**分布式爬虫系统系统设计**



1. **系统功能模块**

设计分布式爬虫架构，架构满足以下功能

1. 分布式：爬虫应该能够在多台机器上分布执行。
2. 可伸缩性：爬虫结构应该能够通过增加额外的机器和带宽来提高抓取速度。
3. 性能和有效性：爬虫系统必须有效地使用各种系统资源，例如，处理器、存储空间和网络带宽。
4. 去重能力：分爬虫所抓取的URL数量众多，会出现重复的情况，应当集中的处理分发。

设计的功能模块的框架图如图1-1所示。

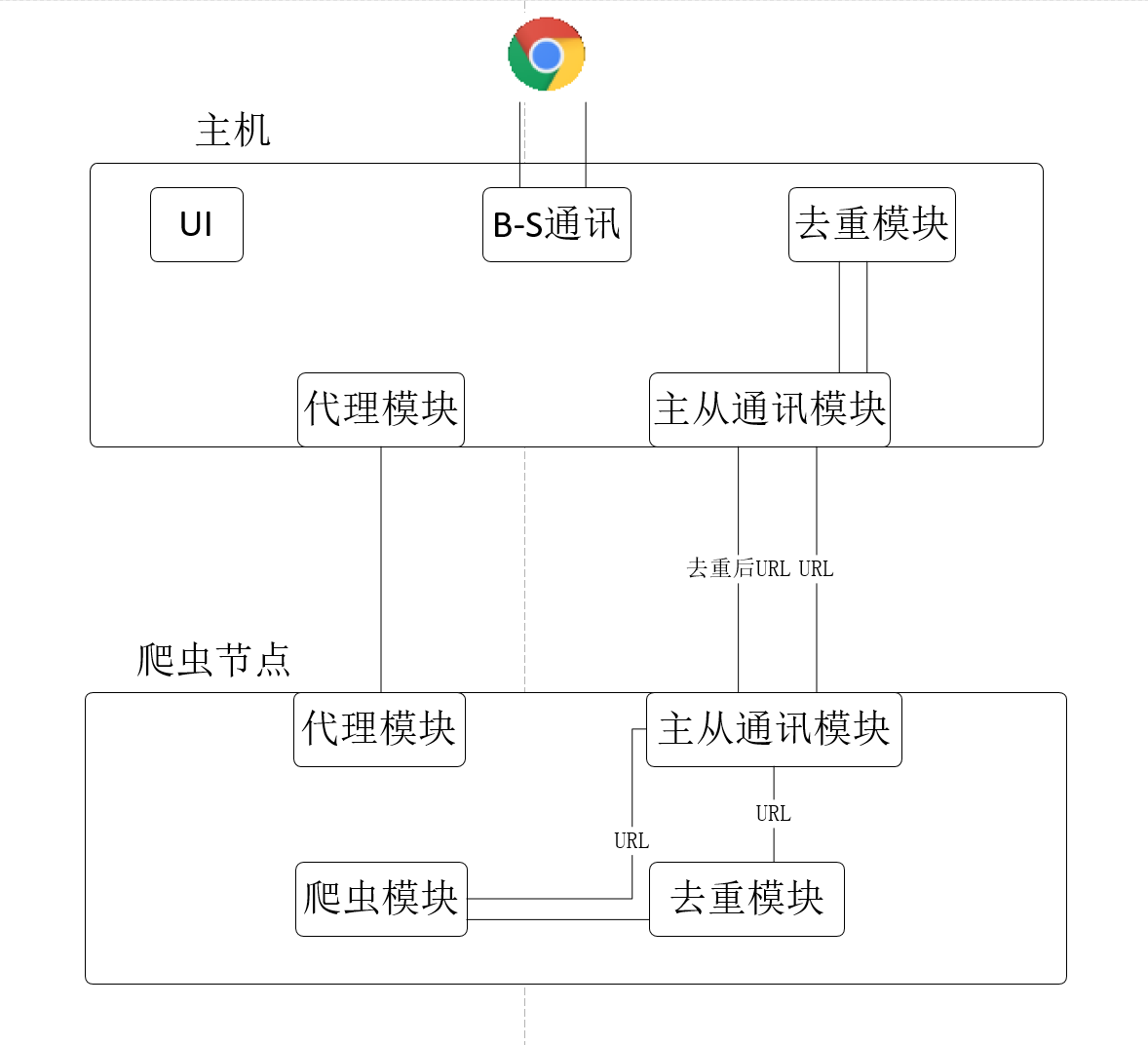


图 1-1 功能模块图

一共有七个模块，分别如下：

1. 去重模块：对采集得URL进行去重。

R-B Tree，全称是Red-Black Tree，又称为“红黑树”，它一种特殊的二叉查找树。红黑树的每个节点上都有存储位表示节点的颜色，可以是红(Red)或黑(Black)。

