的软件,尤其是微处理器内部的结构 构件:一组基本的构成要素,连接件:这些要素之间的连接关系。	分布式应用借助中间件在不同技术之间共享资源 • 位于客户机和服务	多系统所共有的结构和语义特性,并指导如何将各个模块和子系统	KWIC案例 接收按行排列的数据。重复地把第一个词删除,然后接到行末,把所有行 的各种移位结果按照字母表顺序输出	特征: 数据的可用性决定着处理计算单元是否执行: 系统结构:	过滤器状态: 1.停止状态: 处于待启动状态,外部启动过滤器后,过滤器处于处理状态: 2.处理状态: 正处理输入数据队列中的数据。 3.等待状态: 输入数据队列为空此时过滤器等待,当有新的数据输入	事件系统风格 显式调用:各个构件之间的互动是由显性调用函数或过程完成 调用的过程与次序是固定的、预先设定的。	全广播式: 阀度模块将事件广播到所有的模块,但只有感兴趣的模的。 块才去取事件并触发自身的行为;无目的广播,靠接受者自行决定 是否加以处理或者简单抛弃。
接关系上的限制条件性能:质量 架构公式 = 构件 + 连接件 + 拓扑结构 + 约束 + 质量	器的操作系统之上,管理计算机资源和网络通讯。连接两个独立应用 程序或独立系统的软件,即使它们具有不同的接口。通过中间件,应用 程序可以工作于多平台或OS 环境。Mail System不是中间件	有效地组织成一个完整的系统。定义一些构件和连接件类型,施加一组约束描述组合方式 分类 数据流风格:批处理;管道/过滤器;过程控制;	主程序-子过程风格 构件:主程序,子程序 连接器:调用-返回机制 拓扑结构:层次化结构 非结构化程序:所有的程序代码均包含在一个主程序文件中	构件 :数据处理, 构件接口 :输入端口和输出端口,从输入端口读	时过滤器处于处理状态。 管道 作用:在过滤器之间传送数据 • 是单向流 • 可能具有缓冲区 • 数据缓冲区可以是文件、数组、字典、树等集合类型 • 管道形成数	隐式调用 :不直接去invoke一个过程 Event Source:发布事件到EventtManager Event Handlers:可以注册自己感兴趣的事件,并将自己的某个	选择广播式: 调度模块将事件送到那些已经注册了的模块中。 - 点对点模式: 基于消息队列。消息只能够被唯一的消费者所消 个过程 费消费之后即从队列中删除
个较高的层次来考虑组成系统的构件、构件之间的连接,以及由 (CPU和指令集等,操作系统、数据库(不同存储和访问格式等	调用/返回风格: 主程序/子程序; 面向对象; 分层结构 独立构件风格: 事件系统; 虚拟机风格: 解释器; 基于规则的系统;	缺陷:逻辑不清;无法复用;难以与其他代码合并;难于修改,难以测 试特定部分的代码 结构化程序:逐层分解•基于"定义-使用"关系•用过程调用作为	管/小理 然后写到输出端口 连接件 · 数据流 (单向 通常易显步	据传输图 • 管道的先后顺序不影响输出的结果 • 不同的管道中流动 的数据流可能具有不同的数据格式 • 实现机制 限制管道的大小 • 管 道写满时write函数会阻塞 • 管道为空时read函数被阻塞。这解决了	与相应的事件进行关联 Event Manager: 当一个事件被发布,系统自动调用在该事件中 的所有过程(负责调用所有注册到该事件的EventHandler)	 发布-订阅模式:一个事件可以被多个订阅者消费;事件在发送
开发中具有重要影响的设计决策,便于各种人员的交流,反映多种 关注,据此开发的系统能完成系统既定的功能和性能需求	各种不同软件之间在不同平台之间不能移植。或者移植困难。而且,因 为网络协议和通信机制不同,这些系统不能有效相互集成 3.共性凝练和复用:软件应用领域越来越多相同领域的应用系统之间	以数据为中心的风格: 仓库; 黑板; 其他架构风格: MVC; P2P; Grid; SOA 面向对象风格	交互机制 ● 主程序的正确性依赖于它所调用的子程序的正确性 本展: 将大系统分解为若干模块(模块化),主程序调用这些模块实现完整的系统功能。	处理的输出端口传送到另一个处理的输入端口) 拓扑结构:任意拓扑结构的图 警備·讨破器风格	read()调用返回文件结束的问题。从管道读数据是一次性操作数据— 旦被读它就从管道中被抛弃、释放空间以便写更多的数据。 优点:允许对一些如吞吐量、死锁等属性的分析。支持并行执行;	特点:事件的触发者并不知道哪些构件会被这些事件影响,相互 独立 ● 不能假定构件的处理顺序,甚至不知道哪些过程会被调用 个构件之间彼此之间无连接关系,各自独立存在,通过对事件的发	保持 应用于KWIC: 四个功能模块 •共享数据并不直接暴露其格式,而是 • 各 进行封装 • 模块的调用发生在数据发生改变时, 不是主程序控制 这市和 优点: 1.支持实现交互式系统; 2.异步执行 3.容易复用4. 构件替
计比起对算法的 选择和数据结构的设计明显重要得多	许多基础功能和结构是有相似性的。通过中间件提供简单、一致、集 或的开发和运行环境,简化分布式系统的设计、编程和管理 意义:缩短开发周期、节约应用程序开发成本、降低运行成本、降低		优点 : 已被证明是成功的设计方法。可以被用于较大程序 缺点: 代码太多表现不好; 程序太大,开发太慢,测试越来越困难 分解成模块的 四大原则 :	適用场景 :数据源源不断的产生系统需要对这些数据进行若干处理。解决方案:把系统分解为几个顺序的处理步骤。这些步骤之间通过数据流连接一个步骤的输出是另一个步骤的输入;每	使得系统中的物件具有良好的隐蔽性和高内聚、低耦合的特点。 支持软件复用:系统的行为是多个过滤器的行为的简单合成。 在两个过滤器的计算的情况, 不是一个过滤器之间提供适合数据,	注册实现关联 • 可以异步执行 • 一对多通信 构件:对象或过程,并提供如下两种接口: 1.过程或函数,充当事 事件处理器的角色; 2.事件	操不会影响到其它构件的接口; 5. 并发处理提高系统性能; 6.健 件源或 壮性: 一个构件出错将不会影响其他物件, 缺点: 1.分布式的控制方式不利于系统的同步, 验证和调试。2.
