分布式存储架构设计

基于 MongoDB 与 Hadoop 分布式存储

基础架构分析

本架构以三层架构为原型加以改造。

三层架构(3-tier architecture) 通常意义上的三层架构就是将整个业务应用划分为：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。区分层次的目的即为了"高内聚，低耦合"的思想。

架构中由前端服务器、业务服务器集群与后台数据服务集群构成（此配置能承受中小量访问，若出现巨大量访问时，采取横向拓展方式，即增加相应的业务服务器）。而对应的服务器充当三层中的不同角色。

以三层架构来设计的优点在于：

优点
开发人员可以只关注整个结构中的其中某一层；
可以很容易的用新的实现来替换原有层次的实现；
可以降低层与层之间的依赖；
有利于标准化；
利于各层逻辑的复用。
结构更加的明确
在后期维护的时候，极大地降低了维护成本和维护时间

同时也具备缺点：

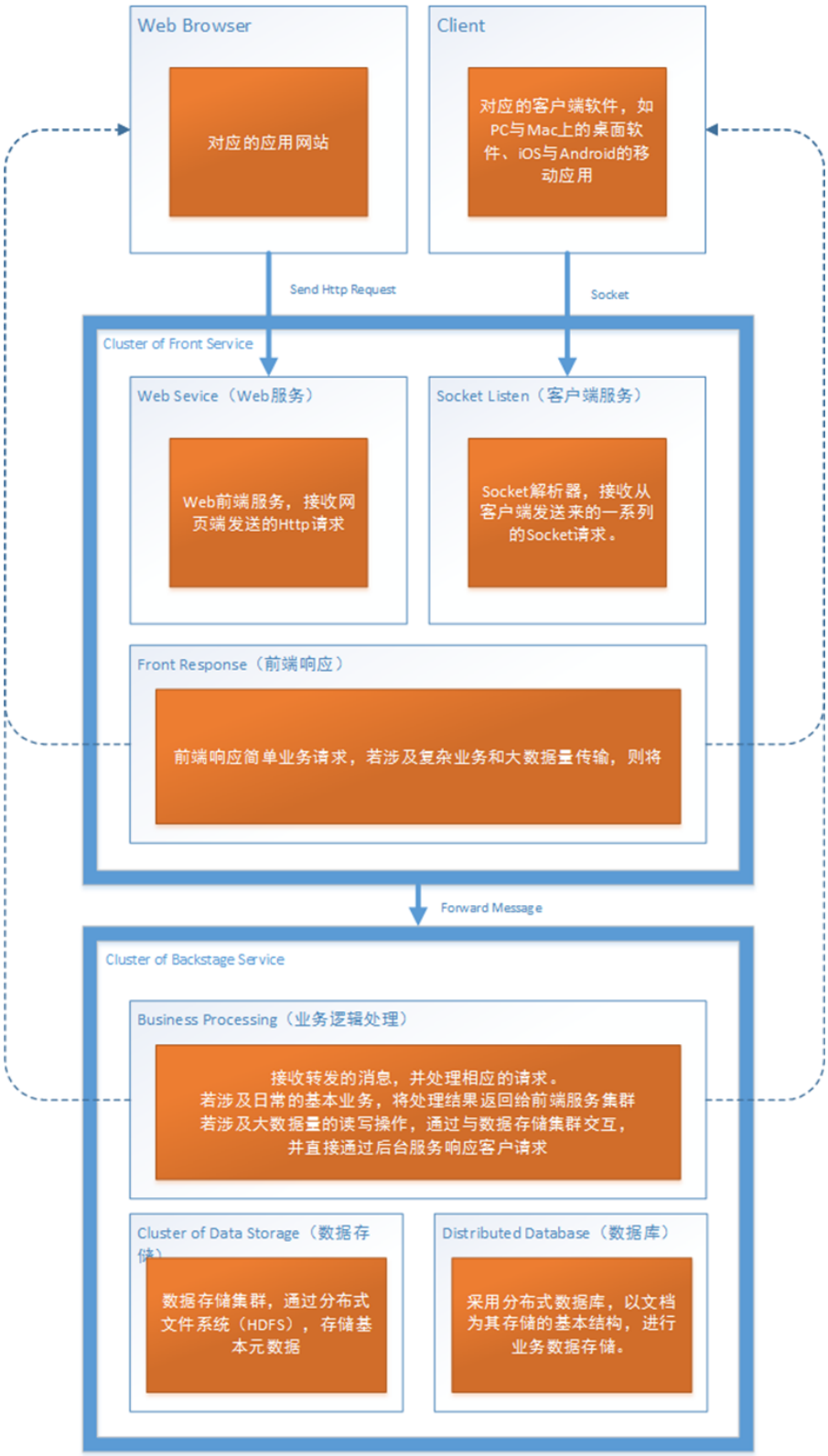
1、降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。

