14 | 谋定而动：爬虫项目需求分析与架构设计

2022-11-10 郑建勋 来自北京

《Go进阶 · 分布式爬虫实战》

课程介绍 >

讲述：郑建勋
时长 10:30　大小 9.59M

你好，我是郑建勋。

这节课，我们来看看爬虫系统的功能与架构。

为了更好地完成爬虫项目，我们需要进入真实的场景中，了解项目的价值、用户的需求，这样我们才能够明白项目应该具备哪些功能，明白为了支撑这些用户需求，我们需要设计出怎样的系统架构。所以这节课，我们就从需求分析出发，反推爬虫系统的功能和架构。

需求调研与分析

我们在第六讲，已经讨论了爬虫的许多商业价值。在这个信息快速流动的时代，先人一步掌握准确的信息能获得惊人的回报。

假设我们通过调研发现了一个商业机会，即通过爬虫聚合各个主流媒体的头条新闻，目的是快速获取用户关心的爆炸性新闻，帮助用户快人一步在资本市场上做出反应。借助机器学习等手段感知到某一类词条和事件传播的速度，精准把握它们可能掀动舆论的时间节点，帮助用户做一只资本市场上的“春江鸭”，提前反应。

以此为基础，我们可以将需求分为三个维度，即业务需求、用户需求和功能需求。

- 对于业务需求来说，关注点应该在业务方希望做什么。这里我们希望构建一个以爬虫引擎为基础的推送系统，为资本市场上的用户提供快速的热点事件和事件预警。
- 对于用户需求来说，我们要关注用户对系统的期待。在这里，用户希望能够快速了解自己感兴趣的最新新闻，并有准确的事件预警机制，帮助自己快速决策。

进一步分析业务需求和用户需求，我们就可以理出系统需要具备的流程和关键环节：

- 用户填写或选择自己感兴趣的话题、感兴趣的网站还有消息接受频率；
- 用户接收最新热点事件的推送；
- 用户通过点击获取与该事件关联的事件，并得到相关的事件预测、预警，甚至可能在网站中进行快速的交易；
- 用户可以查看历史记录，可视化呈现某一个事件的来龙去脉，并进行复盘。

要实现这整个产品，我们需要给每一个关键的流程与环节提供详细的功能和交互，完成它的功能需求，形成一份产品需求文档。

产品需求决定了我们系统的具体设计，这中间包括前端的页面设计，用户交互设计，后端的爬虫引擎设计、数据分析设计、数据推送设计等。我们这节课重点关注支撑这一产品最关键的爬虫引擎的开发和设计，爬虫引擎的工作就是爬取指定数据，并完成数据的收集、清洗和存储。

那么产品的需求如何指导我们完成对爬虫引擎的设计呢？我们可以将爬虫引擎的设计分为两个部分：**一部分是引擎需要具备的功能性模块，它是从产品需求拆解而来的；另一部分是非功能性模块（例如对产品可用性、可扩展性的要求），非功能性模块通常是隐含在需求中的。**

功能性模块的设计

首先让我们来看看功能性的设计。

这里我们要先考虑一下数据的输入。爬虫引擎的任务来自于哪里呢？我们可以指定为两种来源，第一种是我们提前规划好的新闻网站与论坛，这被称为种子网站。另一种是用户通过界面和 API 接口动态增加或修改的任务。

因此，我们需要有一个功能模块负责任务的配置与管理，这个模块可以读取配置文件中预先指定的任务列表，并能够存储后面用户动态新增的任务。同时我们还需要有接口可以灵活地对任务进行增删查改，了解任务当前的状态。