设计者需要的结构形式。支持用户和设计者之间的交流与理解;分为两方面: 外向 目标:建立满足最终用户要求的系统需求; 内向 目	放降率、改善决策、应用系统群集/集成、减少软件维护、提高质量 改进技术、提高产品吸引力 分拳:1.应用服务举中间件:为应用系统提供一个综合的计算环境和支		模块 独立性 :高聚合、低耦合 模块 规模遗中性 :过大分解不充分难理 解;过小开销大接口复杂,模块 复用性 :高扇入+低扇出	个处理步骤由一个过滤器实现;处理步骤之间的数据传输由管道 负责。每个处理步骤(过滤器)都有一组输入和输出,过滤器从管道 中读取输入的数据流经过内部处理,然后产生输出数据流并写入	护和增强系统性能简单:新的过滤器可以添加到现有系统中,旧的可以被改进的换掉;	连接机制: 1.事件 过程绑定: 事件处理器的过程向特定事件注 事件源构件发布事件 • 当某些事件被发布时,向其注册的过程被 调用 • 调用的次序是不确定的 • 2.事件-事件的绑定,一个事件	独立调度模块的数据需要经过调度模块的传递,全局性能和资源管 建成为了系统的瓶颈。
的系统构件构成。 作用: 1.交流的手段: 在软件设计者、最终用户之间方便的交流;	掌平台,包括对象请求代理~、事务监控交易~、Java应用服务器~等 2.应用集成类中间件:提供各种不同网络应用系统之间的消息通信、	功能保持唯一的接口 • 动态绑定: 运行时决定实际调用的操作 复用和维护	17日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日	管道中 构件 : 过滤器 处理数据流 连接件 : 管道,连接一个源和 一个目的过滤器。连接器定义了数据流的图, 形成拓扑结构	缺点:通常导致进程成为批处理的结构:每个过滤器是一个完整的从 输入到输出的转换 ● 不适合处理交互的应用:当需要增量地显示改 变时,这个问题尤为严重 ● 因为在数据传输上设有通用的标准,每个过	能触发其他事件,形成事件链 调试器的例子: 编辑器与变量监视器向调试器注册,接收"断点现	严格 分层: 上层-直接下层 松散分层: 上层-所有下层 横切关注: 事件"在严格分层的基础上每一层都可以与另外的一个组件交互
用到其它的项目、提高大规模重复利用率; 3.关键决策的体现:折 衷,关于性能与安全性、可维护性与可靠性、当前开发费用和未	服务集成和数据集成的功能,如消息中间件、企业集成EAI、企业服务 总线以及相配套的适配器。 3.业务架构类中间件:除了可以将底层共性技术的特征抽象到中间件, 不可以将业务共性抽象至中间性、形成应用模式、如业务流程等	缺点:管理大量的对象;必须知道对象的身份标识;继承引起复杂度,关键系统中值用	宋代城では、「列足京代・からかっぽいと出航。定年中域 - C控制域是CEF,作用域CEFD,作用域、控制域、一旦C做了修改、你就要 去投D的位置因为你的修改可能影响D,一旦修改了你还要去找D在哪 (因为它不受你控制),会带来负面的影响,这是一种耦合性。会对维护带		滤器都增加了解析和合成数据的工作,这样就导致了系统性能下降,并增加了编写过滤器的复杂性。 应用在KWIC:	"一旦自身的活,构成都及中争作,从间底及 编码器 一 交易监测器"则更 就有 "就有相談"将源代码滚动到断点处。 变量监测器。则更 前变量值并显示出来 事件调度策略 无独立(非集中式)调度模块的事件管理器:观察者	本接触, 自闭处立下达20 扩张结构 ,八目
阶段就考虑系统的正确设计与方案选择,为以后开发、测试、维 护各个阶段提供了保证	4+1视图模型 用例视图:描述系统的典型场景与功能 逻辑~:系统的抽象概念与功能(类、接口等)类图,协作图,时序图等;	中,每个对象提供了一个接口,允许其他对象通 过该接口调用对该 对象内封装的数据的操作	来很大的麻烦 应用在KWIC: 1. 分为四个基本功能:输入、移位、排序、输出	理不是收集完然后处理,即在输入被完全消费之前输出就产生了 其他特征: 无上下文信息; 不保留状态; 对上下游的其他过滤器	四个过滤器:输入、移位、排序、输出 每个过滤器处理数据,然后将结果送至下一个过滤器 控制机制是分布式的: 只要有数 据传如,过滤器即开始工作	每个模块都能问其他所有模块注册事件 带有独立调度模块的事件管理器,下面都是带有独立(集中式)调	某一层中的构件一般只与同一级别中的对等实体或较低级别中的 度 构件交互。这种单向交互可以减少不同级别中的构件之间的依赖性。
物理~:系統如何被安装,部署,配置在分布式的环境下,部署图 := == == == == == == == == == == == ==	解释器屈格	缺点:不适合功能的扩展(要么修改已有的,要么新增模块) 计算层的软件架构	 主程序按次序调用这四个模块 3.通过共享的数据存储、使用无约束的读-写协议在模块之间进行 数据交换) 金部的连接省去 fork 进程的操作访问更快、体验更好 	单Reactor多线程 ●主线程中,Reactor对象通过 select 监控连接	过滤器之间的数据共享被严格限制在管道传输 .a. Reactor是 非阻塞同步 网络模型,因为read和send操作都需要用户进程	事件调度模块:负责接收到来的事件并分发它们到其它模块。 要决定怎样分发事件,两种策略: DNS 负载均衡:最简单,最常见,地理级别。优点:●简单、	-4- •加权轮询:根据服务器权重进行任务分配权重一般由根据硬件决定,因为
业务逻辑执行, 数据中转等功能 B/S架构: (表现层:浏览器 ● 逻辑层 Web服务器 应用服务器 ● 数		性能度量: CPU速度(MIPS) ● 网络带宽(Mbps) ● 杏吐量(MIPS、TFLOPS、TPS、QPS)指系統在单位时间内处理请求的数量。 ●	缺点: • 存在父子进程通信复杂、支持的并发连接数量有限的问题 FPC 每次有新的连接就新建一个线程去专门处理这个连接的请求 光点: •线程更轻量级,创建线程的消耗比进程要少得多•多线程是共	事件、收到事件后通过 dispatch 进行分发 ●如果是连接建立的事件,则由 Acceptor 处理,Acceptor 通过 accept 接受连接,并创建一个 Handler 来处理连接后续的各种事件 ●如果不是连接建立	●来了事件我来处理。处理完了我通知你● Proactor Initiator 创建 Proactor 和 Handler,并将Proactor和Handler都通过 Asynchronous	衡设备 ●就近访问,提升访问速度:根据请求来源 IP,解析成距 离用户最近的服务器地址,加快访问速度,改善性能	不同服务器处理能力有差异,但没有考虑根据服务器的状态差异 •负载最低优先:(服务器的角度)• LVS以连接数来判断,• Nginx以HTTP 请 求数判断•CPU 密集型CPU 负载 • I/O 密集型I/O 负载 特点 可以感
据层:数据库服务器) 优点: 系统维护成本低 • 客户端无任何业务逻辑 •良好的灵活性和 可扩展性 • 较好的安全性 • 稳定性,延展性和执行效率 • 容错能力	解釋器和編译器:編译器不执行源程序代码,而是将其翻译为另一种语言(可执行的机器码或目标码),并输出到文件中以便随后链接为可	 并发用户数: 指系統可以同时承载的正常使用系统功能的用户 的数量 	可题 ● PPC fork 代价高● 父子进程通信复杂 缺点: ●创建线程仍然有代价,高并发时 (上万连接/s)有性能问题。	Handler 只负责响应事件,不进行业务处理 Handler 通过 read 读取到数据后 会发给 Processor 进行业务处理	操作。 • AOP 完成 I/O 操作后通知 Proactor。 • Proactor 根据不同事件举型问调不同的 Handler 进行业务处理。 • Handler完成业务处	后,有可能会继续访问修改前的 IP访问失败,影响正常使用业务。 ●扩展性差: DNS 负载均衡的控制权在域名商那里,无法根据。	性能最优类: (客户端的角度)优先将任务分配给处理速度最快的服务器,
