分布式监控

Prometheus

参考文档

官方文档：Overview | Prometheus

github 仓库：Prometheus (github.com)

中文版文档：序言 - Prometheus 中文文档 (fuckcloudnative.io)

参考视频：【尚硅谷】Prometheus+Grafana+睿象云的监控告警系统哔哩哔哩bilibili

简介

系统监控和告警工具包

优势

- 由指标名称和和键/值对标签标识的时间序列数据组成的多维数据模型。
- **监控服务的内部运行状态**（我们需要的）
  - 基于 Prometheus 丰富的 Client 库，用户可以轻松地在应用程序中添加对 Prometheus 的支持，从而让用户可以获取服务和应用内部真正的运行状态
- 强大的数据模型
  - 所有采集的监控数据均以指标（metric）的形式保存在**内置的时间序列数据库（TSDB）**当中，所有的样本除了基本的指标名称外，还包含一组用于描述该样本特征的标签
- 高效：对于单一 Prometheus Server 示例而言它可以处理：
  - 数以百万的监控指标，每秒处理数十万的数据点
- 强大的查询语言 PromQL。
- 不依赖分布式存储；单个服务节点具有自治能力。
- 时间序列数据是服务端通过 HTTP 协议主动拉取获得的。
- 也可以通过中间网关来推送时间序列数据
- 可以通过静态配置文件或服务发现来获取监控目标。支持多种类型的图表和仪表盘。

