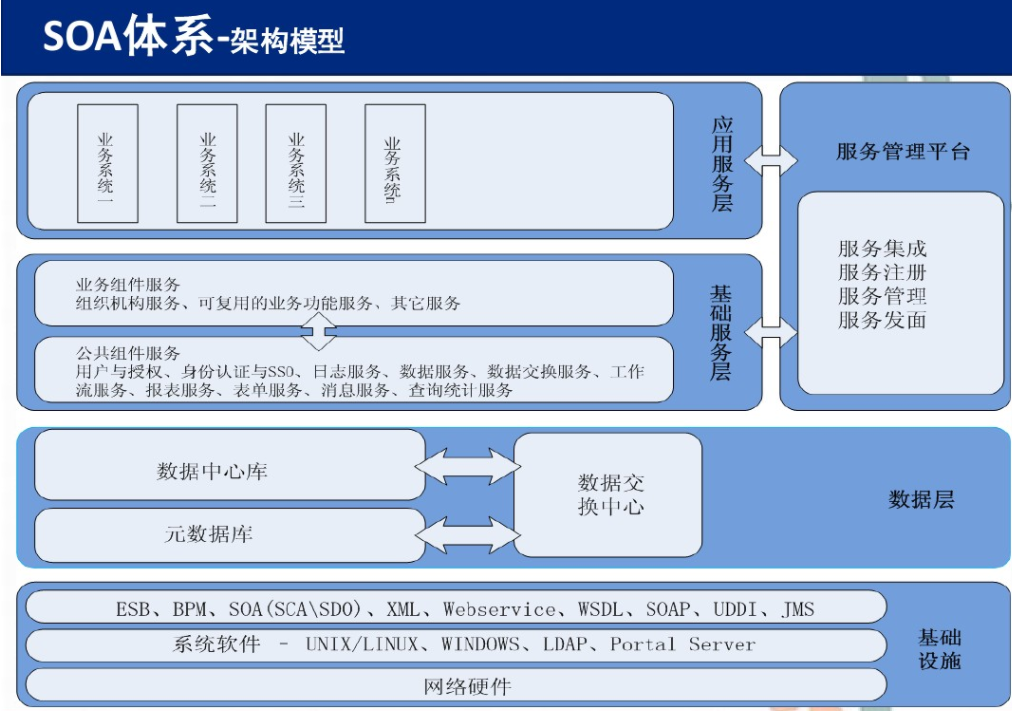
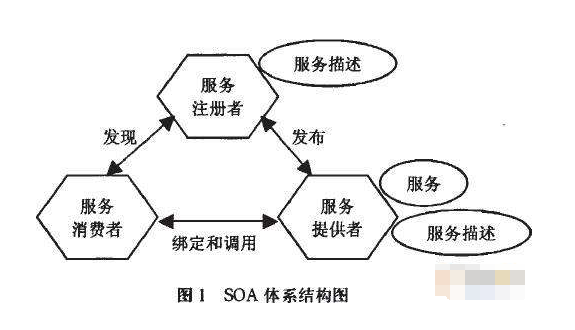
与SOA的关系：SOA是面向服务的架构，将业务逻辑提炼成共用服务，对外提供服务，也不一定就用webservice通信，也可以是socket、EJB等等。目标就是建成一个服务组件的“超市”，供别人自行选用。而分布式，主要还是从部署的角度，将应用按照访问压力进行归类，主要目标是充分利用服务器的资源，避免资源分配不均，譬如流行的三层架构等。

在分布式的环境中，将各种功能都以服务的形式(SOA)提供给最终用户或者其他服务。如今，企业级应用的开发都采用面向服务的体系架构来满足灵活多变，可重用性高的需求。

*技术挑战*

*Web服务（定义、视角）*

1. **面向服务体系架构SOA**（3种角色、3种角色关系、相互间的操作）



***定义****：是一种体系结构风格，它将应用程序的不同功能单元—服务(service)，通过服务间定义良好的接口和契约(contract)联系起来。接口采用中立的方式定义，独立于具体实现服务的硬件平台、操作系统和编程语言，使得构建的系统中的服务可以使用统一和标准的方式进行通信。*