和负载平衡能力 缺点: 每请求一次服务器就要刷新一次页面 ● 受HTTP协议在数据 查询等响应速度上要远低于C/S架构 ● 数据提交一般以页面为单位.	- 解释器的执行速度要慢于编译器产生的目标代码的执行速度,但是 ,可能低于编译器"编译+链接+执行"的总时间	程: 1.用户请求服务器 2.服务器自己的内部处理 3.服务器返回给 7 用户, TPS是每秒能够完成的 访问 (三个过程)的数量	无须进程间通信,但是线程间的互斥和共享又引入了复杂度,可能 一 不小心就导致了死锁问题。●多线程会互相影响,某个线程出现异常时, 可能导致整个进程退出(例如内存越界)● 存在 CPU 线程调度和切换	应结果发给主进程的Handler处理;Handler 收到响应后通过 send将响应结果返回给client。优点:利用多核多CPU 缺点:	理.Handler 也可以注册新的 Handler 到内核进程。 ●效率更高 ●异步 I/O 能够充分利用 DMA 特性让 I/O 操作与计算重叠. 但操作系统需 要做大量的工作 • Windows 通过IOCP实现了真正的异步 • Linux 系	略比较简单: DNS 负载均衡支持的算法少; 不能区分服务器 的差异; 也无法感知后端服务器的状态。	相当于只是通过响应时间衡量服务器状态。复杂度很高:。需要收集和分析每个服务器对每个任务的响应时间,在大量任务处理的场景下,这种收集和统计本身也会消耗较多的性能。工业上使用采样,并调优采样率
数据的动态交互性不强,不利于在线事务处理(OLTP)应用 ● HTML 的表达能力难以支持复杂GUI (如报表等)。 MVC风格	 解析器执行速度之所以慢是因为每次解释执行的时候都需要分析程序的结构而编译代码则直接执行而无需重复编译 	面的一次访问,形成一个TPS,可能产生多次请求 垂直扩展:向一个节点中增加资源,提高硬件配置	代价的问题。连接不多(几百)的情况还是使用PPC prethread 预先创建线程,实现方式: ●主进程 accept,然后将连接交合某个线程处理。 ●子线程都尝试去 accept,最终只有一个线程	及共享数据的互斥和保护机制。以Java 的NIO为例, Selector 是 线程安全的,但是通过 Selector.selectKeys() 返回的键的集合是	統下的AIO 并不完善以Reactor 模式为主 MapReduce: ●2个函数Map和Reduce ●6个过程 input 输入数据 split 对数据进行有依据的分组 map 把分组之后的数据拆分/映射为其	类设备和路由器、交换机类似。如F5和 A10 优点 ●功能强大 :全面支持各层级的负载均衡,支持全面的负载均衡算法,支持	 ●Hash 类: ●将相同Hash值的请求分到同一台服务器上特定的业务需求 ●源地址Hash: 同一个源 IP 地址的任务分配给同一个服务器进行处理适合于存在事务、会话的业务网上银行同一个会话,总是访问同一台服务器
关注点分离:模型、视图和控制器,从开发者的角度看,实现model 与view的解耦 构件 :解释器引擎、存储区(被解释的源代码、解释 器引擎当前的控制状态的表示、程序当前执行状态的表示)	时进行,因此运行时直接将程序代码装入内存并执行即可 分类 传统解释器:纯粹的解释执行	水平扩展:增加新的节点(新的计算机分布式计算) 高扩展性:系统在几乎任何时刻都可被正常访问,通常量化为一 年中正常运行的时间比 •平均故障时间MTTF •平均修复时间	accept 成功 ●比prefork支持更多的并发连接 Reactor: ●创建一个进程池、将连接分配给进程,一个进程可以处理多 个连接的业务。来了一个事件我就有相应的反应" ● Reactor 会根据	取同步措施进行保护。●Reactor承担所有事件的监听和响应,只 在主线程中运行,瞬间高并发时会成为性能瓶颈。	他的新数据 shuffle 把新数据在本地分别有序储存起来 reduce 把 shuffle后各处的数据放在一起进行组装 finalize 输出组装后的数据 设计思路: • Job Tracker-框架中心,定时与集群中机器通信,管理哪些	商用硬件负载均衡, 严格测试,经过大规模使用 ●支持安全防护: : 具备防火墙、防 DDoS 攻击等安全功能 缺点●价格昂贵 ●	● ID Hash: 将某个 ID 标识的业务分配到同一个服务器 ID 一般是临时性 数据的 ID(如 session id)。 网上银行登录用session id hash 可以实现同 一个会话期间总是访问到同一台服务器的目的。
连接器: 对存储区的数据访问	基于字节码的:编译→解释执行。源代码首先被"编译"为高度压缩和优化的字节码。但并不是真正的机器目标代码。因而与硬件平台无关;编译后得到的字节码然后被解释器加以解释;	MTTR ●系統可用性=MTTF/(MTTF+ MTTR) 代码级优化: ● 发现瓶頭: 圧測工具、抽象模型 ● 线程、内存参 3 数调整 ● Java特性优化 ● 減少并发冲突 ● 減少序列化 ● 減少	事件类型来调用相应的代码进行处理 核心组成两部分: ● Reactor ,负 责监听和分配事件 ●处理资源池(进程池或线程池),负责处理事件 ●两 者的数量都可以变化 三种实现方案: 单Reactor单进程/线程	监控连接建立事件,收到事件后通过Acceptor接收、将新的连接分配给某个子进程●子进程的 subReactor将mainReactor 分配的	程序跑在哪些机器上管理所有job •TaskTracker 集群中每台机器都有, 监视自己机器的资源和tasks运行状况 • TaskTracker 把这些信息通过 heartbeat 发送给 JobTracker • JobTracker 会根据信息确定新的job	扩展能力差 软件负载均衡: 优点●简单: 无论是部署还是维护都比较简单 ●便宜: 只要买个 Linux 服务器.装上软件即可; ●灵活: 4 层	消息中间件 沟通模型分类 ●寻址:直接、间接 ●阻塞:同步、异步 ●缓存:有、无 ●消息内容:事件、命令、数据、流 ●确认:无确认、有确认、三次握手
个model可能有多个View; Controller: 负责接收来自客户的请求 、调用model执行业务逻辑、调用View显示执行结果	译,不是编译全部的代码,而是只编译那些被频繁执行的代码段,如被 执行多次的方法、包含多次循环的方法	字符到字节的转换•使用长连接 通信模式优化 PPC 每次有新的连接就新建一个专门的进程,传统UNIX 网络	 Reactor 对象通过 select 监控连接事件、收到事件后通过dispatch 进行分发。 如果是连接建立的事件、由 Acceptor 处理通过 accept 接受连接并创建一个Handler来处理连接后续的各种事件。 	的各种事件。 ● 当有新的事件发生时,subReactor 会调用连接对应的 Handler进行响应 ●Handler 完成 read→业务处理→send	分配运行在哪些机器上 问题• JobTracker 存在单点故障 • JobTracker 任务太多,过多的资源消耗 • 当 job非常多时,内存开销大(上限4000) • 以task 的数目表示资源,没有考虑到 cpu/ 内存的占用情况,内存很大很	和7层负载均衡可以根据业务进行选择;也可以根据业务进行 比较方便的扩展 款点: ●性能一般 ●功能没有硬件负载均衡那 么强大●一般不具备防火墙和防 DDoS 攻击等安全功能。	●接收方数量:点对点、多播、任意播、地理多播、广播 ●沟通方向:单向、全双工、半双工 ●初始化方式:客户端初始化拉取、服务器初始化推送