2、有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

3、增加了开发成本。

所以在本架构的开发初期，由于成本及技术原因，故将前端服务器与业务服务器合并开发，但在开发时候更多考虑降低耦合度的设计，如将部分流程独立成为单独程序，并通过脚本控制。

基本框架


Web Browser
对应的应用网站
Client
对应的客户端软件，如PC与Mac上的桌面软件、iOS与Android的移动应用
Send Http Request
Socket
Cluster of Front Service
Web Sevice（Web服务）
Web前端服务，接收网页端发送的Http请求
Socket Listen（客户端服务）
Socket解析器，接收从客户端发送来的一系列的Socket请求。
Front Response（前端响应）
前端响应简单业务请求，若涉及复杂业务和大数据量传输，则将
Forward Message
Cluster of Backstage Service
Business Processing（业务逻辑处理）
接收转发的消息，并处理相应的请求。
若涉及日常的基本业务，将处理结果返回给前端服务集群
若涉及大数据量的读写操作，通过与数据存储集群交互，
并直接通过后台服务响应客户请求
Cluster of Data Storage（数据存储）
数据存储集群，通过分布式文件系统（HDFS），存储基本元数据
Distributed Database（数据库）
采用分布式数据库，以文档为其存储的基本结构，进行业务数据存储。

前端服务器设计

前端服务器主要用于接受外部的 Http 访问请求，并将访问打包转发给业务服务器进行处理，以及进行简单的业务处理，充当一个中部枢纽的作用。在开发前端模块时，尽量避免了复杂业务的处理，如文件上传、下载等操作，将复杂的业务进行转发由后台的业务服务群进行专门处理。