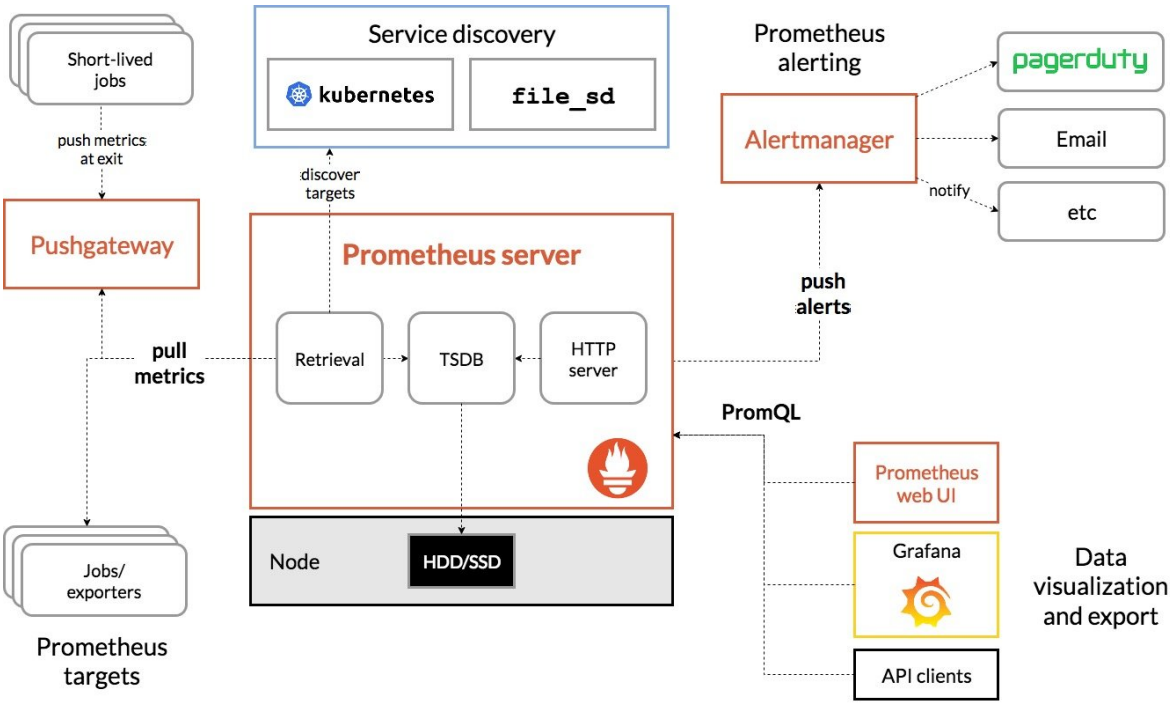
组件

- Prometheus Server 作为服务端，用来存储时间序列数据。
- 客户端库用来检测应用程序代码。
- 用于支持临时任务的推送网关。
- Exporter 用来监控 HAProxy，StatsD，Graphite 等特殊的监控目标，并向 Prometheus 提供标准格式的监控样本数据。
- alartmanager 用来处理告警。
- 其他各种周边工具。

其中大多数组件都是用 Go 编写的，因此很容易构建和部署为静态二进制文件。

Prometheus 的架构

Prometheus 的整体架构以及生态系统组件如下图所示：



Prometheus Server 直接从监控目标中或者间接通过推送网关来拉取监控指标，它在本地存储所有抓取到的样本数据，并对此数据执行一系列规则，以汇总和记录现有数据的新时间序列或生成告警。可以通过 Grafana 或者其他工具来实现监控数据的可视化。

工作原理

- Prometheus 所有采集的监控数据均以指标（metric）的形式保存在内置的时间序列数据库当中（TSDB）：属于同一指标名称，同一标签集合的、有时间戳标记的数据流。除了存储的时间序列，Prometheus 还可以根据查询请求产生临时的、衍生的时间序列作为返回结果。
- Exporter

  在 Prometheus 的架构设计中，Prometheus Server 主要负责**数据的收集，存储并且对外提供数据查询支持**，而实际的监控样本数据的收集则是由 **Exporter** 完成。因此为了能够监控到某些东西，如主机的 CPU 使用率，我们需要使用到 Exporter。Prometheus 周期性的从 Exporter 暴露的 HTTP 服务地址（通常是 /metrics）拉取监控样本数据。

  Exporter 可以是一个相对开放的概念，其可以是一个独立运行的程序独立于监控目标以外，也可以是直接内置在监控目标中。只要能够向 Prometheus 提供标准格式的监控样本数据即可。

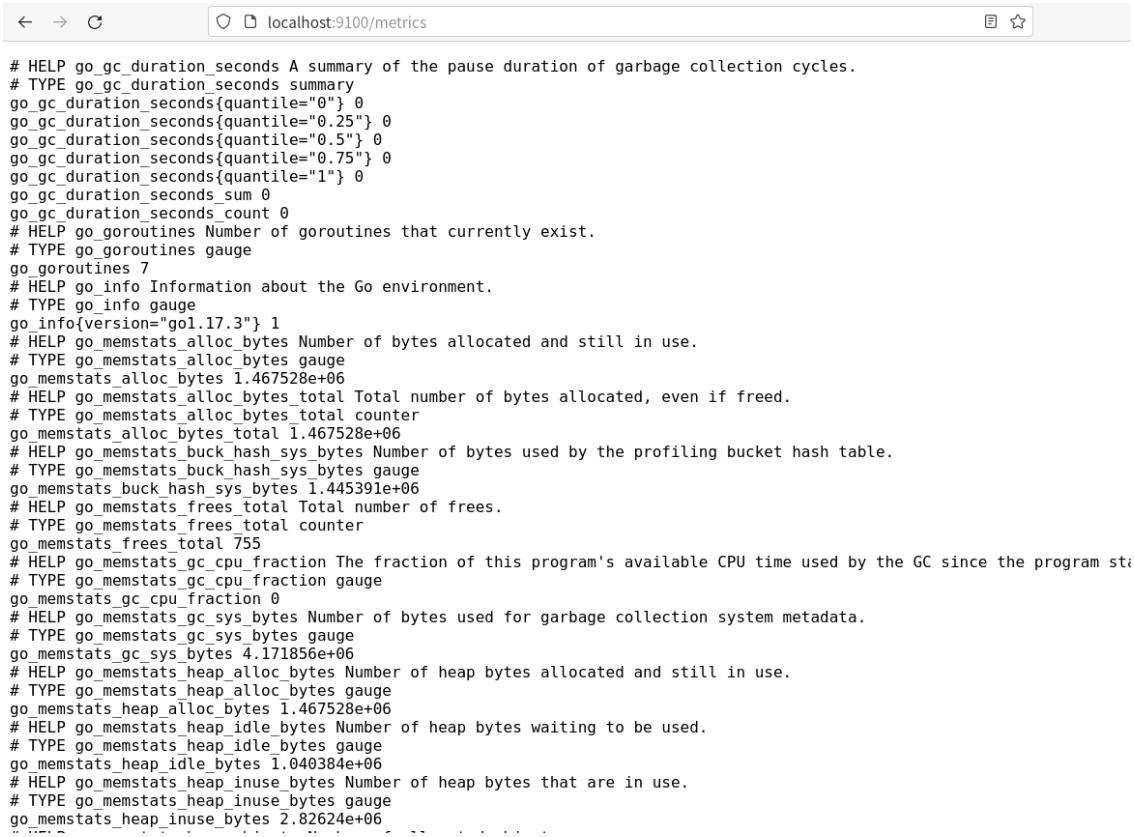
  为了能够采集到主机的运行指标如 CPU, 内存，磁盘等信息。我们可以使用 Node Exporter。