红黑树的特性:

（1）每个节点或者是黑色，或者是红色。

（2）根节点是黑色。

（3）每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL或NULL)的叶子节点！]

（4）如果一个节点是红色的，则它的子节点必须是黑色的。

（5）从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。

注意：

(01) 特性(3)中的叶子节点，是只为空(NIL或null)的节点。

(02) 特性(5)，确保没有一条路径会比其他路径长出俩倍。因而，红黑树是相对是接近平衡的二叉树。

红黑树示意图如图1-2所示：

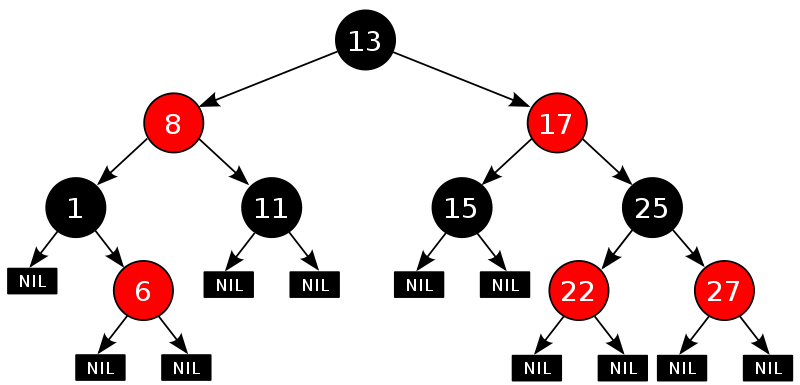


图 1-2 红黑树示意图

红黑树的应用：

红黑树的应用比较广泛，主要是用它来存储有序的数据，它的时间复杂度是O(lgn)，效率非常之高。

例如，Java中的TreeSet和TreeMap，C++ STL中的set、map，以及Linux虚拟内存的管理，都是通过红黑树去实现的。

这里大致介绍下，红黑树和AVL树的差异。AVL树也是特殊的二叉树，它的特性是“任何节点的左右子树的高度之差不超过1”。基本上，用到红黑树的地方都可以用AVL树(自平衡二叉查找树)去替换。但是一般情况下，在执行添加、删除节点时，AVL树比红黑树执行的操作更多一些，效率更低一些；而且红黑树也是相对平衡的二叉树(从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点)。因此，红黑树的效率会高更一点。

R-B Tree时间复杂度

红黑树的时间复杂度为: O(lgn)

下面通过“数学归纳法”对红黑树的时间复杂度进行证明。

定理：一棵含有n个节点的红黑树的高度至多为2log(n+1).

证明：

"一棵含有n个节点的红黑树的高度至多为2log(n+1)" 的逆否命题是 "高度为h的红黑树，它的包含的内节点个数至少为 2^{h/2}-1个"。

我们只需要证明逆否命题，即可证明原命题为真；即只需证明 "高度为h的红黑树，它的包含的内节点个数至少为 2^{h/2}-1个"。

从某个节点x出发（不包括该节点）到达一个叶节点的任意一条路径上，黑色节点的个数称为该节点的黑高度，记为bh(x)。

由红黑树的"特性(4)"可知 bh(x)>=h/2；进而，我们只需证明 "高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"即可。

到这里，我们将需要证明的定理已经由

"一棵含有n个节点的红黑树的高度至多为2log(n+1)"

转变成只需要证明

"高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"。

下面通过"数学归纳法"开始论证高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"。

(1) 当树的高度h=0时，

内节点个数是0，bh(x) 为0，2^bh(x)-1 也为 0。显然，原命题成立。

(2) 当h>0，且树的高度为 h-1 时，它包含的节点个数至少为 2^{bh(x)-1}-1。这个是根据(01)推断出来的！

下面，由树的高度为 h-1 的已知条件推出“树的高度为 h 时，它所包含的节点树为 2^bh(x)-1”。

当树的高度为 h 时，

对于节点x(x为根节点)，其黑高度为bh(x)。

对于节点x的左右子树，它们黑高度为 bh(x) 或者 bh(x)-1。

根据(02)的已知条件，我们已知 "x的左右子树，即高度为 h-1 的节点，它包含的节点至少为 2^{bh(x)-1}-1 个"；

所以，节点x所包含的节点至少为 ( 2^{bh(x)-1}-1 ) + ( 2^{bh(x)-1}-1 ) + 1 = 2^{bh(x)-1}。即节点x所包含的节点至少为 2^{bh(x)-1} 。