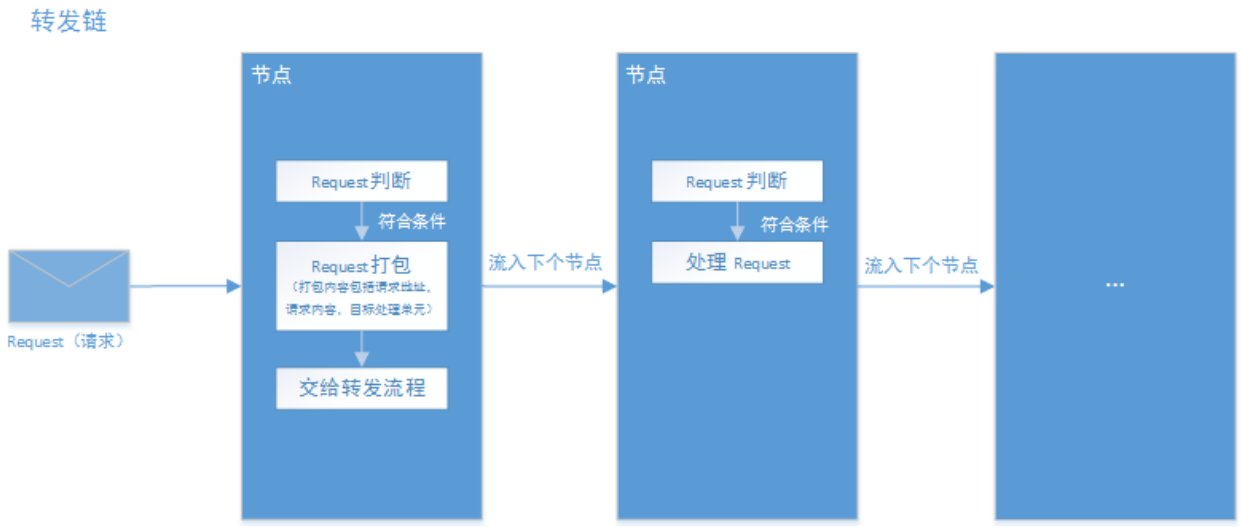
其关键点在于

1.如何处理庞大的连接并发数

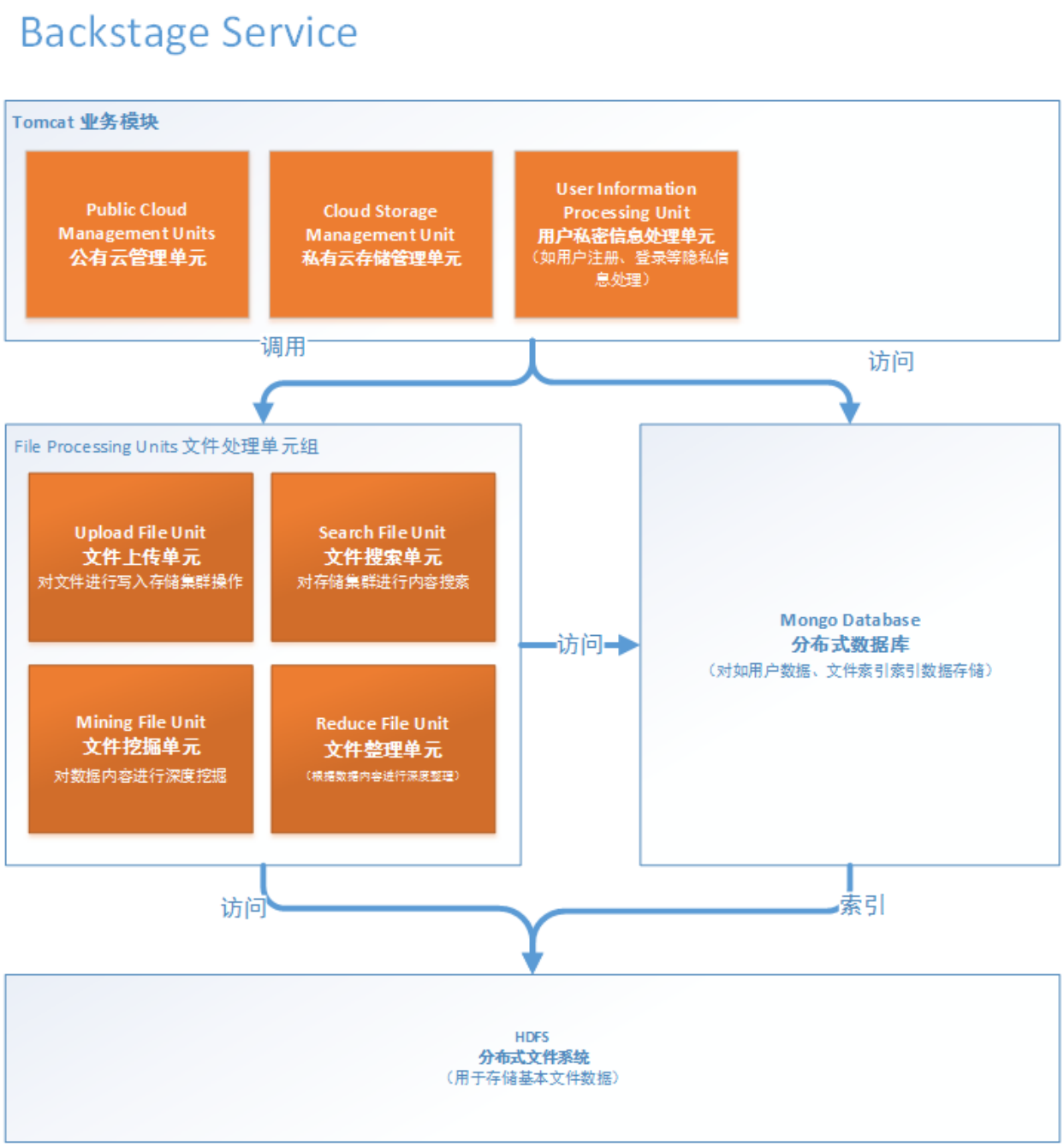
2.如何更有效率的进行转发。

面对第一个问题，我们采用开源服务器 Tomcat 来处理 Web 请求，Tomcat 服务器在中小量访问中效率高，且耗费成本低，维护成本低。

面对第二个问题，我们设计一个转发&处理链进行处理，请求链中有对应的转发或处理过程，每个请求都会流入链中，并从链的某个节点流出，继而转发。每个节点会有相应的请求队列，当该请求符合转发或处理条件，则将请求压入队列，等待转发流程；若不吻合则流入下个节点。节点的顺序通过是动态改变的，每隔一个时间间隔，统计出节点流出数量，针对流出的数量由高到低进行一次排序，重新形成转发处理链。再次接受请求。



