计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(4):99−109 [doi: 10.15888/j.cnki.csa.008413]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Docker 容器的快速开发网页服务器①

谢兆贤, 曹香美, 王　超

(曲阜师范大学 网络空间安全学院, 曲阜 273100)
通信作者: 曹香美, E-mail: 2539008367@qq.com

摘　要: C/S 模式是早期的开发网页服务器架构, 它使用复杂、成本高并且缺少通用性. B/S 模式通过将系统功能集中到服务器中弥补了使用复杂成本高的缺点, 但是没有对用户端的环境和数据传输率进行深入测试和研究, 使得过多的外部用户访问网页, 给服务器带来压力. 针对传统 C/S 模式和 B/S 模式的局限性, 引入 Docker 容器化开发思路, 分别融合 Nginx 和 Flask 方法, 构建 DoNginx 模式与 DoFlask 模式, 通过修改和建立镜像实现网页服务器的开发. 这两种模式综合了 Docker 的轻量级、Nginx 的低消耗、Flask 的稳定性的优势, 实现了模式的优化. 设计 CPU、完整性、吞吐量性能测试实验, 与传统 B/S 模式进行深度分析对比. 实验表明, DoNginx 模式资源利用率高, DoFlask 模式环境兼容性和可靠性更强, 且均优于传统 B/S 模式, 具有很好的框架设计贡献和实验性能.

关键词: Docker; Flask; Nginx; 网页服务器; 镜像; 微服务框架; 负载均衡

引用格式: 谢兆贤,曹香美,王超.基于 Docker 容器的快速开发网页服务器.计算机系统应用,2022,31(4):99–109. http://www.c-s-a.org.cn/1003-3254/8413.html

Rapid Development of Web Server Based on Docker Container

HSIEH Chao-Hsien, CAO Xiang-Mei, WANG Chao

(School of Cyber Science and Engineering, Qufu Normal University, Qufu 273100, China)

Abstract: The C/S mode is an early architecture for developing Web servers, which is complex, costly, and lacking in generality. The B/S mode makes up for the disadvantages of complex use and high cost by concentrating system functions in the server, but it does not conduct in-depth testing and research on the environment and data transmission rate of the user end, which brings too many external visits to the Web page and puts pressure on the server. Given the limitations of the traditional C/S mode and B/S mode, this study introduces the idea of Docker container development and integrates it with Nginx and Flask methods respectively to construct the DoNginx mode and the DoFlask mode, which develop the Web server by modifying and establishing mirror images. The two modes combine the advantages of Docker's lightweight, Nginx's low consumption, and Flask's stability to realize mode optimization. The CPU, integrity, and throughput performance tests are designed to make a comprehensive comparison with traditional B/S mode. Experiments show that the DoNginx mode has a high resource utilization rate, and the DoFlask mode has stronger environmental compatibility and reliability. Both modes are superior to the traditional B/S mode in the above aspects, boasting great contributions to the framework design and good experimental performance.

Key words: Docker; Flask; Nginx; Web server; image; microservices framework; load balancing

B/S 模式是浏览器/服务器模式, 用户界面完全在浏览器中运行. 大多数通过浏览器和各种网络办公系统访问的网站都是采用 B/S 模式开发的. B/S 模式是“瘦客户端”, 核心功能都集中处理在服务器端, 所以需

① 基金项目: 山东省自然科学基金面上项目 (ZR2020MF048)
收稿时间: 2021-06-22; 修改时间: 2021-07-14, 2021-07-20; 采用时间: 2021-07-27; csa 在线出版时间: 2022-03-22

要较低的硬件配置. 但是几乎所有客户端的处理逻辑都运行在服务器端, 给服务器造成很大的压力[1].

随着云计算的快速发展, 人们开始使用容器在云与云之间快速构建软件的环境. 由于容器具备可移植性的特性, 所以容器的使用量有逐年上升的趋势. 早在 2015 年的时候, 仅有 13% 的使用者, 2018 年增加 29% 的使用者. 从文献 [2] 对 997 名 IT 专业人士的调查, 有将近 49% 的受访者反馈曾经使用过 Docker 容器. 以 Docker 容器的市场价值来看, 2017 年的时候仅为 7.6 亿美元[3], 目前也已经增加到 27 亿美元. 以此显示, 容器在云计算领域的重要性, 逐年增加当中.

Docker 是直接建立在操作系统上的虚拟化技术, 它可以直接使用本地的操作系统. 然而, Docker 容器不只提供轻量化的虚拟环境, 与其它的虚拟化工具相比更具优越性. 所以, 在容器的形式打包和部署应用程序渐渐成为新趋势的情况下, Docker 容器已经可以在云基础架构中得到广泛的部署, 例如: Amazon EC2 Container Service, Google Container Engine, Rackspace 和 Docker Data Center[4].

根据云计算平台的架构, 图 1 实现 HTML 的应用, 使用容器技术部署的 Web 服务器, 分成 3 层结构. 第 1 层是基础设施即是服务 (Infrastructure-as-a-Service, IaaS), 是云计算的最底层, 主要用来提供基础资源, IaaS 中间件的解决方案主要由 Amazon, VMWare 和 IBM[5] 等大公司提供. 第 2 层是平台即是服务 (PaaS). 显示一台物理机中, 可以安装多台不同操作系统的虚拟机, 于每个虚拟机内安装 Docker 容器引擎. 在 Docker 容器中下载特定容器镜像, 使用容器构建网页服务器以及进行数据库的管理. 第 3 层为软件即是服务 (Software-as-a-Service, SaaS). 这是与客户连接的一层, 大多数通过网页浏览器进行接入. 在云计算平台上使用容器部署网页应用程序, 使用 HTML 等技术开发网页应用程序, 最后交付给客户最直接的网页浏览网址或者手机软件 APP.