安装

- 在官网Download | Prometheus下载相关压缩包，并解压

```
node_exporter-1.3.1.linux-amd64.tar.gz*
prometheus-2.34.0.linux-amd64.tar.gz*
pushgateway-1.4.2.linux-amd64.tar.gz*
```

  - `tar -zxvf <filename>`
- 安装好 node_exporter 后，打开 `localhost:9100`，即可监控到本机

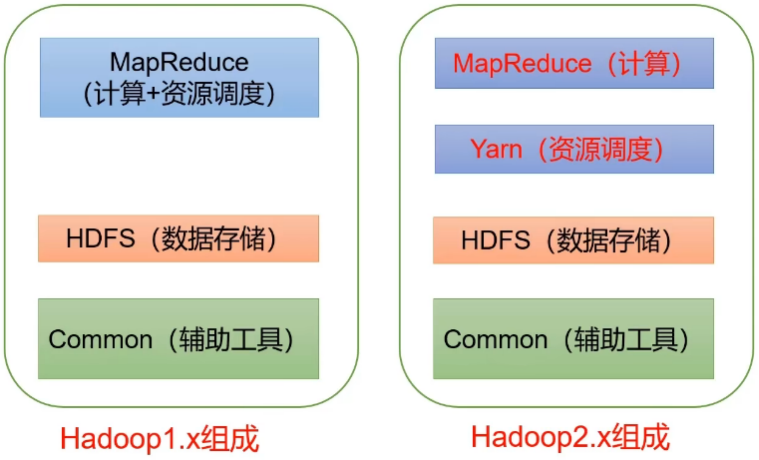


这些监控的指标即可被 Prometheus 获取，存储到其 TSDB 中

Hadoop

简介

- Hadoop 是一个由 Apache 基金会所开发的**分布式系统基础架构**，主要解决**海量数据的存储和分析计算问题**
- 三大发行版本：Apache、Cloudera、Hortonworks
- 优势：
    - 高可靠性：Hadoop 底层维护多个数据副本
    - 高可扩展性：在集群间分配任务数据，可方便地扩展数以千计的节点，可实现动态增加、动态删除节点
    - 高效性：在 MapReduce 的思想下，Hadoop 是并行工作的
    - 高容错性：能够自动将失败的任务重新分配
- 组成：



