**高级网络计算—秦臻**

**题型：6个主观题（15-20分）**

|  |  |  |
| --- | --- | --- |
| 1：C/S架构、B/S架构、IT鸿沟、中间件概念 | 2：网格概念、五层沙漏模型、OSGA | |
| 3：云计算与网格技术的差别、Map-Reduce、Hadoop基本原理、数据一致性理论 | | |
| 4：P2P的概念、DHT的概念、Chord算法、Pastry算法 | | 5：CDN网络概念、DNS重定向技术 |
| 6：物联网体系结构 | |  |

1：C/S架构、B/S架构、IT鸿沟、中间件概念

|  |  |
| --- | --- |
| 概念：C/S 架构是一种典型的两层架构，其全程是Client/Server，即客户端服务器端架构，其客户端包含一个或多个在用户的电脑上运行的程序，而服务器端有两种，一种是数据库服务器端，客户端通过数据库连接访问服务器端的数据；另一种是Socket服务器端，服务器端的程序通过Socket与客户端的程序通信。 C/S 架构也可以看做是胖客户端架构。因为客户端需要实现绝大多数的业务逻辑和界面展示。这种架构中，作为客户端的部分需要承受很大的压力，因为显示逻辑和事务处理都包含在其中，通过与数据库的交互（通常是SQL或存储过程的实现）来达到持久化数据，以此满足实际项目的需要。  优点：1.C/S架构的界面和操作可以很丰富。 2.安全性能可以很容易保证，实现多层认证也不难。 3.由于只有一层交互，因此响应速度较快。  缺点：1.适用面窄，通常用于局域网中。 2.用户群固定。由于程序需要安装才可使用，因此不适合面向一些不可知的用户。 3.维护成本高，发生一次升级，则所有客户端的程序都需要改变。 | 概念：B/S架构的全称为Browser/Server，即浏览器/服务器结构。Browser指的是Web浏览器，极少数事务逻辑在前端实现，但主要事务逻辑在服务器端实现，Browser客户端，WebApp服务器端和DB端构成所谓的三层架构。B/S架构的系统无须特别安装，只有Web浏览器即可。 B/S架构中，显示逻辑交给了Web浏览器，事务处理逻辑在放在了WebApp上，这样就避免了庞大的胖客户端，减少了客户端的压力。因为客户端包含的逻辑很少，因此也被成为瘦客户端。  优点：1）客户端无需安装，有Web浏览器即可。 2）BS架构可以直接放在广域网上，通过一定的权限控制实现多客户访问的目的，交互性较强。 3）BS架构无需升级多个客户端，升级服务器即可。  缺点：1）在跨浏览器上，BS架构不尽如人意。 2）表现要达到CS程序的程度需要花费不少精力。 3）在速度和安全性上需要花费巨大的设计成本，这是BS架构的最大问题。 4）客户端服务器端的交互是请求-响应模式，通常需要刷新页面，这并不是客户乐意看到的。（在Ajax风行后此问题得到了一定程度的缓解） |

如何描述IT鸿沟？

|  |  |
| --- | --- |
| 图2 | C:\Users\ASUS\AppData\Roaming\Tencent\Users\1245998947\QQ\WinTemp\RichOle\XA0GL1MP5GS$ZY[A%OQMY}M.png  缩短开发时间的方法：提高重用性！ |
| IT鸿沟指的是当业务不断发展的时候，现有的IT支持平台的发展不能满足业务的需要，而当IT支持平台发展到可以支持业务的需要的时候，业务又有新的需求和发展，这样业务与IT支持之间持续存在一个鸿沟。 | 存在IT鸿沟的一个重要的原因就是企业中的信息孤岛问题。信息孤岛指相互之间在功能上不关联互助、信息不共享互换以及信息与业务流程和应用相互脱节的计算机应用系统。 |

|  |  |
| --- | --- |
| 提高重用性（针对公共服务、共性服务）  与数据库有关,与分布式有关,与安全可靠有关 | IT鸿沟的解决办法——EAI（企业应用集成）  C:\Users\ASUS\AppData\Roaming\Tencent\Users\1245998947\QQ\WinTemp\RichOle\}FRLYX4OS9PUIR2(L`]DOKL.png  IT鸿沟的解决办法——EAI（企业应用集成）  应用整合层之上是流程整合层，它将不同的应用系统连接在一起，进行协同工作，并提供商业流程管理的相关功能，包括流程设计、监控和规划，实现业务流程的管理。  到了最上端的用户交互层，则是为用户在界面上提供一个统一的信息服务功能入口，通过将内部和外部各种相对分散独立的信息组成一个统一的整体，保证了用户既能够从统一的渠道访问其所需的信息，也可以依据每一个用户的要求来设置和提供个性化的服务。 |
| IT鸿沟的解决办法——EAI（企业应用集成）  EAI(Enterprise Application Integration)企业应用集成，具体到技术层面上的划分，我们认为一套完整的 EAI 技术层次体系应该包括应用接口层，应用整合层，流程整合层和用户交互层四个大的层面。  EAI 技术层次体系最下面的一层是应用接口层，它要解决的是应用集成服务器与被集成系统之间的连接和数据接口的问题。  再往上去就是应用整合层，它要解决的是被集成系统的数据转换问题，通过建立统一的数据模型来实现不同系统间的信息转换。 |