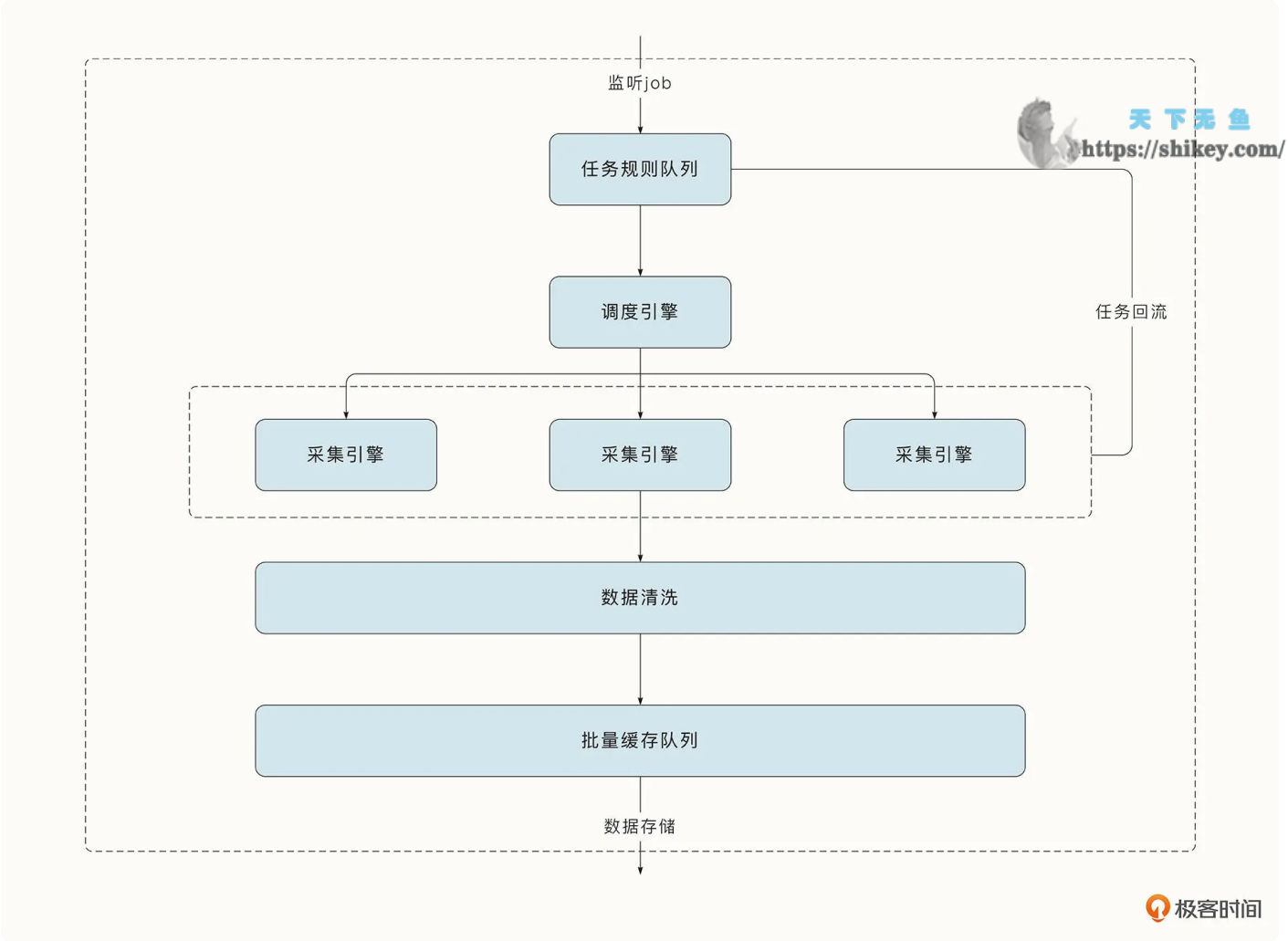
每个任务中除了包含任务要爬取的 URL，还包含了任务相关规则的描述。例如需要抓取页面中的哪一些信息、哪一些网址需要进行爬取、页面爬取的深度、以及请求超时时间等。任务相关的规则可能比较复杂并且有关联，例如 A 网站爬取到 B 网站，B 网站爬取到 C 网站，每一个网站都需要有对应的处理规则。因此我们用一个规则引擎模块处理规则。

有了任务之后，我们需要合理地安排任务的执行，我将这个模块称为调度引擎，调度引擎会将任务均衡地分发到对应的采集模块中，分别进行处理。同时调度引擎还需要接收新的任务并存储到队列中。

采集引擎获取到任务与规则后会进行相应的采集工作，解析出相关的页面信息。不同的页面可能需要不同的访问方式，例如有些页面可以直接访问、有些需要模拟浏览器访问，有些页面需要提前登录才能够访问，这都是通过选择不同的采集引擎完成的。此外，采集引擎还会按照深度优先搜索或者广度优先搜索等算法循环爬取页面，并将任务回流给调度引擎处理。

在采集引擎的基础上，还会有一些辅助的任务管理模块。这些模块包括，限制器、代理器、去重器、随机 UA、任务优先级队列、失败处理器。其中，限制器负责控制任务的采取频率；代理器用于隐藏源 IP，突破服务器的反爬机制；去重器用于避免重复任务；随机 UA 用于生成随机的 User-Agent；任务优先级队列用于为任务分级，高优先级的任务先执行；失败处理器可以处理采取失败后的重试问题。