在Hadoop1.x时代，Hadoop中的MapReduce同时处理业务逻辑运算和资源的调度，耦合性较大。

在Hadoop2.x时代，增加了Yarn。Yarn只负责资源的调度，MapReduce只负责运算。

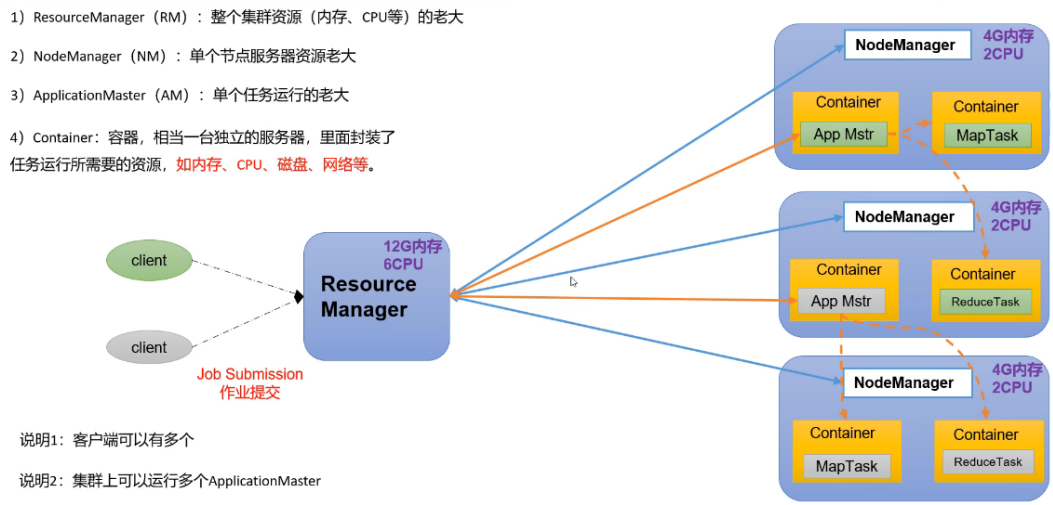
Hadoop3.x在组成上没有变化。

HDFS

- Hadoop Distributed File System，简称 HDFS，是一个**分布式文件系统**
- 组件：
  - **NameNode（nn）**：存储文件的**元数据**，如文件名、文件目录结构、文件属性（生成时间、副本数、文件权限），以及每个文件的**块列表**和**块所在的 DataNode** 等。
  - **DataNode（dn）**：在本地文件系统**存储文件块数据**，以及快数据的校验和
  - **Secondary NameNode（2nn）**：每隔一段时间**对 NameNode 元数据备份**

YARN

- 是 Hadoop 的**资源管理器**（主要管理 CPU 和内存）
- 包含
  - **Resource Manager**：管理整个集群资源
  - **NodeManager**：管理单节点服务器资源
  - ApplicationMaster：管理单个任务
  - Container：容器，相当于一台独立的服务器，里面封装了任务运行所需要的资源，如内存、CPU、磁盘、网络等



MapReduce

- 负责 Hadoop 中计算的功能
- MapReduce 将计算过程分为两个阶段
    - Map：并行处理输入数据
    - Reduce：对 Map 结果进行汇总

HDFS、YARN、MapReduce 关系

- 通俗来讲，HDFS 负责存储，YARN 负责资源管理，MApReduce 负责计算
- 当一个客户端（client）提交任务请求，ResourceManager 会在某个节点，创建一个容器（Container），容器内即为客户提交的任务（ApplicationMaster）。

  ApplicationMaster 创建好后，会根据任务量向 ResourceManager 申请资源，ResourceManager 创建 Map 任务，在多个节点上创建 Map Task，进入 MapReduce 中的 Map 阶段。每个 Map Task 独立工作，负责检索其所在的 DataNode。

  Map Task 会将检索结果存储到某个 DataNode，即为 Reduce 阶段。更新 NameNode。

Hadoop 运行环境搭建

安装配置 CentOS

- 安装过程与一般创建虚拟机过程无异，使用 vmware，建议将此虚拟机命名为 hadoop100，方便后续管理与克隆
- 可以在安装操作系统阶段修改主机名（也可后续命令行修改）
    - 注：可以选“最小安装”，也就是纯命令行界面，也可以选择有图形化界面的
    - 又注：算了， 还是别最小安装了，连 vim 都得自己安装配置.
- 配置 IP 地址
    - 配置 VM 的 IP 地址：编辑-虚拟网络编辑器-VMnet8-更改设置-VMnet8