业务服务器设计


Backstage Service
Tomcat 业务模块
Public Cloud Management Units
公有云管理单元
Cloud Storage Management Unit
私有云存储管理单元
User Information Processing Unit
用户私密信息处理单元
（如用户注册、登录等隐私信息处理）
调用
访问
File Processing Units 文件处理单元组
Upload File Unit
文件上传单元
对文件进行写入存储集群操作
Search File Unit
文件搜索单元
对存储集群进行内容搜索
Mining File Unit
文件挖掘单元
对数据内容进行深度挖掘
Reduce File Unit
文件整理单元
（根据数据内容进行深度整理）
访问
Mongo Database
分布式数据库
（对如用户数据、文件索引索引数据存储）
访问
索引
HDFS
分布式文件系统
（用于存储基本文件数据）

后台数据集群设计

后端使用两个分布式组件混合使用，分别是 Mongodb 和 Hadoop。

分布式存储数据库设计

需求目标

本数据库意在连接业务层与文件系统，作为一个承上启下的作用。使业务层方便访问相关的数据，后台亦能根据相应的数据对其进行整理，维护。更重要一点，能使各个组件的耦合度降低。便于每个组件的更新开发。

目标
对后台分布式文件系统 Hadoop 进行管理并建立索引.
对业务中的基本数据，如用户资料等数据，进行合理存储，提高数据安全与进行高速访问。
兼顾日后的数据挖掘的任务。
能更容易进行横向扩展（硬件扩展）与纵向扩展（数据存储结构更新）

组件选用与介绍

选择 MongoDB 作为本架构的核心数据库。与创投的数据库相比，其具有以下优势：

优势	说明
性能	支持云计算层次的扩展性。
部署	其数据库支持 Mac，Windows，Linux 等多种操作系统，同时支持 java，c++等多种语言的开发。
使用	面向集合形式存储，与传统数据库的以表结构存储十分相似。
	模式自由（补充说明）
	以文档为其存储的基本结构，文档内部以键值对存储

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。Mongo 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

它的特点是高性能、易部署、易使用，存储数据非常方便。

主要功能特性
面向集合存储，易存储对象类型的数据。
模式自由。

支持动态查询。
支持完全索引，包含内部对象。
支持查询。
支持复制和故障恢复。
使用高效的二进制数据存储，包括大型对象（如视频等）。
自动处理碎片，以支持云计算层次的扩展性
支持 RUBY，PYTHON，JAVA，C++，PHP 等多种语言。
文件存储格式为 BSON（一种 JSON 的扩展）
可通过网络访问

所谓"面向集合"（Collenction-Orented），意思是数据被分组存储在数据集中，被称为一个集合（Collenction)。每个 集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库（RDBMS）里的表（table），不同的是它不需要定 义任何模式（schema)。

模式自由（schema-free)，意味着对于存储在 mongodb 数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库里。

存储在集合中的文档，被存储为键-值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各中复杂的文件类型。我们称这种存储形式为 BSON（Binary Serialized dOcument Format）。