中间件概念

|  |  |  |  |
| --- | --- | --- | --- |
| 中间件（Middleware）  中间件位于C/S计算模型的操作系统、网络和数据库之上，应用软件的下层，是连接两个独立应用程序或独立系统的软件；  中间件为上层的应用软件提供运行与开发的环境，帮助用户灵活、高效地开发和集成复杂的应用软件；  分布式应用软件可借助中间件在不同的技术之间共享资源，相连接的系统即使它们具有不同的接口，但通过中间件相互之间仍能交换信息；  中间件屏蔽了底层分布式环境的复杂性和异构性，简化了分布式应用程序的开发，使分布式应用程序的健壮性、可扩展性、可重用性更好。 | | 中间件定义:中间件（Middleware）是一种软件，处于系统软件（操作系统和网络软件）与应用软件之间，它能使处于应用层中的各应用成分之间实现跨网络的协同工作（也就是互操作），这是运行各应用软件之下所涉及的“系统结构、操作系统、通信协议、数据库和其他应用服务”各不相同。 | |
| 中间件的作用  提供了通道功能，可保证应用系统数据的可靠传输；  起着桥梁作用，达成异构应用资源之间的互联互通；  是分布式应用集成开发框架，促成企业分布式应用的有机集成；  作为平台，可有效实现分布应用的分阶段部署；  起着过滤网作用，可有力保护关键信息的安全。 | |
| 中间件所提供的最基本的系统服务  通信服务：用户屏蔽低层网络通信接口，为高层应用提供可靠的消息通信和远程访问；  名字服务：提供全局的名字服务空间，屏蔽低层实体分布的具体物理位置；  存储服务：提供全局的一致数据访问，集成低层分布的局部文件系统和数据库；  分布式事务服务：提供分布式事务处理，包括事务管理和调度、负载均衡和失败恢复等；  安全服务：提供存取访问控制，在中间件层次保证分布式系统的安全性。 | |
| 中间件的优越性  企业应用方面：缩短应用开发周期、减少项目开发风险、应用程序质量可维护、增加产品吸引力；  应用程序方面：透明地同其他应用程序交互、与运行平台提供的网络通信服务无关、具有良好的可用性和可靠性、具有良好的可扩展性。 | |
| 中间件技术的主流标准  DNA（Distributed Internet Applications）标准。微软在推出Windows2000系列操作系统平台的基础上，扩展了分布式计算模型以及改造Back Office系列服务器端分布计算产品后发布的分布式计算体系结构和规范。  DNA架构由三层组成：表示层，可以由浏览器组成的客户，也可以是具有一定商业逻辑处理能力的胖客户；商业逻辑层，由COM/COCOM+/COM等组件组成；数据层，可以是数据库，也可以是外部应用程序。 | 中间件技术的主流标准  CORBA（Common Object Request Broker Architecture）标准。它是OMG（Object Management Group）组织基于众多开放系统，制订的一种标准的面向对象应用程序体系规范。  CORBA标准主要分为三个层次：对象请求代理，使得在分布式环境下的对象之间能够透明的发送请求和接收响应；对象服务，为使用和实现对象而提供的基本对象集合，这些服务（事件服务、生命周期服务、关系服务等）应独立于应用领域；公共设施，是能被多个应用共享的一系列服务。 | | 中间件技术的主流标准  J2EE（Java 2 Platform, Enterprise Edition）标准。它是Sun公司把企业用户、厂商及技术专家聚集到JCP（Java Community Process）开发的企业级Java API标准，以此来简化在瘦客户级环境下的应用开发。  J2EE核心是一组技术规范与指南，其中所包含的各类组件、服务架构及技术层次，均有共同的标准及规格，让各种依循J2EE架构的不同平台之间，存在良好的兼容性，解决过去企业后端使用的信息产品彼此之间无法兼容，企业内部或外部难以互通的窘境。 |

2：网格概念、五层沙漏模型（瓶颈）、OSGA

|  |  |  |  |
| --- | --- | --- | --- |
| 什么是网格计算？  网格计算在每个节点上安装中间件进行彼此间的连接，使用网格计算的用户需要参与提供计算机资源。网格计算利用互联网上计算机的CPU闲置资源处理能力来解决大型计算问题。  优点：提供高效的资源共享服务，平衡多台计算机上的负载，可把程序放到最适合的位置上。  目的：高性能计算。  网络：覆盖网络/应用层网络。 | | 网格计算定义  定义：网格计算是动态多机构虚拟组织中的一个协调的共享资源和解决问题的过程（Ian Foster定义）。  网格必须满足三个条件：  在非集中控制环境下协同使用资源；  使用标准的、开放的、通用的协议（标准化）；  提供非凡的服务。 | |
| 网格计算定义  网格将网络上分布的资源聚合为一体，建立计算和数据处理的通用系统支持平台，实现资源共享和联合工作。  资源：计算、数据、存储、设备、软件和人员等；  本质：资源的共享和协同，统一和横向设计；  技术：虚拟组织之间的资源共享和协同工作技术；  方法：建立广泛的资源共享和协同的标准协议；  最终目标：数字信息基础设施。 | |
| 网格分类  计算网格：分布式计算体系结构。  信息网格：支持用户有效共享和管理信息资源，为用户提供透明的信息服务。  知识网格（语义网格）：使用户有效的共享和管理知识资源，为用户提供知识服务。 | |
| 网格体系结构 以协议为中心的5层沙漏结构  以服务为中心的开发网络服务体系结构OSGA (Open Grid Services Architecture) | | | |
| 网格体系结构——5层沙漏结构  基本思想和概念：共享、互操作性、协议、服务。  共享：深层次、广泛、动态、具有多种形式的有条件受控的共享。  互操作性：共享关系可以跨域不同的平台、语言和编程环境，跨越不同的组织边界、使用策略和资源类型。  协议：为了实现特定的操作而定义的分布式系统元素之间交互的方式以及交互过程中交换的信息结构。  服务：由所使用的协议和所实现的行为来定义。服务抽象掉了与资源相关的信息。 | | | |
| 中心思想：采用以协议为中心的分层结构。  5层沙漏结构根据各组成部分离共享资源的距离，将对共享资源进行操作、管理和使用的功能分散在五个不同的层次中，越向下层就越接近共享的物理资源，与特定资源相关的成分就越多；越向上层就越感觉不到共享资源的细节特征，也就是说上层是更加抽象的共享资源的表示。 | |  | |
| 5层结构中各层协议的数量是不同的，对于最核心的协议，既要能实现上层各种协议向核心协议的映射，同时又要能实现核心协议向下层协议的映射，核心协议在所有支持网格计算的地点都应该得到支持，因此核心协议的数量不应该太多，这样核心协议就形成了协议层次结构中的瓶颈，形成沙漏形状。 | |  | |
| 五层沙漏模型（对内容进行展开）  构造层：提供网格服务的基础硬件，包括：计算、存储、网络等。  连接层：将硬件连接起来，并能方便地查找和使用及为保证使用安全。  资源层：将硬件转换成资源，并为资源提供管理和维护功能。有限的资源构成瓶颈。  汇集层：将所有单个的资源形成一个网格。  应用层：使用资源的用户应用程序。 | | 5层结构应用： | |
| 构造层：其功能是向上提供网格中可共享的资源(物理或逻辑实体)。常用的资源包括处理能力、存储系统、目录、网络资源、分布式文件系统、分布式计算机池、计算机集群等。 | | 是网格中网络事务处理、通信与授权控制的核心协议。构造层提交的各种资源间的数据交换都在这一层的控制下实现。各资源间的授权验诋、安全控制也在这里实现。资源间的数据交换通过传输、路由及名字解析等机制实现。 | |
| 资源层：该层的作用是对单个资源实施控制，与可用资源进行安全握手、对资源做初始化、监测资源运行状况、统计与付费等有关的资源使用数据。 | | 汇聚层：其作用是将资源层提交的受控资源汇集在一起，供虚拟组织的应用程序共享、调用。为了对来自应用层的共享进行管理和控制，汇聚层提供目录服务、资源分配、日程安排、资源代理、资源监测诊断、网格启动、负荷控制、账户管理等多种功能。 | |
| 应用层：这层是网格上用户的应用程序。应用程序通过各层的API调用相应的服务，再通过服务调用网格上的资源来完成饪务。应用程序开发涉及大量的库函数。为了便于网格应用程序的开发，需要创建支持网格计算的库函数。 | |