      子网 IP 设置为：192.168.10.0

      NAT设置：网关IP：192.168.10.2 （网关最后一个数字通常是2）
    - windows（即宿主机）：设置-网络-更改适配器选项-VMnet8-右键点击-属性-双击“Internet 协议版本 4 (TCP/IPv4)”

      默认网关：192.168.10.2

      首选 DNS 服务器：192.168.10.2

      备用 DNS 服务器：8.8.8.8
    - 进入虚拟机，打开终端，输入 `su root`，输入密码，进入 root 用户

      `vim /etc/sysconfig/network-scripts/ifcfg-ens33`

      把 BOOTPROTO 修改为 static，即将 IP 设置为静态 IP

      在最后加以下内容：

      IPADDR=192.168.10.100

      GATEWAY=192.168.10.2

      DNS1=192.168.10.2

      效果如下图

- 修改主机名称：`vim /etc/hostname` 建议设置为 hadoop100，方便后续管理与克隆
- 配置主机映射：即将主机名与其 IP 地址建立映射，作用相当于 C 语言的 #define，也即后续出现 IP 地址的地方，均可用主机名代替，后续需要修改本机 IP 地址时，无需在修改后，将之前主机内所有出现 IP 地址的地方都进行修改

  `vim /etc/hosts`

  添加如下内容：

  192.168.10.100 hadoop100

  192.168.10.101 hadoop101

  192.168.10.102 hadoop102
- 上述配置好以后，重启下虚拟机，`reboot`

kubernetes

- 调研这个的原因：

> 随着容器技术的迅速发展，Kubernetes 已然成为大家追捧的容器集群管理系统。Prometheus 作为生态圈 Cloud Native Computing Foundation（简称：CNCF）中的重要一员,其活跃度仅次于 Kubernetes, 现已广泛用于 Kubernetes 集群的监控系统中。

OpenResty

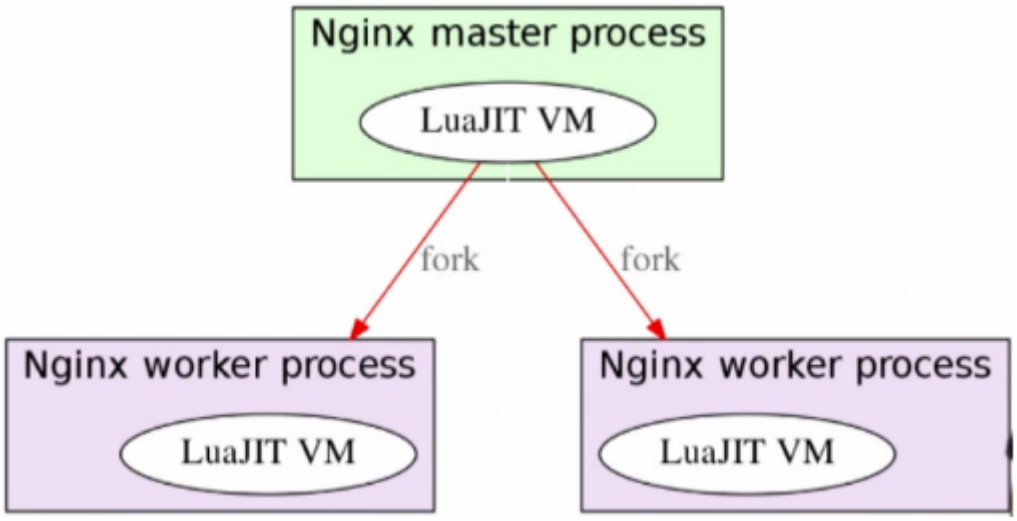
原理

- 使用安装教程 OpenResty 使用介绍 | 菜鸟教程 (runoob.com)