MongoDB 服务端可运行在 Linux、Windows 或 OS X 平台，支持 32 位和 64 位应用。

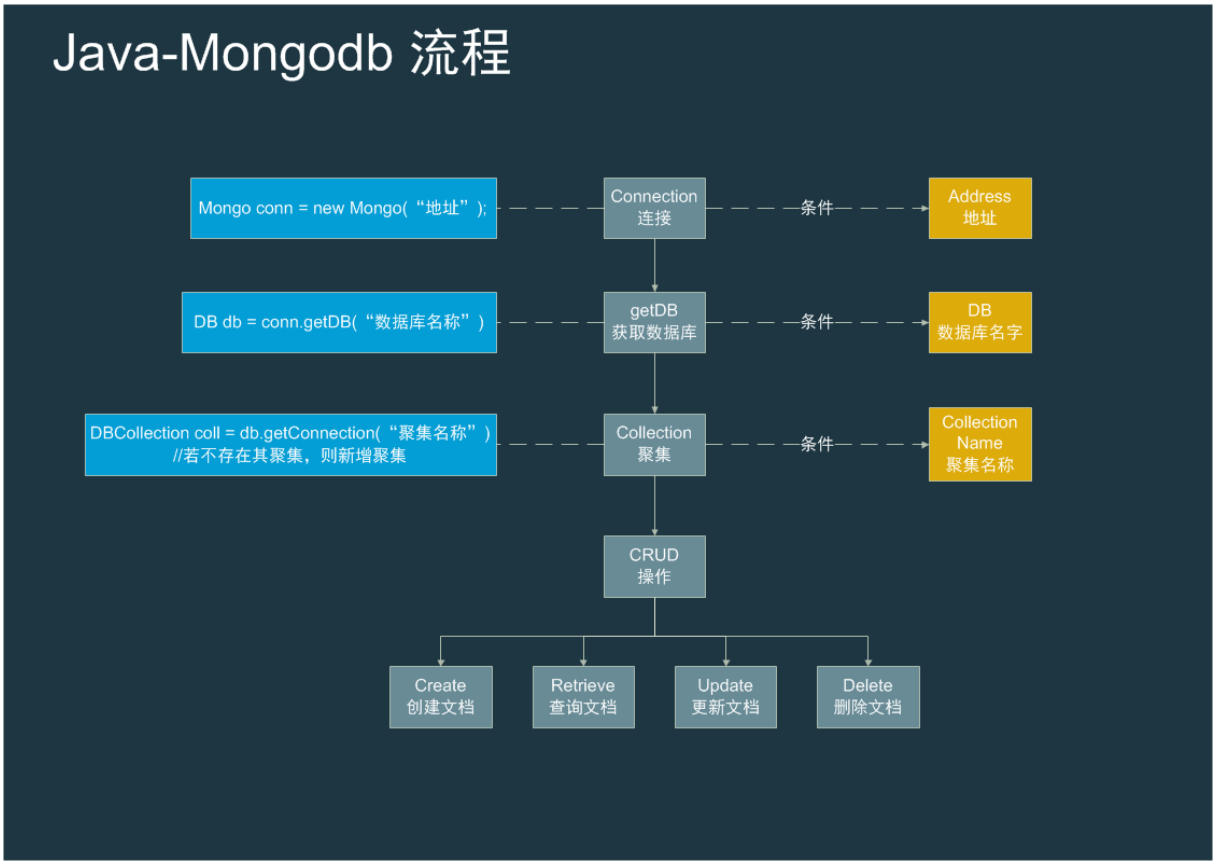
数据库设计方案

本数据库设计采用**关系&文档型数据库设计**（关系模型与文档模型相结合）。全部集合可以分成两部分：基本数据集合和业务数据，两者关系类似关系模式里面的主从表。

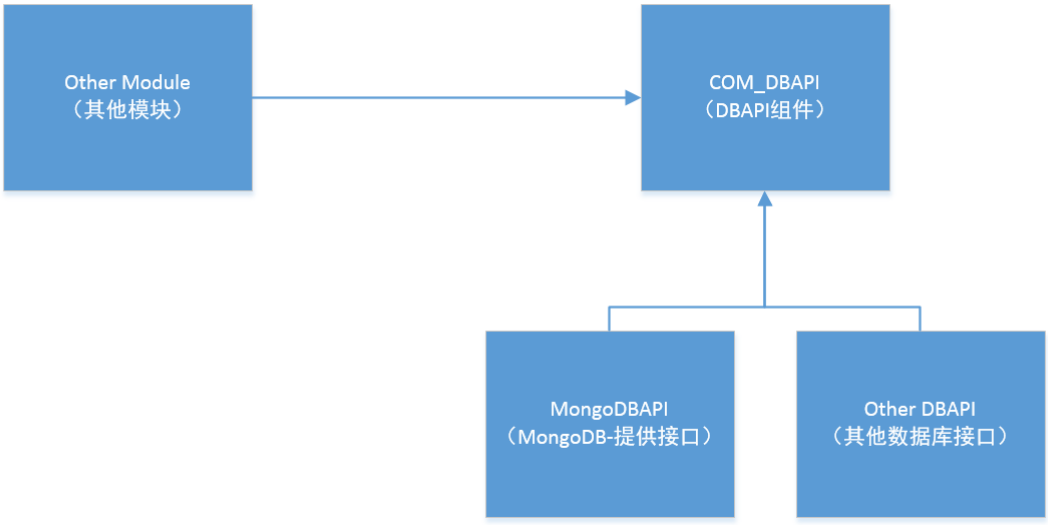
集合	说明
基础数据集合	基本数据集合存储系统的元数据，用于管理以及关联文件系统中的基础文件。使文件系统与业务层耦合降低，便于对文件系统的一些统计优化操作。同时也减少文件数据灾害所带来的风险。
业务数据集合	业务数据集合存储由元数据与一系列业务逻辑数据组合构成的数据，此部分数据用于为上层业务区提供服务。