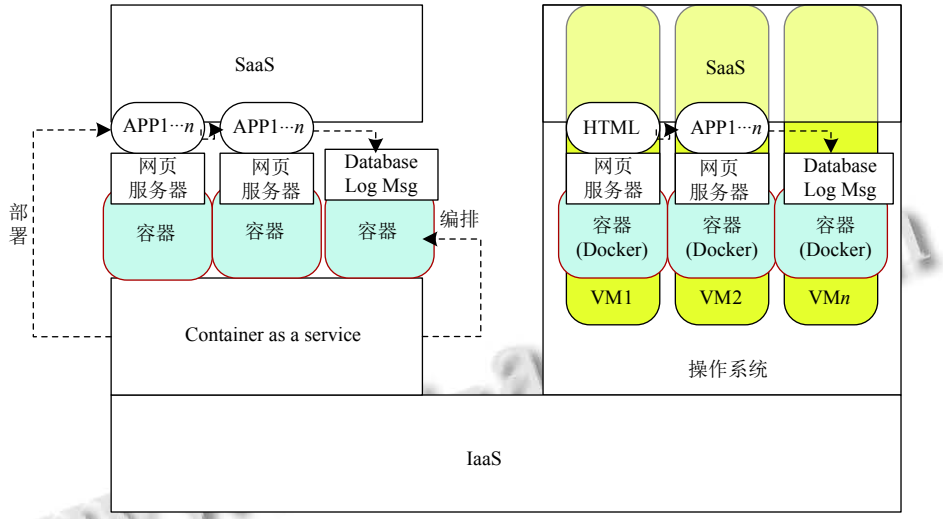

图 1 容器于网页应用的云计算平台

所以, 文献 [6] 初步提出在 Docker 环境下, Java Web 快速搭建平台的设计理念, 为开发网页服务器提供一个新的思路. 然而, 在网页服务器的开发过程, 经常遇见产品的开发环境和客户的使用环境不尽相同. 所以, 每次面对一个新硬件环境的时候, 就需要重新安装一次. 这样, 容易发生人为操作的错误, 导致产品的开发、测试、维护成本增加的问题; 此外, 还会造成重复加载, 造成整体开发速度缓慢的问题. 为了有效地避免发生以上的问题, Docker 容器做为一个轻量级的应用容器引擎, 不只可以解决环境不兼容的问题, 还可以保持操作环境的一致性, 减少重复安装环境的行为, 提升开发效率. 即便目前使用 Docker 容器技术应用在网页服务器的研究很多, 例如: Docker 容器应用于 Web 边缘设备服务的部署[7]、Web 应用自动缩放的功能[8] 和网页服务器的负载均衡[9] 等. 但是, 尚未有对于 Docker 容器于网页服务器开发模式的研究. 所以, 本文将以 Docker 容器为基础, 分别结合 Nginx 技术和 Flask 技术, 建立有效开发网页服务器的模式.

总之, 本文针对传统 B/S 模式部署的网页服务器进行研究, 主要的贡献有以下 3 点. (1) 提出 DoNginx 和 DoFlask 两种新的部署网页服务器模式, 以 Docker 容器为基础, 应用于云平台开发网页服务器的方案. (2) 分析这两种模式与传统的模式的区别, 测试 CPU 占用率、产品交付完整性和数据吞吐量 3 种性能. 得出以 Docker 容器开发的网页服务器, 更具高效性和完整性. (3) 实现网页开发的标准化和流程化, 提升轻量级网页服务器的开发效率.

文章结构上, 引言说明服务器的使用和背景, 分析目前开发网页服务器存在的问题, 介绍虚拟化技术, 其余的说明如下. 在第 1 节, 介绍网页服务器相关的研究工作. 第 2 节讲述 B/S、DoNginx 和 DoFlask 等 3 种开发网页服务器的模式. 第 3 节, 对这 3 种模式的性能进行分析和比较. 最后的结束语, 总结本文以及未来的研究方向.

1 相关工作

1.1 国内外研究近况

文献 [7] 提出一种基于 Web 服务部署的方法, 为了解决 Docker API 使用困难的问题, 研究更加抽象的 HTTP API, 用于部署、更新、终止边缘设备上的容器.

从文献 [8] 可得, 人们为了满足 Web 高效率的需求, 会购买大量服务器或是高阶服务器进行简单的工作, 过度高效的硬件容易造成资源的浪费, 低配置的硬件将无法满足高度流量的环境. 所以, 提出混合云无限缩放 Web 应用的构思, 设计一个基于 Docker 容器的混合云平台, 实现 Web 应用的自动扩展, 进行容器部署和具备自动缩放特性的应用程序.

文献 [10] 中提到, Nginx 是具备开源、轻量级、和高性能特性的服务器, 由于它的高性能和易于扩展的优点, 近年来受到开发人员的欢迎, 并且应用广泛. 通过 Nginx 和 HTTP 的处理流程, 得以动态地调整服务器的权重, 取得较佳的服务器性能.

文献 [11] 对 Nginx 的体系模块、框架结构的内部架构和实现的原理进行研究. 使用 C++语言, 对 Nginx 的源码进行封装和扩展.

文献 [12] 验证 Docker 镜像在 Python 语言环境下运行的可行性, 将一台虚拟机上的 Python 文件打包成镜像, 上传到云平台. 然后, 在另一台虚拟机中拉取镜像, 完成两台虚拟机之间软件的交付. 这种做法, 为本文使用的 Flask 技术奠定基础, 因为 Flask 技术是基于 Python 的 Web 开发技术, 可以快速建立可交付部署的网页平台.

文献 [13] 说明 FTP 服务器上传和下载文件的功能, 同时设计与实现 FTP 的远程管理系统. 最终, 通过 FTP 远程管理系统实现 FTP 服务器, 缩短传输数据和提交数据的时间. 它不仅使普通客户和管理员不再记录复杂的 FTP 命令, 而且大大减少管理员的工作量, 提高工作效率.

1.2 符号定义

本文中涉及到的符号与意义, 见表 1.

表 1 符号表

符号	定义
T_{pull}	拉取镜像所需总时间
T_{run}	启动镜像所需时间
T_{Nginx}	Nginx运行时间
T_{web}	网页加载时间
T_{file}	文件编译加载总时间
W_i	单个文件编译加载时间
F_i	单个文件复制到Nginx容器时间
P_i	拉取单个镜像时间
T_{begin}	实验开始时间
T_{return}	实验结束时间
T_{Flask}	Flask运行时间
T_{build}	镜像创建时间
T_{push}	上传镜像到Docker Hub时间
T_{upload}	上传文件时间
T_{sql}	数据管理时间

2 系统架构和模式

首先介绍 FTP 工具在物理机和虚拟机之间传输信息的流程, 然后介绍 Docker 容器的框架. 最后, 分别提出 3 种模式的设计与实现.