  原理简介 OpenResty工作原理 - 简书 (jianshu.com)

  安装 (7条消息) CentOS下OpenResty安装详解*nmtcttn*的博客-*CSDN*博客openresty安装
- OpenResty 是一个基于 NGNIX 的可伸缩的 Web 平台。它是一个强大的 Web 应用服务器，Web 开发人员可以使用 Lua 脚本语言调动 Nginx 支持的各种 C 以及 Lua 模块，更主要的是性能方面，OpenResty 可以快速构造出足以胜任 10K 以上并发连接响应的超高性能 Web 应用系统
- Nginx 采用的是 `master-worker` 模型，也就是一个 `master` 进程管理多个 `worker` 进程，基本的时间处理都放在 `worker` 进程中，`master` 进程负责全局初始化以及对 `worker` 进行的管理。

OpenResty 中，每个 `worker` 进程使用一个 LuaVM，当请求被分配到 `worker` 时，将在这个 LuaVM 中创建一个 `coroutine` 协程，协程之间数据隔离，每个协程都具有独立的全局变量。



Nginx

- Nginx 是一个高性能的 HTTP 和反向代理服务器，特点是占有内存少，并发能力强
- Nginx 设计为主进程和多个工作进程的工作模式，每个进程是单线程来处理多个连接，每个工作进程采用了非阻塞 I/O 来处理多个连接，从而减少线程上下文切换，从而实现高性能、高并发。
- Nginx 采用多进程模式，对于每个 `worker` 进程都是独立的，因此不需要加锁，节省了锁带来的性能开销。
- Nginx 采用异步非阻塞的方式去处理请求，异步非阻塞就是当一个线程调用出现阻塞而等待时，其他线程可以去处理其他任务

ngx_lua

- ngx_lua 是将 Lua 嵌入 Nginx，让 Nginx 执行 Lua 脚本，并且高并发、非阻塞地处理各种请求
- 开发者可以采用串行地方式编写程序。ngx_lua 会自动地在进行阻塞地 I/O 操作处中断，保存上下文，然后将 I/O 操作委托给 Nginx 事件处理机制，在 I/O 操作完成后，ngx_lua 会回复上下文，程序继续执行，这些操作都是对用户程序透明的。
- `ngx_lua` 模块的原理
    - 每个工作进程 `worker` 创建一个 Lua 虚拟机（LuaVM）,工作进程 `worker` 内部协议共享 VM。
      -每个 Nginx I/O 原语封装后注入 Lua 虚拟机，并允许 Lua 代码直接访问。
    - 每个外部请求都由一个 Lua 协程处理，协程之间数据隔离。
    - Lua 代码调用 I/O 操作等异步时，会挂起当前协程，而不阻塞工作机进程。
    - I/O 等异步操作完成时，还原相关协程相关协议的上下文，并继续运行。

OpenResty 配置安装

- [OpenResty - OpenResty® Linux 包]
- 可通过修改 /etc/openresty/nginx.conf 更改 127.0.0.1 的显示内容，更改后，使用命令 `service openersty start` or `service openersty restart` 即可
- 我的安装目录在 `usr/local/openresty`

Lua

安装与入门

- 在葡萄牙语里代表美丽的月亮。事实证明她没有糟蹋这个优美的单词，Lua 语言正如它名字所预示的那样成长为一门简洁、优雅且富有乐趣的语言。
- Lua 从一开始就是作为一门方便嵌入(其它应用程序)并可扩展的轻量级脚本语言来设计的，因此她一直遵从着简单、小巧、可移植、快速的原则，官方实现完全采用 ANSI C 编写，能以 C 程序库的形式嵌入到宿主程序中。LuaJIT 2 和标准 Lua 5.1 解释器采用的是著名的 MIT 许可协议。
- 在 Windows 和 Linux 下的安装教程 Lua 环境搭建 · OpenResty最佳实践 (gitbooks.io)
- 教程 Lua 教程 | 菜鸟教程 (runoob.com)