连接件: 隐式调用、显式调用、或者其他机制Http 两个分离原则: 展示与模型分离;控制器与视图分离 优点: 代码易开发易维护;同一信息可以有不同的显示方式;业务	行时 运行环境下的编译器将其翻译为本地机器码 禁点易川模煳了	缺点: ● fork 代价高: 创建一个进程需要分配很多内核资源,内	 如果不是连接建立事件,则Reactor 会调用连接对应的Handler (第 2 步中创建的 Handler)来进行响应● Handler 会完成 read→业务处 理→send 的完整业务流程。●优点: 简单,没有进程间通信和竞争,全 	程完成后续的业务处理。 • 交互很简单, 父进程只需要把新连接 传给子讲程, 子讲程于须返回数据。	Yarn优点: 减小了JobTracker的资源消耗,并且让监测每一个Job子任务	理位置决定返回哪个机房的 IP ●集群~: F5 收到请求后进行 集群级别的负载均衡 ●机器~: Nginx 收到用户请求后将请	特点: 异步交互、客户端服务器松散耦合、可靠送达(散据持久存储)、利于应用集成、消息智存在队列里、进程通过中间消息服务器来传递信息、对于数据库集成很自然 消息模式 : 一对一、一对多、多对一、多对多发
逻辑更易测试	执行Java: 1."编译"得到字节码(与硬件平台无关); 2.Java解释器 直接执行该字节码	发连接数量有限 prefork 系统在启动的时候就预先创建好进程,然后才开始接受	邵在同一个进程内● 缺点: 一个进程无法发挥多核 CPU 的性能,只 能部署多个系统来利用多核 CPU,这样需要维护多套系统●Handler 只	read、send 等无须同步共享, 业务处理可能需要同步共享)		负载均衡算法 ●轮询:按顺序轮流分配,最简单的策略,无须关	布/订阅模式 队列:发送者和接受者之间扮演一个中间地址目标 各个进程 独立发送和失效,各个进程间的失效和交流失败可忽略——实现了时间解 耦、位置解耦 信息优先级:最高优先级优先、权重优先发
冗余高可用计算 主备架构 : 主机执行所有计算任务。例如读写数据、执行操作等 ●当主机故障(定机)时,任务分配器不会自动路计算任务发送给备。	集群方案 •主备架构和主从架构简析: 架构简单: 通过冗余—台服务器来提升 可用性 •问题: 人工操作效率低、容易出错、不能及时处理故障	异地多活 • 异地: 地理位置上不同的地方, • 多活: 不同系统都能够提供服务 • 判断一个系统是否符合异地多活,需要满足两个标准:	并行范式●Task-FarmingMaster-Worker Model: Master把内容给分解成小的任务,分发给workers,并且汇聚最后产生的结果, work stealing: 每个worker维护—个任务队列,为空时随机去其他的	质的总称,是用于输入电子计算机进行处理,具有一定意义的 3	如果主备间数据复制延迟,由于备机并不对外提供读写操作,因此对业务 2.有影响,但如果延迟较多,恰好此时主机又宕机了,则可能丢失较多数据, 3此对于复制延迟也不能掉以轻心。一般的做法是做复制延迟的监控措	并且只要与中介断开连接,就将自己降级为备机。因此可能出现	使用主备,各机不能读,如果使用主从,从机也可以读 复杂度高。多各即多通道,增加了主机的复制压力,同时增加了对正常 读写的压力(实践中、需要考虑降低该压力)。多通道,情况不一、容易
机,此时系统处于不可用状态 ●如果主机能够恢复,任务分配器继续 将任务发送给主机 ●如果主机不能够恢复(如机器硬盘损坏):人工:	■言可用生料方安・可用性亜少面加严格的採品由 自动空动机场操作	1.元比切问哪一门地点的亚牙系统。静脉多特别正确的亚牙液牙 2.某个地方业务异常的时候,用户访问其他地方正常的业务系统。 能够得到正确的业务服务 • 地理位置分类:同域异区,跨域异地,跨国异地	worker創队列中stealing任务 • 单程序多数据(SPMD)每个进程执行相同的代码段、但在数据的不同部分上执行。在可用处理器之间拆分数据。流水线:取指译码执行	◆数据是信息的表达、载体,信息是数据的内涵,是形与质的	5.当廷迟数据量较大时及时领警。由人工干预处理6.益: •对于客户端来说不需要感知备机的存在。即使灾难恢复后,原来的各机被人工修改为主机后对于客户端来说、只是认为主机的地址换了,无	机将主机升级为主机。如果是网络中断导致主机与中介断连.	後39月1日7月(東京)
新的机器作为备机、根据备机状态的不同分类: 冷备架构:备机上的程序包和配置文件都准备好.但备机上的业务	采取某种策略(随机、轮询等) 将计算任务分配给集群中的不同服务器 当集群中的某分服务器故障后,不再将任务分配给它,分配给其他服	单点故障 的优化: ●找出扩展性能瓶颈 ●找出单一故障点 ●找出宕 和影响区域 ●水平/乗車扩展/打公	河回5月15代,任何发生882年1987分战后,600次以,或许是300年 河回5月校文、充分利用资源。今治:一个问题被划分为两个成多个子 问题,每个子问题都独立解决,并将它们的结果组合在一起。预测并行: 采用不同的算法来解决同一个问题——第一个给出最终解决方案的算	用系統 非功能需求 ● 存储高性能: 读写分离、数据缓存、分 需 库分表、NoSQL ● 存储高可用: 主从、CAP理论 行	號如道是原来的各机升级为主机了 ●对于主机和各机来说,双方只需要进 于数据复制即可,无须进行状态判断和主备倒换等复杂操作。 *点:备机仅是备份,不提供读写操作,硬件浪费;故障后无法自动恢复		ヨエットのサル、対け国連の成立、・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
将备机的业务系统启动,并将任务分配器的任务请求切换发送给备	负载均衡集群的设计关键点在于两点: ●选取分配策略(轮询和随机基本够用) ●检测服务器状态: 稍微复杂, 既要检测服务器的状态(服务器	硬件垂直拆分 6. 数据库水平扩展 7.数据库水平和垂直拆分 8.分 离集合 9.缓存 10.反向代理/HTTP 加速器	采用不同的算法本解决问。「「问题——第一「名面版经解决方条的算 法就是选择的算法●数据流水线 适用于细粒度并行性。也适用于涉及 多个执行阶段的应用,但需要对大量数据集进行操作 ●参数计算模型、Nimrod-G - 用于计算密集型参数应用。	读写分离 将数据库读写操作分散到不同的节点上 ●数据库服务器搭建主从集群,一主一从、一主多从都可以。	E从复制(与主备复制的区别)。只有主机可以写,但主机和从机都可以读主机出故障时,写操作不可用,但是读操作可以正常执行 15。 • 主机出故障时,连操作不可能。 • 人机提供连操作,发挥了硬件的	模拟式: 主备机之间并不传递任何状态数据,而是备机模拟成	均衡性:保证数据分区基本均衡 ●容错性:部分服务器故障后,这些服务器上的数据分区需要分配给其他服务器 ●可伸缩性:当集群容量
即可。 ●冷备可以节省一定的能源,但温备能够大大减少手工操作 时间,因此一般情况下推荐用温备的方式	时间过长),常用的做法是任务分配器和服务器之间通过心跳来传递信息(心跳检查),包括服务器信息和任务信息,然后确定状态判断条件	分布式配置框架 ●管理Client服务器、配置下发功能 ● 拉取模式: Client主动拉収、Client规模上万适用 ● 推送模式: ConfigServer主动推 Client规模数干适用	Gridbus Broker – 用于分布式数据密集型参数应用。 阿姆达尔定律: 忽略所有系统或通信开销的 1	通过复制将数据同步到从机。每台数据库服务器都存储了所有 性 的业务数据。 • 业务服务器将写操作发给数据库主机将读操作 绐	4.66 - 13 000 (1997)	优点: 实现更简单,省去了状态传递通道的建立和管理工作。	不够,扩充新的服务器后,算法能够自动将数据分区迁移到新服务器, 并保证扩容后所有服务器的均衡性 ●必须要有一个角色来负责执行数据 分配算法:可以是独立服务器,如HDFS架构,也可以是集群选举出的服 条器、世称之为主机。但服带完全不同。如Elasticsearch