2.1 系统架构

2.1.1 FTP 传输原理

FTP 协议是一个在计算机网络上, 能够对用户端和服务器之间进行文件传输的应用层协议. 同时, FTP 服务器进程可以为多个用户端的进程提供服务. 当 FTP 用户端连接 FTP 服务器, 遵循 FTP 的协议与服务器传送文件, 完成上传下载文件任务[14].

通过 FTP 工具与虚拟平台之间进行通信. 在虚拟的环境中, 将文件复制到 Docker 容器下, 实现 Nginx 与 HTML 和 Flask 与 HTML 的交互通讯, 使 Nginx 和 Flask 可以执行 HTML、JavaScript 和 CSS 文件. 此时,

在虚拟机中的 Docker, 便可以拉取 Nginx 镜像与创建 Flask 镜像, 分别将 Nginx 和 Flask 下的镜像配置文件进行部署与执行. FTP 工具与 Docker 的关系, 如图 2 所示.

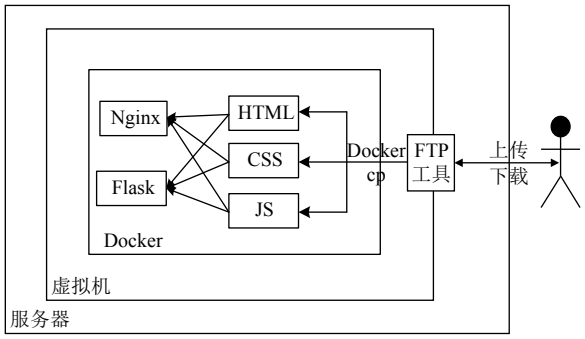

图 2 FTP 传输文件关系

2.1.2 Docker 架构

图 3 的 Docker 网页服务器架构, 显示 Nginx 技术和 Flask 技术的关系, 组成说明如下:

(1) Docker client: Docker 的用户端, 是操作命令的终端. 可以在主机直接执行 Docker 命令, 在用户端同时与多个 Docker 的守护程序通信.

(2) Docker daemon: 守护 Docker 的进程, 监听 Docker API 的请求与管理 Docker 的对象, 例如: 镜像、容器、网络、和卷. 守护程序还可以与其它的守护程序通信, 进行管理 Docker 的服务.

(3) Docker image: Docker 的镜像, 是一个只读文件, 用来创建运行容器. 一个镜像可以创建很多个容器, 以 Dockerfile 文件指令集创建 Docker 镜像. 镜像包含 Ubuntu、Nginx、MySQL 和 PHP 等官方镜像, 也包括自创建镜像文件.

(4) Docker container: Docker 容器, 负责应用程序的运行. 容器独立运行一个或一组应用或服务, 每个容器之间各自独立, 容器的开启、停止、和删除, 与其它容器无关. 启动镜像后生成的容器, 会自动生成容器 ID 和名称, 也可以自行命名容器的名称.

(5) Docker registry: Docker 的仓库, 集中存放镜像文件的场所. 仓库分为公开仓库 (public) 和私有仓库 (private) 两种形式, 仓库内存放的每个镜像文件, 都有不同的标签. 在 Docker Hub 官网上创建的 Docker 账户, 容许新创建的镜像到官网, 实现镜像再利用.

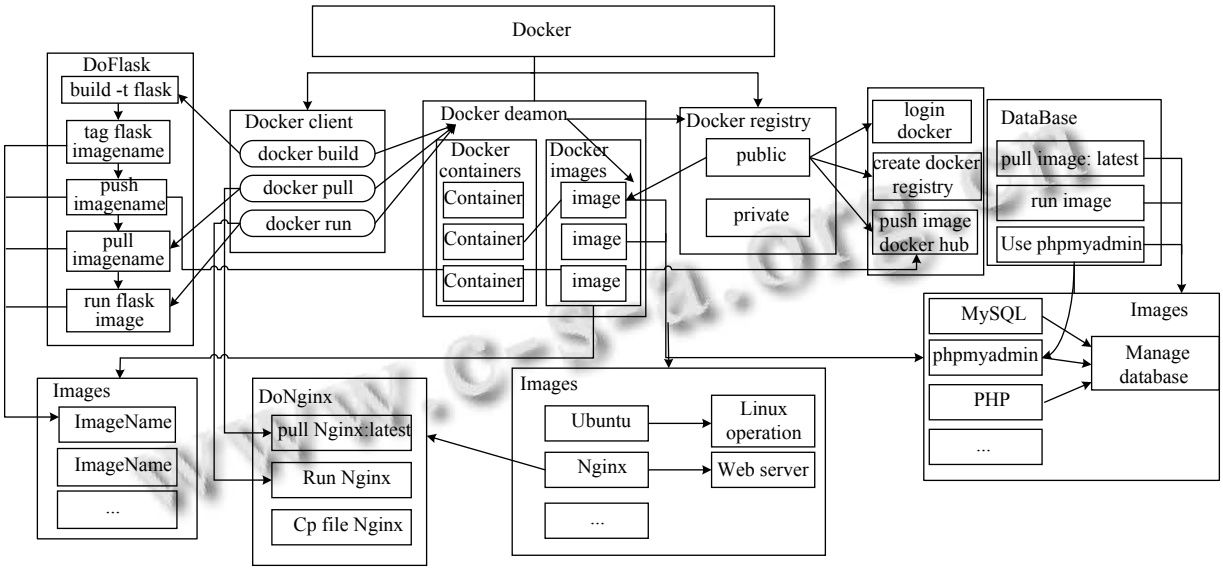

图 3 Docker 架构图

2.2 系统模式

本节对 DoNginx 模式和 DoFlask 模式分别设计运行序列图, 以及运行时间优化算法, 与传统 B/S 模式作对比, 对网页服务器进行设计与实现.

2.2.1 传统 B/S 模式

B/S 模式即浏览器 (browser) 和服务器 (server) 架构模式, 是对 C/S 模式的改进架构模式. 用户端界面是通过浏览器来实现, 极少事务逻辑在前端实现, 主要事务逻辑在服务器端实现. 这种模式将实现系统功能的主要工作集中到服务器上, 具有无需安装用户端、易于在 Web 上发布信息和易于扩展等优点[15]. 同时, 简化了系统的开发、维护和使用.

图 4 显示 B/S 模式构建网页服务器的运行序列关系, 包含用户端界面、控制机制、服务器和数据库. 用户端界面即浏览器, 负责显示结果, 是用户操作系统的接口. 控制机制即中间件, 运行在浏览器和服务器之间, 负责传递信息. 服务器负责与数据库进行连接, 提供数据服务, 通过数据库返回结果.