Lua 特性

- **轻量级**：它用标准 C 语言编写，编译后仅仅一百余 K，可以很方便地嵌入别的程序里
- **可扩展**：Lua 提供了非常易于使用地扩展接口和机制，由宿主语言（通常是 C 或 C++）提供这些功能，Lua 可以使用它们，就像是本来就内置地功能一样
- **其他特性**：
    - 支持面向过程编程和函数式编程
    - 自动内存管理，只提供了一种通用类型的表（table），可以实现数组、哈希表、集合、对象

Lua 基本语法

- 命令行运行 `Lua File.lua`
- 注释：
    - 单行注释：`--`
    - 多行注释：`--[[  ]]--`
- 标识符：字母（区分大小写）、下划线、数字
    - 一般约定，以下划线开头连接一串大写字母的名字（比如 _VERSION）被保留用于 Lua 内部全局变量。
- 全局变量：
    - 在默认情况下，变量总是认为是全局的。全局变量不需要声明，给一个变量赋值后即创建了这个全局变量，访问一个没有初始化的全局变量也不会出错，只不过得到的结果是：nil。如果你想删除一个全局变量，只需要将变量赋值为nil。

Lua 数据类型

Lua 是动态类型语言，变量不需要类型定义，只需要为变量赋值

nil

- 对于全局变量和 table，nil 还有一个“删除”作用
- nil 作比较时应该加上双引号 `type(x)=="nil"`
    - `type(X)==nil` 结果为 false 的原因是 type(X) 实质是返回的 **"nil"** 字符串，是一个 string 类型

boolean

- Lua 把 false 和 nil 看作是 false，其他的都为 true，**数字 0 也是 true**

number

- Lua 默认只有一种 number 类型：double 型

string

- 字符串由一对双引号或单引号来表示
- 也可以用 2 个方括号 `[[ ]]` 来表示"一块"字符串。
- 在对一个数字字符串上进行算术操作时，Lua 会尝试将这个数字字符串转成一个数字
- 字符串连接操作：`..`

  求字符串长度：`#`，放在字符串前面

table

- 在 Lua 里，table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。也可以在表里添加一些数据，直接初始化表
- Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是**字符串**

```
a = {}
a["key"] = "value"
key = 10
a[key] = 22
a[key] = a[key] + 11
for k, v in pairs(a) do
    print(k .. " : " .. v)
end
```

- Lua 中，表的默认初始索引一般以 1 开始
- table 不会固定长度大小，有新数据添加时 table 长度会自动增长，没初始的 table 都是 nil

function

```
function factorial1(n)
--something here
end
```

thread

- 在 Lua 里，最主要的线程是协同程序（coroutine）。它跟线程（thread）差不多，拥有自己独立的栈、局部变量和指令指针，可以跟其他协同程序共享全局变量和其他大部分东西。
- 线程跟协程的区别：线程可以同时多个运行，而协程任意时刻只能运行一个，并且处于运行状态的协程只有被挂起（suspend）时才会暂停。

uesrdata

- userdata 是一种用户自定义数据，用于表示一种由应用程序或 C/C++ 语言库所创建的类型，可以将任意 C/C++ 的任意数据类型的数据（通常是 struct 和 指针）存储到 Lua 变量中调用。

DisGraFS

What

- **图文件系统**：逻辑上抛弃树状结构，文件之间用"**关系**"连接。基于语义的局部性：有共同特征的文件相连
- **分布式图文件系统**：底层存储采用**分布式存储**，语义识别采用**分布式计算**，用**图结构**描述文件之间的关系