人工执行,系统实现很简单 •缺点:需要"人工操作" •比较适合内部、后台管理系统这类使用人数不多、使用频率不高的业务,不太:	中节点 ID 最小为Master)来区分不同服务器的角色,不同的角色执行 不同的任务(如Master-Slave 角色),任务分配器将不同任务发送给不	分布式消息框架 ●异步消息 ●系統解構 ●分开调用者与被调用者的处理逻辑、解决处理语言差异、数据结构差异以及生产者与消费者速度差异等 ●保证最终一数性和消息有序性	$r_s + r_s$ 设计 三个方法•串行算法的直接并行化 •从问题描述开始设计并行算	写入到数据库主服务器后立刻进行读取,此时读操作访问的是 提 从机,主机还没有将数据复制过来,所以读不到最新数据 开	第一次 - 主备机直接建立状态传递的渠道。可以是网络连接(如各 干一个端口),也可以是非网络连接(用串口线连接)。可以是三机发送状 5给备机,也可以是各机拉取主机的状态。可以和数据复制通道共用,		两者区别 ●数据读写 集中集群中客户端只能将数据写到主机。分散集群架构中,可以向任意服务器中读写数据。 ● 应用场景 数据集中集群适合
主从架构:从机也要执行任务,任务分配器需要将任务进行分类。 哪些发送给主机执行,哪些任务发送给备机执行。 ●主机故障时:	服务器故障,不需要重新分配角色,只需要将其剔除即可。	并行计算架构 串行: ●指令依次执行 ●在单个处理器上执行 ● 任何时候都只能执行—条指令 并行: ●将问题分解为可同时解决的	法。借用已有算法求解新问题 四个步骤。划分:分解成小的任务,开 拓井发性、先坡分解再功能分解,使数据集和计算集互不相交。通讯: 确定诸任务间的数据交换,监测划分的合理性。 四种通讯模式:周部/ 全局通讯、结构化/非线构化通讯、静态/动态通讯。同步/异步通讯	从机失败后再读一次主机 ●关键业务读写操作全部指向主机。 也 非关键业务采用读写分离 是	B可以独立一条通道。秋态传递通道可以是一条,也可以是多条,还可以 是不同类型的通道混合(如网络+串口) 倒装方案: 。主备机共享一个对于 各个时类型的通道混合(如网络+串口) 侧装方案: 。主备机共享一个对于 各户端来说唯一的地址(如虚拟)P)。客户端记录于备机各自的NP。备机县	等の対応制度を対する。 会内域只需要将读写操作发送给主机B即可反之亦然。◆如果故障主机能够恢复则客户端继续访问两台主机两台主机整 续相互复制对方数据◆如果故障主机不能恢复则需要人工操	数据量不大集群机器数量不多(个位数)的场景,如ZooKeeper集群; 数据分散集群,由于其良好的可伸陷性,适合业务数据量巨大、集群 机器数量庞大(上千台)的业务场景,如Hadoo内集群 数据度:存储物理的文件的集合数据库率模操作数据的程序
续发送给主机。不管这些任务执行是否成功。如果主机能够恢复(人工或者自动),任务分配器继续按照原有的设计策略分配任务。如果	为不同类型并分配给不同角色的集群节点。2.角色分配策略实现比较复杂: 可能使用 ZAB、Raft 这类复杂的算法来实现 Leader 的选举。	 采用整体控制/协调机制 	至母面代 宏持化中部中心通机 解决力的点面机 阿沙升沙面机 ●组合:依据任务的局部性组合成更大的任务。合并小尺寸任务、减少 任务数,通过增加任务的粒度和重复计算,可以减少通讯成本、保持映 射和扩展的灵活性,随低数件工程成本,涌讯量与任务子集的表面成	 客户机无论读写操作。都发送给主机、备机不对外提供任何读有服务 主机故障情况下、备机也不读写数据、整个系统处于 	「拒绝服务的能力 缺点: ●状态传递通道本身故障了,则各机会主动升 股为主机。虽然可以通过多通道来降低通道故障的机率,但是通道越多, 5续的状态决策越复杂,特别是容易收到多种矛盾的信息	作,增加一台新的机器为主机•原有故障主机不能恢复的情况 下,成功写入原有故障主机但没有复制到正常主机的数据会丢	数据序: 仔细数结的文件的集合 数据序头的"架下数据的程序 分库分表: 本质是数据拆分,对数据进行分而治之,将一个表结构分为多 个表,或者将一个表的数据分片后放入多个表,这些表可以放在同一个 库里,也可以始邻末间的库里,其至可以放在不同的数据库率侧上
为从机。任务分配器继续按照原有的设计策略分配任务。	器都是任务分配器,Follower 收到请求后判断:如果是写请求就转发给 Leader,如果是读请求就自己处理。●角色指定: 通过 ZAB 算法来选举	●过程函数・中粒度・程序员・共享消息、消息传递 ●循环指令块 细粒序 编译器 共享变量	现代的,根据30人后任,特别从行工生现外,通讯编号任务了条约及组成 正比,计算量与任务子集的体积成正比 ●映射:将每个任务分配到处 理器上,提高算法的性能	主机能够恢复(人工或自动),客户端继续访问主机。主机继续将 数据复制给备机 ●如果主机不能恢复,则需要人工升级备机为 传	中介式: 主备机之间不直接连接,而都去连接第三方中介,通过中介来 传递状态信息•主备机无须再建立和管理多种类型的状态传递连接通道, 录等接得即介加可,实际上降低了主备机的连接复杂度•状态决策更	据,在另一台主机上读取不到。优点:两台主机,无倒换概念;	 ●垂直拆分:根据业务的维度、将原本的一个库/表拆分为多个库、每个库与原有的结构不同。●水平拆分:根据分片算法,将一个库/表拆分为多个库。每个库依旧保留原有的结构(仅仅改变了规模和大小)
发展阶段 ●单库单表: 所有用户数据都存放在同一数据库的 代題	行选举,直到新的 Leader 选举出来后才继续对 Client 提供服务 ·9· ■分片 在应用层和数据库层之间增加一个代理层,把分片的路由规 则	●多指令发射/内存交叉存取-非常细粒度-硬件处理器 据库中对应的记录。分片维度:按哈希切片:●对数据的某个字 段束哈希、再除以分片总数后取模。取模后相同的数据为一个分 添	.10. 点: ●同步阻塞问题: 当参与节点占有公共资源时,其他节点访问公共	写入主机但还没有复制到备机的数据可能会永远丢失 简 最大努力保证模式: 适用干对一致性要求并不十分严格,但是对 公	前单:无须考虑多种类型的连接通道获取状态信息如何决策的问题。11- 分库分表引起的 问题。扩容与迁移 处理方法 1.按照新旧分片规则对新 日数据库进行双写 2.将双写前按照旧分片规则写入的历史数据.根据新分	数据集中集群:●即一主多备/从,写操作只能发给主机。如果 在條介層●内左: 最快的选择 于需额外的(/○开销 轴占是)	-12 -
结构完全一样的表,如User0, User1UserN,所有表的数据的的的总量不变•多库多表: 水平切分,将切分的数据库和表水平实现	T发人员不用关心分片规则,只需关心业务逻辑的实现,待业务 逻辑 见之后,在代理层配置路由规则即可 ●代理层的引入增加了一层 网络	片,这样将数据分成多个分片,●优点:数据切片比较均匀,对数据 下 压力分散的效果较好 •缺点:数据分散后,对于查询需求需要进 但	去(尤其在第二阶段),如果是协调者挂掉,可以重新选举一个协调者; 是如果协调者宕机,那么无法解决●数据不一数,在二阶段提交中,当	源的提交尽量延后到最后一刻处理,如果业务流程出现问题,则 所有资源更新都回滚,保持事务一致。以消息队列消息消费和 F	自数站手拉竹从今:47从今前这州门分,他的一个指别文家总品的部分 特规则注移写入新的数据库 3.将按照旧的分片规则查询改为按照新的分 特规则查询 4.将双写数据库逻辑从代码中下线,按照新的分片规则写入 按据 5.删除按照旧分片规则写入的历史数据•数据—数性问题由干数据	●硬盘: 很多缓存框架会结合使用内存和硬盘,在内存分配空间 满了或是在异常的情况下,可以被动或主动的将内存空间数据持	頁 度慢的问题。●将一个服务器的内容平均分布到多个服务器上;智能 4久 识别服务器,让用户获取离用户最近的服务器,提高访问速度。