数据库组件调用流程

参考 mongodb 的 javaapi 进行调用，首先通过连接数据库获取连接用的对象，并绑定具体的数据库。对其查找相应的集合，进行 CRUD 操作。（流程与传统数据库十分相似）



为了保持接口独立性，应该相应封装一层接口，用于调用数据库，来降低组件对系统的耦合程度，便于日后由于业务需要而更换组件时的代码修改。封装的接口给其他的业务模块进行使用。



分布式文件存储系统设计

需求目标

该组件为整个架构的底层部分，用于对基本数据的存储。所谓的基本数据未知其具体的内容的二进制数据。其基本数据的相关信息（包括如何读取、解析的方法）则由分布式数据库进行统一管理。文件存储系统仅管理数据的本身，面对如何有效快捷访问数据，如何以最小空间存储更多数据，如何保证数据完整性等问题。

组件选用与介绍

我们使用 HDFS 作为该系统的核心组件。

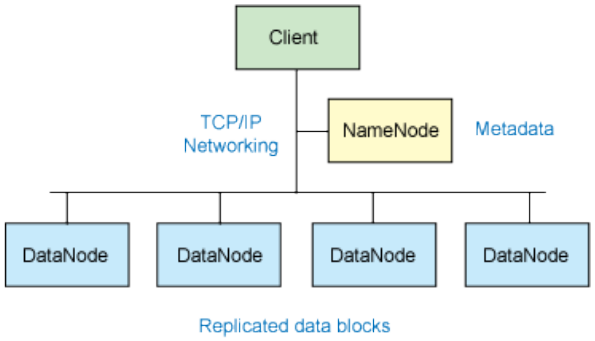
Hadoop 是一个能够对大量数据进行分布式处理的软件框架。但是 Hadoop 是以一种可靠、高效、可伸缩的方式进行处理的。Hadoop 是可靠的，因为它假设计算元素和存储会失败，因此它维护多个工作数据副本，确保能够针对失败的节点重新分布处理。Hadoop 是高效的，因为它以并行的方式工作，通过并行处理加快处理速度。Hadoop 还是可伸缩的，能够处理 PB 级数据。此外，Hadoop 依赖于社区服务器，因此它的成本比较低，任何人都可以使用。

Hadoop 是一个能够让用户轻松架构和使用的分布式计算平台。用户可以轻松地在 Hadoop 上开发和运行处理海量数据的应用程序。它主要有以下几个优点：

优势	说明
高可靠性	Hadoop 按位存储和处理数据的能力值得人们信赖。
高扩展性	Hadoop 是在可用的计算机集簇间分配数据并完成计算任务的，这些集簇可以方便地扩展到数以千计的节点中。
高效性	Hadoop 能够在节点之间动态地移动数据，并保证各个节点的动态平衡，因此处理速度非常快。
高容错性	Hadoop 能够自动保存数据的多个副本，并且能够自动将失败的任务重新分配。

Hadoop 带有用 Java 语言编写的框架，因此运行在 Linux 生产平台上是非常理想的。Hadoop 上的应用程序也可以使用其他语言编写，比如 C++。

HDFS 就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件，等等。但是 HDFS 的架构是基于一组特定的节点构建的，这是由它自身的特点决定的。这些节点包括 NameNode（仅一个），它在 HDFS 内部提供元数据服务；DataNode，它为 HDFS 提



供存储块。由于仅存在一个 NameNode，因此这是 HDFS 的一个缺点（单点失败）。

存储在 HDFS 中的文件被分成块，然后将这些块复制到多个计算机中（DataNode）。这与传统的 RAID 架构大不相同。块的大小（通常为 64MB）和复制的块数量在创建文件时由客户机决定。NameNode 可以控制所有文件操作。HDFS 内部的所有通信都基于标准的 TCP/IP 协议。

使用 HDFS 的原因在于，Hadoop 能提供高可靠，高效率，易伸缩的处理。由于其是按照 Google 的 GDFS 进行仿照设计实现，而且其背后有大量的开源工作者为其提供技术支持，有足够的开发说明文档。这些优点能让架构开发迅捷，难度简单，成本减少。

在架构的设计中，我们重点关注如何节省空间已存储更多数据的问题。由于系统会根据文件的膨胀而时占用空间日益增加。使得 HDFS 在某特定时间间隔后必须进行横向扩展（即进行硬件扩展，如增加存储空间）。如何能延长时间间隔成为问题关键。