因此，原命题成立。

由(1)、(2)得出，"高度为h的红黑树，它的包含的内节点个数至少为 2^bh(x)-1个"。

因此，“一棵含有n个节点的红黑树的高度至多为2log(n+1)”。

1. 代理模块：获取代理IP和端口。

开启线程检测代理堆栈是否为空，若为空则发送请求索要新的代理IP（期间设有信号量阻塞请求）。代理的检验环节交给爬虫分节点以避免中心节点负担过重。

1. 爬虫主-从通讯模块：主机爬虫和节点爬虫进行通讯。

主机负责集中去重，任务调配还有代理的抓取还有分发。从机负责验证代理可用性，爬取信息，采集数据，分布式存储，回传待去重的URL。

1. B-S通讯模块

B-S通讯模块基于websocket的通讯，介绍一下websocket。

WebSocket是HTML5出的协议，也就是说HTTP协议没有变化，但HTTP是不支持持久连接的（长连接，循环连接的不算）。  
 首先HTTP有1.1和1.0之说，也就是所谓的keep-alive，把多个HTTP请求合并为一个，但是Websocket其实是一个新协议，跟HTTP协议基本没有关系，只是为了兼容现有浏览器的握手规范而已，也就是说它是HTTP协议上的一种补充，可以通过图1-3理解他们之间的关系。

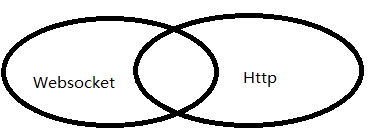


图1-3 websocket和http的关系

Websocket是一个持久化的协议，相对于HTTP这种非持久的协议来说，websocket的主要是为了解决在web上应用长连接进行灵活的通讯应用而产生，但websocket本身只是一个基础协议，对于消息上还不算灵活，毕竟websocket只提供文本和二进制流这种基础数据格式．在实际应用则更偏向于对象消息的处理，而在这基础上更希望集成一系列的消息路由机制来解决消息处理上的问题．为了解决以上问题beetle针对websocket进行了一些高层次的封装，让服务端处理消息变得更简单灵活。

1. 爬虫模块：爬取网页数据。

使用面向对象方法封装PhantomJS的一系列操作，请求的方法中附带有触发器，在请求次数达到上限的时候自动进行IP的切换。使用正则表达式引擎模糊提取信息，存放在数据库中。

1. UI模块

UI模块用基于websocket通信的Echarts的动态数据展示柱状图和折线图，时刻向用户展示爬取数据的信息和爬虫系统的性能。ECharts，一个纯 Javascript 的图表库，可以流畅的运行在 PC 和移动设备上，兼容当前绝大部分浏览器（IE8/9/10/11，Chrome，Firefox，Safari等），底层依赖轻量级的 Canvas 类库 ZRender，提供直观，生动，可交互，可高度个性化定制的数据可视化图表。ECharts 提供了常规的折线图，柱状图，散点图，饼图，K线图，用于统计的盒形图，用于地理数据可视化的地图，热力图，线图，用于关系数据可视化的关系图，treemap，多维数据可视化的平行坐标，还有用于 BI 的漏斗图，仪表盘，并且支持图与图之间的混搭。