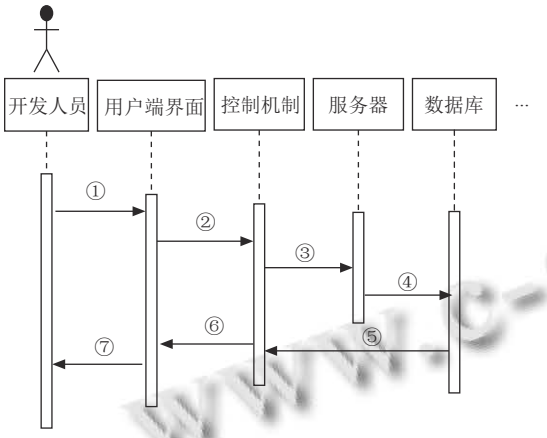

图 4 B/S 运行序列图

序列图显示开发过程, 有以下 3 个步骤.

首先, 开发人员通过过程①创建命令, 生成静态网页, 在用户端界面显示. 用户通过用户端界面通过过程②向服务器端发出请求, 由控制机制接收, 再通过过程③传送给服务器.

然后, 服务器接收请求命令, 通过过程④与数据库进行连接, 对数据进行存取等操作. 数据库使用过程⑤将结果反馈到控制机制, 控制机制将通过过程⑥结果继续反馈给用户端界面.

最后, 用户端界面将返回来的结果进行处理, 可以将系统的逻辑功能更好地展示出来, 再通过过程⑦显示界面.

通过式 (1) 计算编译运行代码文件的时间, 式 (2) 计算构建网页并运行的全部时间.

$$T_{\text{file}} = \sum_{i=1}^{n} W_i \tag{1}$$

$$T = T_{\text{return}} - T_{\text{begin}} = T_{\text{run}} + T_{\text{file}} \tag{2}$$

B/S 模式构建网页服务器的时间函数 BSOpTime (n, T), 算法如算法 1.

算法 1. BSOpTime

输入: n.
输出: T.

T_{begin}=clock();
for i=1 to n
    T_{file}+=W_i;
Send Web to user;
T_{return}=clock();
T=T_{return}−T_{begin};
return T;
end

2.2.2 DoNginx 模式

Nginx 是一款高性能和轻量级的 Web 服务器. 它依据全新的服务器开发模型, 采用事件驱动架构处理请求, 不只可以隐藏自己的 IP 地址, 性能优异且占据内存小, 整体运行效率远超过传统的 Apache 和 Tomcat[10]. 在虚拟平台, 使用 Docker 和 Nginx 技术开发 Web 服务器, 称之 DoNginx 模式.

图 5 显示 DoNginx 的运行序列关系, 包含客户所在的物理机、虚拟平台、Docker Hub、Nginx 容器、FTP 工具和管理平台等 6 个会话. 开发人员需要先创建开发项目代码, FTP 工具负责将开发过程中的代码文件传到虚拟平台, 管理平台负责执行代码文件. 序列图显示的开发过程, 依序有以下 3 个步骤.

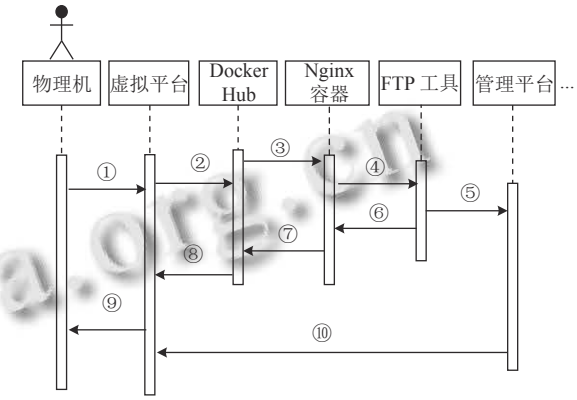

图 5 DoNginx 运行序列图

首先, 开发人员通过过程①启动虚拟平台, 然后通过流程②向 Docker Hub 请求拉取 Nginx 镜像, 使用⑧返回拉取结果, 若拉取成功会返回成功结果, 如果失败返回失败信息. 接下来使用过程③启动镜像生成容器并命名, 若启动成功执行过程⑦返回容器 ID, 若失败返回错误信息.

然后, 使用 IP 地址连接 FTP 工具, 进行过程④, FTP 工具对密钥进行检测. 连接成功后, 选择要执行的代码文件上传到虚拟机, 记作 VM. 然后查看上传的结果, 若上传成功, 则在虚拟机中查看文件信息; 若失败,

则返回错误的信息. 过程⑥会从虚拟机中复制开发人员上传的 n 个文件到 Nginx 容器内部指定的文件, 形成代码文件集 $F=\{F_1, F_2, \cdots, F_n\}$.

最后, 过程⑤使用管理平台设置虚拟开发环境, 执行代码文件集 F. 打开网页浏览器访问 Nginx 容器内部网页配置代码文件 F, 成功出现需要开发的网页. 使用过程⑩返回给虚拟平台反馈信息, 然后物理机中通过过程⑨访问构建的网页.

通过式 (3) 得到上传文件到虚拟机的时间以及将所有文件复制到 Nginx 镜像的全部时间. 同时, 式 (4) 可以得出 Nginx 镜像运行的时间与构建网页时间的和. 加入拉取镜像的时间后, 可以取得全部的计算时间, 如式 (5).

$$T_{\text{file}} = T_{\text{upload}} + \sum_{i=1}^{n} F_i \tag{3}$$

$$T_{\text{run}} = T_{\text{Nginx}} + T_{\text{web}} \tag{4}$$

$$T = T_{\text{return}} - T_{\text{begin}} = T_{\text{pull}} + T_{\text{run}} + T_{\text{file}} \tag{5}$$

DoNginx 模式构建网页服务器的时间函数 DoNginx-OpTime(n, T), 算法如算法 2.

算法 2. DoNginxOpTime

输入: n.
输出: T.

```
T_begin = clock();
Send file.html + file.css + file.js to VM;
T_file = T_upload;
Send file.html + file.css + file.js to Docker.Nginx;
for i = 1 to n
    T_file += F_i;
T_run = run Nginx.image + Web;
Send Web to user;
T_return = clock();
T_1 = T_return−T_begin;
if T_1 == T_pull + T_run + T_file;
    return T = T_1;
end
```

2.2.3 DoFlask 模式

此模式是结合 Docker 和 Flask 技术, 开发网页服务器. 目前采用 Python 开发网页项目的主流框架, 包括 Django 和 Flask. Django 框架是广泛而全面; Flask 框架偏向轻量级、简洁和灵活, 主要面向一些功能简单、需求简单和项目周期比较短的轻应用.