相关技术

通过对各个云存储服务的研究、猜测，以及对文件存储的数据进行挖掘，我们能得出以下结论：文件系统中出现一定量的数据冗余，而这种冗余直接导致存储空间的浪费！比如：某份文件记录在系统中可能会出现若干次，而其若干次的内容完全一致。这就是典型的文件重复。出现这种重复的原因是由于多个用户上传内容一致的文件导致。

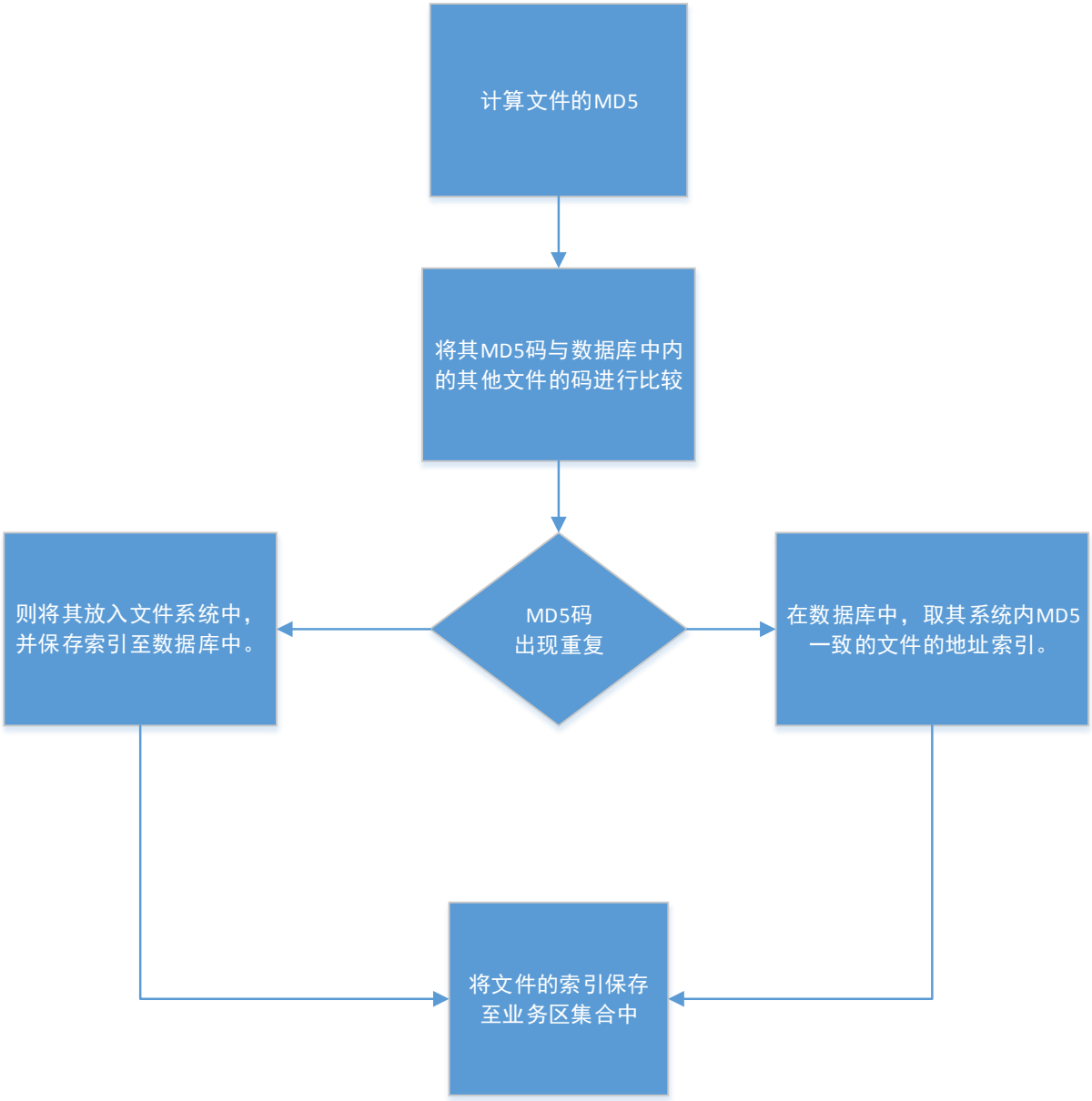
在架构的上层，即系统的业务上，我们提供每个用户都有一定容量的私有存储空间。但这个存储空间仅仅是个业务层面的记录数字。其存储的目录同为后台的文件存储系统。而文件系统本身是不关心文件的相关业务信息（文件的拥有者、文件的上传时间等等）。这种松耦合的设计能在后台设计算法，对存储系统优化，如进行文件去除操作。