什么时候需要 ●数据库中表的数量达到了一定量级,需要分解 ●可 单表的大数据量对索引查询带来的压力,并方便对索引和表结 如G	用的框架:Cobar和Mycat等。支持事务的分布式数据库:很多产品 OceanBase等对外提供可伸缩的体系领机并提供一定的分布式事务专	的分片 优点: ●单库单表的数据保持在一定的量级有助于性能 的提高 ●切分的表的结构相同,应用层改造较少,只需要增加路由 到	调者向参与者发送commit请求之后,发生了局部网络异常或者在发送 mmit请求过程中协调者发生了故障,这会导致只有一部分参与者接受 了commit请求。而在这部分参与者接到commit请求之后就会执行	消息 4.更新数据库 5.提交数据库事务 6.提交消息事务 事务补偿机制:在数据库分库分表后,如果涉及的多个更新操 规	■大,通常会造成不一致问题,因此,通常先清理旧数据,洗完后再迁移到新 观则的新数据库下,再做全量对比。 还需要对比评估迁移过程中是否有数	数据库,而是key-value存储结构的特殊数据库(Berkeley DB和 Redis),响应速度和吞吐量都远高于关系型数据库等。	缓存中可以存放的最大元素的数量 ●一旦缓存中元素数量超过这个值
例,利用多个数据库实例来分解大量的数据库请求带来的系统 者近 压力 •如果希望在扩容时对应用层的配置改变最少,就需要在 用于	5明,使用者不需要直接控制这些特性 ●目前不太适用于交易系统,较多 F大数据日志系统, 统计系统、查询系统、社交网络等	据是分散的,很难利用数据库的Join操作,跨库Join性能差 • 拆分 提规则难以抽象 • 分片事务的一致性难以解决 • 数据扩容的难度	ommit操作。但是其他部分未接到commit请求的机器则无法执行事务 交。于是整个分布式系统便出现了数据不一致性的现象。 新段提交协议:针对单点故障问题,在第一,二阶段间加入准备阶段,当	务保证一数性。●对于跨库的多个操作可通过补偿和重试使其 需 在一定时间窗口内完成操作 ●这样既保证了事务的最终一致性,程	居更新,如果有需要迭代清洗,直至一致。 如果数据量巨大,无法全量对比, 需要抽样对比,抽样特征需要根据业务特点进行选取。 线上记录迁移过 全中的更新操作日志,迁移后根据更新日志与历史数据共同决定数据的最	基本类型 本地缓存 应用中的缓存组件 ●优点: 应用和cache是不同一个进程内部,请求缓存非常快速,没有过多的网络开销等。 在单应用不需要集群支持或者集群情况下各节点无需互相通知	元素值往往可以一定程度上提高缓存的命中率,从而更有效的使用缓 的 存。基本操作 命中与验证 ●HTTP 再验证If-Modified-Since,看缓存
实例 : 16亿条数据,分解到4个数据库实例里,每个数据库实例 块的 包含2个数据库,每个数据库里有4个表,那么一共有32个表 平 ●可	物數据可以分散到不同数据库服务器(User, Pay, Commodity) 以冷热分离,根据数据的活跃度将数据进行拆分 ●冷数据:更新频率	题;存在跨节点Join的问题;存在跨节点合并排序、分页的问:" 题;存在多数据源管理的问题 不同点:垂直更偏向于业务拆分如	调者故障后,参与者可以通过超时提交来避免阻塞 •canCommit阶段 和两阶段的请求阶段很像,协调者向参与者发送commit请求,参与者 果可以提交就返回yes,否则返回no •preCommit阶段:协调者根据	偿机制,则在遇到问题时,需要记录遇到问题的环境、信息、步 3 骤 状态等 后结译过重试机制使其达到最终一种性	所状态,以达到迁移数据的最终一数性。 ●动静数据分离问题:对于一些 动静敏感的数据,如交易数据,最好将动静数据分离。 选取时间点对静历 P数据进行迁移。	多个应用程序无法直接的共享缓存,各应用或集群的各节点都	需 缓慢命中), 200(再验证不命中), 404(已删除) 缓 存清洗策略 : 当缓存 存 空间被用满时需要删除一部分缓存, 这样才能引入新的缓存。策略:
如何分库分表: •客户端分片: 使用分库分表的数据库的应用层 表中 直接操作分片逻辑, 分片规则需要在同一个应用的多个节点间 而料	P的内容划分为多个表,例如冷数据的表放在查询性能高的服务器, B热数据的表并部署到更新性能高的服务器 优点。 业务清晰,拆分规	CAP理论: ●Consistency(一致性)任何一个读操作总是能读取 理之前完成的写操作结果,也就是在分布式环境中,多点的数 有	与者canCommit阶段的响应来决定是否可以继续事务的preCommit作: 1.所有参与者的反馈都是yes:那么进行事务的预执行,协调者向所参与者发送preCommit请求,并进入prepared阶段。参与者接收到	表的多个数据源路由事务。如果更新操作在一个数据库实例内 分 发生,便可以使用数据源的事务来处理。对于跨数据源的事务。 18	(3)就品打116%。 查詢问题: 在分库分表以后,如果查询的标准是分片的主键,则可以通过 分片规则再次路由并查询,但是对于其他主键的查询、范围查询、关联查 句、查询结果排序等,并不是按照分库分表维度来查询的。	字典等常用数据。 • 缓存介质: 硬盘, 内存 • 实现方法: 应用编 • 中间件, 如Ehcache, Guava Cache等 分布式缓存 与应用分离 缓存组件或服务。 • 优点: 自身就是一个独立的应用。 与本地应用	
通过依赖Jar包来实现: 方式: ●在应用层直接实现: 直接在应用 同的 层读取分片规则,然后解析分片规则,据此实现切分的路由逻辑。计相	9机器上,便于管理 ●便于实现动静分离、冷热分离的数据库表的设 输式 ●数据维护简单 缺点:●部分业务表示法关联、只能通过接口方	的时间内返回,也就是系统随时都是可用的 ◆Partition(分区容 事 忍性) 在出现网络分区的情况下,分离的系统也能正常运行 易	eCommit请求后会执行事务操作,并写undo和redo日志中,成功执行 务后,返回ACK响应,并等待最终指令 2.参与者的反馈有一个是No或 等待超时之后协调者都没收到响应:那么就中断事务,协调者向所有的	事务的一数性。这就需要通过编写程序来选择数据库的事务管 理器、即事务路由、●自动提交事务路由、通过依赖JDBC数据源 有	解决方案: ●在多个分片表查询后合并数据集(效率很低) ●按查询需求 定义多分片维度形成多张分片表(空间换时间) ●通过搜索引擎解决,如果 有实时要求,还需要实时搜索。(难度大) 分布式事务问题: 多库多表分布	隔离,多个应用可直接的共享缓存。 • 应用场景: 缓存经过复杂 法首组出的数据:缓存存储系统中断整访问的执占数据:减轻存	A 从cache中取到数据●~失效:应用程序没有从cache取到数据,那 种储 么从数据库中取数据,取到后放到缓存中●~更新:先把数据存到数 应 据库中,成功后,再让缓存失效。●Read Through是在查询操作中
行定义,容易调试维护。但要求开发者既要实现业务逻辑,还需 瓶颈 要实现框架需求。该实现方式会让数据库保持的连接比较多, 水 型	页,不易进行数据扩展和提升性能 ●事务处理复杂 P切分:不是终表讲行分类 而是终其按照某个字段的某种抑则分散到	AP两种选择 ● CP模式: 保证分布在网络上不同节点数据一数 仍	未收到协调者请求,执行事务中断 • do Commit阶段:协调者根据参与 preCommit影影的响应来决定是否可以继续事务的do Commit影件	5 交事务,不需要开发人员手动操作事务和配置事务,只能满足 同简单的业务逻辑需求 ●可编程事务路由 通觉采用Spring的声 3	式所引发的一数性问题。 同组数据跨库问题 :要尽量把同一组数据放到 同一台数据库服务器上,不但在某些场景下可以利用本地事务的强一致性 还可以使这组数据实现自治。	用服务器,处理所有对WEB服务器的请求、请求的页面在代理服务器上有缓存的话,代理服务器直接将缓冲内容发送给用户。 则供向WEB服务器发出请求 取回数据 在本地缓存后再发送	股 更新破仔,当破仔头戏的时候以知或比如换出),Cacne Aside是出 调用方负责把数据加载入缓存,而Read Through则用缓存服务自己 会 来加载,从而对应用方是透明的 ●Write Through 在更新数据时发