采集好数据之后，接下来就进入到数据的清洗和存储阶段了。收集到的数据类型各异，也可能有无效或不符合规范的数据，这就需要我们统一进行数据清洗并完成数据的存储工作。同时，为了提高程序的性能，我们还需要一个缓存队列，以便在收集到一批数据后批量将数据写入数据库中。在数据存储方面，我们需要一个叫做存储引擎的功能模块，帮助我们将数据存储到不同类型的数据库或文件当中。Worker 处理的流程图如下所示：



主要的功能性设计就是这些，接下来我们再考虑一下非功能性需求衍生出来的非功能性设计。

非功能性模块的设计

非功能性的设计首先要考虑的是可扩展性。我们希望服务能够随着任务数量的增加而扩展，希望它能够快速增加 Worker 程序的数量，帮助我们应对更多的爬虫任务。这要求 Worker 服务是无状态的，任务可以在任何 Worker 中运行，并且具有相同的行为。

不过还需要注意的是，我们的爬虫任务和其他一些快速返回的 HTTP 请求有很大不同，我们的任务将会在 Worker 中长时间存在，因此，我们会遇到许多新的问题：

- 当我们增加一个新的 Worker，该怎么保证任务可以重新分配，或者确保新的任务一定能分配到新的 Worker 呢？
- 每一个任务的负载是不同的，例如一些任务会比另一些任务消耗更多的 CPU 与内存。那么我们要如何合理分配任务，才能让每一个 Worker 负载均衡呢？

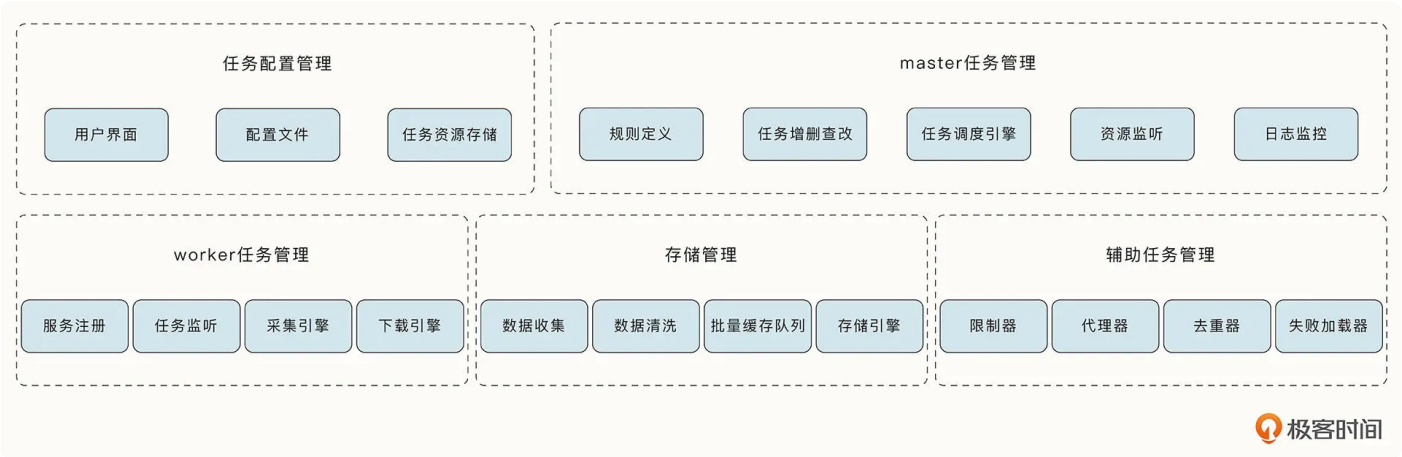
显然，单独的 Worker 无法实现这样的功能，我们需要有另外的 Master 服务作为任务的调度器。这就构成了 **Master-Worker 架构模式**。

除了扩展性，我们还要考虑服务的可用性。我们假设产品对于服务的可用性有比较高的要求，如果少数服务崩溃，不应该影响到服务的运行。这时我们就需要考虑分布式系统的容错问题。由于现在我们有了两种类型的服务，一个是 Master，一个是 Worker。因此我们需要单独考虑它们的容错性。

