轻量级 Web 服务器的高并发技术研究与实现

李思莉,杨井荣,苟 强
(成都理工大学 工程技术学院 电子信息与计算机工程系,四川 乐山 614000)

摘 要:主要研究并实现了使用 Vert. x 框架将同步非阻塞模式作为 Web 开发的 IO 模型,在轻量级 Web 服务器 Nginx 上利用高并发技术开发部署学分管理系统。该系统不仅重构了 SpringMvc 三层架构,而且将原来的 3 层模型扩充为 5 层模型,缓解了高并发数据量对系统的压力。在数据安全方面,改变了传统的利用数据库隔离机制来保证数据安全的低效做法,将对数据安全的保证放到持久化层的 Server 层。在并行数据接收方面,利用线程池,数据缓冲技术,进一步提高了学分管理系统的处理效率。除此之外,又通过创建多路复用的通信通道,着重在百万级并发的通信层面上对传统 Web 的开发方式进行了优化。最后,通过实验与传统的 Web 的 IO 方式作对比,得出异步输入输出在 Web 的应用中完全能胜任百万级甚至更高的并发量的结论。

关键词:高并发;Vert. x;同步非阻塞模式;SpringMvc;持久化层
中图分类号:TP302. 1 **文献标识码**:A **文章编号**:1673-629X(2020)10-0075-04
doi:10. 3969/j. issn. 1673-629X. 2020. 10. 014

Research and Implementation of High-concurrency of Light Weight Web Server

LI Si-li, YANG Jing-rong, GOU Qiang
(Department of Electronic Information and Computer Engineering, The Engineering & Technical College of Chengdu University of Technology, Leshan 614000, China)

Abstract: We mainly study and implement the use of Vert. x framework to take synchronous non blocking mode as the IO model of Web development, and adopt high concurrency technology to develop and deploy credit management system on the lightweight Web server Nginx. This system not only reconstructs the three-tier structure of SpringMvc, but also expands the original three-tier model to five-tier model, which alleviates the pressure of high concurrent data volume on the system. In terms of data security, the traditional inefficient method of using database isolation mechanism to ensure data security is changed, and the guarantee of data security is put into the Server layer of persistence layer. In terms of parallel data receiving, the processing efficiency of credit management system is further improved by using thread pool and data buffer technology. In addition, by creating a multiplex communication channel, the traditional Web development mode is optimized on the level of millions of concurrent communication. Finally, by comparing the experiment with the traditional IO mode of Web, it is concluded that asynchronous input and output can be used in the application of Web with a concurrency of millions or more.

Key words: high-concurrency; Vert. x; NIO mode; SpringMvc; persistence layer

0 引 言

大量用户在同一个时间点同时访问某个相同的站点称为高并发。高并发现象在如今的互联网行业应用中非常普遍,如 12306 铁路购票网站,双 11 时阿里巴巴、京东、唯品会等电子商务网站要处理的并发数通常都高达每秒百万级。但如何处理高并发却是一个非常难的技术瓶颈。该文研究的是在单机无集群的情况[1],以 NIO 为基础的同步非阻塞 IO,而非传统的 IO 方式,结合 Vert. x[2] 的事件驱动完成同步通信与异步事件处理的并行运算,是数据通信部分百万级别的并发。并在此研究基础上利用 Java Spring 线程池,完成了课外学分管理系统。通过大量的实验数据,与传统 Web 应用的 IO 方式进行对比,得出论文研究并实现的 MVC 层的扩展、数据安全优化、同步非阻塞模式与 NIO 在 Web 的应用中完全能胜任百万级甚至更高的并发量的结论。同时,由于这种异步事件处理方式是

收稿日期:2019-12-13 **修回日期**:2020-04-15
基金项目:四川省教育自然科学重点项目(18ZA0077);乐山市科技计划项目(19JRK229)
作者简介:李思莉(1974-),女,硕士,副教授,研究方向为云计算、软件系统架构。

