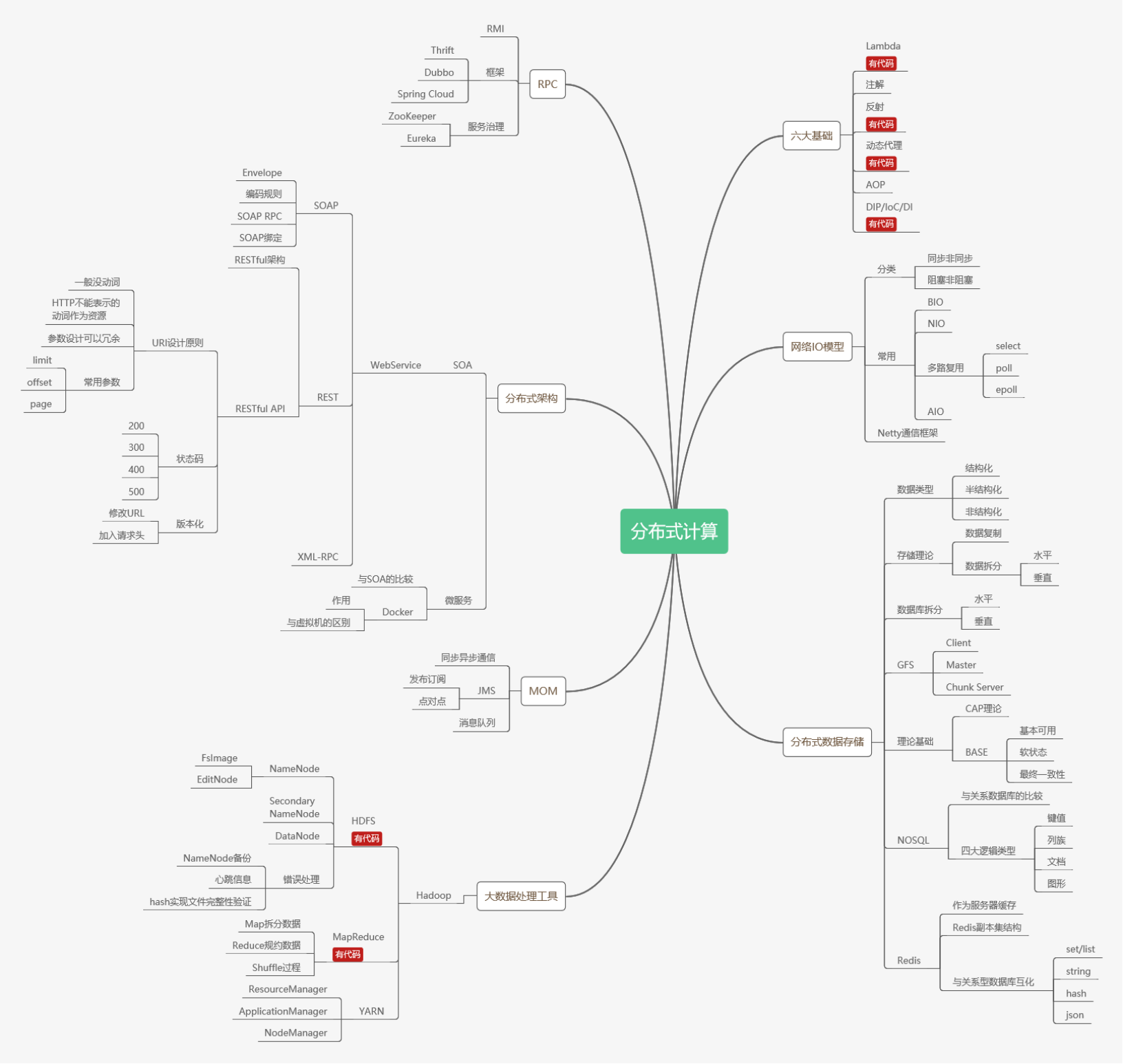

分布式计算
六大基础
Lambda
有代码
注解
反射
有代码
动态代理
有代码
AOP
DIP/IoC/DI
有代码
网络IO模型
分类
同步非同步
阻塞非阻塞
常用
BIO
NIO
多路复用
select
poll
epoll
AIO
Netty通信框架
分布式数据存储
数据类型
结构化
半结构化
非结构化
存储理论
数据复制
数据拆分
水平
垂直
数据库拆分
水平
垂直
GFS
Client
Master
Chunk Server
理论基础
CAP理论
BASE
基本可用
软状态
最终一致性
NOSQL
与关系数据库的比较
四大逻辑类型
键值
列族
文档
图形
Redis
作为服务器缓存
Redis副本集结构
与关系型数据库互化
set/list
string
hash
json
大数据处理工具
Hadoop
HDFS
有代码
FsImage
EditNode
NameNode
Secondary NameNode
DataNode
错误处理
NameNode备份
心跳信息
hash实现文件完整性验证
MapReduce
有代码
Map拆分数据
Reduce规约数据
Shuffle过程
YARN
ResourceManager
ApplicationManager
NodeManager
MOM
JMS
同步异步通信
发布订阅
点对点
消息队列
分布式架构
SOA
WebService
SOAP
Envelope
编码规则
SOAP RPC
SOAP绑定
REST
RESTful架构
RESTful API
URI设计原则
一般没动词
HTTP不能表示的动词作为资源
参数设计可以冗余
常用参数
limit
offset
page
状态码
200
300
400
500
版本化
修改URL
加入请求头
XML-RPC
微服务
Docker
与SOA的比较
作用
与虚拟机的区别
RPC
框架
RMI
Thrift
Dubbo
Spring Cloud
服务治理
ZooKeeper
Eureka

1. Lambda

写法：

两种都 OK：

```
(x, y) -> x + y
(x, y) -> { x = 1; y = 2; }
```

优点：

简化 JAVA 代码的繁重，提升性能

2. 注解

定义：

JAVA 代码的一种辅助修饰元素
提高程序的动态配置能力
大量第三方框架工具采用
可以自定义或者使用现有的注解类

3. 反射

定义：

程序可以访问、检测和修改它本身的状态或行为的一种能力

JAVA 中指对任意一个类都可以知道它的所有属性和方法，对任意一个对象都能够调用它的任意一个方法，是一种动态获取信息和动态调用对象的机制

优点：

只要知道的类名和所属的包名，就能够在程序运行中动态地调用任何一个类
提高了程序的可扩展性、灵活性和自适应能力，降低了耦合

缺点：

反射实际上是一种解释操作，其性能要远远低于直接调用的代码，一般只在对动态性和可扩展性要求很高的框架中使用

应用场景：

依赖注入
动态代理模式
编译器 API

写法：

四个步骤：

```
A）Class<?> rect = Class.forName("包名.Rectangle");
B）Method method = rect.getMethod("area", double.class, double.class);
C）Object obj = rect.getDeclaredConstructor().newInstance();
D）System.out.println(method.invoke(obj, 10, 20));
```