开放网格体系服务结构OGSA

|  |  |
| --- | --- |
| 基本思想和概念：以服务为中心的模型。  五层沙漏结构试图实现的是资源的共享，OGSA实现的将是对服务的共享。OGSA将一切（计算资源、存储资源、网络、程序、数据库、设备等）都看作网格服务，将网络看作可扩展的网格服务的集合。  OGSA在五层沙漏结构基础上，对Web Services进行了扩展，提出动态服务（即网格服务）的概念。网格服务是一组接口（发现、动态服务创建、生命周期管理、通知等）定义明确、遵守特定惯例的Web Services。 | OGSA的两大支撑服务  Globus：Globus是一种开放结构、开放源码的服务的集合，也是支持网格和网格应用的软件库，该工具包解决了安全、信息发现、资源管理、数据管理、通信、错误检测以及可移植性等问题。  Web Services：建立在Internent和XML基础上的服务协议标准。  两者关系：Web Services的核心是在大的异构网络上将各种应用连接起来，借助于各Web标准（UDDI、WSDL、XML等）将Internet从一个通信网络进一步发展到一个应用平台。 |
|  |

3云计算与网格技术的差别、Map-Reduce、Hadoop基本原理、数据一致性理论

|  |  |  |  |
| --- | --- | --- | --- |
| 云计算和网格计算的区别？  首先，网格计算的思路是聚合分布资源，支持虚拟组织，提供高层次的服务，例如分布协同科学研究等。而云计算的资源相对集中，主要以数据中心的形式提供底层资源的使用，并不强调虚拟组织的概念； 其次，网格计算用聚合资源来支持挑战性的应用，这是初衷，因为高性能计算的资源不够用，要把分散的资源聚合起来；后来逐渐强调适应普遍的信息化应用，特别在中国，做的网格跟国外不太一样，就是强调支持信息化的应用。但云计算从一开始就支持广泛企业计算、Web应用，普适性更强。第三，在对待异构性方面，二者理念上有所不同。网格计算用中间件屏蔽异构系统，力图使用户面向同样的环境，把困难留在中间件，让中间件完成任务。而云计算实际上承认异构，用镜像执行，或者提供服务的机制来解决异构性的问题。当然不同的云计算系统还不太一样，像Google一般用比较专用的自己的内部的平台来支持。 第四，网格计算用执行作业形式使用，在一个阶段内完成作用产生数据。而云计算支持持久服务，用户可以利用云计算作为其部分IT基础设施，实现业务的托管和外包。第五，网格计算更多地面向科研应用，商业模型不清晰。而云计算从诞生开始就是针对企业商业应用，商业模型比较清晰。最后，云计算是以相对集中的资源，运行分散的应用（大量分散的应用在若干大的中心执行）；而网格计算则是聚合分散的资源，支持大型集中式应用（一个大的应用分到多处执行）。但从根本上来说，从应对Internet的应用的特征特点来说，他们是一致的，为了完成在Internet情况下支持应用，解决异构性、资源共享等等问题。 | | | |
| 云计算与网格计算比较  云计算和网格计算在目标、结构和技术方面有共同之处，但也有许多的区别。  相同之处  目标：减少计算成本、增加可靠性、增加灵活性  面临很多共同问题：管理大型设备、定义用户发现、请求和使用资源的方法。  不同之处：业务模型、体系结构、资源管理、编程模型、应用模型、安全模型 | | 业务模型  云计算：面向企业或单个用户，按需使用资源，按使用收费，如电、水、气等的使用方式一样，由单一的公司资源建立云计算环境，提供服务，资源可以使高性能设备，也可以是PC机。（我为人人）  编程模型网格计算：面向工程，集中于解决大型复杂的计算问题，一项任务可由来自于不同成员贡献的资源共同完成，资源一般是高性能的计算、存储设备。（人人为我） | |
| 体系结构  构造层：包括各种计算、存储、网络等硬件资源。  资源层：通过虚拟化计算，抽象为被上层或终端用户使用的资源。比如，虚拟的计算机、集群、文件系统、数据库等。  平台层：在资源层之上，整合了一系列的工具、中间件、服务等，提供应用部署平台服务。  应用层：包括云环境下运行的各种应用程序。 |  | | |
| 资源管理——计算模型  网格：批量计算模型，不支持交换式应用  云计算：资源同时被所有用户共享，支持延迟敏感的应用 | | | 资源管理——数据位置  云计算：能够将数据分块，存储于不同的位置，可以将任务调度到靠近数据存储位置的地方执行，提高效率  网格计算：数据存储依赖于共享的文件系统，难于利用数据位置信息，需要改进任务调度器为数据意识的。  关键需要的是计算和数据资源管理的综合，在访问模式中利用数据位置信息使数据移动的数据量最小，改善终端应用的性能和可扩展性。 |
| 资源管理——数据模型  网格：在网格环境下有专门的数据网格处理数据集中式的应用，出现了虚拟数据的概念  云计算：网络计算是数据、云计算和客户端计算的结合 | | |
| 资源管理——虚拟化  网格：每一个组织自己控制和管理自身的资源，对虚拟化的依赖程度不如云计算高  云计算：虽然不是必要的，但在实践中几乎成为不可缺少的元素  编程模型  网格：MPI, Globus Tookit, WSRF 等  云计算：MapReduce, Hadoop等 |
| 安全模型  网格：更为严格，网格安全的核心是单点登录、代理和数据隐私等。  云计算：更为简单，具有更弱的安全性。采用网络管理为用户创建账户，并允许用户通过邮件在不安全和未加密的通信中重设密码和接收新的密码。新用户通过邮件能方便容易地使用云服务。  安全模型：云计算安全是云计算研究中的重要课题，普通云用户面临的安全风险主要体现在以下方面：  授权的用户接入：企业敏感数据要求只有特权用户才能访问以保证机密性。  数据隔离：保证不同用户之间的数据达到完全隔离。  数据恢复：有良好的容错机制以保证系统出错时数据不丢失。 | | |
| 应用模型  网格：支持多种不同的应用（高性能计算、高吞吐量计算等）。  云计算：能处理和网络计算类似的应用，由于网络环境的不确定性，在高性能计算方面，云计算要稍逊一筹。支持交互式应用。  浏览器和Web 2.0技术在用户和网格与云计算交互的过程中扮演重要角色。 |