***SOA核心理念****：让企业应用彻底摆脱面向技术的解决方案的束缚，轻松应对企业商业服务变化、发展的需要。业务决定服务，服务决定技术。*

***SOA与传统架构的区别：***

*a.传统软件架构为了保持自由开放的特性，对软件的执行没有限制，一旦软件开始执行，就会获得相应的系统资源，并且认为软件的每个组成部分都是可靠的*

*b.在SOA中，软件的执行过程中不再像过去那样将要在本地安装所有的部分，而是动态地从网上根据需要请求服务，软件的各个部分分散在Internet中，需要时进行动态地组合。这种分散的软件结构，将安全责任由传统的软件使用者全部负担，转换为由软件的使用者和提供者共同负担，因而明确了责任，降低了单个用户的安全风险*

*c.传统软件包是被编写为独立的（self-contained）软件，即在一个完整的软件包中将许多应用程序功能整合在一起。实现整合应用程序功能的代码通常与功能本身的代码混合在一起。*

*d.SOA旨在将单个应用程序功能彼此分开，以便这些功能可以单独用作单个的应用程序功能或“组件”*

*Web服务策略（目标、框架、安全性4个基本要求）*

*Web服务管理*

*Web服务中Qos主要需求*

1. **SOA基本特征**

*SOA优势：*

*一，SOA可通过互联网服务器发布，从而突破企业内网的限制，实现与供应链上下游伙伴业务的紧密结合。通过SOA架构，企业可以与其业务伙伴直接建立新渠道，建立新伙伴的成本得以降低。*

*二，SOA与平台无关，减少了业务应用实现的限制。要将企业的业务伙伴整合到企业的“大”业务系统中，对其业务伙伴具体采用什么技术没有限制。*

*三， SOA具有低耦合性特点，业务伙伴对整个业务系统的影响较低。在企业与各业务伙伴关系不断发生变化的情况下，节省的费用会越来越多。*

*四， SOA具有可按模块分阶段进行实施的优势。可以成功一步再做下一步，将实施对企业的冲击减少到最小。*

*五， SOA的实施可能并不具有成本显著性。*

**第八章 云计算架构**

*云计算解释*

*云计算分类*

1. **现有IT系统的主要问题**（云计算提出原因）

a.数据孤岛。按照传统的方式建设的方式必然产生数据孤岛，能够被共享的业务核心数据被分散到各个应用，并且各个应用的开发商很多不一样的甚至是同一个开发商由于各种问题导致数据编码标准不一致，数据不一致和不可信等问题；这些问题导致不能够形成完整和精准的数据视图，导致很难进行数据分析和支撑业务流程的运营。

b.应用竖井。应用系统集成复杂度和难度很高，没有好的设计规范和架构以及代码质量，基本上代码重用程度都很低；从界面集成、数据集成、应用集成这三个方向来做都很困难），甚至出现无法集成。

c.对于一个性能要求很高的系统（例如高负载量和高数据量的系统，和将来会做的全省大集中系统和新一代的应用系统），我们会利用大量的硬件资源和相应的集群技术等技术进行相应的数据分区、集群和均衡负载来应对峰值的访问情况。要设计和很好地实施如此复杂的集群牵涉到非常多的技术，所以我会经常看到当系统压力上来的时候会遇到各种各样的问题并且这方面的问题很严重。这些应用系统按照传统的项目建设方式，即每建设一套业务应用系统基本上都要购买新的硬件设备（例如服务器、存储等）和平台系统软件（例如数据库、中间件等）。带来了大量的硬件资源的浪费和占用大量的空间、电力的浪费、运维成本的提高.对于一些高负载和高数据量的应用系统，我们对硬件资源的要求是按照此应用系统高峰值的需要来进行购买以应用此应用的需要，但是此应用的高峰期是具有周期性的

d. 目前没有一个完整地、 端到端的整体IT系统监控解决方案，无法对我们整体IT系统（例如主机、存储、网络、数据库、应用服务器、应用）的运行情况进行一个很好的监控，通过监控产生的数据来指导我们优化我们的IT系统

e. 随着将来市场需求的变化、国家政策的变化、新技术和新方式的出现等一系列的变化，如果我们在技术、架构、设计、应用方式、管理不进行优化（甚至重构）的话，我们将会面临更大的痛苦：

我们在将来肯定会建设一些应用系统，来满足我们的员工、供应商、分销商、大众的需要。比如建设新一代的电子商务系统和新的Web 2.0应用等，对于这种新一代的应用它的特点就是负载量大（例如高并发量和高数据量），数据类型复杂，业务种类繁多和需求变化快。即我们需要把应用的访问方式扩展到新一代智能手机上;所以我们有些应用的功能需要运行在Google Android 和Apple iOS(例如iPhone 、iPAD )上。并且由于移动设备本身存在的局限性，相应大量的业务应用的执行能力大量移到后台。那就意味着我们应用的特点更是负载量大，数据类型复杂,业务种类繁多和需求变化快。