基于 Spring 管理的线程池,在系统扩展上,很容易实现分布式系统完成更多的并发与集群架设。

1 高并发学分管理系统架构

客户/服务器模式(C/S)不能应对多平台带来的开发时间、开发效率、开发投入等多方面要求,加之各 PC 之间操作系统不同,为了兼顾过时的 Windows XP 系统,在开发 PC 端系统时通常出现两种情况:(1)开发多个版本;(2)兼顾 XP 不使用高版本 Windows 系统的特性和高效率 API。这两种情况都不好,因此在开发学分管理系统时,放弃 C/S 架构,使用 B/S[3] 架构。这套架构,在客户端上只需要前端网页和可运行在系统上的浏览器就可满足用户对于多平台,不同系统设备的需求,节约开发时间和开发成本。在具体的程序内部架构设计上,传统的三层架构已经无法满足系统高并发需求,数据传输中传统的 I/O 设计模式和传统的 I/O 传输必将面临性能瓶颈甚至会导致整个课外学分管理系统的崩溃。因此在实际的开发过程中,将系统设计成 5 层模式,由外向内展开依次是:

(1)负责与前端信息交互的 restfulApi 层;

(2)负责管理处理逻辑的中央组件管理层;

(3)负责管理并发线程的调度和管理的并发层;

(4)负责处理信息的逻辑层;

(5)负责持久化信息的 ORM 层。

通过实验证明,该架构在技术上是可行的,在并发请求每秒 10 万数量级上依然保持稳定。

2 高并发系统分析

2.1 处理高并发请求的 MVC 层

要完成十万级,百万级的并发请求,普通的 IO 会导致系统性能急速下降,这将导致系统无法正常运行。因此,在具体的开发实现中,使用了非阻塞式 IO。非阻塞式 IO 分为异步非阻塞 IO 和同步非阻塞 IO。通过对学分管理系统的需求分析,得出整个流程不需要消耗很多的等待时间,因此,采用同步非阻塞 IO 模式。加之非阻塞 IO、零拷贝、事件驱动等特性,在开发生态圈里有很多经验可取,在框架的设计上也能利用现有的同步非阻塞 IO 框架,不必重头开发底层。

2.2 处理高并发请求的 Server 层

Server 层的高并发,着重体现在线程安全上,在数据处理上,不能出现很多线程去同时操作运算数据的情况。对于线程安全,在整个 Server 层实现上完全使用了线程安全的数据结构,如:ConcurrentHashMap,SynchronizeList 等,需要注意的是要避免使用过时的线程安全的数据结构,如:vector,HashTable 等,这会降低整体的效率。

除了线程安全的数据结构,很多方法的逻辑也不允许多线程同时操作,一般的解决方案是使用 Synchronize 关键字对需要加锁的方法或者代码块进行修饰,但这是一种悲观锁,如果发生异常,会出现阻塞,这对系统是致命的,不仅会导致后续的操作挂起,还会导致程序崩溃。要避免发生这种情况,在 Server 层实现上采用了非阻塞的并发算法 CountDownLatch[4],它是 Java 提供的原生非阻塞并发算法,可以有效实现学分管理系统的线程同步。

2.3 持久化、并行数据接收与 Restful 层

利用数据库的隔离机制完成数据安全是一种低效的做法。在学分管理系统持久化层的设计中,将数据安全因素放到调用持久化层的 Server 层里面去实现[5]。持久化层事务的传播机制统一采用 Spring 的传播机制,并利用缓存技术,减少系统响应时间。在初始化时,采用快速数据库连接池初始化一个足够大的数据库连接池交给持久化层使用[6]。

并行数据接收是并发的开始,这里采用了成熟的模式设计,即一个接收的总线 Boss 线,多个负责传输转发到相应处理的 Server 逻辑的 Worker 线程,将多个 Worker 线程初始化为一个线程池由 Spring 统一管理,它存在于整个 Springboot 程序中。这个模式实现了代码复用,减少了初始化、调用等冗余代码,也能更好地融合在主框架里。