对整体应用版方部记的推广特定成注 J ●加过定制DBC所以 多个 实现,可让开发者集中精力实现少务逻辑。无须关心分库分表的 数线 实现。让就是针对业务逻辑层提供与JDBC—数的接口,分库分 ●路	7年中,在每个表中包含一部分数据。所有表加此来是全量数据。即将 据按一定规律,按行切分,并分配到不同的库表里,表结构完全一样。 由过程:分库分表后,数据将分布到不同的分片表中,通过分库分表规	可用性和分区容忍性,但弱化了数据一致性要求。两阶段提交 1. 协议: ●请求阶段: 事务协调者通知每个参与者, 在本地执行事务, 响 写redo和undo日志但不提交。请求阶段,参与者将告知协调 者	协调者从参与者得到了ACK的反馈:协调者接收到参与者发送的ACK 应.那么它将进入提交状态.并向所有参与者发送doCommit请求。参与 接收到请求后.执行正式的事务提交.并在完成事务提交之后释放所有	明式的事务米管理数据库事务,在分库分表时,事务处理是 5 个问题,在一个需要开启事务的方法中,需要动态地确定 开 启哪个数据库实例的事务,也就是说在每个开启事务的方法调 #	歐雅遊行 國雅遊行 第2:●用于存储数据的硬件或软件组件,以使得后续更快访问响应的数 据●缓存中的数据可能是提前计算好的结果,数据的副本等	用户。降低了向WEB服务器的请求数,从而降低了WEB服务器	 的 生。当有数据更新的时候、如果没有命中,直接更新数据库、然后返 回。如果命中了,则先更新缓存,然后再由缓存更新数据库(同步)。 Write back,更新数据的时候,只更新缓存,不更新数据库,用缓
ORM框架实现: 分片规则实现到ORM框架中或者通过ORM 什么	■找到对应的表相阵的过程叫做路田 ●仕设计表的需要确定对表按照	者自己的决策: 同意或取消事务的执行•提交阶段: 基于投票结 事 果讲行决策: 当日仅当所有的参与者同意揭交事务 协调者才通 到	务资源,并向协调者发送haveCommitted的ACK响应。 那么协调者收 1这个ACK响应之后,完成任务 2.协调者从参与者没有得到ACK的反馈。	用削就必须确定开启哪个数据源的事务。●声明式事务路由,即 作 在实现的方法上直接声明事务的处理注解,注解包含使用哪个 据	\$月: ●主要解决高并发,热点数据访问的性能问题 ●提供高性能的数 B快速访问。原理: ●将数据写入到读取速度更快的存储 ●将数据缓存	CDN缓存:内容分发网络●通过在现有互联网中增加一层新的原络架构(CDNS),将网站内容发布到最接近用户的网络边缘,让	可存异步地批量更新数据库(速度很快,性能很高,但是不能保证一致性,
导致内存泄露, 那么就会遇到内存溢出错误●堆外内存: jdk5之后	一台服务器(逆时针)之间的数据 ●如果增加一台服务器,受影响的也 只有新服务器和它前一台服务器之间的数据 ●服务节点大小时,容易	要到后端数据库系统进行查询,使数据库压力过大甚至崩溃【解决】 将空值缓存, 再次接收到同样的查询请求时,若命中缓存并值为空就	友生死视等的开友问题,读与速度下清严重 ●成本局: 开旦随着系统的规模变大,成本将不断上升 ●有限的支撑容量: 现有关系型还	式,经常查询的数据存储在同一个文档中,而不是存储在表中,适 由于存储不同的属性以及士量的数据的应由程序	性能的 ●计算能力,利用Zookeeper作为协同服务、稳定服务和 Failover机制 Dynamo ●高可用性,高扩展性,去中心化的分布式系	載、帯宽限制了响应速度 ●解决方案:单任务分解为多任务,多任 务导步处理 ●技术方案:1.浏览器要求加载动态内容。2.浏览器	泰翻译为具体的质量属性及其场景 ●输出: 风险, 非风险, 关键点. 折中 点 其于度量的 ~ ●设计质量指标的具体度量方式(数学公式) 在度量时
存 实现方法:编程直接实现: ●通过静态变量一次获取到缓存内存中,减少频繁的I/O读取,静态变量实现类间可共享,进程内可共享,缓存的实时性稍差 ●为了解决本地缓存数据的实时性问题,	造成大量数据集中到一个节点上(数据倾斜问题),引入虚拟节点即对每一个服务节点计算多个哈希,每个计算结果位置都放置一个此服务节点 数据淘汰策略 • Volatile-Iru:已设置过期时间-最近最少	问这个缓存key的请求量较大,多个请求同时发现缓存过期,因此多	: 无法支撑大型互联网应用级的海量的数据存储,如Google等。 非关系型数据库 共有原则 •NoSQL实现都建立在硬盘、机器和网络 都会失效的假设之上 •这些失效不能彻底阻止,因此需要让系统能	按CAP分类: ◆CP: 关注一数性和分区容忍性, 这类数据库可支持在 无法确保数据一数性的情况下拒绝提供服务, 损失可用性, 主要和 BigTable(列), Hypertable(列), HBase(列), MongoDB(文档),	 (← 由多个物理异构的机器组成;每台机器存放一部分数据;数据 有的备份同步完全由系统自己完成;每台机器可随意添加或去除;● 单台机器故障不会影响系统对外的可用性●数据存储格式;键值● 	渲染静态内容。3.浏览器引擎要求不同服务器给数据。4.各服务器异步返回数据。5.浏览器引擎通知浏览器数据到了。6.浏览器 异步渲染返回的数据 面向自适应的前端架构 ●问题:多屏适配	 可采用自动化的工具对质量属性进行计算,得到结果,三个基本活动: 从软件体系结构文档中获取度量信息并计算,得到结果。建立质量属性和度量结果之间的映射原则,即确定怎样从度量结果推出系统具有什么样
目前大量使用的是结合ZooKeeper的自动发现机制,实时变更本地静态变量缓存,编程直接实现缓存。优点,直接在heap区内读写,快并且方便。缺点。同样是受heap区域影响,缓存的数据量	使用	个请求会同时访问数据库来查询最新数据,并且回写缓存,这样会造 成应用和数据库的负载增加性能降低甚至崩溃【解决】分布式锁: 保证对于每个key同时只有一个线程去查询后端服务 ●本地锁:与	够在即使非常极端的条件下也能应付这些失效●对数据进行分区● 最小化失效带来的影响,也将读写操作的负载分布到了不同的机器 上●保存同一数据的多个副本:大部分NOSQL实现都基于数据副本	Terrastore(文档), Redis(Key-value), Scalaris(Key-value) MemcacheDB(Key-value), Berkeley DB(Key-value) AP: 关注可用性和分区容忍性、沒迷數据库主要以立即 "最终—	数据分区:Consistent Hashing •数据复制:在Consistent	北本 松端情况 尺寸态化尺寸能灵活诱应各体田怪器 a 络—	的质量属性。根据映射原则分析推导出系统的某些质量属性。•优点: 结果比较客观。精确。è战点: 很多质量属性无法给出具体的计算公式; 能给出计算公式的质量属性往往是针对部代码级影的质量操性如代码 行数,而这些属性对评价往往缺乏足够的意义;需要在A设计基本完成
非常有限,同时缓存时间受GC影响。主要满足单机场景下的小 数据量缓存需求,同时对缓存数据的变更无需太敏感感知,如一 纷积影響等理,其缺熱不数据等场景, 中间价Fbcache 。 a 体速 多	内部实现●消极方法:在主键被访问时如果发现它已经失效,那么就 删除它,触发时机是在实现GET,MGET,HGET,LRANGE等所有 油及到证则数据的命令时机会用用。 和码方法 图即性他从设置了生	分布式換突似。通过本地被限制只有一个线程去数据阵中查询数据。 节点数量较多时并未完全解决缓存并发的问题 ● 軟过期: 对缓存中 的数据设置生物时间 不使用缓左眼条组供的过期时间 一個左電腦	的於倫切米保证主採的高可用生、自仔第一个以家的,可以任为家级 指定希望保存的副本数 ●查询支持、不同的实现的一个共性在于哈希 来中的Kovavalugum即 保占。■以鄉倫对左続 结构不同党 可以减少	数性" 来确保可用性和分区容忍性, 主要有: Cassandra (列) CouchDB (文档) SimpleDB (文档) Riak (文档) Dynamo(键值)	会被路由到某个节点上去读取 ●易于扩展 ●以Dynamo为基础,结	结构机class名,用问一股css代码定义样式,用问一个js函数定义 交互 ●面度组件化:每个组件的html.css.is都独立封装管理。	以后才能进行;需要评估人员对待评估的体系结构十分了解。 质量属性 •终端用户的视角· 性能· 可靠性· 可用性· 安全性 • 商业视角· 上市时间
