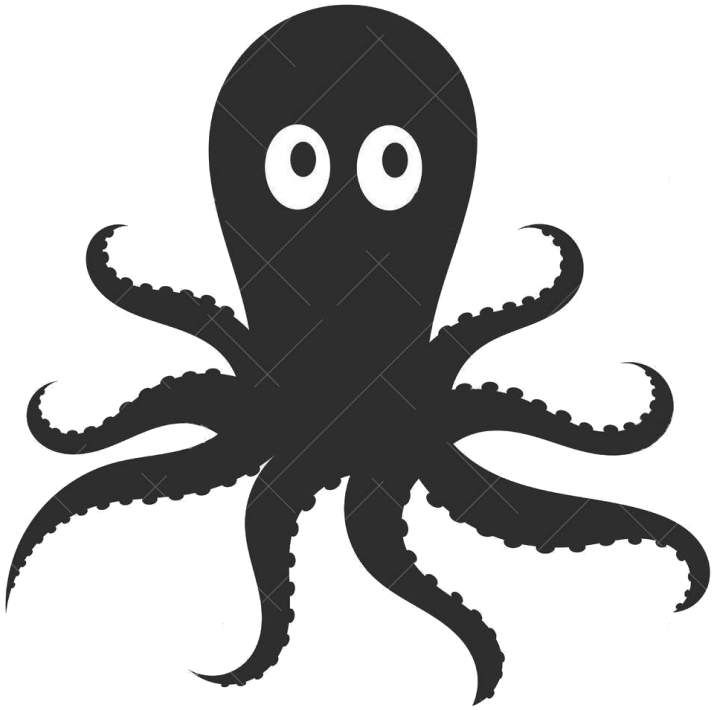
版本	1.0
编写人	夏天
修订日期	2018– 2– 27

Octopus — Leading Web Cralwer



目　录

1 Octopus 特色

2 技术体系

2.1 背景与目标

在 Web2.0、3.0 时代，新闻报道、论坛、即时通信、博客、微博等在线社会网络媒体形式不断涌现,Web 不仅是人们获取信息的重要手段, 同时成为人们参与社会活动、快速发表个性化见解的重要媒介。为方便分析互联网舆论动态，高效的分布式采集器成为必不可少的核心部件。

传统的聚焦爬虫或者面向主题的采集器，多采用多线程方式实现，但是多线程在状态共享方面较为复杂，实现的可靠性不易保证。因此，Octopus 采用最新的 Akka Actor 技术，将 HTTP 采集、任务分发、采集结果的解析、保存等交由 Actor 异步完成，大幅度提高 HTTP 请求的并发处理能力。

采集器的设计目标如下：

(1) 支持分布式，每个客户端可以运行在不同机器上，每台计算机里面的客户以多线程方式运行
(2) 统一管理所有待抓取的 URL，保持爬虫尽量分布在不同机器上
(3) 限制同一个域名下爬虫数量不能过多
(4) 把网页类型分为导航网页和数据网页两种，导航网页必要时可以再次抓取，以获取更新后的内容；数据网页一般长久保存到数据库中，不再更新。
(5) 导航网页和数据网页两类抓取任务分开，降低相互影响，拥有各自的爬虫队列；导航网页的爬行队列在必要时可以清空，以重新开始。
(6) 抓取任务分为两类：1. 当前内存中正在或者等待处理的任务；2. 保存在 NoSQL 数据库中，可以持久化存储的任务两类。任务的分发在内存中完成，同时 Octopus 会自动启动一个注入器，定期把任务由数据库向内存队列注入，从而被主控制器调度和分发。

2.2 定义

(1) Scala：类似于 Java 的多范式静态类型的编程语言，集成了面向对象编程和函数式编程的各种特性。
(2) MongoDB：一个基于分布式文件存储的数据库，由 C++ 语言编写，旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。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 bson 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。

(3) Promise Future 模式

2.3 参考资料

(1) https://www.mongodb.com/
(2) http://scala-lang.org/
(3) http://reactivemongo.org/
(4) http://akka.io/

3 总体设计

为便于分离爬虫和任务控制逻辑的处理，我们把整个爬虫分为可跨主机独立运行的两大部分，即:

- **Master**
  Master 负责控制待抓取的 URL 队列，接收 Fetcher 的抓取结果，并维护 URL 的分派，重复检测等各类逻辑处理，通过 Master 的控制，可以控制站点的速度、更新时间、采集深度。Master 同时提供了一个 HTTP Restful 协议的 API 接口，用于外部程序提交采集任务，查看当前采集状态。
- **Fetcher**
  Fetcher 负责向远程 Master 请求获取一个待抓取的 URL，得到后，则执行抓取动作，并同时进行网页抽取处理，将抽取结果异步保存到数据库中。

3.1 处理流程

(1) 爬虫在启动之前，需要事先配置抓取板块 board，每个 board 里面设置了爬虫抓取的重要信息，如入口地址，抓取深度，接受进一步抓取的 url 正则表达式，文章 url 的正则表达式等。爬虫从入口地址开始抓取内容。
(2) 爬虫每遇到一个待抓取的 url，首先把该 url 保存到正在等待抓取的 url 队列中，避免重复抓取，然后抓取内容，进行解析。
(3) 如果 url 符合文章页面的正则表达式，则进行文章自动抽取

(4) 否则，进一步判断该 url 所隶属的板块有无定义文章正则表达式的规则，若未定义，则需要进一步自动判别是否为文章页面。如果通过正文自动抽取抽取出大量文本，则为文章页面，否则为导航页面。
(5) 为避免每次抓取导航页面都重复判断是否为文章页面，需要把识别出来的导航页面类型缓存。
(6) 抽取网页中的子链接，把符合进一步抓取条件（利用板块 board 对象中的正则表达式）的子链接抽取出来，进行抓取.

4 URL 管理

URL 的基本处理策略：

(1) 所有数据类型的最终页面（文章），默认永远不再更新，因此，在 Redis 中记录每个已经抓取过的数据页面的哈希值，用于避免重复采集。
(2) 导航页面根据刷新周期的设置定期更新，在实现时，记录每个导航页面的入库时间，采用 Redis 的哈希结构存储，该结构中每一个字符的 Key 为导航页面的哈希值，value 为最后一次抓取时间，每次遇到一个导航页面，都与该哈希结果中的对应值进行比对，决定是否入库。
(3) 抓取队列分为两类：数据页面队列和导航页面队列，以保证遇到数据页面能够及时被抓取到。数据页面队列和导航页面队列都直接放在 Redis 中，队列长度会限制在一定长度，默认为 100 万。
(4) 初始的导航页面/数据页面链接，通过 Inject 过程注入，该过程根据需要可以定期重复执行，保证入口地址能够及时更新
(5) 对于导航页面，对抽取出的每一个链接判断其类型，判断规则为：（A）如果满足人工指定的数据页面规则，进入数据队列；（B）SKIP：如果通过主题链接自动抽取得到大量子链接，该类子链接直接标记为数据页面链接，进入数据页面队列。如果没有得到大量子连接，说明原先判别可能有误，则尝试作为数据页面抽取，如果成功，则重新标记为数据页面。其他所有有效子连接作为导航页面进入队列等待抓取
(6) 数据页面：采用正文自动抽取的方式提取内容，以异步处理方式入数据