云计算的先行者Google的云计算平台能实现大规模分布式计算和应用服务程序，平台包括Hadoop框架、 MapReduce分布式处理技术、分布式的文件系统GFS、结构化的BigTable存储系统以及Google其他的云计算支撑要素。

MapReduce分布式处理技术、分布式的文件系统GFS、结构化的BigTable存储系统是Google的三个核心技术。

|  |  |  |  |
| --- | --- | --- | --- |
| 描述Map-Reduce（的整个过程）  目标  任务分配（过程）  Master的责任（任务）  Map worker的责任（任务）  Reduce worker的责任（任务） | | | MapReduce可解决哪些算法问题  MapReduce待处理的数据集可以分解成许多小的数据集,而且每一个小数据都可以完全并行地进行处理，因此不能解决不可分拆的计算任务，或者相互间有依赖关系任务，如Fibonacci函数: Fk+2 = Fk + Fk+1 。  MapReduce可解决的基本算法：各种全局数据相关性小、能适当划分数据的计算任务。如：分布式排序、关系代数操作（选择，投影，求交集、并集，连接，成组，聚合等）、矩阵向量相乘、矩阵相乘、词频统计(word count)、文档倒排索引等。  MapReduce可解决的复杂算法：Web搜索引擎（网页爬取、网页排序、搜索算法）、Web访问日志分析（分析和挖掘用户在Web上的行为）、数据/文本统计分析（如专利文献引用分析和统计）、图算法并行化宽度优先搜索（最短路径问题）、机器学习、数据挖掘等。 |
| MapReduce的编程思想——分而治之 | | |
|  | | |
|  | | | |
| MapReduce分布式处理技术  MapReduce定义了Map和Reduce两个抽象的编程接口，由用户编程实现：  Map: 对一组数据元素进行某种重复式的处理  Reduce: 对Map的中间结果进行某种进一步的结果整理 | | MapReduce分布式处理技术——Map端  map: (k1, v1) → list(k2, v2)  输入：键值对(k1, v1)表示的数据  处理：文档数据记录（如文本文件中的行，或数据表格中的行）将以“键值对”形式传入map函数；map函数将处理这些键值对，并以另一种键值对形式输出处理的一组键值对中间结果list(k2, v2)。  输出：键值对(k2, v2)表示的一组中间数据  备注: list(k2, v2) 表示有一个或多个键值对组成的列表  MapReduce分布式处理技术——Reduce端  reduce: (k2, list(v2)) → list(k3, v3)  输入：由map输出的一组键值对list(k2, v2) 将被进行合并处理，同样主键下的不同数值合并会到一个list(v2)中，故reduce的输入为(k2, list(v2))。  处理：对传入的中间结果列表数据进行某种整理或进一步的处理，并产生最终的某种形式的结果输出list(k3, v3)。  输出：最终输出结果list(k3, v3)。 | |
| 各个map函数对所划分的数据并行处理，从不同的输入数据产生不同的中间结果输出；  各个reduce各自并行计算，各自负责处理不同的中间结果数据集合；  进行reduce处理之前，须等到所有的map函数做完，并且在进入reduce前会对map的中间结果数据进行整理(Shuffle)，保证将map的结果发送给对应的reduce；  最终汇总所有reduce的输出结果即可获得最终结果 | |
| 73978_1_1_l.jpg | | | |
|  | | | |
| MapReduce分布式处理技术——实现机制  (1) MapReduce函数库首先把输入文件分成M块，每块大概16MB到64MB。接着在集群的机器上执行处理程序。  MapReduce算法运行过程中有一个主控程序，称为master。主控程序会产生很多作业程序，称为worker。并且把M个map任务和R个reduce任务分配给这些worker，让它们去完成。  (2) 被分配了map任务的worker读取并处理相关的输入(这里的输入是指已经被切割的输入小块splite)。它处理输入的数据，并且将分析出的键/值(key/value)对传递给用户定义的reduce()函数。map()函数产生的中间结果键/值(key/value)对暂时缓冲到内存。  (3) map()函数缓冲到内存的中间结果将被定时刷写到本地硬盘，这些数据通过分区函数分成R个区。这些中间结果在本地硬盘的位置信息将被发送回master，然后这个master负责把这些位置信息传送给reduce()函数的worker。 | (4) 当master通知了reduce()函数的worker关于中间键/值(key/value)对的位置时，worker调用远程方法从map()函数的worker机器的本地硬盘上读取缓冲的中间数据。当reduce()函数的worker读取到了所有的中间数据，它就使用这些中间数据的键(key)进行排序，这样可以使得相同键(key)的值都在一起。  (5) reduce()函数的worker根据每一个中间结果的键(key)来遍历排序后的数据，并且把键(key)和相关的中间结果值(value)集合传递给reduce()函数。reduce()函数的worker最终把输出结果存放在master机器的一个输出文件中。  (6) 当所有的map任务和reduce任务都已经完成后，master激活用户程序。在这时，MapReduce返回用户程序的调用点。  (7) 当以上步骤成功结束以后，MapReduce的执行数据存放在总计R个输出文件中(每个输出文件都是由reduce任务产生的，这些文件名是用户指定的)。通常，用户不需要将这R个输出文件合并到一个文件，他们通常把这些文件作为输入传递给另一个MapReduce调用，或者用另一个分布式应用来处理这些文件，并且这些分布式应用把这些文件看成为输入文件由于分区(partition)成为的多个块文件。 | | |