1. 数据库模块：存储爬取数据。

数据库采用redis，redis数据库具有存储数据快，多线程支持好，可移植性强等特点。

**2.程序流程图**

整体的程序流程图如图2-1所示。

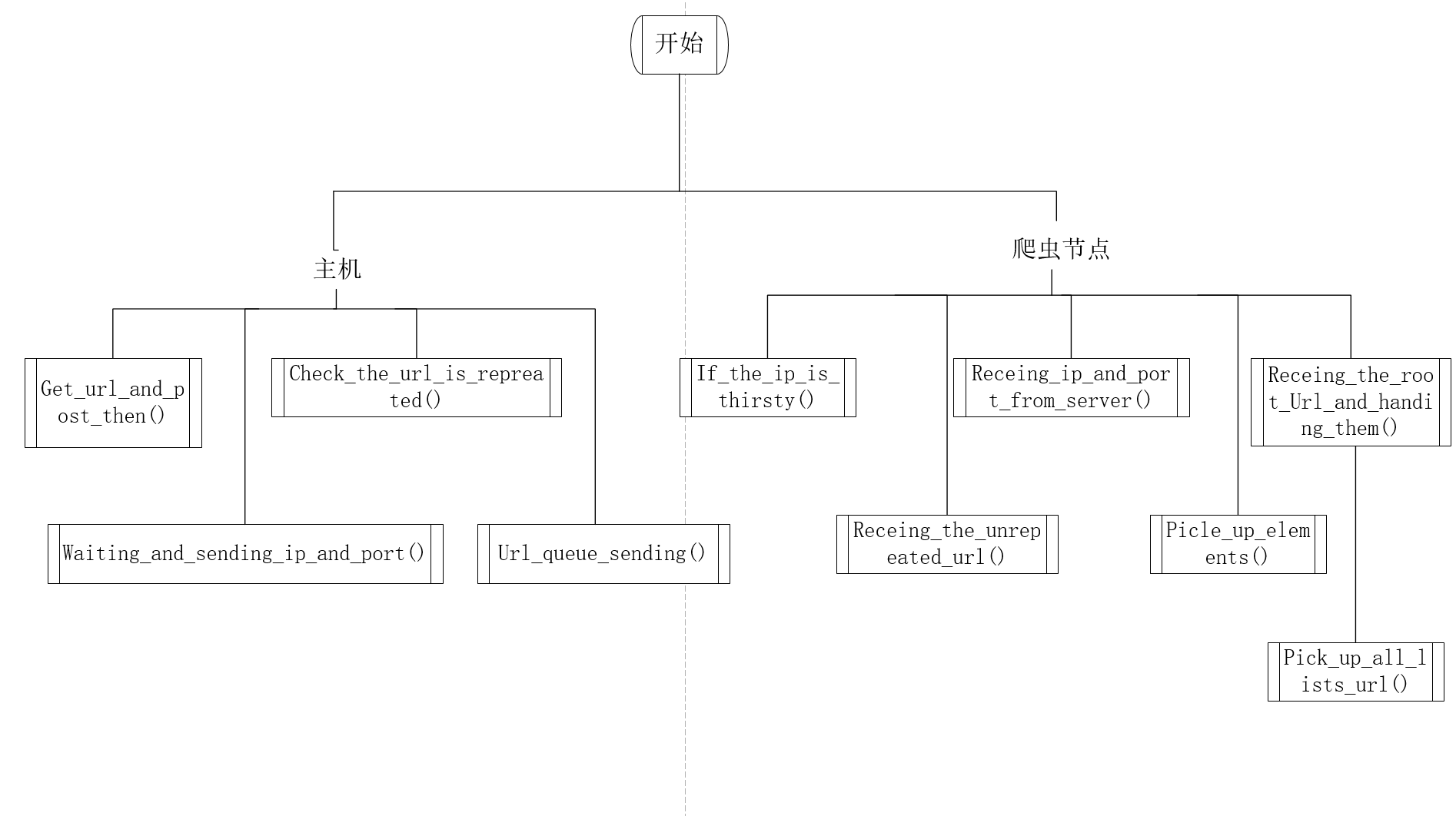


图 2-1 程序流图

整体的流程思路是：用户从浏览器界面输入待提取的网页，如www.taobao.com.浏览器将该网页传给爬虫主机，爬虫主机获得最高域名的网页www.taobao.com后开始爬取出主页面下的各个URL，主机将这一批URL平均分发给各个爬虫分节点爬取，各个爬虫分节点获取URL，自己过滤后再传给主机爬虫，主机爬虫将这些URL记录去重后再分发给爬虫分节点爬取，各个分节点爬取的数据储存在各个分节点的数据库中。