开发人员在 Docker 内, 创建 Flask 镜像, 如图 6 的 DoFlask 运行序列图, 呈现开发网页的序列流程关系. 它包含服务器、用户端、Docker Hub、虚拟平台以及管理平台. 在虚拟平台中, 开发人员需要先创建开发项目代码, 管理平台负责执行代码文件. 两者不同的地方是, Nginx 需要虚拟平台从 Docker Hub 拉取镜像, Flask 则需要服务器的虚拟平台自行创建镜像, 然后上传到 Docker Hub, 再使用用户端虚拟平台拉取镜像. 如图 6 的服务器做为开发端服务器, 负责创建镜像, 用户端的虚拟平台负责测试结果.

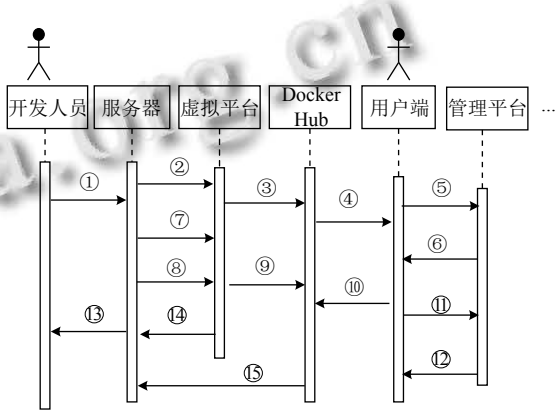

图 6 DoFlask 运行序列图

依照序列图显示, 整体的开发过程, 有以下 4 个步骤.

首先, 开发人员通过过程①创建镜像文件, 创建 Python 文件、requirements 文件和 Dockerfile 文件. requirements 文件用于记录所有依赖包及其精确的版本号, 以便新环境的部署. 然后, 导出打包项目, 依赖不同环境的依赖包和环境包, 以便后续迁移项目的使用. 在通过 Dockerfile 文件, 便可以自动创建镜像或自行定义创建. 然后, 启动开发端服务器, 在虚拟平台中通过②和⑦安装 Python 环境和 Flask 软件包. 若是安装成功, 则返回成功信息; 若失败, 则返回错误信息.

其次, 使用过程⑧创建 Web 包镜像, 即把过程①创建的文件打包, 使用如下命令.

$ sudo docker build <DockerfilePath> -t <ImageName> [:TAG]

当使用⑭检查创建结果, 若创建成功, 则返回成功信息. 使用过程③登录 Docker Hub 账户, 输入用户名密码, 进行检测. 通过过程⑨将镜像文件上传到 Docker Hub 中的仓库, 使用过程⑮返回上传结果, 若上传成功, 则在 Docker Hub 仓库中可以找到此镜像.

第三, 在用户端中使用过程④拉取上传到 Docker

Hub 仓库的镜像, 并且使用过程⑩返回拉取结果. 通过过程⑤, 启动拉取成功的镜像; 使用过程⑥, 返回启动的容器 ID.

最后, 在管理平台中, 通过过程⑪访问启动的 Flask 容器, 查看内部的 Web 配置文件. 使用 IP 地址访问浏览器, 从过程⑫返回访问结果.

式 (6) 得出上传镜像和拉取镜像的全部时间, 从式 (7) 可以计算全部的时间. 然而, 式 (8) 是运行 Flask 镜像时间与构建网页时间的和.

$$T_{\text{file}} = T_{\text{push}} + T_{\text{pull}} \tag{6}$$

$$T = T_{\text{return}} - T_{\text{begin}} = T_{\text{run}} + T_{\text{build}} + T_{\text{file}} \tag{7}$$

$$T_{\text{run}} = T_{\text{Flask}} + T_{\text{web}} \tag{8}$$

DoFlask 模式构建网页服务器的时间函数 DoFlaskOpTime(n, T), 算法如算法 3.

算法 3. DoFlaskOpTime

输入: n.
输出: T.

T_{begin} = clock();
Server:
  Build image in Docker.Flask;
  T_{push} = send image to Docker Hub;
User:
  T_{pull} = pull image in Docker Hub;
  T_{run} = run Flask.image + Flask.Web;
  Send Web to user;
  T_{return} = clock();
  $T_1 = T_{\text{return}} - T_{\text{begin}}$;
  if $T_1 == T_{\text{build}} + T_{\text{run}} + T_{\text{file}}$;
    return $T = T_1$;
  end

2.3 数据库管理

图 7 的数据库管理运行序列关系, 包含虚拟平台、管理平台、Docker Hub 和 Docker 容器. 在虚拟平台的阶段, 使用 Docker 和 phpmyadmin 技术管理数据库. 开发人员需要先拉取镜像, FTP 工具负责将开发过程中的代码文件传到虚拟平台, 管理平台则负责设置虚拟开发环境. 总之, 图 7 的序列图显示开发过程, 总结以下 3 个步骤.

首先, 开发人员通过过程①启动虚拟平台, 然后通过流程②向 Docker Hub 请求分别拉取 MySQL、PHP、phpmyadmin 镜像. 使用过程⑥返回拉取的结果, 若是拉取成功则会返回成功结果; 反之, 则返回失败信息. 接下来使用过程③run 命令启动 3 种镜像, 分别生成各自的容器并命名. 若是启动成功, 则执行过程⑤, 返回容器 ID; 若失败, 则返回错误信息.

然后, 使用 IP 地址访问 phpmyadmin 浏览器网页. 进行过程④, 管理平台对输入的用户名密码进行检测. 若是检测成功, 则登录到数据库. 使用网页版数据库管理工具对数据库进行操作, 并使用过程⑦, 返回结果; 若失败, 则返回错误信息.

最后, 通过 SQL 语句对数据库进行操作, 使用过程⑧, 反馈信息给虚拟平台, 让物理机可以直接访问数据库网页.