响应国家的节能减排建设低碳社会的号召，我们都知道我们现在传统的建设数据中心和应用的思路带来很大的浪费，即带来占用更多的存储空间，浪费更大的电能。所以我们需要利用相应的硬件技术和软件技术让我们的数据中心的PUE达到在1.5左右，并且降低对空间的占用和降低运维成本和提高运维效率。

随着3G/4G LTE技术的发展，我们需要将移动互联网和桌面互联网连接在一起，真正实现广义上的互联互通。将我们的应用与移动互联网，将开发者与使用者，将社会与个人，将企业与个人，将个人与个人之间的关系全部打通，全面提升信息化水平。

*云计算解决的问题：*

*大规模的存储空间，用于存储海量的数据。*

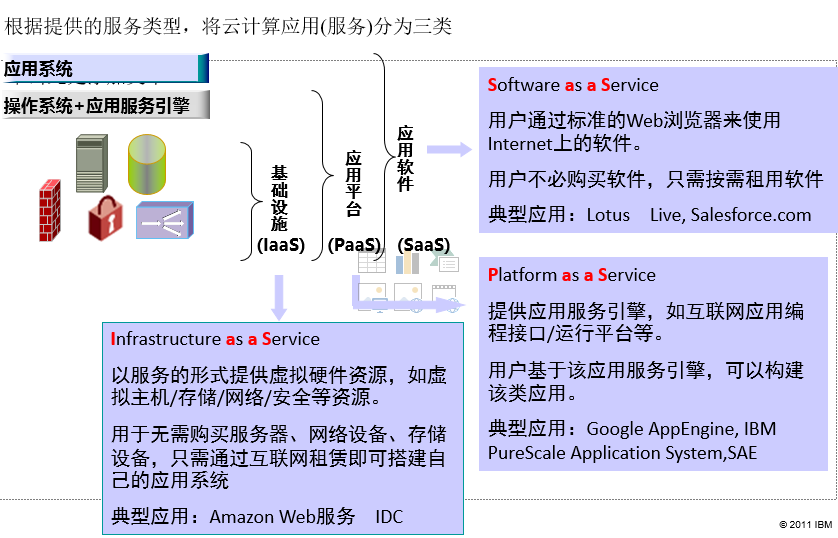
*随着业务的发展，新的数据源源不断地增加，存储空间需要相应扩大，即：扩展性，scalability。*

*系统的硬件设备必须便宜，通常使用大宗产品（commodity），譬如PC，或者价格便宜，中等性能的Dell server。*

*在设计这个集群的时候，必须保证不能因为个别机器死机，导致整个系统的崩溃。也就是系统的稳定性要好，reliability。*

1. **云计算架构**（三种不同的层次-每个架构针对什么样的用户、特征）

根据提供的服务类型，将云计算应用(服务)分为三类：基础设施层(IaaS);应用平台层(PaaS);应用软件层(SaaS). 这三个层结合起来，就形成了典型的云计算的SPI模型。



计算包括三个层次的服务:基础架构即服务（IaaS），平台即服务（PaaS）和软件即服务（SaaS）。

IaaS通过互联网提供数据中心、基础架构硬件和软件资源，还可以提供服务器、操作系统、磁盘存储、数据库和/或信息资源。

Paas则提供了基础架构，软件开发者可以在这个基础架构之上建设新的应用，或者扩展已有的应用，同时却不必购买开发、质量控制或生产服务器。

SaaS是最为成熟、最出名，也是得到最广泛应用的一种云计算。它是一种软件分布模式，在这种模式下，应用软件安装在厂商或者服务供应商那里，用户可以通过某个网络来使用这些软件，通常使用的网络是互联网。

IaaS、PaaS和SaaS之间的关系可从两个角度来看：从用户体验角度而言，它们之间的关系是独立的，因为它们面对不同类型的用户；而从技术角度而言，它们并不是简单的继承关系(SaaS基于PaaS，而PaaS基于IaaS)，因为首先SaaS可以是基于PaaS或者直接部署于IaaS之上，其次PaaS可以构建于IaaS之上，也可以直接构建在物理资源之上。

IaaS、PaaS和SaaS这三种模式都采用了外包的方式，以减轻企业负担，降低管理、维护服务器硬件、网络硬件、基础架构软件和应用软件的人力成本。从更高的层次上看，它们都试图去解决同一个商业问题——用尽可能少甚至是为零的资本支出，获得功能、扩展能力、服务和商业价值。

在云计算实施的前期或者在很多场景的时候，主要关注点在于应用的可靠运行、快速开发和部署、机器资源的充分利用、以及方便的运维等问题；对于这个时候我们应该定位于主要采用Iaas云计算架构（即很依赖于硬件虚拟化技术）和部分采用Pass云计算架构来解决.