4.1 URL 的分派策略

URL 的分派策略是爬虫设计的核心，不考虑 Master 和 Fetcher 的交互细节，URL 的分派策略可以通过图1呈现。

如图1所示，每一个 Fetcher 仅与 Master 的某一个桶进行关联，同时，Master 默认

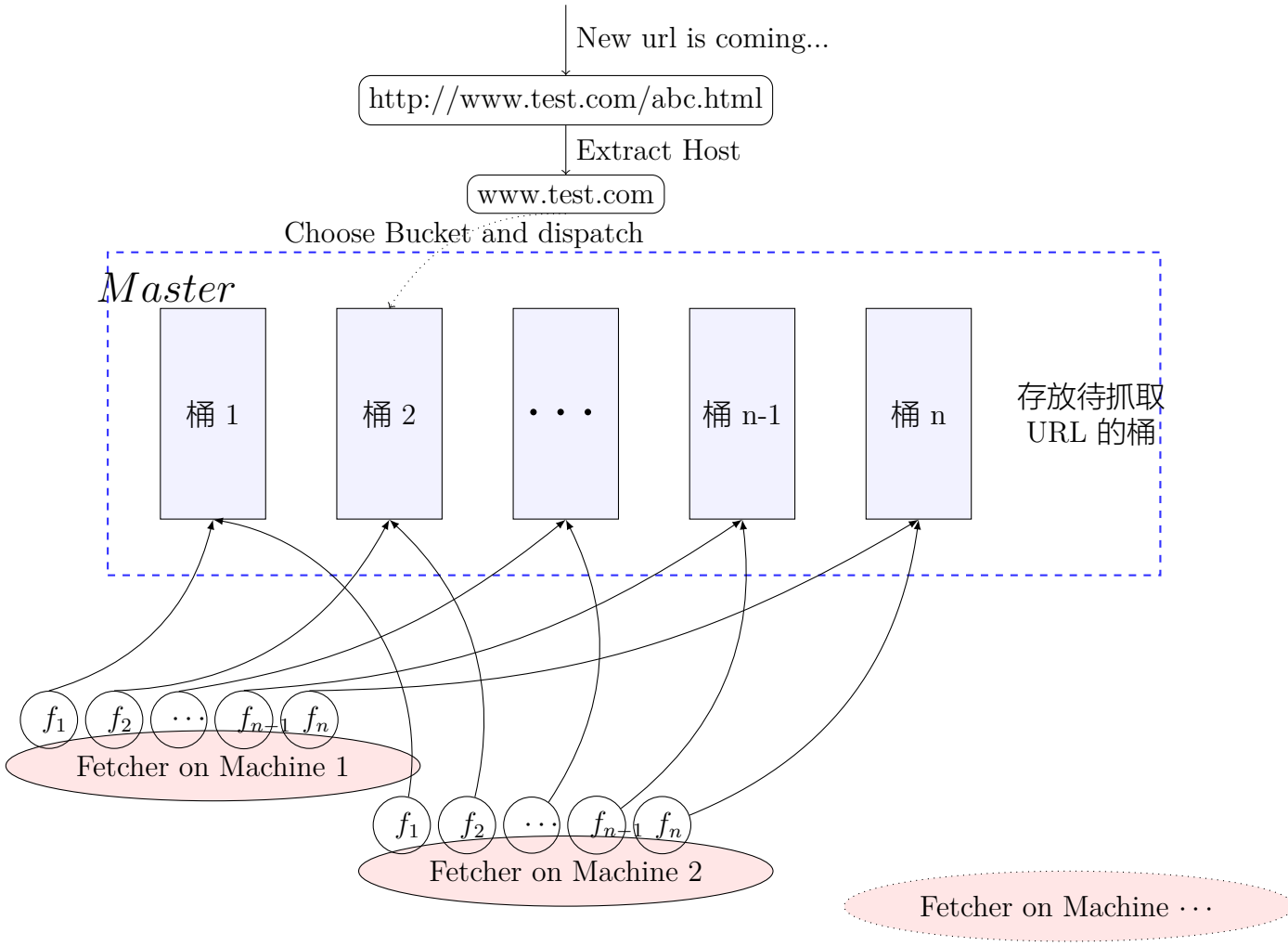


图 1: 基于内存桶的 URL 分派策略

会把来自同一个主机下的所有待抓取链接放入相同的桶中，这就避免了同一主机下的多个 URL 被众多的 Fetcher 同时抓取，而导致采集目标服务器压力过大，甚至被封锁的问题。

在具体实现时，每当一个新的 URL 需要进入桶中排序以等待抓取时，Master 会抽取出该 URL 所在的主机名称，进一步按照主机名称进行散列（取该字符串的 MD5 值），再通过和桶的数量取余数，放入指定的桶中。

即，URL 所在的桶为：

$$BucketIdx(url) = \frac{MD5(Host(url))}{BucketSize}$$

以上默认策略在抓取大量网站时，会遇到如下极端情况：

假设两个主机 a.com 和 b.com，都拥有大量待抓取 URL，同时，这两个主机按照默认策略，所选择的桶为同一个桶，即这两个主机的 URL 在同一个桶中排序，而有的桶则非常空闲。在这一情况下，两个主机的 URL 只能被串形抓取，由于分配策略不完善而无法充分利用系统资源。

为解决上问题，Octopus 采集器支持高级 URL 的桶分配策略，能够进一步发挥系

统性能。基本思想：在必要的时候，能够将部分主机的 URL 从任务较多的桶转向任务较少的桶，处理逻辑尽可能简单，不需要完全精确的刻画当前桶分配现状，保证系统运行的整体效率。基于该思想，桶分配高级策略处理如下：