Restful 层的设计采用 Vert. x Web 框架而放弃了低性能的 SpringMvc[7],Vert. x 是事件驱动的,整个处理过程基于事件总线而非单独的控制器。

3 高并发关键技术实现

3.1 SpringBoot 配置说明

整个系统采用 YAML 配置模板,Server 配置了 Http 访问端口,访问根路径和内嵌的 Tomcat 编码[8]、响应时间等配置,其关键参数如表 1 所示。

表 1 关键配置参数

Port	uri-encoding	max-threads	min-spare-threads	pool size	instances	max-file-size	max-request-size
10086	UTF-8	800	30	60	30	100 M	100 M

3.2　Restful 层的实现

Restful 层的设计采用 Vert. x Web 框架,它采用异步模式,通过事件循环调用存储在异步任务队列中的任务,大大降低了传统阻塞模型中线程对于操作系统的开销[9]。

整个学分管理系统实现高并发通信高效率的核心步骤是创建多路复用的通信通道。为了减少冗余代码,主框架采用 Springboot。将复用通道交由 Spring 统一管理,在此之前要创建由 Spring 管理的 Worker 线程池,部分代码如下:

```
@Component
public class SpringVertxFactory implements VerticleFactory, ApplicationContextAware {
private ApplicationContext applicationContext;
@Override
public String prefix() {
return "credit"; }
@Override
public boolean blockingCreate() {
return true;
    }
@Override
public Verticle createVerticle ( String s, ClassLoader classLoader) throws Exception {
        String clazz = VerticleFactory. removePrefix(s);
return (Verticle) applicationContext. getBean(Class. forName(clazz));
}
@Override
public voidsetApplicationContext(ApplicationContext applicationContext) throws BeansException {
this. applicationContext = applicationContext; }}
```

上述代码完成了三个目标:

(1)实现了 VerticleFactory 和 ApplicationContextAware 接口,VerticleFactory 接口能产生 Vert. x 工作线程, ApplicationContextAware 接口是当 SpringContext 初始化完成后,用于获取 SpringContext 的接口,其目的是将产生的 Vert. x 工作线程 Verticle 加入到 SpringContext 中,达到由 Spring 容器统一管理 Verticle 线程池的目的[10]。

(2)初始化通道总线和事件总线,注册 RestfulApi 到 Vert. x,并设置相关联的属性。使用了线程同步的方式保证初始化顺序执行。

(3)初始化 Vert. x 核心容器 Vertx,并设置最大线程量和最大连接响应时间。注册 Vert. x 工作线程到 Vert. x 容器,检查初始化过程中是否超时,初始化过程中是否有错误,以及是否全部线程都已经初始化完成。

由于 Vert. x 的工作线程由 Spring 容器统一管理[11],只有当 Spring 容器初始化完毕后才能使用 Spring 容器里的 Vert. x 工作线程,故需要监听 Springboot 的启动消息事件。Vert. x 中的 RestfulApi 没有一套现成的能直接完成映射注册的开发注解或模板类,因此 Vert. x 的 RestfulApi 需要自己去实现。

该文定义的 RestfulApi 必须继承 AbstractVerticle 这个抽象类,才能被 Vert. x 核心容器接收作为通信处理链上的一部分。并且会去执行该对象类里面的 start 方法,所以一定要重写这个方法。这个方法里面首先要创建处理逻辑的代理接口,而且这个接口也必须要能被 Vert. x 核心容器接受,然后注册访问地址[12]到 Vert. x 的核心路由上面,由于整个过程是事件驱动的,所以要设立监听端口。将事件处理的逻辑结果写入到 Router 的 routingContext 中,这样才可到前端解析[13]。