- 当 Worker 崩溃后，运行在其中的任务将无法运行。因此，我们希望 Master 能够监控到 Worker 的数据变化，并且 Master 能够通过任务的重新分配将崩溃节点的任务分散到其他 Worker 中。当 Worker 重新恢复后也有类似的过程。不过由于 Worker 无状态，不用考虑 Worker 崩溃之后数据的不一致问题。
- 对于 Master 节点，它的主要任务是完成调度工作，本身不需要实现多高的并发量，任务的调度和分配在一个程序中就能搞定。但是为了解决故障容错问题，我们需要有多个 Master 随时待命，这会需要对多个 Master 进行选主，客户端只能与 Master 的 Leader 节点进行交互，并且只有 Master 的 Leader 节点能够调度任务到 Worker。

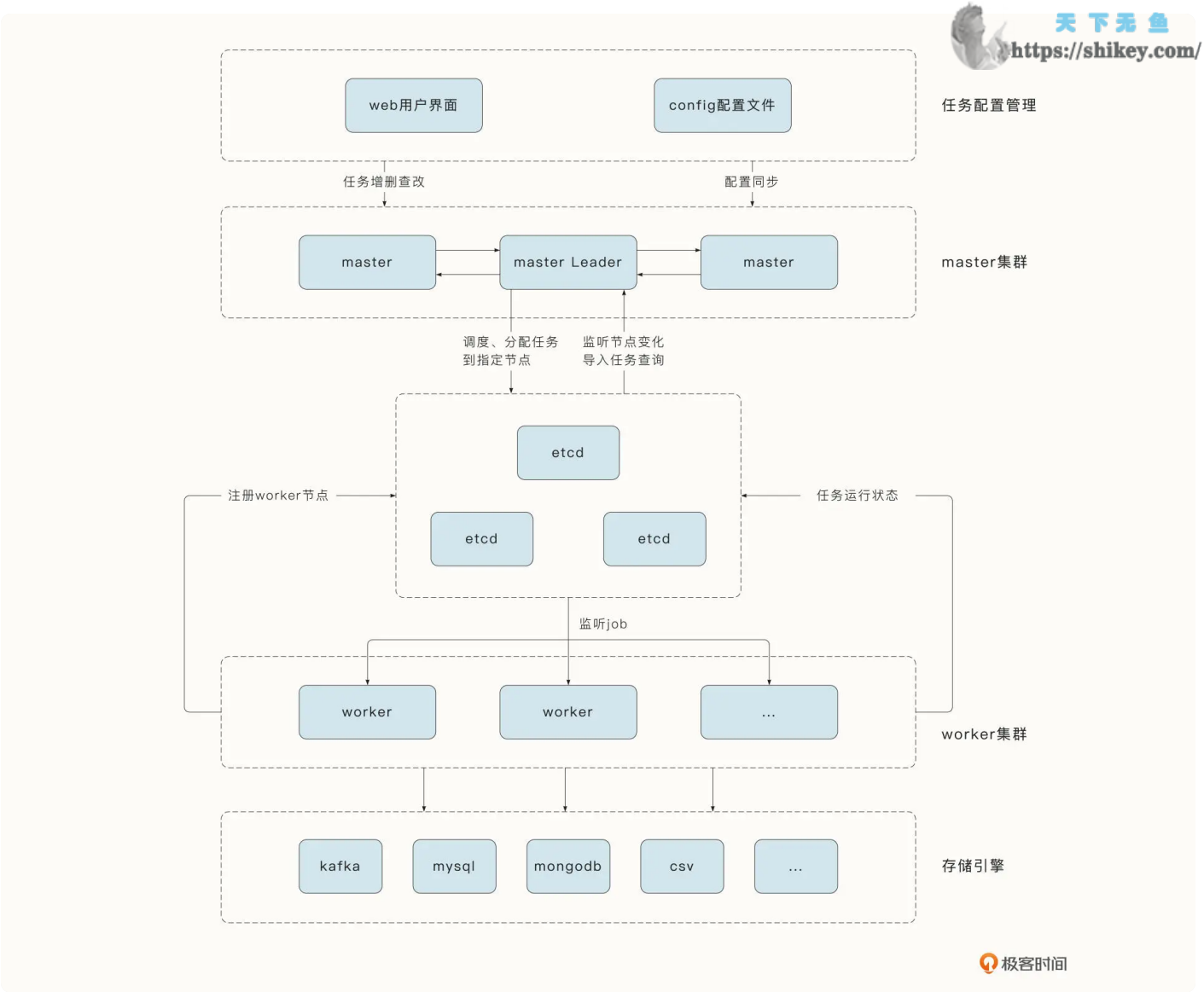
好在，我们在实践中并不需要自己去实现分布式算法。细数一下，**我们需要监听 Worker 节点，完成服务发现、任务的动态分配、还有 Master 节点的选主工作**，这几点都是分布式协调服务可以做到的。因此，我们可以引入一个新的组件 etcd，帮助我们完成分布式系统的协调工作。