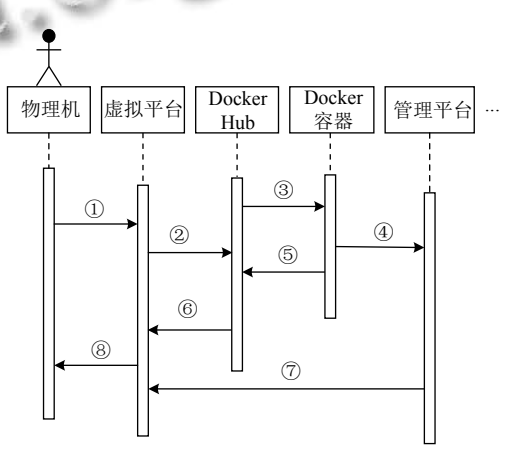

图 7 数据库管理运行序列图

式 (9) 计算拉取全部镜像的时间, 式 (10) 计算启动全部镜像与构建网页时间的和. 式 (9) 加式 (10) 加编译 SQL 语句时间为总时间, 如式 (11).

$$T_{\text{pull}} = \sum_{i=1}^{n} P_i \tag{9}$$

$$T_{\text{run}} = \sum_{i=1}^{n} P_i + T_{\text{web}} \tag{10}$$

$$T = T_{\text{return}} - T_{\text{begin}} = T_{\text{pull}} + T_{\text{run}} + T_{\text{sql}} \tag{11}$$

连接静态网页服务器和数据库的时间函数 DBMOpTime(n, T), 算法如算法 4.

算法 4. DBMOPTime

输入: n.
输出: T.

T_{begin} = clock();
for i = 1 to n
  T_{pull} += P_i;

```
    T_run += P_i;
    T_run = T_run + T_web;
    Send Web to user;
    T_return = clock();
    T_1 = T_return − T_begin;
    if T_1 == T_pull + T_run + T_sql
        return T = T_1;
end
```

3 实验结果与分析

本节通过搭建 Windows 和虚拟机环境的实验场景, 设计了测试 CPU、完整性、吞吐量的实验. 实验的配置环境的物理机操作系统为 Windows 10, 64-bit X86 系统. 物理机的处理器为 Intel(R) Core(TM)i5-8265U CPU 1.60 GHz, 内存为 8 192 MB RAM, 硬盘为高速固态硬盘, 512 GB. 实验用的虚拟机操作系统为 Ubuntu 16.04 LTS, 包括 1 个 CPU 和 20 GB 内存. 内设的实验软件环境, 如表 2.

3.1 实验结果

通过传统 B/S、DoNginnx 和 DoFlask 三种模式, 对开发 Web 应用程序的过程进行实验, 以 Google 的 cadvisor 技术监控容器. 检测的过程, 针对 CPU 的占用情况、产品交付的完整性和服务器的吞吐量, 得到下面的结果.

3.1.1 CPU 测试

图 8 显示 CPU 测试的结果, 观察各种方法在 CPU 的占用率. 依序图 8(a) 为传统 B/S 模式的 CPU 占用率, DoNginx 模式 CPU 占用率为图 8(b), 以及图 8(c) 的 DoFlask 模式 CPU 占用率. 图上显示, 传统方法的 CPU 占用率逐渐上升后趋于平稳. 其次, DoNginx 模式的 CPU 虽然起伏不定, 但是占用率为最低. 最后, DoFlask 模式在实验初期消耗大量 CPU, 后期逐渐下降并趋于平稳. 这几个实验横轴均为 $4T$, 每个 T 为 30 s. 依照式 (12) 得出图 8 的平均值, 依序分别为 56%, 24%, 和 43%. 由此可知, DoNginx 模式占用的 CPU 最低, 而 B/S 模式的占用率最高, 消耗的资源最多.

$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{12}$$

表 2 实验软件及版本说明

软件	版本	存储驱动器类型
Webstorm	2017.1.4	NA
VMware-workstation	12.5	vStorage VMFS
Docker	20.10	OverlayFS
Nginx	1.8.1	OverlayFS
Flask	1.0.1	OverlayFS

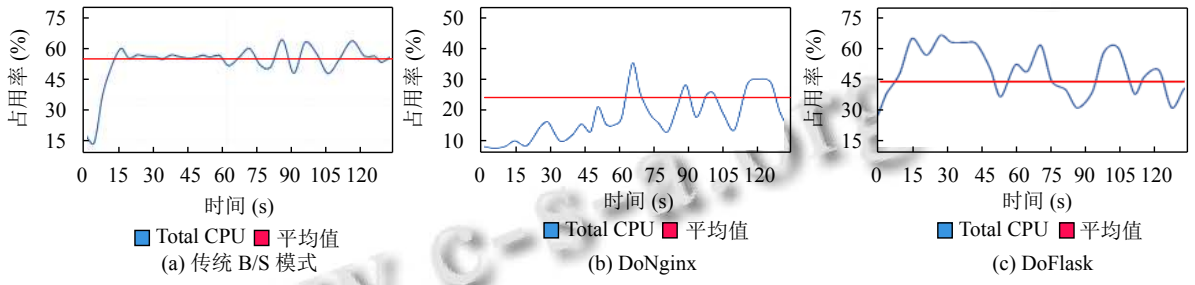

图 8 各种方法的 CPU 占用率

3.1.2 吞吐量测试

图 9 为吞吐量的测试分析图, 显示各种方法的数据吞吐量. 依图 9(a)–图 9(c) 的顺序显示, 传统 B/S 模式、DoNginx 模式和 DoFlask 模式的吞吐量. 由此可知, 传统方法的吞吐量在 2.5 Mb/s 与 7.5 Mb/s 之间, 其次, DoNginx 模式的吞吐量在中间部分逐渐上升后又稍微下降并趋于平稳. 然而, DoFlask 模式的吞吐量起伏不定, 却始终高于 5 Mb/s. 依照式 (12) 可以得出平均值, 依序为 4.8 Mb/s、3.8 Mb/s 和 4.2 Mb/s. 故此, 结论是 B/S 模式数据吞吐量最大, 传输数据效率最快, 而 DoNginx 模式数据吞吐量最少.

3.1.3 产品交付完整性测试

测试的过程, 主要经过两个步骤: (1) 在服务器部署 Web 应用程序, 完成后传输到用户端. (2) 在用户端, 对 Web 应用程序进行检测.