Google Hadoop架构

|  |  |
| --- | --- |
| 在Google发表MapReduce后，2004年开源社群用Java搭建出一套Hadoop框架，用于实现MapReduce 算法，能够把应用程序分割成许多很小的工作单元，每个单元可以在任何集群节点上执行或重复执行。  此外，Hadoop 还提供一个分布式文件系统GFS（Google file system），支持大型、分布式大数据量的读写操作，其容错性较强。而分布式数据库（BigTable）是一个有序、稀疏、多维度的映射表，有良好的伸缩性和高可用性，用来将数据存储或部署到各个计算节点上。 | 在架构中MapReduce API提供Map和Reduce处理、GFS分布式文件系统和BigTable分布式数据库提供数据存取。基于Hadoop可以非常轻松和方便完成处理海量数据的分布式并行程序，并运行于大规模集群上。  MEL)ZDLYUSQ_KLDMGPH`5LA |

数据一致性理论

|  |  |  |  |
| --- | --- | --- | --- |
| CAP理论  其中字母“C”、“A”、“P”分别带表以下特征：  强一致性（Consistency）。系统在执行某项操作后仍然处于一致状态。在分布式系统中，更新操作执行成功后所有用户都应该读取到最新的值，这样的系统被认为具有强一致性。  可用性（Availability）。每一个操作总是能够在一定的时间内返回结果。  分区容错性（Partition Tolerance）。分区容错性可以理解为系统在存在网络分区的情况下仍然可以接受请求（满足一致性和可用性）。 | | CAP理论是在分布式环境中设计和部署系统时需要考虑的三个重要的系统需求。根据CAP理论，数据共享系统只能满足这三个特性中的两个，而不能同时满足三个条件。因此系统必须在这三个特性之间做出权衡。 | |
| 放弃P：如果想避免分区容错性问题的发生，一种做法就是将数据放到一台机器上。虽然无法100%地保证系统不会出错，但不会碰到由分区带来的负面效果。当然，这个选择会严重影响系统的可扩展性。  放弃A：相对于放弃“分区容错性”，其方面就是放弃可用性。一旦遇到分区容错故障，那么受到影响的服务需要等待数据一致，因此等待期间系统无法对外提供服务。  放弃C：这里所说的放弃一致性，并是不完全放弃数据的一致性，而是放弃数据的强一致性，而保留数据的最终一致性。以网络购物为例，对只剩最后一件库存商品，如果同时接收了两份订单，较晚的订单将被告知商品售罄。 | |
| CAP理论  在网络分区场景下，如发生网络故障，分区G1发送的数据消息不能传达到另一分区G2，数据将处于不一致状态，不能满足一致性要求。采用如阻塞、加锁、集中控制等技术来保证数据的一致性，但同步操作必定要消耗一定时间，必然会影响到系统的可用性和分区容错性，尤其在网络规模较大的时候。 | |
| 分布式系统一般通过复制数据来提高系统的可靠性和容错性，并且将数据的不同副本存放在不同的机器上，由于维护数据副本一致性代价很高，许多系统采用弱一致性来提高性能。不同的一致性模型：  强一致性、弱一致性、最终一致性 | | 最终一致性：是弱一致性的一种特例，在这种一致性下系统保证用户最终能够读取到某操作对系统特定数据的更新。  此种情况下，“不一致性窗口”的大小依赖于交互延迟、系统负载，以及复制技术中replica的个数（可以理解为master/slave模式中，slave的个数）。  DNS系统是在最终一致性方面最出名的系统，当更新一个域名的IP以后，根据配置策略以及缓存控制策略的不同，最终所有客户都会看见最新值。  终一致性模型根据其提供的保证可以划分为多个模型：  因果一致性（Causal Consistency）：假如有相互独立的A、B、C三个进程对数据进行操作。进程A对某数据进行更新后并将该操作通知给B，那么B接下来的读操作能够读到A更新的数据值。但是由于A没有将该操作通知给C，那么系统将不保证C一定能够读取到A更新的数据。  读自写一致性（Read Your Own Writes Consistency）：用户更新某个数据后，读取该数据时能够获取其更新后的值，而其他用户读取该数据时则不能保证读取到最新值。  会话一致性（Session Consistency）：值读自写一致性被限制在一个会话范围内，也就是说提交更新操作的用户在同一个会话里读取该数据时能够保证数据是最新的。  单调读一致性（Monotonic Read Consistency）：指用户读取某个数据值，后续操作不会读取到该数据更早版本的值。  时间轴一致性（Timeline Consistency）：要求数据的所有副本以相同顺序执行所有更新操作，也称为单调写一致性（Monotonic Write Consistency）。 | |
| 强一致性：要求无论更新操作在哪个数据副本上执行，之后所有的读操作都要获得最新的数据。对于单副本数据来说，读写操作在同一数据上执行，容易保证强一致性。对于多副本数据来说，需要使用分布式事物协议（如Paxos）。  弱一致性：在这种一致性下，用户读到某一操作对系统特定数据的更新需要一段时间。这段时间成为“不一致性窗口”。 | |

|  |  |
| --- | --- |
| 事务是用户定义的一个数据库操作序列，这些操作要么全不做，要么全做，是一个不可分割的单位，ACID是事务所具有的特性。  原子性（Atomicity）：事务中的操作要么都做，要么都不做。  一致性（Consistency）：系统必须始终处于强一致状态下。  隔离线（Isolation）：一个事务的执行不能被其他事务所干扰。  持续性（Durability）：一个已提交的事务对数据库中数据的改变是永久性的。  ACID特性是传统关系型数据库中事务管理的重要任务，也是恢复和并发控制的基本单位。 | BASE方法通过牺牲一致性和孤立性来提高可用性和系统性能，其中BASE分别带表：  基本可（Basically Availability）：系统基本能够运行、一直提供服务。  软状态（Soft-state）：系统不要求保持强一致性。  最终一致性（Eventually consistency）：系统需要在某一刻后达到一致性要求。 |
| 数据一致性实现技术  分布式系统在不同节点的数据采用什么技术保证一致性，取决于应用对系统一致性的需求，在关系型数据管理系统中一般会采用悲观的方法（如加锁），这些方法代价比较高、对系统性能影响较大，而在一些强调性能的系统中则会采用乐观的方法。  Quorum系统NWR策略、两阶段提交协议、时间戳策略、PAXOS算法 |