基于上面对功能性需求与非功能性需求的分析，我们可以列出系统需要具备的核心功能：



架构设计

接下来，就要进行架构设计了。系统完整的系统架构如下所示：



解释一下。在这里，Master 总览全局，为用户操作提供接口，并作为任务的调度器完成如下工作：

- 提供任务增删查改的 API；
- 实现任务的调度；
- 动态获取和监听 Worker 节点的变化，实现任务动态的负载均衡；
- 借助 etcd 实现选主，完成可用性的保证；
- Master 集群中只会有一个 Leader，其他 Master 接收到的请求会转发到 Leader 中处理。

etcd 集群负责实现 Master 与 Worker 的分布式协调工作：

- 为 Worker 与 Master 实现注册中心的功能；
- 实现事件的监听和通知机制；
- 存储每个 Worker 需要执行的任务，这些任务由 Master 完成分配；
- 提供 Master 选主能力。

Worker 负责监听任务的变化，完成具体任务的采集工作：

- 动态监听 Master 为其分配的任务；
- 注册服务到 etcd 中；
- 完成海量并发任务的爬取、解析、清洗、存储工作。

到这里，一个爬虫项目的核心功能与分布式架构就介绍完毕了。这个架构从大的方向上划分了模块和功能，保证了分布式系统的容错性与扩展性。如果以问题为导向，我们会发现这些流程、组件都是为了应对具体问题而出现的。不过，还有一些比较细节的设计我们这节课并没有提到，例如使用何种框架与协议，使用哪一种高并发模型，程序的扩展性设计，这些问题我们都将在后面的课程中一一解决。

总结

好了，这节课，我们通过对一个真实需求的一步步剖析，梳理出了我们的爬虫系统需要具备的核心功能，这些功能覆盖了功能性的需求与非功能性的需求，这些需求也决定了我们如何设计系统架构。总的来说，我们实现了 Master-Worker 架构，并借助分布式协调服务 etcd 完成了选主、服务注册、节点监听、任务存储等功能。Master-Worker 内部的架构其实仍然有很多值得思考和设计的地方，在接下来的开发篇我们还会进一步讨论。

课后题

学完这节课，我也给你留一道思考题吧。

对 Go 这种静态编译的语言，你知道怎么在不重新编译和运行程序的情况下，动态添加一个新的爬虫任务和新的爬虫规则吗？

欢迎你在留言区和我交流讨论，我们下节课再见！

赞 2　提建议

上一篇　13丨智慧之火：详解分布式容错共识算法

下一篇　15丨众人拾柴：高效团队的Go编码规范

精选留言 (8)

写留言

风铃
2022-11-10 来自北京

咱们实战课程大概什么时候开始了

> 作者回复: 下周开始进入实战

3

风铃
2022-11-10 来自北京

咱们什么时候可以开始了

> 作者回复: 下周开始进入实战

3

会飞的大象 神
2022-11-10 来自上海

将爬虫任务和规则维护到etcd配置中心，进行配置热加载

3

shuff1e
2022-11-10 来自北京

有代码吗？上下github的链接。talk is cheap, show me the code

作者回复: 15讲后进入实战，到时都会有相对应的项目代码给到，保持一些耐心哦

2

拾掇拾掇

2022-11-10 来自浙江

配置中心或者监听配置文件

2

老猫

2022-11-11 来自江苏

1. 任务，通过用户界面添加到数据库里，然后任务调度器读取任务进行就好了。
2. 规则，两种思路，如果可以将规则写成配置文件，如yaml,json之类的，那直接动态添加这个规则。但如果这个规则比较复杂，可以考虑为规则单独起一个服务，将服务配置到任务调度器里

1

逗逼章鱼

2022-11-10 来自北京

对 Go 这种静态编译的语言，你知道怎么在不重新编译和运行程序的情况下，动态添加一个新的爬虫任务和新的爬虫规则。
是把任务写进数据库里读取吗？

作者回复: 这是一种方式，但这需要我们的程序提前提前写好了解析任务的代码，这就是静态的。 还有动态的方式，这种方式可以依靠虚拟机实现，我们后面会实战两种方式

1

Realm

2022-11-10 来自浙江

猜测一下：用协程订阅etcd中的/jobs、/rules，
在etcd中增加任务和爬虫规则时，会通知订阅者，达到动态配置的效果.

1