How

系统架构

DisGraFS分为5个组成部分：索引服务器、分布式存储集群、分布式计算集群、网页端和客户端。

- **索引服务器**：进行分布式存储集群与分布式计算集群的通信、网页端部署的位置，目前也负责构建与维护图数据库（但若有需要，也可将图数据库的部分分离出去）；
- **分布式存储集群**：基于 Juicefs 的分布式储存系统，管理、存储和调度分布式存储系统中的所有文件；
- **分布式计算集群**：基于 Ray 的分布式计算系统，将文本语义识别、图像识别、语音识别以及元数据提取等任务分散给计算集群中的多个计算机；
- **网页端**：直观显示文件所构成的图，并将用户在图上的操作以友好方式展示。
- **客户端**：客户端负责直接接收用户对文件系统的操作，并针对不同的平台对其进行实现。

- 具体技术

> 分布式存储集群：JUICEFS + 阿里云 OSS
>
> 图数据库：NEO4J
>
> 网页前端：使用 neo4j 官方提供的 d3.js 和 pototo.js 框架
>
> 文件语义识别
>
> 分布式计算集群：RAY + VLAB

操作实现

新增文件

1. 用户在网页端启动客户端，将分布式存储集群挂载在本地电脑上；
2. 用户将需要上传的文件直接拖入 JuiceFS 对应的盘，此时分布式存储系统对文件进行切分并通过合理调度将文件分布式地存储在存储集群中；
3. 分布式存储集群发信息给索引服务器，索引服务器将信息转发给分布式计算集群，开始对文件进行内容识别并且打出标签；
4. 打标完成后，分布式计算集群将标签以及文件其他信息一起发送返回给索引服务器，索引服务器根据收到的标签以及文件信息更新图数据库。

文件搜索

1. 用户在网页端提出文件搜索请求，网页端将搜索关键字（可以是标签，也可以是文件名）上传至索引服务器；
2. 索引服务器根据关键字创建搜索语句，在图数据库中搜索，将具有相关标签的所有文件通过图和列表两种方式返回至网页端；
3. 用户可以根据网页端返回的图，直接通过点击获得某一文件的标签与信息，或者获得具有某一标签的所有文件，实现根据文件内容进行搜索以及在图上直接搜索邻顶的目标。

文件获取

1. 用户在关键词搜索返回的文件中找到自己需要的文件，点击打开文件的按键，服务器将消息传给 JuiceFS 分布式存储集群；
2. 分布式存储集群找到需要打开的文件，将其下载到用户本地存储空间并将其打开。

删除文件

1. 用户在客户端提出删除文件的请求，客户端将目标文件名上传至索引服务器；
2. 索引服务器将信息传递给分布式存储集群，分布式存储集群将文件删除；
3. 索引服务器根据文件名删除图数据库中对应的节点，更新图数据库。

关于分布式文件系统的思考

- 其一，DisGraFS 复现存在较大困难，而且据该项目的成员 phr 学长说，其稳定性也存在问题，所以我建议：不把在 DisGraFS 部署分布式就监控作为首要任务。
- 其二，现在已经成熟的分布式文件系统也有许多，我们可以找到一些**开源的、工程量小的（即易于实现和学习其原理）**的分布式文件系统，作为我们开发分布式监控的**第一阶段**的依托实体。毕竟我们的重点在于——**分布式监控**
    - 关于现有分布式文件系统的知乎回答：(81 封私信 / 64 条消息) 开源的分布式文件存储系统大家在使用什么？ - 知乎 (zhihu.com)
    - 常见开源分布式文件系统架构对比 - 简书 (jianshu.com)
- 其三，在第一阶段完成后，即我们已经实现了分布式监控，可以综合考虑**时间、分布式监控实现难度、DisGraFS 的学习难度、DisGraFS 与我们第一阶段开发的分布式监控的兼容性（即需要做那些修改，才能部署到其上）**等多方因素，决定是否开启分布式监控第二阶段的开发，即部署到 DisGraFS 上。当然，我是很希望我们可以做到这一点，希望这个我们科大人自己开发的分布式文件系统能够在一年年的接力中，不断丰富完善。