第一步：获得 Class 对象
第二步：获得指定函数名和参数类型的 Method 对象
第三步：创建对象的 Object 实例（无参构造）
第四步：打印 method 的 invoke 结果，传入实例对象和实际参数

4. 动态代理

目标：

获得一个介于买家和卖家之间、全程包办、任意灵活定制的中间商

定义：

一种设计模式，又称为委托模式
本质是一种面向接口的间接访问对象的编程思想

优点：

灵活扩展原有功能，在原有功能的基础上加上前置处理和后置处理
降低代码模块之间的耦合度

应用场景：

添加监控、审查处理等功能扩展

写法：

三个步骤：

```
A) public interface Dividable {
        int divide(int a, int b);
}
/*
        创建一个接口，被代理的类不一定要继承这个接口，但是这个接口的
        方法名最好和被代理的方法同名
*/
B) public class DivisionProxy implements InvocationHandler
//代理类定义
private Object obj;
//代理类的 Object 私有数据成员
public void setObj(Object obj) {
        this.obj = obj;
}
//代理类的 setter
public Object invoke(Object proxy, Method method, Object[] args)
//代理类 invoke 方法定义
method.invoke(obj, args);
//代理类中获得原本方法的执行结果
C) InvocationHandler handler = new DivisionProxy(new Division());
//获得代理的 handler
Dividable dividable = (Dividable) Proxy.newProxyInstance
        (Division.class.getClassLoader(),
        Division.class.getInterfaces(),
        handler);
//获得代理接口
dividable.divide(5, 3);
//执行代理方法
```

5. AOP 面向切面编程

定义：

通过预编译方式和运行期间动态代理实现程序功能的统一维护的技术

应用场景：

需要在方法前后进行功能增强和扩展

需要对方法的执行顺序进行可配置的动态调整

优势：

切面业务和主业务逻辑剥离

扩展功能不破坏原有逻辑

专注于主要业务逻辑

代码可复用

模块间解耦

6. DIP 依赖倒置原则

依赖：

程序员依赖电脑

定义：

高层模块（调用者）不依赖于底层模块（被调用者），他们都依赖于抽象

细节（具体实现）也依赖于抽象（接口或抽象类）

7. IoC 控制反转

定义：

一种设计模式，也是 DIP 的一种具体实现方案

将被依赖的低层模块的生产交给 IoC 容器完成，高层模块通过相关的容器控制程序在外部 new 一个低层模块并且注入到自己的引用中

是一种面向对象设计法则——“好莱坞法则”：别找我们，我们找你

8. DI 依赖注入

定义：

所依赖的对象由外部容器（如 IoC 容器）注入进去

方法：

Setter 注入

构造函数注入

接口注入

写法：

两个步骤：

A）
```
public static Food getFood(String foodName) {
        Class<?> foodClass = Class.forName(包名+foodName);
        return (Food) foodClass.getDeclaredConstructor()
                        .newInstance();
}
//Food 是一个接口，所有食物都要 implements 这个接口并且实现 eat()方法
```

B）
```
//Restaurant 类有一个 Food 私有数据成员
restaurant.setFood(Kitchen.getFood(food));
//setter 注入
Restaurant restaurant = new Restaurant(Kitchen.getFood(food));
//构造函数注入
```

/*
实际上是用一个接口代替了所有类型的食物，向第三方申请食物的过程变成了向第三方申请一个接口，只要拥有接口就可以拥有食物
*/

9. RPC 远程过程调用

定义：

一种广义远程调用技术框架，提供一种网络上组件间通信和传递数据的方式
发出请求的是客户程序，提供服务的是服务器

实现方案：

基于 JAVA 的 RMI
第三方框架：Netty 通信框架、Dubbo、Thrift
SOA 架构（web service）
RESTful
微服务

原理：

服务器打开对应端口的服务，并且在注册中心注册
客户端向注册中心订阅服务
注册中心给客户端返回服务地址
客户端请求服务

过程：

客户端把服务调用请求交给本地 stub
本地 stub 将参数和调用方法等信息进行编码
本地 stub 根据找到的服务地址把请求发送给服务器
服务器 stub 接收到请求后，对客户端请求进行解码
服务器 stub 根据解码结果调用本地服务，并且获得服务执行结果
服务器 stub 对执行结果编码，并且将执行结果发送给客户端
客户端 stub 接收到执行结果后解码
得到服务调用的最终结果

10. RMI 远程方法调用

基本原理：

客户端调用存放在本地的远程对象存根（stub）
存根把数据编码后发送给存放在服务器的远程对象骨架（skeleton）
骨架调用对应方法，并且把相应结果发送给存根
存根给客户端返回值或异常

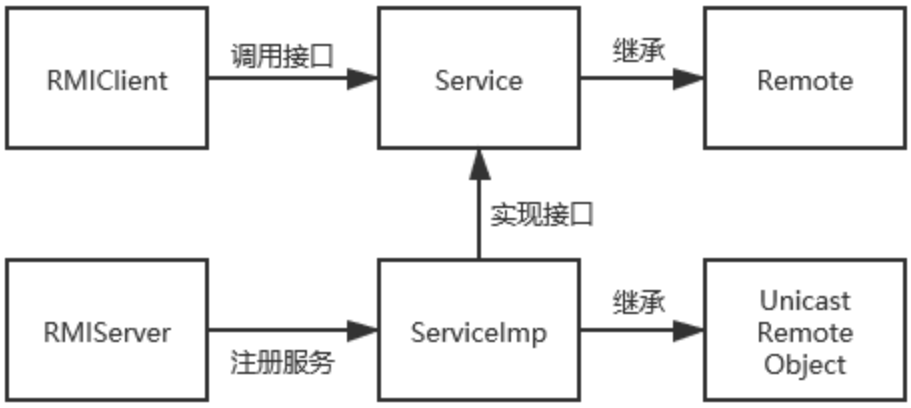
与 RPC 的不同：

RMI 基于存放在客户端的 stub 进行调用，RPC 是发送一个通讯协议
RMI 基于 JAVA，RPC 则是一个通用网络服务协议，与语言无关
RMI 的调用结果可以是一个对象