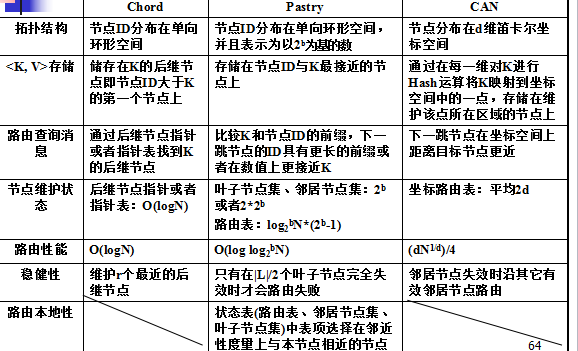
4：P2P的概念、DHT的概念、Chord算法、Pastry算法

|  |  |
| --- | --- |
| 什么是P2P？  对等网络(Peer-to-Peer Networks)是分布式系统和计算机网络相结合的产物，在应用领域和学术界获得了广泛的重视和成功，被称为“改变Internet的一代网络技术”。 | P2P: Peer to Peer对等网络。  peer指网络结点，在行为上是自由的——任意加入、退出，不受其它结点限制，匿名；在功能上是平等的——不管实际能力的差异；在连接上是互联的——直接/间接，任两结点可建立逻辑链接，对应物理网上的一条IP路径。  充分利用网络带宽、节点资源，提高工作效率。 |
|  | P2P有几代（区别）【没找到确切的答案】  第二代P2P  没有集中的目录服务器，但是拓扑结构有意义。这个结构意味着P2P网络拓扑被紧紧的控制。比如： Mesh, Ring , d-dimension Torus, K-ary tree。  使用DHT技术，有较好的可伸缩性和查询效率。提供负载均衡和确定性的搜索保证。但是容错性或弹性不好，尤其是在恶意攻击下。文件不是被 随机地而是以特定的位置放置，这样使得连续的查询更加容易满足。  它使用精确的定位算法和特殊的路由协议使得搜索效率提高。支持精确查询不能支持模糊查询。 |
| 为什么基于离散的hashtable不受限制【没找到确切的答案】  。 |
|  | DHT原理  Hash函数可以根据给定的一段任意长的消息计算出一个固定长度的比特串，通常称为消息摘要（MD：Message Digest），一般用于消息的完整性检验。  Hash函数有以下特性：  给定 P，易于计算出 MD（P）  只给出 MD（P），几乎无法找出 P  无法找到两条具有同样消息摘要的不同消息  Hash函数  MD5：消息摘要长度固定为128比特  SHA-1：消息摘要长度固定为160比特 |
| DHT的概念、主要思想  什么是DHT：把网络上资源的存取像Hashtable一样，可以简单而快速地进行put、get，该思想的诞生主要是受第一代P2P（Napster）网络的影响。DHT更强调的是资源的存取，而不管资源是否是一致性的。 |
|  | 将内容索引抽象为<K, V>对  K是内容关键字的Hash摘要：K = Hash(key)  V是存放内容的实际位置，例如节点IP地址等  所有的<K, V>对组成一张大的Hash表，该表存储了所有内容的信息  每个节点都随机生成一个标识(ID)，把Hash表分割成许多小块，按特定规则（即K和节点ID之间的映射关系）分布到网络中去，节点按这个规则在应用层上形成一个结构化的重叠网络  给定查询内容的K值，可以根据K和节点ID之间的映射关系在重叠（Overlay）网络上找到相应的V值，从而获得存储文件的节点IP地址  定位(Locating)  节点ID和其存放的<K, V>对中的K存在着映射关系，因此可以由K获得存放该<K, V>对的节点ID  路由(Routing)  在重叠网上根据节点ID进行路由，将查询消息最终发送到目的节点。每个节点需要到其邻近节点的路由信息，包括节点ID、IP等  网络拓扑  拓扑结构由节点ID和其存放的<K, V>对中的K之间的映射关系决定  拓扑动态变化，需要处理节点加入/退出/失效的情况 |
|  |
|  |

|  |  |
| --- | --- |
| Chord  目标Hash节点IP地址－>m位节点ID(表示为NID)，Hash内容关键字－>m位K(表示为KID)。<K, V>存放在顺时针方向的离该节点最近的下一个节点。Chord中每个节点上都有一张局部的指针表，表中存放2i 个节点信息（i=0，1，……，n-1）。  每个节点仅维护其后继节点ID、IP地址等信息  查询消息通过后继节点指针在圆环上传递  直到查询消息中包含的K落在某节点ID和它的后继节点ID之间  这种方法的时间效率是log（n），相当于进行折半查找。 | Chord：Hash表分布规则    简单查询过程  基于指针表的扩展查找过程 |
| 采用环形拓扑(Chord环)  其核心思想就是要解决在P2P应用中遇到的基本问题：如何在P2P网络中找到存有特定数据的节点  Chord使用一致性哈希作为哈希算法，在Chord协议中将其规定为SHA-1。 |
| 应用程序接口  Insert(K, V)：将<K, V>对存在放到节点ID为Successor(K)上  Lookup(K)：根据K查询相应的V  Update(K, new\_V)：根据K更新相应的V  Join(NID)：节点加入  Leave()：节点主动退出 |
| Chord：小结：1算法简单、2负载平衡：所有的节点以同等的概率分担系统负荷，从而避免某些节点负载过大、3可扩展：查询过程的通信开销和节点维护的状态随着系统总节点数增加成对数关系(O (log N)数量级) 、4可用性：要求节点根据网络变化动态更新查询表，能够及时恢复路由关系，使得查询可靠地进行。5拓扑失配问题 | |