(1) 在向桶中注入 URL 遇到桶溢出的时候，启动桶分配优化处理，将当前 URL 插入到负荷较小的桶中，并记录 (主机 → 桶) 的当前映射关系。
(2) 桶分派程序维持了所有非默认分派的 (主机 → 桶) 的映射关系，同时记录了该桶中所包含了该主机下待抓取 URL 数量
(3) 根据链接选择要分派的桶时，首先检查 (主机 → 桶) 映射关系表，如果表中存在，则选择表中指向的桶，并更新桶中该主机待抓取 URL 的数量，否则采用默认散列结果指向的桶。
(4) 当 URL 从桶中弹出并交给 Fetcher 抓取时，如果是动态指派的桶，则异步更新桶中该主机待抓取 URL 的数量，当数量降低为 0 时，从 (主机 → 桶) 动态关系映射表中移除该条目。

系统默认为哈希分配方式，如果要启用高级桶分配策略，需要在 “my.conf” 配置文件中，设置如下配置项：

master.bucket.picker = “advanced”

5 系统配置

启用高级桶分配策略：

master.bucket.picker = “advanced”

5.0.1 硬件环境

(1) 8 核以上 CPU
(2) 50G 磁盘空间
(3) 8GB 以上内存
(4) 50G 以上硬盘空间
(5) 2M 以上网络带宽

5.0.2 软件环境

(1) Windows Server、Linux 等能够运行 JVM 和 MongoDB 数据库的操作系统
(2) 数据库：MongoDB 3.0 以上或者 MySQL 5.0 以上
(3) JDK 1.8 以上

5.1 搜索引擎跳转链接的处理

百度等搜索引擎为了统计用户点击数据，对于搜索结果的 URL，输出的是百度自己的一个特殊链接，用户点击该链接时，百度进行后台的统计计数等处理，然后跳转到目标页面，为了获取页面的真实 URL，采集器支持如下参数配置：fetcher.jumpingUrls = ["https://www.baidu.com/link
?url=.+", "https://www.bing.com/link
?url=.+"] 进行此项配置后，系统会将抽取出的符合以上任一模式的链接，获取其跳转后的真实 URL，把真实的文章 URL 链接存入抓取队列，进行后续处理。

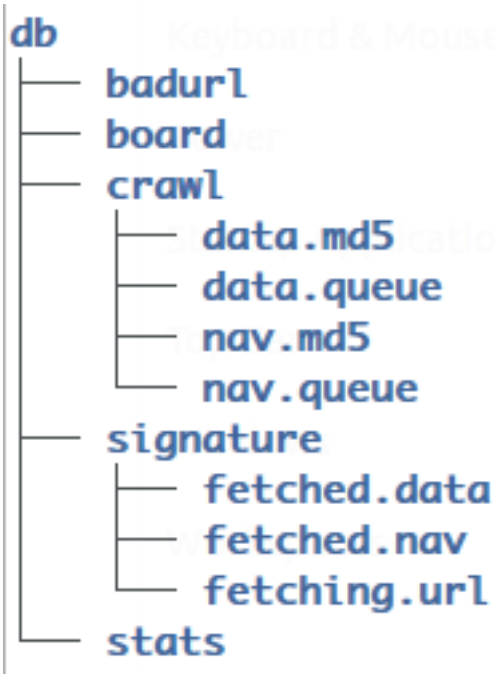
6 核心数据结构

6.1 待抓取 URL 队列

为实现低依赖、方便配置，待抓取 URL 队列由原先保存在 Redis 中的 List，转为底层采用 RocksDB 存储结构、自主设计的队列结构，即 QueueDB，QueueDB 本质上是一个循环队列，只是该队列的数据存储保存在了 RocksDB 之中。

7 抓取数据库目录结构

Octopus 运行后，会把中间状态数据保存在高性能文件数据库中，所有数据库默认位于当前运行目录的 db 子目录中，结构如下图所示：



各个目录的存放数据如下表所示：

目录名称	存放数据
db/badurl	保存了所有的坏链和不存在的主机地址
db/board	保存了所有的采集任务，每个任务代表某个频道的监控源
db/crawl/data.md5	在数据库中排队等待抓取的文章 URL 签名
db/crawl/data.queue	在数据库中排队等待抓取的文章 URL 队列
db/crawl/nav.md5	在数据库中排队等待抓取的列表 URL 签名
db/crawl/nav.queue	在数据库中排队等待抓取的列表 URL 队列
db/signature/fetched.data	已经采集过的文章 URL 的 MD5 签名
db/signature/fetched.nav	已经采集过的列表页 URL 的 MD5 签名
db/signature/fetching.url	分配给某个爬虫客户端正在抓取的 URL 的 MD5 签名，2 分钟自动失效
db/stats	抓取统计数据，如每天、每小时抓取的文章数量