4　实验结果

4.1　程序编写

Java 原生的 NIO API 在开发中显得过于繁琐,也未封装成一个高并发的架构。为了减少开发带来的时间消耗和框架封装的性能消耗,采用现有的 Vert. x 框架。现有主流的 Web 开发中 Spring 是必不可少的,将两者结合由 Spring 管理 Vert. x 的部分组件能用工程化的开发流程去简化异步 Web 程序的开发。

将部分 ajax 请求接口更改为 Vert. x 开发,应用更多 Spring 带来的方便且规范的服务,减少在后续服务带来的开发难度和性能消耗。

整合 Web 其余所有部分通过 Spring 与 Vert. x 协同工作,并借此管理 Vert. x 的异步线程池,动态地申请资源,减少性能浪费。

4.2　相同固定时间和压力内测量吞吐量和响应时间

为了保证实验的可行度和可信度,采用由 Apache 基金会开发的 JMeter 压力测试工具[14-15],对该项目进行测试,并且实验是基于课外学分管理系统设计的,这两个不同接口会运行在同一个 Java 虚拟机中,最大程度地保证了在运行环境、参数、性能等各方面的一致性,得出的实验结果对比也更有说服力。

设定为百万级并发请求:让程序能模拟一百万个用户对同一个接口模块请求。

设定图形结果计算包括吞吐量和响应时间。

固定时间为一分钟或者一分钟又几秒钟(结束上百个线程会消耗几秒时间)。

接口调用的逻辑和功能完全一致。

设定线程请求无延迟,即延迟 0 ms。

4.3　学分管理系统实验结果及实验数据处理

在实验过程中,为了保证发送的数量是一样的,应当同时启动两个线程组,且设置完全一模一样,设置在

同一个测试组中,启动整个测试组。

在此期间密切关注线程数量变化,记录线程非满载的情况下的测试数据,在后期处理数据时需要除去这一部分不合格的启动数据。观察后台是否已经崩溃,因为在百万级的并发下 SpringMvc 大概率会假死,如果已经崩溃或者假死则数据上没有对比的必要性。