|  |  |
| --- | --- |
| Pastry  Hash节点IP地址－>m位节点ID(表示为NID)  Hash内容关键字－>m位K(表示为KID)  NID和KID是以2b为基的数，共有m/b个数位  <K, V>存储在离它最接近的节点上  每个用户维护一个状态表：路由表、邻居节点集、叶子节点集。 | Pastry：概述  考虑网络的本地性，解决物理网络和逻辑网络的拓扑失配问题  基于应用层定义的邻近性度量，例如IP路由跳数、地理距离、往返延时等  节点ID分布采用环形结构 |
| Pastry： Hash表分布规则 | 每个节点维护一个状态表  路由表、邻居节点集、叶子节点集  Pastry：节点维护状态表  路由表R  包括 m/b行，每行包括2b 个表项  第n行与节点ID的前n-1个数位相同，第n个数位不同，取值从0到2b -1，也称n-1数位前缀相同  表中的每项包含节点ID，IP地址等  根据邻近性度量选择距离本节点近的节点  b过大，节点要维护的路由表大，但存储的邻居节点多，在转发时更为精确，b的选择反映了路由表大小和路由效率之间的折衷 |
| 邻居节点集M  存放在真实网络中与当前节点“距离”最近的|M|个节点的信息  “距离” 类似IP路由协议中的距离， 考虑转发跳数、传输路径带宽、QoS等综合因素后所得的转发开销  |M|的典型值为2b或者2×2b  邻居节点集通常不用于路由查询消息，而是用来维护本地性 |  |
| 叶子节点集L  存放在键值空间中与当前节点距离最近的|L|个节点的信息，其中各有一半的节点标识大于或小于当前节点  |L|的典型值为2b或者2×2b  路由时，首先检查叶子节点集 |
| 节点状态表和查询 |
| Pastry：总结  逻辑网络路由跳数O(log2b N)  路由表开销log2b N \*(2b -1)  路由本地性：状态表（路由表、邻居节点集、叶子节点集）中的表项选择在邻近性度量上与本节点相近的节点  稳健性：只有在|L|/2个叶子节点完全失效时才会路由失败 |

基于DHT的结构化P2P比较



CDN网络概念、DNS重定向技术

|  |  |
| --- | --- |
| 什么是CDN：CDN的全称是Content Delivery Network，即内容分发网络。  其目的通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络“边缘”，使用户可以就近取得所需的内容，解决Internet网络拥挤的状况，提供用户访问网站的响应速度。  从技术上全面解决由于网络带宽小、用户访问量大、网点分布不均等原因，解决用户访问网站的响应速度慢的根本原因。 | CDN工作原理  当用户访问某域名上的内容信息时，首先需用通过域名解析系统将该域名转换为存储该内容信息的内容服务器的IP地址，再根据该IP地址访问内容服务器以获取内容信息。如果该域名使用了CDN技术，该域名上的内容信息将会被复制并发布到广泛分布的、位于不同地理位置的CDN网络的边缘节点服务器。用户访问使用CDN技术的域名时，其地理位置附近的CDN网络的边缘节点服务器将向其提供相关内容信息。  传统的未加缓存服务的访问过程  http://pic001.cnblogs.com/images/2011/1/2011111514082838.jpg  使用CDN缓存后的网站访问过程  http://pic001.cnblogs.com/images/2011/1/2011111514091332.jpg |
|  |
|  |
| 传统访问模式 VS CDN访问模式  传统的DNS解析过程在将网站主机域名转换为IP地址时，并不预先判断该服务器是否正常工作；即使该服务器已经宕机不能提供服务了，用户的请求仍将被发往这台服务器，造成服务的中断。  不同网络间的兼容以及不同网络运营商ISP之间的传输瓶颈等问题使得数据的流通受到限制。  互联网以数据包传输为基础，任何一个数据包的丢失或出错都必须重新发送，而平均一个重传过程需要3秒钟，从而导致延迟。并且现有的HTTP协议也有诱发延迟的因素，据调查，完整下载一个网页，需要在用户和服务器之间往返20-100次。  现有的路由技术以路由器工作状态的历史数据为依据来确定当前数据包的传输路径，无法真实反应当前的路由和网络连接状况。这往往会导致数据传输所经过的路径并不是当前的最佳路径。而且，众多的路由器和交换机不但使数据传输的时间延迟增大，还会增加出错的几率，因为任何一个路由器出现问题都会影响到整个传输过程。 | Akamai CDN在全球建立部署了几千个节点，以物理节点距离用户最近而著称；通过对各节点间的路由优化，实现信息的分发；主要提供WEB、下载、流媒体等加速服务。 |
| LimeLight通过在全球最重要的城市建立“超级节点”，且所有“超级节点”间采用自有光纤直连，实现全球多运营商网络的互联互通；如同在全世界建立了一个自有的、具有互联互通的“BGP”网络。  LimeLight在每个“超级节点”部署了成千上万的服务器、存储、网络设备。形成了分布式的全球“信息”存储中心。  通过“超级节点”网络结构的使用，LimeLight能够更好地满足和支持富媒体全球分布式分发，特别适合于为大资料库、大文件、海量观众提供高效率分发服务。 |
| Akamai和LimeLight差异比较 |
| CDN系统的两大技术派系：综合当今众多CDN服务商的技术体系，无不受到当今世界两大CDN服务商，即 Akamai和LimeLight（以下简称LL）的技术架构影响。可以说Akamai和LimeLight的技术架构引领着当今CDN行业的技术发展。 |