JNDI：

JAVA 命名与目录接口，一套完整的命名服务与容器绑定的 API

RMI 结构：



RMIClient 调用远程方法接口 Service
RMIServer 注册远程方法，并且实现 ServiceImp
ServiceImp 继承 UnicastRemoteObject，实现 Service 接口
Service 接口继承 Remote 接口

主要优势：

简单，易于维护和开发

主要问题：

基于 JAVA，难以多语言
注册中心不容易实现
效率和性能有瓶颈
安全性差
灵活性和扩展性差

11. 网络 I/O 模型

系统 I/O 基本方式：

阻塞：执行条件未满足时会一直等待
非阻塞：执行条件未满足时会报错
同步：多个任务执行必须一个一个来
异步：多个任务可以并发执行

常用 I/O 通信模型：

BIO：同步阻塞 I/O
NIO：同步非阻塞 I/O
多路复用 I/O
AIO：异步 I/O

多路复用 I/O：

适合用于高并发，无须多线程，速度和性能较高，同一端口可以绑定多个协议（TCP/UDP）

但是并非真正的 AIO

主要技术是 select、poll 和 epoll

select 和 poll 都是轮询并发的 socket

epoll 是 select 和 poll 的增强版

callback 机制

无连接限制

减少内存数据拷贝

12. Netty 通信框架

定义：

一个 NIO C/S 框架，用于最大程度上快速、简单、流水线化地开发协议服务器和客户端等网络应用，例如 TCP/UDP socket 服务器

应用场景：

RPC 的网络通信和传输

特殊网络服务器的通信接口（如 http 协议）

特殊网络要求（如长连接）

13. 各种 RPC 和服务资源注册与协调框架

Thrift 优点：

多语言

支持多种信息格式

支持阻塞 I/O 和多路复用 I/O

并发性能高

Thrift 缺点：

业务变化后要修改 IDL 接口定义语言

为了生产环境的可用性，接口的稳定性就得不到保证，一部分已经部署，另一部分尚未更新

某个系统的更新难以通知到其它系统

zookeeper 定义：

分布式、开源的应用程序资源协调服务，hadoop 的重要组件

zookeeper 目标：

封装易出错的关键服务，给用户提供简单高效、易用的接口

zookeeper 主要功能：

管理集群中统一或独特的信息

集群状态监控

协调资源抢占

分派计算任务

Thrift+zookeeper 优缺点：

跨语言跨平台

服务治理机制的实现比较复杂（例如访问权限、版本控制和性能措施等）

Dubbo 定义：

阿里 AOP 服务化治理方案的核心框架

Dubbo 功能：

透明化的远程服务调用
容错机制和软负载均衡
服务的自动注册和发现
网络通信模型采用 Netty，服务注册中心采用 zookeeper

Spring Cloud 定义：

一系列框架的集合，拥有基于 Spring Boot 的开发便利性，提供了一整套企业级分布式系统云应用的开发解决方案

Spring Cloud 功能：

服务治理
配置中心
消息总线
负载均衡
断路器
数据监控
分布式会话和集群状态管理

eureka 定义：

包含在 Spring Cloud 中，用于服务注册和发现的服务治理框架

eureka 服务注册与管理

服务器向服务发现中心注册/续约/注销服务
用户向服务发现中心获取服务列表，并且向服务器调用服务

基于服务的分布式系统开发方案：

RMI 基于 JAVA
Thrift+zookeeper：高效快速、跨语言跨平台
Dubbo+zookeeper：服务治理功能强大
Spring Cloud 解决方案（包含 eureka）：Spring 有庞大生态圈，功能完善

14. zookeeper 与 eureka

CAP 含义：

C：数据一致性
A：服务可用性
P：分区容错性

CAP 原则：

任何分布式系统最多只能保证上述三条性质中的两条

二者区别：

zookeeper 侧重于 CP，eureka 侧重于 AP
数据高峰使用 eureka，数据核对使用 zookeeper

15. 一套完整的数据瓶颈和并发访问解决方案

采用 Dubbo 的负载均衡，配置多台服务器
Eureka 和 zookeeper 混合使用
使用 redis 存储经常使用的数据，加快查询速度
课程做数据库拆分