其中爬虫模块的流程图如图2-2所示。



图 2-2 爬虫模块流程图

爬虫模块的时序图如图2-3所示。

数据库

结构化

爬虫节点

分配

初始化URL

去重

1.提交

2.分配

3.爬取

4.存储

5.提交URL

6.去重

7.分配

8.爬取

9.存储

图 2-3 爬虫模块时序图

总体的框架接口有：

（1）用户输入和主机接口：用户输入想要爬取的URL，通过浏览器传给后端的主机

（2）主机和爬虫节点接口：主机将获得的URL分配给分爬虫节点，分爬虫节点将爬取的URL传给主机，主机去重后再将URL分配给各个分爬虫节点。

（3）爬虫节点和代理服务器的接口：获得代理ip和端口。

（4）爬虫节点和数据库的接口：将爬取的数据分布式的存储在数据中。

（5）爬虫节点和网商服务器接口：爬取电商网站得数据。

总体的接口图如图2-4所示。

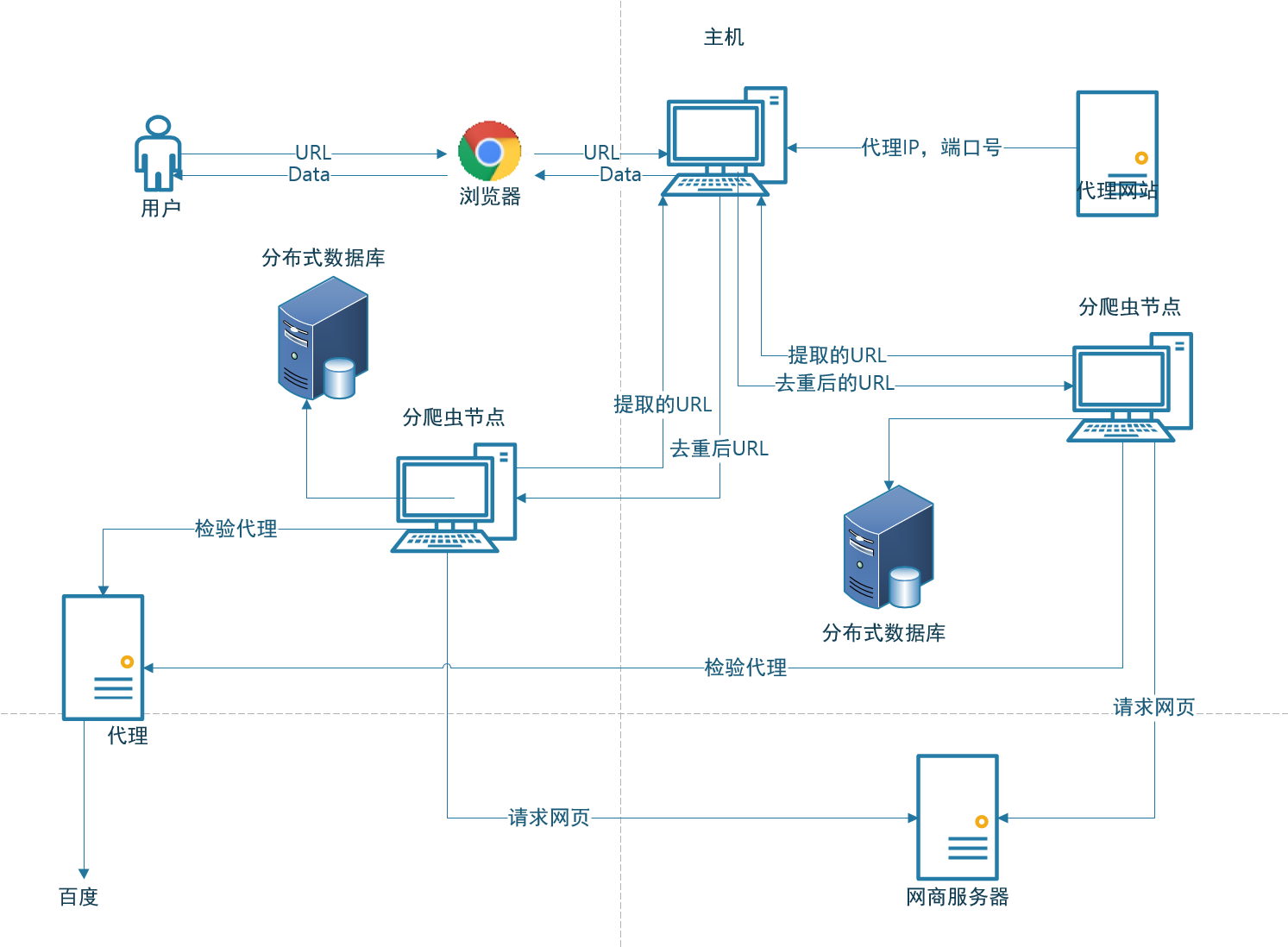


图2-4 总体框架接口图