1.重点采用Iaas 云计算架构中的硬件虚拟化技术等技术（服务器虚拟化、网络虚拟化、存储虚拟化）以提高硬件的利用率、降低机房占用空间和功耗。

2.快速和方便地给应用提供应用所需要的服务器资源（VM)、网络资源、存储资源。

3.快速和方便地给应用提供应用所需要依赖的平台软件资源，例如数据库系统（DB2)、J2EE应用服务器（WAS)、WEB 服务器（IHS)等。

4.快速和方便地自动地把应用部署到相应的硬件环境中。

5. 硬件虚拟化技术（例如VMWare，PowerVM,xen等）的能力需要在这体现。

注意：

1.对于在这种情况下，每台服务器所具有的CPU core数目和内存数量越大越好。不要弄一些性能较差的机器干这种事情。

2. SSD和大内存对数据库性能的提升是很明显的。

3. 这个就是我们通常所说的以大变小

*数据既是服务*

*数据开发平台分成以下几大部分：*

*数据开放平台数据层：即需要相应的数据库服务器存储我们企业中以前积累和将来积累的一系列数据经过加工后的有用、完整、精确的数据，例如主数据、元数据、业务数据等。这个数据的形成就需要利用相应的ETL技术、数据仓库技术等技术。*

*数据规则层：即需要建立一系列的数据所有权规则、数据共享规则、数据可用性规则来限制数据的创建、利用等；反应到数据开放平台这个整体就是一系列的算法/代码。*

*数据逻辑层：为了让在数据开放平台中的数据能够给各个业务应用利用，我们需要开发大量的业务逻辑来处理和加工数据开放平台中的数据。这样的话就会形成一系列的支撑应用（例如统一用户管理应用、主数据管理应用、数据分析应用（例如提供报表等等））。*

*服务API层：如何让各个业务应用更好和方便快捷地使用我们的数据呢？我们需要提供一系列简单的服务API给各个应用使用。*

*统一用户管理中心（UMC）*

*用户认证和用户授权*

*应用集成和流程整合*

*大并发和大数据良的挑战*

*NoSQL（特点、BA数据挖掘分析系统特点、引入Hadoop）*

**第九章 大数据**

1. **大数据的概念**（了解为主、四个维度-四个V）

4V特征：Volume: 数据量异常庞大，一般达到PB量级

Variety: 数据呈异构化，数据来源呈多样性

Velocity: 数据处理要求时效性

Value: 单个数据无价值，但大规模数据拥有巨大价值

*互联网企业在大数据处理遇到的难题*

*离线批处理模型、内存计算模型、交互计算模型的区别*

* 离线批处理模型： GFS/HDFS/NoSQL/MapReduce，业界主流模式，技术成熟，数据规模大，但时效性差*

* 大内存计算模型： Hana,MemCloud，计算速度快，但需要大规模集中式内存结构支持（若为分布式则受制于网络传输速度），技术成熟度不够*

* 交互式计算模型： Google有Dremel, PowerDrill,Apache有Drill通过data locality/in-memory buffer/columnar data structure等技术来提高计算速度，以现有计算架构和软件技术为基础，具可行性；但目前技术分散，缺乏一个集成平台*

**第十章 大数据计算体系**

**1. 大数据计算体系**

组成：数据存储系统、数据处理系统、数据应用系统

1. **数据存储系统**（非结构化数据、结构化数据；两种存储系统的区别；非关系型数据库是否分区）

非结构化数据是数据结构不规则或不完整，没有预定义的数据模型，不方便用数据库二维逻辑表来表现的数据。包括所有格式的办公文档、文本、图片、XML, HTML、各类报表、图像和音频/视频信息等等。

非结构化数据库是指其字段长度可变，并且每个字段的记录又可以由可重复或不可重复的子字段构成的数据库，用它不仅可以处理结构化数据（如数字、符号等信息）而且更适合处理非结构化数据（全文文本、图象、声音、影视、超媒体等信息）。