16. MOM 面向消息中间件

同步与异步通信（与 I/O 的同步异步区别）：

同步：应用发送请求后一直等待直到获得结果，请求时服务必须在线

异步：应用发送请求，并且在将来检查请求是否被处理，请求时服务不一定在线

MOM 定义：

面向消息中间件允许一个网络应用向另一个应用通过第三方中间件发送消息，无论另一个网络应用是否在线

同步优缺点：

编程容易、输出立即可知、更易发现和恢复错误、实时性能更好

服务必须在线、请求服务会占用大量服务器资源、要求面向连接的通信协议

异步优缺点：

请求无需指定服务器、服务器无需在线、资源可以及时释放、可以使用非面向连接的协议

响应时间不可预测、错误恢复更困难、编程难度较大

消息队列特性：

双方不必同时在线

既可以是发送方又可以是接收方

应用之间可能一对多，也可能多对一

不用关心 OS、编程语言、底层通信协议

JMS（JAVA Message Service）：

SUN 提出的统一各种 MOM 的系统接口的规范，包含点对点和发布/订阅两种消息模型，提供可靠消息传输、事务和消息过滤等机制

JMS 与 MQ：

JMS 不是消息队列，也不是消息队列协议，而是一套规范的 JAVA API 接口

JMS 与 MOM 厂商无关，需要各个厂商去实现，大部分 MOM 都支持 JMS

可以使用 JMS API 连接使用 Stomp 协议的产品（例如 ActiveMQ）

类似 JDBC 和 mysql 的关系

JMS 两种模型：

点对点：每个消息只能有一个消费者，并且只能消费一次，消费者未收到消息时就会处于阻塞状态（MOM 上有多个 channel）

发布/订阅：每个消息可以有多个消费者，订阅某个主题的消费者只能消费自他订阅主题以来发布的消息（MOM 上有多个 topic）

ActiveMQ 定义：

Apache 出品，最流行的、功能强大的开源消息总线，是一个完全支持 JMS 和 J2EE 规范的 JMS Provider 实现

广泛应用于实际生产环境中，社区成熟，学习文档丰富

ActiveMQ 特性：

支持多语言和多协议

支持持久化和事务

内嵌至 Spring

支持部署到 JBOSS 服务器

支持集群

界面友好、测试方便

17. 分布式数据存储

分布式存储概念：

充分利用网络中分散的机器上的磁盘空间
它们构成成一个虚拟的存储设备
物理上分散，逻辑上集中

分布式存储类型：

结构化数据：由用户定义的严格的数据，如二维表
半结构化数据：数据格式不统一，如 json
非结构化数据：数据本身不存在结构，如图片、声音

分布式数据存储理论：

数据分片：

水平分片：按行拆分
垂直分片：按列拆分，拆分后的每一列都含有主键
混合分片：在水平或垂直分片的基础上再垂直分片

数据复制：维护关系 r 的多个完全相同的副本，各个副本存储在不同的节点上
（全复制：系统中每个节点都有 r 的一个副本/主副本：人为指定，简化副本管理）

数据库拆分（与数据分片区别）：

水平拆分：对单个业务的数据库进行拆分，得到多个储存同一业务的数据库
垂直拆分：对多个业务进行拆分，得到不同单个业务的数据库

水平拆分方法：

选择一个拆分字段
通过路由算法确定数据存放在哪个数据库中
查询时同样可以利用该算法进行查询

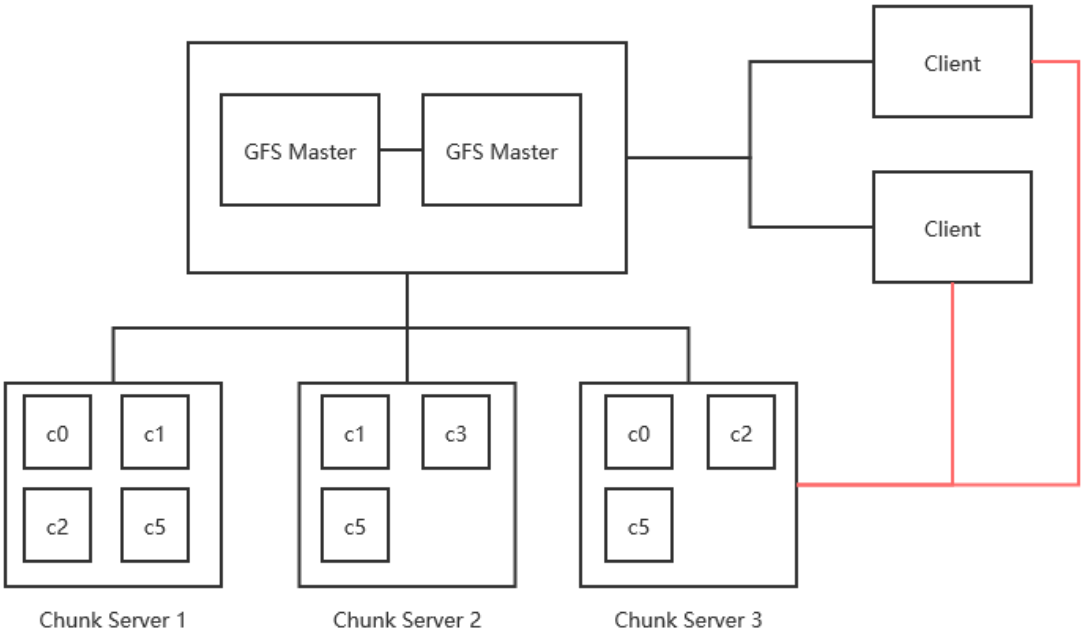
GFS 功能：

Google File System
数据冗余、低成本数据快照、附加文件记录

GFS 三个角色：

Client：一组 GFS 提供给用户的接口，应用程序可以直接调用
Master（主服务器）：GFS 的管理节点，存储与数据文件相关的元数据
Chunk Server（数据块服务器）：负责具体的存储，可以有多个（GFS 采用副本的方式实现容错，每一个 Chunk 默认有三个副本）