文件去重技术

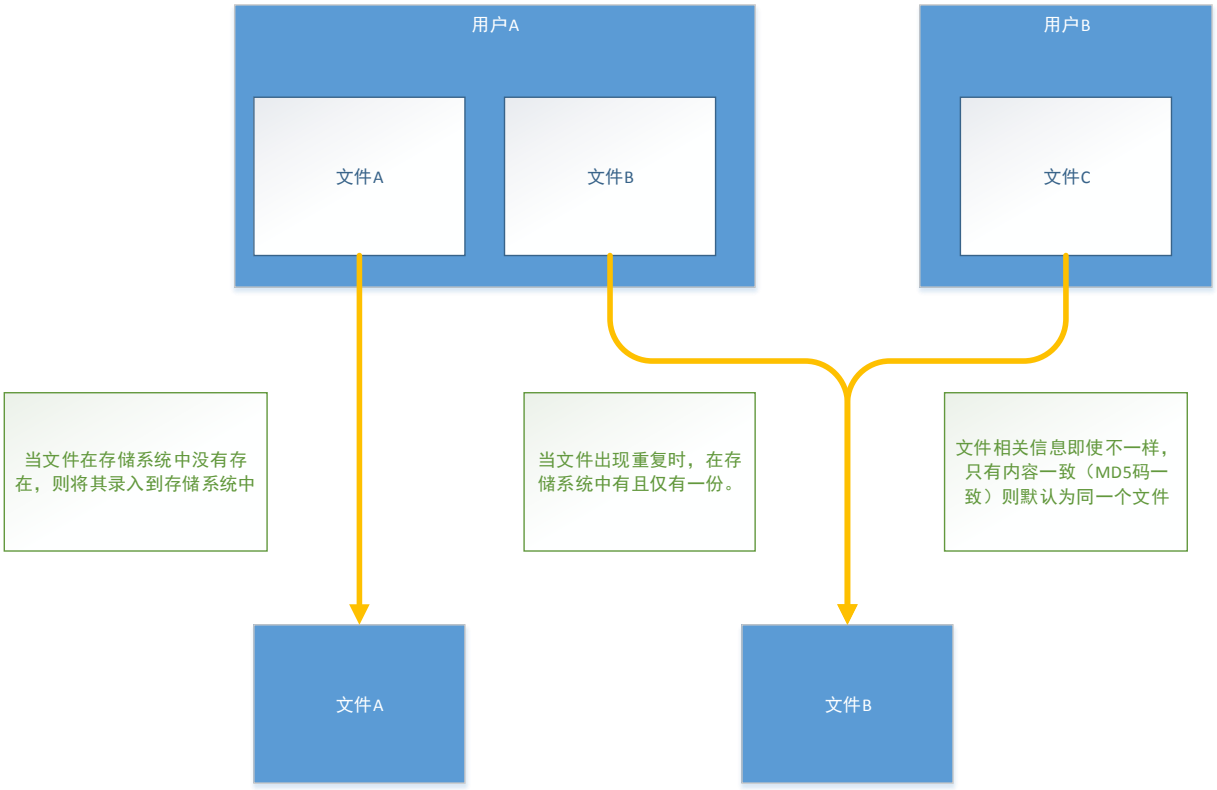
需求：当文件数量不断膨胀时，出现文件重复的几率则会增加，从而导致系统出现冗余数据，故需要特定算法，对存储系统进行优化、去重操作。

算法核心：对文件的内容进行 MD5 编码，当两个文件的内容一致的时候，即其二进制数据一致时，其 MD5 编码则为一致，故仅对其文件的 MD5 码进行比较，则能获知两文件是否一样，后进行文件去重的操作。

文件去重流程


计算文件的MD5
将其MD5码与数据库中内的其他文件的码进行比较
MD5码
出现重复
则将其放入文件系统中，并保存索引至数据库中。
在数据库中，取其系统内MD5一致的文件的地址索引。
将文件的索引保存至业务区集合中

文件索引示意图


用户A
文件A
文件B
用户B
文件C
当文件在存储系统中没有存在，则将其录入到存储系统中
当文件出现重复时，在存储系统中有且仅有一份。
文件相关信息即使不一样，只有内容一致（MD5码一致）则默认为同一个文件
文件A
文件B