结构化数据可以通过固有键值获取相应信息，且数据的格式固定，比如数据库。

HDFS/GFS优缺点

NoSQL数据库性能需求——CAP理论

NoSQL数据库事务要求——BASE法则

四类数据库比较

同一数据访问接口类型：ODBC、JDBC、DAL、UDAL

1. **数据处理系统**

数据分析算法：回归分析类算法、决策树类算法、贝叶斯类算法

计算处理模型：MapReduce、图并行计算、交互式处理、流计算

计算平台与引擎

**4. 数据应用系统**

**第十一章 Hadoop生态系统**

1. **Hadoop总体架构**
2. **HDFS文件系统**

主从节点自己的功能部署

HDFS存储结构优势

命名空间的管理

第二名称节点作用

容错回复机制

机架感知

心跳检测机制

1. **分布式存储架构（HBase）**

region、store、HFile

HBase表特征、表索引、表值形式

二次索引表机制

1. **Hadoop资源管理与作业调度**

三个组件（功能、服务、业务流程）：

Zookeeper 业务流程

Oozie提供工作流的位置、节点概念

YARN部署方式

**第十二章 MapReduce计算模型**

*分布式并行计算系统*

1. **MapReduce计算架构**

分治法——MapReduce的本质思想

批量化计算的设计模式

四大组件、和物理节点之间的映射）（图）

主要任务

JobTracker、TaskTracker和Oozie 的区别

分片定义、与Block的相互映射

Map数目（了解）

三个工作阶段

**第十三章 交互式处理**

一次写入、多次读取

1. **Dremel数据模型与存储结构**

逻辑上嵌套式结构

物理存储结构（行和列存储的存储效率，列更好）

P9 还原嵌套结构（R值和D值）

1. **并行查询**
2. **Drill**

整体架构、运算过程与Dremel相似

**第十四章 流计算**

1. **流计算模型**

P5 两者模型的区别（延时、吞吐率）

P8 （计算响应延时）

2. Storm计算架构

3. Storm工作机制

映射关系（Spark中的流计算不涉及）

**第十五章 内存计算模型**

1. **内存计算模型**

为解决什么问题

1. **分布式缓存体系**

缓存服务器概念、非永久存储媒介

永久性存储器是指不用继续通电也能持久保存数据的东东.比如 U盘、硬盘等。

非永久性存储器是一直通电的情况下才保存数据的东东，一断电就丢失数据。比如，CPU的高速缓存、内存等。

内存技术（数据压缩、列存储结构、分区、指插入差异数据）

Memchache工作流程、内存管理

1. **内存数据库**

整体结构

两个架构的比较、优缺点（全内存架构和读写分离架构）

1. **MemCloud计算架构（不涉及）**

计算题：

系统吞吐率（system throughput）是指单位时间内系统处理的数据量或完成的任务数。

系统响应时延（response delay）是基于客户端来计算的，它等于客户端向服务器端提交一个任务到计算结果返回之间的时间间隔。



假设客户端一个任务完成的时间可分为如图15-6所示的三部分：

delay time = network latency + server latency + network latency

= 2 \* network latency + server latency

其中，network latency是网络传输时间以Ln表示；server latency是服务器处理一条数据所需时间，以Ls表示。

对于Native Stream Processing System而言：

客户端系统延迟 delay time = 2Ln + Ls

服务器端系统吞吐率 throughput = 1/(delay time) = 1/(2Ln + Ls)

如果我们采用一次把10条数据打成一个包（batch）发送处理的方式，网络传输时间不变，仍然为2 \* network latency，但服务器处理时间变为10 \* server latency，因为需要处理10条数据。

对于这种Micro-batch Processing System而言：

客户端系统延迟 delay time = 2Ln + 10Ls

服务器端系统吞吐率 throughput = 10/(delay time) = 10/(2Ln + 10Ls) = 1/(0.2Ln + Ls)

比较两种模式的delay time和throughput可知，只要Ln > 0和Ls > 0，则有：

2Ln + Ls < 2Ln + 10Ls

1/(2Ln + Ls) < 1/(0.2Ln + Ls)

由此可知，两种模式比较，Native Stream Processing System的时间延迟性好于Micro-batch Stream Processing System，但系统吞吐率则低于Micro-batch Stream Processing System。另外，Native Stream Processing System容错性成本也较高，因为逐条数据备份或恢复的开销大于成批次的处理成本。