GFS 结构：



（注意数据传输是直接发生在 Chunk Server 和 Client 之间的，Master 负责管理元数据）

NOSQL 数据库：

应该被称作非关系型数据库
分布式、轻量级
主要用于处理半结构化数据
支持水平扩展
一般不支持 A（原子）C（一致）I（隔离）D（持久）原则

NOSQL 兴起的原因：

高并发
海量数据
高可扩展性和高可用性
不要求严格数据库事务
不要求严格读写实时性
不包含大量复杂 SQL 查询

NOSQL 与关系数据库的比较：

A）关系数据库优点：有完整的关系代数理论，支持 ACID，基于索引可以高效查询，技术成熟

缺点：可扩展性较差，不能支持海量数据，数据模型过于死板，不能支持 WEB2.0，事务机制影响了系统性能，牺牲存储空间交换了查询性能

B）NOSQL 优点：支持超大规模数据，数据类型灵活，横向扩展能力强

缺点：缺乏数学理论支撑，查询效率不高，维护困难，大多不支持 ACID

NOSQL 和关系数据库应用场景：

NOSQL：互联网企业的非关键业务（例如数据分析）
关系数据库：电信、银行等关键领域，需要事务强一致性

BASE 理论：

基本可用：分布式系统一部分不可用时，其他部分仍可以正常工作

软状态：状态可以有一段时间不同步

最终一致性：可以暂时处于不一致状态，但是系统必须保证一定时间后能读取到更新后的数据

NOSQL 四大逻辑类型：

A）键值数据库：键是字符串，值是任意类型数据

应用场景：频繁读写，数据模型简单，例如内容缓存

优点：扩展性和灵活性高，大量写操作时性能高

缺点：无法存储结构化信息，查询效率低

不适用：利用值做查询，需要存储数据之间的关系，需要事务支持

B）列族数据库： 存储列族（类似一种动态定义的二维表）

应用场景：拥有动态字段，可以容忍副本短期不一致，分布式大数据

优点：查找速度快，容易进行分布式扩展，复杂性低

缺点：功能少，大都不支持事务强一致性

不适用：需要 ACID 支持的情形

C）文档数据库：存储版本化的文档（一组自我描述的数据记录）

应用场景：存储、索引和管理面向文档或类似半结构化的数据，例如后台大量读写和使用 json 的程序

优点：支持高并发，可以将经常查询的数据存储在同一个文档中，数据结构灵活，索引可以构建在键上或者文档本身的内容上

缺点：缺乏统一的查询语法

不适用：不支持文档间的事务

D）图形数据库：存储图形结构

应用场景：处理具有高度相关互联关系的数据，例如模式识别、社交网络、推荐系统

优点：支持复杂的图形算法，可以构建灵活的关系图谱

缺点：复杂性很高，不支持大规模数据处理

Redis 定义：

开源、支持网络、基于内存、键值对 NOSQL 数据库

Redis 特点：

高性能

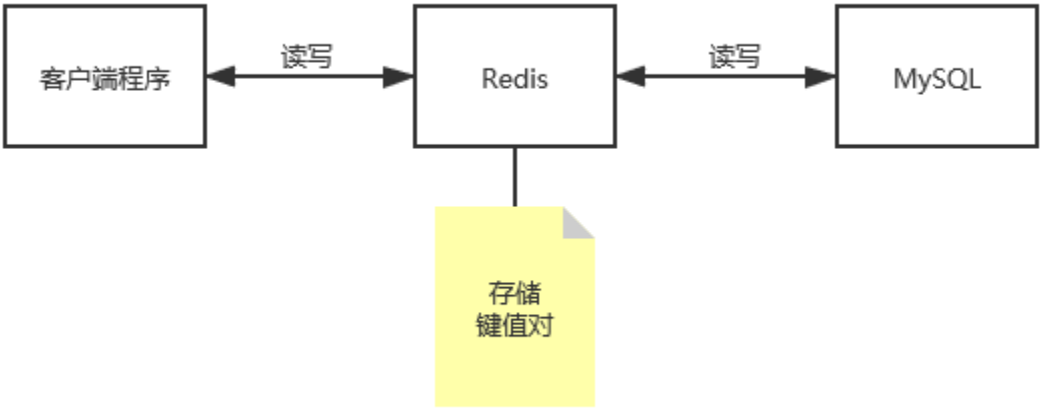
数据类型丰富

持久化

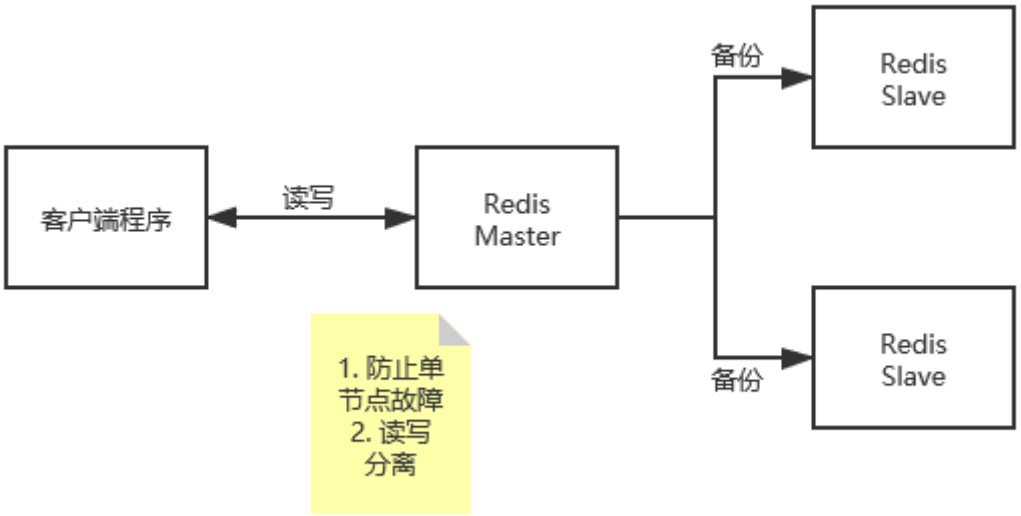
单线程

订阅/发布模型

Redis 作为服务器缓存结构：



Redis 副本集结构：



关系型数据库转化为 Redis：

Set/List 存储所有信息
利用 string 结构：set (a:b:c, d)
利用 hash：hset (a:b, c, d)
利用 json 格式的 string：set (a, {“b”:b, “c”:c, “d”:d})