在进行 30 个产品的传输后, 发现环境不兼容导致的错误有模块错位、数据消失以及样式出错等. 图 10 是测试后得出的结果, 传统 B/S 模式开发的产品, 传输到与开发环境不同的用户端后, 70% 会出现以上错误, 需要开发人员继续进行维护. 而且出错率不稳定, 难以

预测结果. 以 DoNginx 模式开发的产品, 传输到不同环境用户端后, 20% 会出现错误. 使用 DoFlask 开发的产品, 出错率仅为 5%, 并且数据保持稳定, 可以对未来的实验结果进行推测.

总结实验测试的结果. 首先, 使用 DoNginx 技术的 CPU 占用率最低、效率高、内存占用较少和产品交付的出错率较低; 其次, 使用 DoFlask 技术对 CPU 的占用率较低、占用内存最少, 并且具有环境兼容性, 大大提高产品交付的稳定性; 最后, 传统的 B/S 模式对 CPU 占用率最高、占用的内存最多和产品交付的出错率最高.

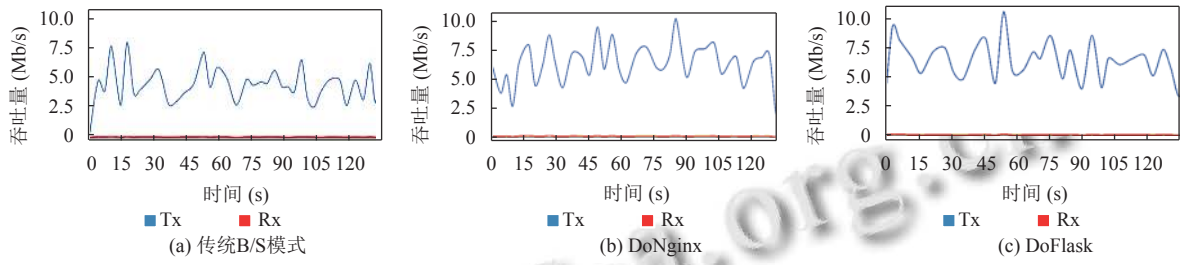

图 9 各种方法的吞吐量

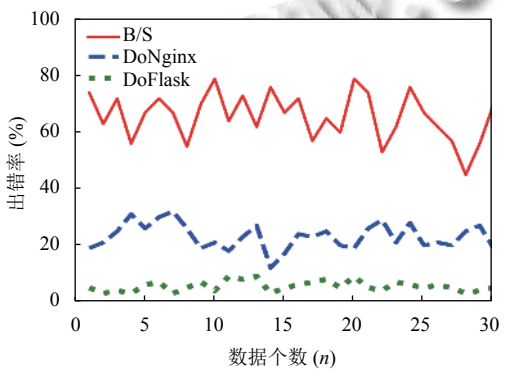

图 10 各种方法的完整性测试

3.2 实验分析

本节对 3 种模式性能测试实验结果进行分析.

(1) 分析 CPU 占用率. 第一, 使用 DoNginx 设计方案得到的产品运行速率快、数据传输效率高、在浏览器调试代码方便、开发效率高且 CPU 占用率低. 第二, 使用 DoFlask 的设计方案使产品可靠性与稳定性, 大幅提高; 通过可以打包生成镜像的技术, 提高产品重复利用率. 第三, 经过大量验证性实验得到的数据知道, 使用 Docker 轻量级容器技术, 将会比使用传统 B/S 技术节约 20% 左右的 CPU 利用率, 平均减少 50% 的内存使用量.

综合以上分析, 图 11 显示系统每 T s (实验设置为 30 s) 收集 CPU 的利用率, 以及这 3 种模式在不同阶段的比较.

在图 11 的 0–T 时间段部分, 可以看出此时 DoFlask 占用 CPU 最多. 然而, 传统 B/S 模式占用的 CPU 利用率比较少, 这是因为此时处于修改代码的状态; 2T–4T 时间段是处于程序执行的状态. 在 T–2T 时间段的部分, 可以发现 DoNginx 和 DoFlask 都达到最大值, 因为这两种方案都处于执行的状态, 消耗资源最大. 然而, 在 2T–3T 的时间段, DoFlask 处于创建镜像的状态, 无需消耗大量的资源. 3T–4T 的时间段内, 在另一台服务器的 DoFlask, 处于拉取镜像和执行程序的状态, 此时消耗资源较多. 所以, 可以得知无论在哪一阶段, DoNginx 都是 CPU 占用率最低的技术, DoFlask 紧随其后.

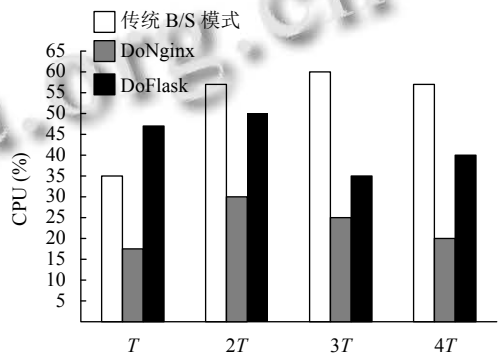

图 11 CPU 占用率比较图

(2) 分析数据吞吐量. 经由图 12 的吞吐量比较图乃是以每一个 T s (30 s) 为单位, 收集数据传输吞吐量后分析, 有以下两点. 第一, 传统 B/S 模式部署网页的优点是传输数据速度快, 数据吞吐量大, 并且较为平稳. 第二, DoNginx 和 DoFlask 技术在前期的数据传输量小, 后期逐渐增加, 这是因为 T–3T 时间段任务是拉取和生成镜像; 3T–4T 时间段任务是执行程序.

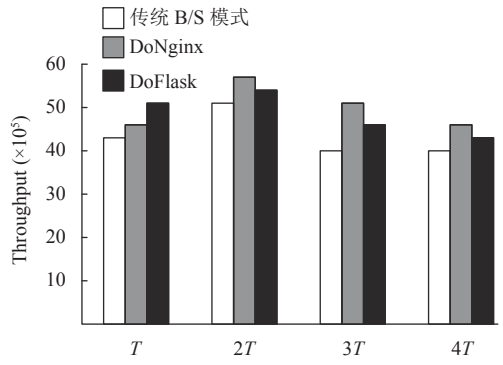

图 12 数据吞吐量比较图

(3) 分析交付完整性. 使用 DoFlask 模式创建网页, 将开发网页服务器的文件打包生成镜像, 然后上传到云空间, 可以在其它服务器下载并启动此镜像. 这种方法完美的解决环境不兼容的问题, 保证交付产品的完整性. 所以 DoFlask 模式交付完整性最高.