|  |  |
| --- | --- |
| CDN关键技术（内容路由技术）  内容路由技术，即CDN实现负载均衡技术。  负载均衡是整个CDN的核心，负载均衡的准确性和效率直接决定了整个CDN的效率和性能。  负载均衡技术将网络的流量尽可能均匀地分配到几个能完成相同任务的服务器或网络节点上进行处理，避免部分网络节点过载而另一部分节点空闲的不利状况，既可以提高网络流量，又可以提高网络的整体性能。  CDN负载均衡系统实现CDN的内容路由功能，将用户导向到CDN网络中最佳的节点。  LimeLight CDN的主要实现方法为IP Anycast。  Akamai CDN的主要实现方法为DNS重定向。 | 首先，域名解析器访问域名xxxx.pcworld.com的域名服务器，请求解析该域名。该域名的域名服务器返回一个CNAME（Canonical NAME）类型的DNS记录。该CNAME类型的DNS记录指示了所请求解析的域名xxxx.pcworld.com的别名a1506.g.akamai.net。其作用是通过返回一个Akamai CDN网络的内部域名，将域名解析请求重定向到Akamai CDN网络的域名服务器，即根Akamai域名服务器。然后，域名解析器访问根Akamai域名服务器，请求解析域名a1506.g.akamai.net。根Akamai域名服务器根据域名解析器的IP地址和网络状况等动态信息，返回CNAME类型的DNS记录，即域名a1506.g.akamai.net的CNAME类型的DNS记录为za.akamaitech.net，将域名解析请求重定向到一级Akamai域名服务器。同样的方法，域名解析器访问一级Akamai域名服务器，请求解析域名za.akamaitech.net。一级Akamai域名服务器也根据域名解析器IP地址和网络状况等动态信息，返回CNAME类型的DNS记录，即域名za.akamaitech.net的CNAME类型的DNS记录为nog.akamai.net，再将域名解析请求重定向到二级Akamai域名服务器。最后，域名解析器访问二级Akamai域名服务器，请求解析域名nog.akamai.net。二级Akamai域名服务器根据域名解析器IP地址和网络状况等动态信息，返回A类型的DNS记录，即Akamai CDN网络边缘节点服务器的IP地址。通常情况下，二级Akamai域名服务器将返回2个IP地址，用户可选择其中的任意一个边缘节点服务器发出访问请求。 |
| Akamai CDN内容路由的主要实现方法为DNS重定向。    用户向域名解析器发送域名解析请求，请求解析某域名，以获取内容服务器的IP地址；域名解析请求中包含了域名解析器的IP地址，并被发送到该域名的域名服务器；  该域名的域名服务器通过返回一个CNAME类的DNS记录，将此域名解析请求重定向到CDN网络的域名服务器；  CDN网络的域名服务器对该域名解析请求进行智能解析，根据域名解析器的IP地址、网络的拥塞状况、各CDN网络边缘节点服务器的负载情况等，将最合适的CDN网络边缘节点服务器的IP地址返回给域名解析器（域名解析器再转发给用户）。 |
| CDN主要特点（优势）  本地Cache加速：提高了企业站点(尤其含有大量图片和静态页面站点)的访问速度，并大大提高以上性质站点的稳定性。  镜像服务：消除了不同运营商之间互联的瓶颈造成的影响，实现了跨运营商的网络加速，保证不同网络中的用户都能得到良好的访问质量。  远程加速：远程访问用户根据DNS负载均衡技术智能自动选择Cache服务器，选择最快的Cache服务器，加快远程访问的速度。  带宽优化：自动生成服务器的远程镜像cache服务器，远程用户访问时从cache服务器上读取数据，减少远程访问的带宽、分担网络流量、减轻原站点WEB服务器负载等功能。  集群抗攻击：广泛分布的CDN节点加上节点之间的智能冗余机制，可以有效地预防黑客入侵以及降低各种D.D.o.S攻击对网站的影响，同时保证较好的服务质量。 |
| CDN应用领域 |

物联网体系结构

|  |  |
| --- | --- |
| 联网的概念分为广义和狭义两方面  广义来讲，物联网是一个未来发展的愿景，等同于“未来的互联网”，或者是“泛在网络”，能够实现人在任何时间、地点，使用任何网络与任何人与物的信息交换，以及物与物之间的信息交换；  狭义来讲，物联网是物品之间通过传感器连接起来的局域网，不论接入互联网与否，都属于物联网的范畴。 | 物联网与互联网的不同  物联网是RFID技术与互联网结合而产生的新型网络，主要解决物品到物品（Thing to Thing，T2T），人到物品（Human to Thing，H2T），人到人（Human to Human，H2H）之间的互连。其中，H2T是指人利用通用装置与物品之间的连接，H2H是指人之间不依赖于个人电脑而进行的互连。  物联网具有与互联网类同的资源寻址需求，以确保其中联网物品的相关信息能够被高效、准确和安全的寻址、定位以及查询，其用户端是对互联网的延伸和扩展，即任何物品和物品之间可以通过物联网进行信息交换和通讯。  因此，物联网又在不同应用领域的专用性、高度的稳定性和可靠性、严密的安全性和可控性等方面有别于互联网。 |
| 物联网的一种定义是：通过射频识别（RFID）、红外感应器、全球定位系统（GPS）、激光扫描器等信息传感设备，按约定的协议，把任何物品与互联网连接起来，进行信息交换和通讯，以实现智能化识别、定位、跟踪、监控和管理的一种网络。 |
| 物联网可划分为一个由感知层、网络层和应用层组成的三层体系。  感知层：包括二维码标签和识读器、RFID标签和读写器、摄像头、CPS（Cyber Physical Systems，网络化物理系统）、各种传感器、传感器网络（指由大量各类传感器节点组成的自治网络，具有自组织、自愈合的特点）、T2T&H2T&H2H终端和传感器网关等，感知层的主要作用是感知和识别物体，采集并捕获信息。  网络层：包括各种通信网络与互联网形成的融合网络，这被普遍认为是最成熟的部分，除此之外还包括物联网管理中心、信息中心等利用网络的能力对海量信息进行智能处理的部分。也就是说网络层不但要具备网络运营的能力，还要提升信息运营的能力。网络层是物联网成为普遍服务的基础设施。  应用层：是将物联网技术与行业专业技术相结合，实现广泛智能化应用的解决方案集。物联网通过应用层最终实现信息技术与行业的深度融合，对国民经济和社会发展具有广泛影响。应用层的关键问题在于信息的社会化共享以及信息安全的保障。 |
| 物联网的内涵主要体现:  互联网特征：对需要联网的物一定要能够实现互联互通的互联网络；  识别与通信特征：纳入物联网的“物”一定要具备自动识别与物物通信的功能；  智能化特征：网络系统应具有自动化、自我反馈与智能控制的特点。  全面感知、可靠传送、智能处理是物联网的基本特征。 |

按五层结构阐述

感知层、接入层、网络层、中间件层、应用层

|  |  |
| --- | --- |
| 知层由各种传感器以及传感器网关构成，包括二氧化碳浓度传感器、温度传感器、湿度传感器、二维码标签、RFID 标签和读写器、摄像头、GPS等感知终端。感知层的作用相当于人的眼耳鼻喉和皮肤等神经末梢，它是物联网获识别物体，采集信息的来源，其主要功能是识别物体，采集信息。  接入层用来读取识别到的物体的信息并将读到的内容连到Internet。 | 可将物联网分成 5 层结构：感知层、 接入层、 网络层、支撑层和应用层。感知层主要完成 数据的采集，一般包括传感器、 RFID 标签、 摄像头等设备。 接入层主要实现将采集到的数据传入到传输网络中去，例如宽带有线网络、 卫星网络，以及一些无线网络如 3G 网络、 4G 网络、WiFi 网络等。网络层主要完成数据的远距离传输，一般由各种私有网络、 互联网或云计算平台等组成。支撑层主要完成数据的处理。应用层是用户与物联网的接口，它要服务于行业，满足行业的需求，实现例如智能交通、 智能物流、 智能环保等方面的智能应用。 |
|  | |