线程机制·大型高并发系统场景 ●简单,配置可直接使用,无需其他依赖 ●多种的缓存策略,灵活 ●两级缓存数据有:内存和磁盘,	效时间的主键中选择一部分失效的主键删除,触发时机是Redis的时间事件,即每隔一段时间就完成一些指定操作 持久化方法 ●RDB:	指缓存服务器重启或者大量缓存集中在某一个时间股内失效、业务 系统需要重新生成缓存,给后端数据库造成瞬时的负载升高的压力, 基至崩溃【解决】。	时间和空间的开销●对于一数性要求可以降低,而可用性的要求则 更为明显,弱一数性理论BASE(反ACID)● 可以替代关系型数据库	以NoSQL为主: 鈍NoSQL 在一些数据库结构经常变化,数据结构 不定的系统中,就非常适合使用NoSQL来存储以NoSQL为数据 源: 在四限序口令直接控数据NoSQL 国通过NoSQL 国地位别	了关系数据库的优点,使它处于关系数据库和键值数据库之间。	定义好框架和加载方式,内容在加载时从外部填充。改变开发的 协作方式。大家不再是按页面分工,而是按组件来分工。弱化了 相互间的依赖关系。重在维护:组件化在项目初期的准备工作	成本和收益、项目周期,目标市场、与原有系统的集成性。开发者的视角: 可维护性、轻便性、复用性、可测试性 软件架构测试方法 性能测试。 负载测试不限制软件的运行资源,测试软件的数据吞吐量上限。压力
能更简单方便的进行缓存实例的监控管理 • 支持多缓存管理器实例,以及一个实例的多个缓存区域。 • Ehcache的超时设置主要	中 ●AOF: 以协议文本的方式,将所有对数据库进行过写入的命令及 其参数记录到 AOF文件 缓存的问题	证:只有一个效程能进行该仔更新 ●失效时间分方机制: 对不同的数据使用不同的失效时间,甚至对相同的数据、不同的请求使用不同	持 SQL这样的工业标准,将会对用户产生一定的学习和应用迁移成本,尽管现在有第三方工具或中间件缓解此弊端,但仍为非标准化	課・個別程分次三直接付数額NOSQL,丹園辺NOSQLの形別次、 把每次写入,更新,删除操作都复制到MySQL数据库中● 根据应 用的逻辑来决定去相应的存储获取数据● 同时,也可以通过复制 协议把数据同步复制到全文检索,实现强大的检索功能 ● 需要考虑		会增加一定工作量,但随时间推移会发挥出巨大的优势。 评审方式 基于调查问卷的 ~ ●评估过程: 多个评估专家考察系统、然后回答问卷中的问题; 对多个评估结果进行综合。得到最终的结果	测试:长时间或超大负荷地运行测试软件,对系统的稳定性进行测试 ●容量测试:确定测试对象在给定时间内能够持续处理的最大负载或工作量。网络性能测试:测试网络带宽 定识 . 负载和端口的变代对用户的响向
是针对整个cache实例, 而不针对单独的key, 失效的缓存元素无 法被GC回收, 时间越长, 缓存越多的内存占用越大, 内存泄露的 概率也越大。	数据不一数: ●先与缓存,再与数据库(错误的做法): 缓存与成功,但 写数据库失败或响应延迟,则下次读缓存时,就出现脏读【解决】先写	线程来更新缓存 ●缓存集群:可以做缓存的主从与缓存水平分片 缓存高可用:缓存局不高可用 需要根据实际的场景而完 并不易所	支持 ●支持的特性不够丰富:现有产品所提供的功能都比较有限, 大多数NoSQL 数据库都不支持事务,也不像MS SQL Server和 Oracle那样能提供各种 附加功能,比如BI和报表等 ●现有产品在逐	数据复制的延迟问题,与主从中的复制延迟问题一样	据不断写入磁盘; 使用类似MySQL的日志方式,记录每次更新的 。日志。前者性能较高,但可能会引起一定程度的数据丢失;后者相	 ・优点: 目由, 灵活,可评估多种质量属性,也可在SA设计的多个阶段进行。◆缺点: 结果很大程度上来自评估人员的主观推断, 因此不同的评估人员可能会产生不同其至截然相反的结果, 评估人 	时间的影响~步骤:制定目标和分析系统、选择测试度量的方法、选择 相关技术和工具、制定评估标准、设计测试用例、运行测试用例,分析测 试线里~增标:响应时间 内存 祕盘 处理器 网络 压力测试 6 在同一
分布式缓存 平滑迁移: 双写->迁移历史数据->切读->下线双写。 具体过程与分库分表的扩容与迁移相同 停机迁移: ●停机应用先	存失败,则下次读取(并发读)缓存时,则读不到数据; 【解决1】根据写	计例如临界点是否对后端的数据库造成影响。解决方案:●分布式 实现数据的海量缓存●复制:实现缓存数据节点的高可用	· 步成熟中: 大多数产品都是开源产品的新贵,有待继续完善,包括 与之相关的生态构建。按数据模式分类 • 列式: 主要围绕着列进行数	本高,内子数据易失却个各忽视 键图行义化模式。 大部分互联内 应用的特点都是数据访问有热点,也就是说,只有一部分数据是 被插数值用的 ■ 把热占数据进行出方层级 五 非热占数据左联到域	库和非关系数据库之间,数据模型不局限于关系或Key-Value ●面 向集合存储,易存储对象类型的数据:数据被分组存储在数据集中,	员对领域的熟悉程度、是否具有丰富的相关经验也成为评估结果是否正确的重要因素。基于场景的~●分析SA对系统应用的	、成品来"猪棒"。明显时间,1947、放盆、处理感,1941 运刀商机。 在1时 时间内或某一时间内,向系统发送预期数量的交易清求,并发交易清求, 递增交易清求,并发递增交易清求 自的 ·发现性能振發,评价系统性能 对系统资源进行优化,提高响应时间与吞吐量 测试流程图:。 测试计划。
到新的缓存数据集群中 ●更改应用的数据源配置,指向新的缓存集 群 ●重新启动应用。该方式简单,高效,能够有效避免数据的不一	库,再回写缓存 ●缓存异步刷新: 指数据库操作和写缓存不在一个操作步骤中,比如在分布式场景下,无法做到同时写缓存或需要是步刷新	缓存服务器压力过大。解决:复制多份缓存副本,把请求分散到多 个缓存服务器上减轻缓存执占导致的单分缓存服务器压力	Column Family特性: 每次重问都会处理很多数据,但涉及的列开不 多,比较适合汇总和数据仓库类应用 • Key-value: 如HashTable, 速	DIPKED, PLACE STABILITY TO SEE STABLE SEEDING	包含无限数目的文档 ●存储在集合中的文档,被存储为Key-Value	求的满足程度◆软件体系结构分析方法(SAAM) ◆体系结构权衡 分析方法ATAM ◆输入: 场景(用例场景-常规应用场合、变化性	对系统反感近170亿、矩向响应时间与台电温 國風風隆區: ● 测试月 初
致,但需要由业务方评估影响,一般在晚上访问量较小,或者非核心 服务的场景下比较适用。	【解决】根据日志中用户刷新数据的时间间隔,以及针对数据可能产生不一致的时间,进行同步操作 -17.	大分數据库 玩鳥: 个语的注解,爲穩定性,使用简单,功能強大, 被业界广泛认可 问题 ●扩展困难:由于存在类似Join这样多表查	度快、大的数据存放量和高并发操作,非常适合通过主键对数据进行 查询和修改等操作,不支持复杂的操作,但可通过上层的开发来弥补。	列族 •利用Hadoop HDFS作为其文件存储系统,高可靠性的底层	各种复杂的文件类型●高性能, 易部署, 易使用, 存储方便。 - 19 -	*************************************	(株: ●硬件平台: 服务器CPU,内存及硬盘 ●网络平台: 负载,延迟, 传输故 障 ●软件平台: 数据库,中间件●应用级别: 线程级别,会话级别,代码级制。