根据以上的 3 点分析, 归纳内容如表 3. 它显示各个系统模式的性能测试结果.

表 3 各模式的性能测试表

编号	系统模式	CPU占用率 (%)	吞吐量 (Mb/s)	内存占用量 (Mb)	完整性 (%)
1	B/S	56	4.8	200	30
2	DoNginx	24	3.8	50	80
3	DoFlask	43	4.2	40	95

(4) 分析变异数状态. 根据收集的数据, 式 (13) 得出标准差, 使用式 (14) 得出变异数. 从变异数变异的程度, 可以看出 3 种模式彼此之间的不同.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X})}{N}} \tag{13}$$

$$Var(x)=\frac{\sigma}{\overline{X}} \tag{14}$$

总结得出表 4, 发现 CPU 的变异程度大于完整性测试, 完整性测试大于数据吞吐量. B/S 模式的 CPU 占用率和吞吐量的变异程度较高, 完整性测试变异程度最高. DoNginx 模式的 CPU 占用率变异程度最高, 吞吐量和完整性测试变异程度较低. DoFlask 模式的 CPU 占用率和完整性测试变异程度最低, 吞吐量变异程度较高. 所以, B/S 模式总体变异数最大也最不稳定; DoNginx 模式变异数较低, 较为稳定; DoFlask 模式变异数最低, 最为稳定.

综合 B/S 模式、DoNginx 和 DoFlask 模式建构的网页服务器过程, 分析测试的性能后得出结论: 利用 DoFlask 模式建构网页服务器比传统 B/S 模式和 DoNginx 下的建构交付完整性提高 20%–65% 个级别, CPU 占用率较低, 并且缩小内存占用量, 明显具备开发上的优势.

表 4 CPU 和吞吐量的变异数状态

编号	系统模式	CPU (%)	吞吐量变异 (%)	完整性测试 (%)
1	B/S	17.1	8.7	11.2
2	DoNginx	18.6	8.5	8.4
3	DoFlask	15.1	9.8	5.3

4 结论与展望

针对现有的 B/S 模式开发网页服务器存在数据传输慢、资源占用大且不兼容的缺点, 提出了新型的 Docker 技术分别结合 Nginx 和 Flask 的模型. 对 3 种开发模型进行研究和测试, 根据 3 种模式在开发性能和兼容性的不同, 推断出最适合开发网页的模式为 DoFlask 模式, 为开发人员和运维人员节省大量开发交付与等待的时间, 降低产品环境更新过程中可能的出错率. 此外, 服务器虚拟化技术打破单应用单服务器的传统部署模式, 提高硬件资源的利用率, 简化服务器的运维管理工作, 从而降低管理费用.

无论用 DoNginx 或者 DoFlask 结合数据库技术, 都可以快速地搭建 Web 系统. 对于一般的初学者而言, 需要了解和掌握全部的技术, 还是需要一定的时间学习和探索. 未来可以结合这两种技术, 提供即便没有开发经验的初学者, 只要填入需求, 依照发展的新算法, 便能够自动推荐并且直接生成基本网页的系统.

参考文献

1 Zhang SJ. Development and forecast analysis of english network teaching aid platform under B/S technology framework. 2020 12th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). Phuket: IEEE, 2020. 68–71. [doi: 10.1109/ICMTMA50254.2020.00023]

2 Elliott D, Otero C, Ridley M, *et al*. A cloud-agnostic container orchestrator for improving interoperability. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). San Francisco: IEEE, 2018. 958–961. [doi: 10.1109/CLOUD.2018.00145]

3 Sultan S, Ahmad I, Dimitriou T. Container security: Issues, challenges, and the road ahead. IEEE Access, 2019, 7:

52976–52996. [doi: 10.1109/ACCESS.2019.2911732]
4 陈雁, 黄嘉鑫. 基于 Kubernetes 应用的弹性伸缩策略. 计算机系统应用, 2019, 28(10): 213–218. [doi: 10.15888/j.cnki.csa.007106]
5 Kozlovszky M, Törőcsik M, Schubert T, *et al*. IaaS type cloud infrastructure assessment and monitoring. 2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija: IEEE, 2013. 249–252.
6 税正威. 基于 Docker 的 Java Web 快速开发部署平台的设计与实现 [硕士学位论文]. 北京: 北京邮电大学, 2018.
7 Ha JH, Kim J, Park H, *et al*. A Web-based service deployment method to edge devices in smart factory exploiting docker. 2017 International Conference on Information and Communication Technology Convergence (ICTC). Jeju: IEEE, 2017. 708–710. [doi: 10.1109/ICTC.2017.8190760]
8 Li YC, Xia YM. Auto-scaling web applications in hybrid cloud based on docker. 2016 5th International Conference on Computer Science and Network Technology (ICCSNT). Changchun: IEEE, 2016. 75–79. [doi: 10.1109/ICCSNT.2016.8070122]
9 Bella MRM, Data M, Yahya W. Web server load balancing based on memory utilization using docker swarm. 2018 International Conference on Sustainable Information Engineering and Technology (SIET). Malang: IEEE, 2018. 220–223. [doi: 10.1109/SIET.2018.8693212]
10 Qin E, Wang YL, Yuan LP, *et al*. Research on Nginx dynamic load balancing algorithm. 2020 12th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA). Phuket: IEEE, 2020. 620–624. [doi: 10.1109/ICMTMA50254.2020.00138]
11 戴伟. 基于 Nginx 高性能 Web 服务器的理论研究与性能改进 [硕士学位论文]. 南京: 南京邮电大学, 2019.
12 张延冬, 邢艳芳. 基于 Docker 的运维平台设计. 计算机时代, 2018, (4): 16–18, 22.
13 Li P, Zhang XZ. The design and implementation of Web-based FTP remote management system. 2012 Second International Conference on Business Computing and Global Informatization. Shanghai: IEEE, 2012. 774–777. [doi: 10.1109/BCGIN.2012.207]
14 Bin C. Formalized description and analysis of FTP on petri net. 2015 International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP). Adelaide: IEEE, 2015. 176–179. [doi: 10.1109/IIH-MSP.2015.118]
15 Xie JH, Liu ZY. Based on C/S and B/S mixed mode telephone marketing system. 2012 International Conference on Industrial Control and Electronics Engineering. Xi ’an: IEEE, 2012. 874–876. [doi: 10.1109/ICICEE.2012.232]