18. 分布式架构

SOA 概念：

面向服务的体系结构
将应用程序功能发送给用户或者其他服务
粗粒度、松耦合
服务之间利用简单、精确的接口通讯，不涉及底层通讯模型

SOA 特点：

服务着眼于具体业务
服务虽然粗粒度，但是可控且可重用
集成的目的是形成新的服务
SOA 需屏蔽细节
SOA 让各个业务保持松散

SOA 优点：

使用接口通信，降低耦合度
把项目拆分成小项目，便于开发
增加功能时只需增加小项目
可进行分布式部署

SOA 缺点：

远程通信接口开发增加工作量
业务增多后 SOA 的结构会越来越复杂
新技术发展促使 SOA 进步

WebService 定义：

一种经典的、常用的 SOA 服务接口实现技术
SOA 是一种架构，WebService 是利用标准实现的服务

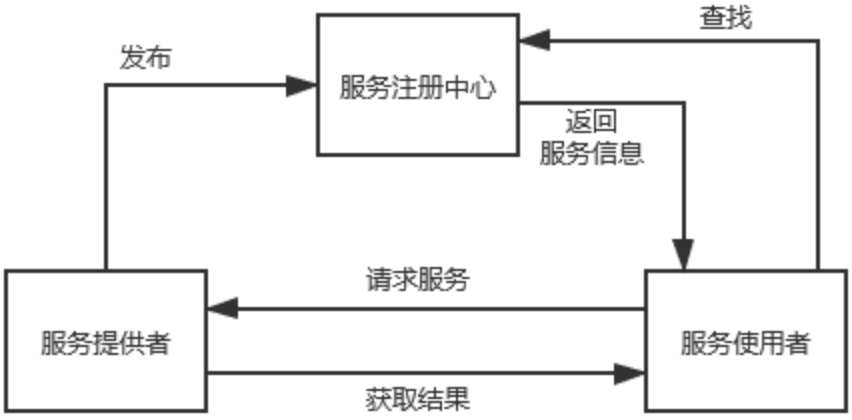
WebService 特点：

封装性良好
松耦合
高度可集成性，跨平台
易于构建

WebService 应用场合：

跨防火墙通信
应用程序集成
B2B 集成
软件和数据的重用

WebService 结构（类似 RPC 结构）：



服务提供者：在注册中心注册并发布服务，响应服务
服务注册中心：一边接受服务，一边查找使用者需要的服务
服务使用者：根据具体应用需求调用服务

三种主流 WebService 实现方案：

SOAP

REST

XML-RPC

SOAP 定义：

简单对象访问协议

一种轻量级、跨平台的数据交换协议，主要用于 WebService，基于 XML 构建的格式使得它可以被第三方网站调用

不仅描述了消息格式，还描述了如何利用 HTTP 传输消息

SOAP 结构：

SOAP Envelope：描述消息如何处理的框架

SOAP 编码规则：数据的编码规则

SOAP RPC：远程过程调用和应答的协议

SOAP 绑定：定义了一种底层传输协议完成数据交换

REST 定义：

表述性状态转移

分布在各处的资源由 URI 确定，客户端应用通过标准 HTTP 方法（get/post/put/delete）来获取资源的表述（json、XML）并且实现转化

不是协议，不是架构，是一种抽象的软件设计理念

XML-RPC 定义：

用 XML 封装调用函数，使用 HTTP 协议作为传输机制

逐渐发展成了 SOAP

三种方法比较：

XML-RPC 逐渐被 SOAP 取代

SOAP 比 REST 更成熟、更安全

REST 效率更高

SOAP 应用：

假设有以下六个服务：

Country()：输入国家编码，输出国家名称

YellowPages()：输入企业名称，输出企业代码、所在国家编码

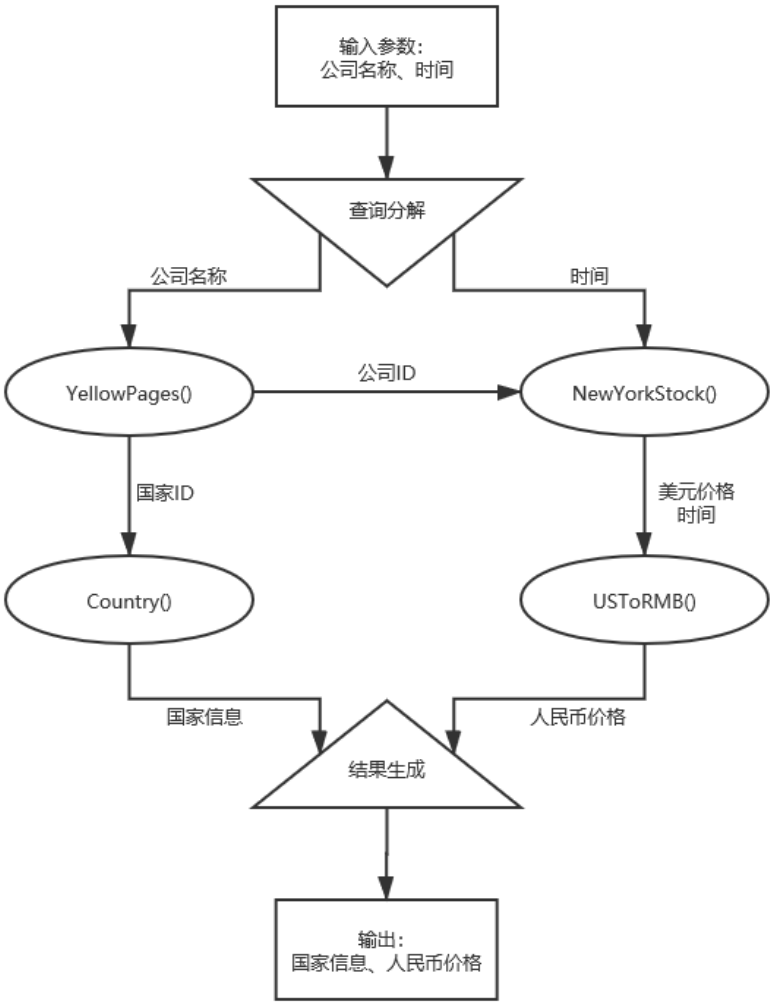
NewYorkStock()：输入公司代码、时间，输出该公司在纽约的股票价格

LondonStock()：输入公司代码、时间，输出该公司在伦敦的股票价格

USToRMB()：输入美元价格、时间，输出对应的人民币价格

UKToRMB()：输入英镑价格、时间，输出对应的人民币价格

用户想通过“跨国公司名称”和“时间”找出“该跨国公司在纽约的股票折合成人民币的价格”和“该公司所在国家的信息”



RESTful 架构：

符合 REST 原则的一种互联网软件架构

每一个 URI 表示一种资源

客户端与服务器间传递资源的表现层

客户端通过 HTTP 动词操作服务端资源，实现“表现层状态转化”