在实验数据监听器中取得相应数据和统计图形,首先在 SpringMvc 组里面取得吞吐量和响应时间结果,如图 1 和图 2 所示。

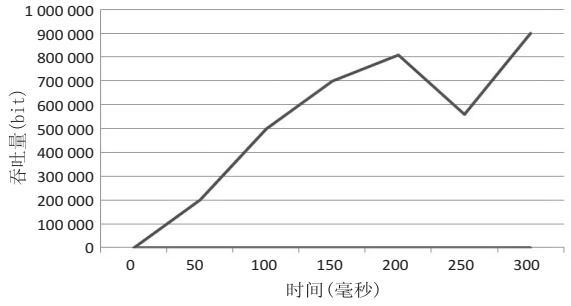

图 1 学分管理系统 SpringMvc 吞吐量

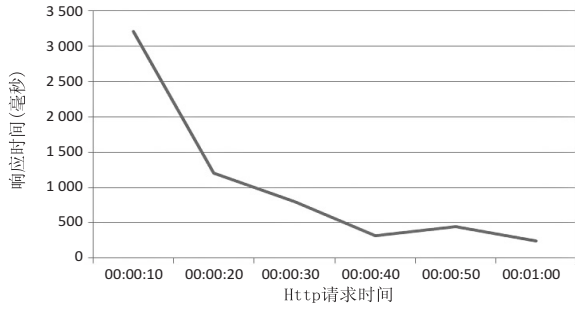

图 2 学分管理系统 SpringMvc 响应时间

Vert. x 的响应时间和吞吐量如图 3 和图 4 所示。

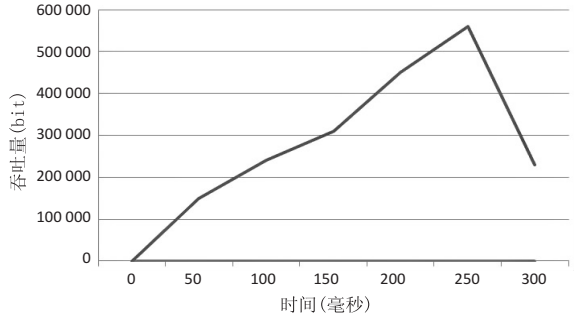

图 3 学分管理系统 Vert. x 吞吐量

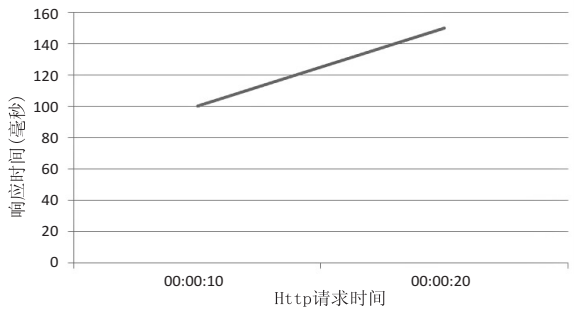

图 4 学分管理系统 Vert. x 响应时间

由上面四幅图片可以获得的信息,仍然需要比对 SpringMvc 和 Vert. x,需要排除不合格的测试量。首先排除前 10 秒钟的线程启动时测试的数据,再减去 20 秒后衰减的线程量这样的响应时间才是合格的比对样本,其结果如图 5 所示。

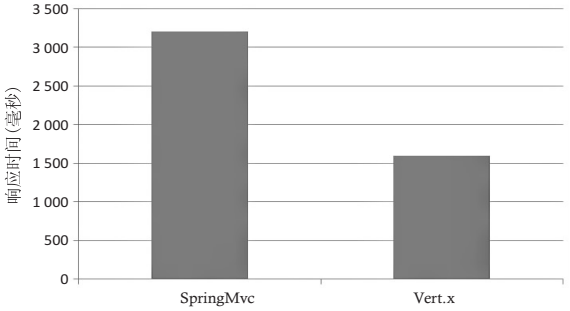

图 5 响应时间对比

相同时间发出的数量能保证在误差范围内,故可以记录所有的量一次吞吐代表完成一次请求,结果如图 6 所示。

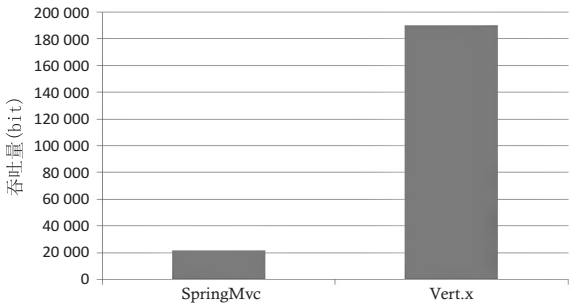

图 6 吞吐量对比

5 结束语

在实验数据的对比下能发现,在响应时间是万倍的差距,在吞吐量上是数十倍的差距,在同一 JVM,同一功能,执行同一逻辑,同一线程组中排除不合格数据得出的数据对比中可以得到如下结论:

(1)相较于传统且主流的 SpringMvc 的 IO 模式,NIO 更能胜任高并发环境,而且这个性能是提升巨大的,能在主要的两方面中体现出指数倍的差距;

(2)能在相同逻辑下大幅度减少通信时间;

(3)相同条件下,NIO 通信的程序能处理更多的请求。

参考文献:

[1] 孙明刚. 一种使用环形缓存和环形队列实现 UDP 高效并发的方法[J]. 中国新技术新产品,2016(11):6-7.

[2] 阮晓龙,董凯伦. 基于 ELK 的 Nginx 反向代理日志分析与业务服务质量评价体系研究与探索[J]. 网络安全技术与应用,2019(12):15-18.

[3] 夏德宏. 浅析网络程序设计中的并发复杂性[J]. 电脑知识与技术,2016,12(14):223-224.

(下转第 85 页)