RESTful API：

符合 RESTful 架构的一套互联网 API 设计理论

RESTful API 举例：

[GET] http://api.test.com/books/1

查询编号为 1 的图书

[DELETE] http://api.test.com/books/2

删除编号为 2 的图书

URI 设计原则：

一般不含动词，使用名词复数

如果某动作不能用 HTTP 表示，则把它作为一种资源

参数的设计可以有冗余

常见参数：

limit：返回记录数量

offset：返回记录开始位置

page、per_page：指定页码和每页记录数

状态码：

2xx：成功

3xx：重定向

4xx：客户端错误

5xx：服务端错误

版本化两种方法：

在 URL 中表示版本信息：http//api.test.com/v1

在请求头中传输版本信息

WebService 总结：

SOAP：技术成熟稳定、安全性高，用于银行、证券等大型企业级解决

REST：轻量级解决方案，性能优秀，用于普通公司的业务集成和移动终端

微服务定义：

一组独立部署的服务

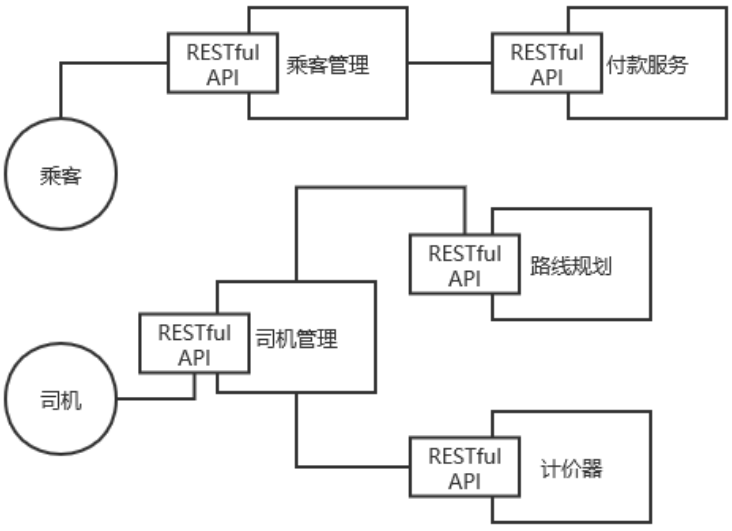
基于业务划分

各自运行不同的进程

通过 HTTP 等轻量级通信协议互相通信

最小化集中管理

微服务架构举例：



微服务与 SOA 的比较：

SOA 粗粒度，微服务细粒度

SOA 高度依赖 ESB（企业通信总线），微服务的服务集成相对松散

Docker 作用：

一组服务运行的宿主

为服务运行提供一致的运行环境

轻松地迁移服务

支持持续交付和部署

维护和扩展方便

比传统虚拟机更方便、高效

Docker 容器与虚拟机的区别：

Docker 上没有客户机操作系统

Docker 容器由 Docker 引擎进行统一管理，资源消耗少

虚拟机使用管理程序操纵硬件，Docker 容器可以直接操纵硬件

19. 大数据处理工具

Hadoop 定义：

一个 Apache 开发的分布式系统基础架构

用户可以在不了解分布式底层细节的情况下开发分布式程序，并且充分利用集群的威力进行高速运算和存储

最核心的两大技术是 HDFS 文件系统和 MapReduce 数据计算模型

HDFS 主要组件：

NameNode：

- 储存元数据在内存中
- 储存文件、Block 和 DataNode 之间的映射关系
- 有两种结构：
  - FsImage：维护文件树结构和所有文件夹的元数据
  - EditLog：记录对于文件的修改、增加、删除操作日志

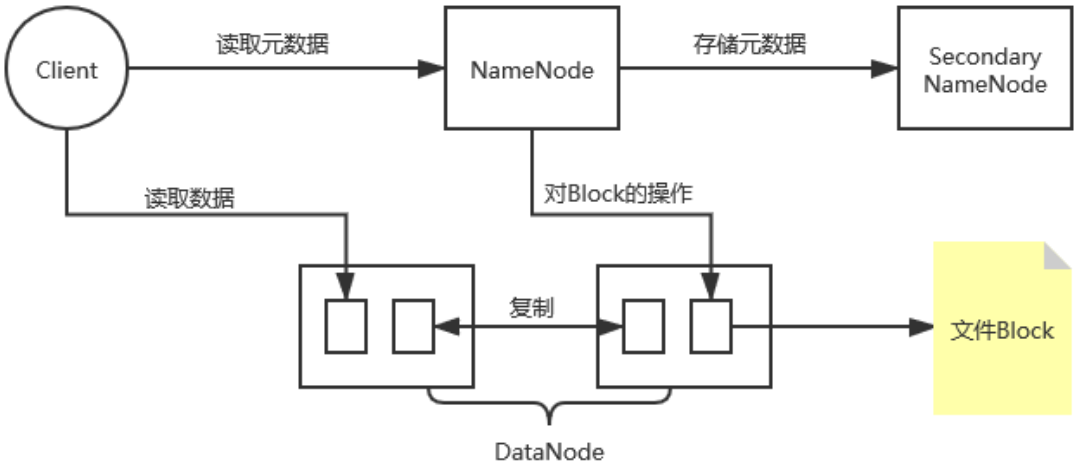
DataNode：

- 存储文件内容在磁盘
- 维护了 Block id 到本地文件的映射关系
- 向 NameNode 定期发送所储存的块的列表
- 也负责客户端或者 NameNode 的调度时文件的检索和读取

SecondaryNameNode：

- 存储 NameNode 元数据的备份
- 单独运行在一台机器上
- 解决 EditLog 不断变大的问题，加快重启速度

HDFS 架构：



HDFS 数据错误与恢复：

名称节点出错：利用 SecondaryNameNode 进行 FsImage 和 EditLog 的备份

数据节点出错：数据节点定期发送“心跳”信息报告状态，数据节点故障导致备份的减少后 HDFS 会为这些块自动生成新副本

数据出错：采用 MD5 或 SHA1 对文件进行校验，如果出错则读取备份，NameNode 定期检查损坏文件并且重新复制

HDFS 命令：

五种命令：

A）hadoop dfs -mkdir -p /test1/test2/test3

-p 可以直接创建一个不存在的目录的子目录

B）hadoop dfs -mv /test4 /test1

把 test4 文件夹移动到 test1 目录下

C）hadoop dfs -ls /test1

查看 test1 目录下的所有文件

D）hadoop dfs -put test.txt /test1/test4

把服务器当前目录的 test.txt 上传到 Hadoop 的/test1/test4 目录下

E）hadoop dfs -rm -f -r /test1/test4 /test1/test2

-f 不显示任何提示信息 -r 递归删除该目录下的所有文件，包括该目录本身

-skipTrash 不放入回收站

注：Linux 创建文件使用 touch test.txt，Hadoop 获取帮助使用 hadoop dfs

YARN 定义：

Yet Another Resource Negotiator

一个分层通用资源管理系统，为上层应用提供统一的资源管理和调度

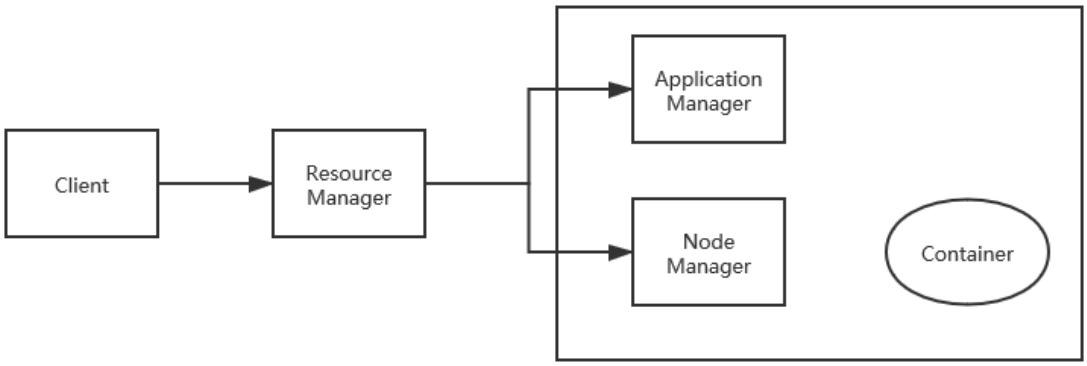
YARN 优点：

减少资源消耗

检测子任务状态分布式化

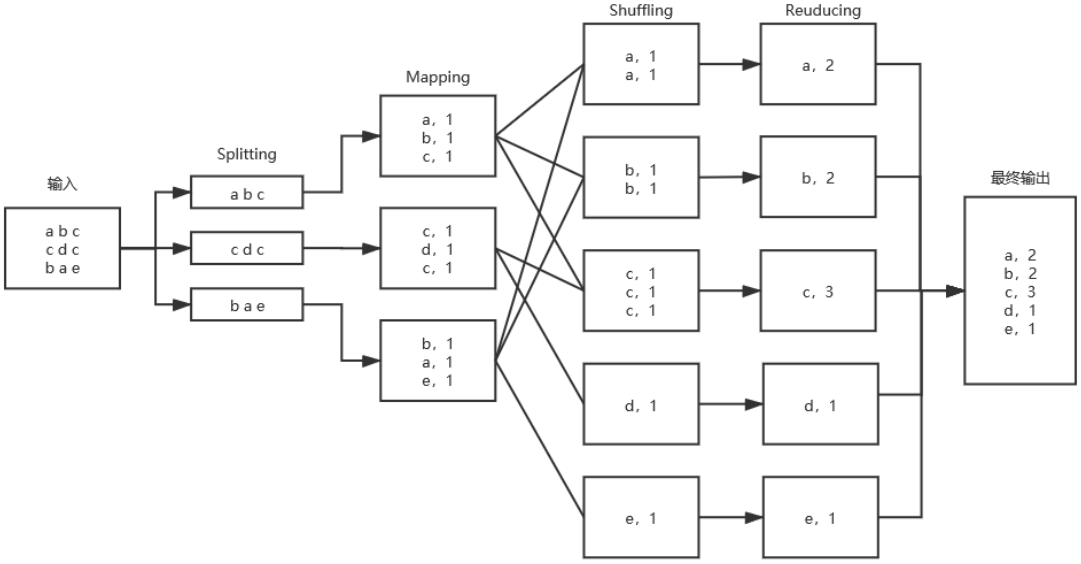
更安全且优美

YARN 体系结构：



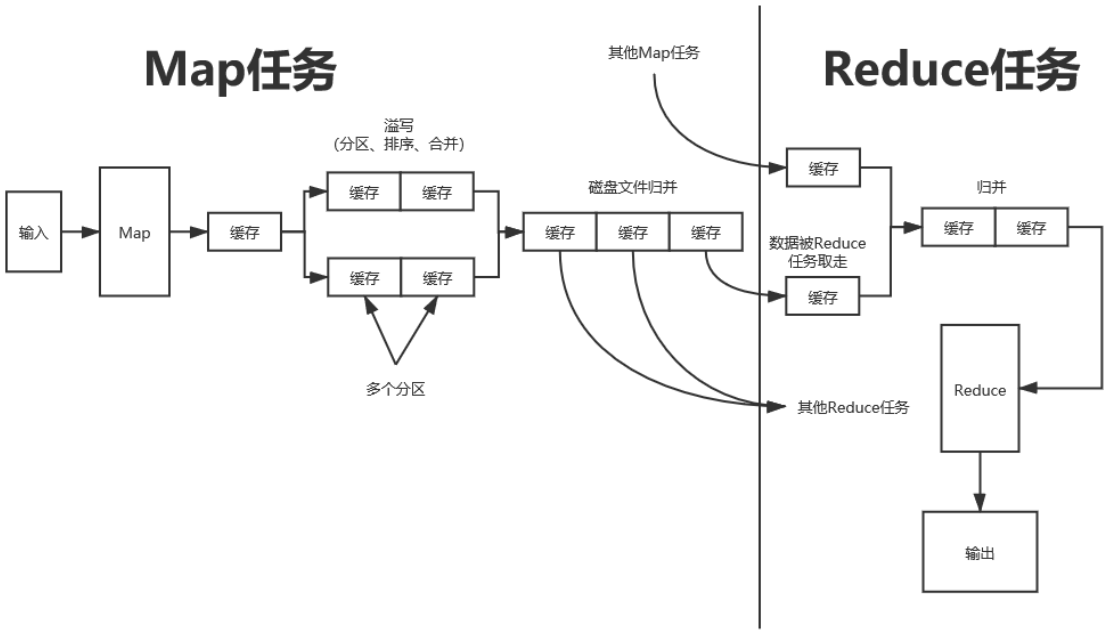
ResourceManager：
处理客户端请求
启动/监控 ApplicationMaster
监控 NodeMaster
资源分配、调度
ApplicationMaster：
为应用程序申请资源，并分配给内部任务
任务调度、监控和错误处理
NodeManager：
单个节点上的资源管理
处理 ResourceManager 和 ApplicationManager 的命令

MapReduce 工作流程：



Map：把大量数据进行拆分
Reduce：对拆分后的数据进行规约
不同的 Map 和 Reduce 之间不会进行通信，数据交换由框架本身执行

Shuffle 过程：



MapReduce 代码：

MapReduce 和 JAVA 数据类型对应关系：

Long　　LongWritable
String　　Text
Integer　　Intwritable
null　　NullWritable

（注意 Text 转 String 用 toString()，IntWritable 转 Integer 用 get()）

一些案例：

A）计算单词的出现次数

Mapper：

```
//拿到传入进来的一行内容，把数据类型转化为 String
String line = value.toString();
//将这一行内容按照分隔符进行一行内容的切割 切割成一个单词数组
String[] words = line.split(" ");
//遍历数组，每出现一个单词  就标记一个数字 1  <单词，1>
for (String word : words) {
	/*
		使用 MapReduce 程序的上下文 context  把 mapper 阶段处理的数据发送
		出去，作为 reduce 节点的输入数据
	*/
	context.write(new Text(word),new IntWritable(1));
}
```

Reducer：

```
//定义一个计数器
int count = 0;
//遍历一组迭代器，把每一个数量 1 累加起来就构成了单词的总次数
for(IntWritable value:values){
       count += value.get();
}
//把最终的结果输出
context.write(key,new IntWritable(count));
```

B）计算某个部门员工的平均工资

在保证员工不重复出现的情况下，Mapper 输出部门和工资的键值对，Reducer 把工资求和并且每扫描到一个键值对就把 peopleCnt 加一，peopleCnt 就是部门人数

C）找出比工资高于平均值的员工姓名

Mapper 输出第一种键值对是 0 和员工工资，第二种键值对是 1 和员工姓名+员工工资，Reducer 先统计键为 0 的员工总工资和员工人数，在第一次统计到键为 1 的员工姓名+员工工资时计算平均工资，这样可以在一次循环中就完成两个任务，之后进行比较即可

D）找出工资前三的员工姓名

Mapper 输出员工姓名和员工工资的键值对，Reducer 设置三个私有成员变量，遍历一次找出前三的姓名和工资，利用 Context 写入即可