面向 Spring 的热点代码在线部署方法研究

万嘉龙,况立群*,熊风光,薛红新,韩　燮
(中北大学 大数据学院,山西 太原 030051)

摘　要:随着 Spring 生态不断发展,越来越先进的部署方式降低了部署的复杂度,提高了不同环境下的部署效率,但是在预生产环境下,对频繁改动的热点代码,其部署效率不是很理想,一些简单的代码修改就会引发对所有依赖服务的重新编译部署,给项目部署、运维以及测试带来很多预期之外的影响。在线部署机制针对这个问题进行了改进,该机制使用自定义注解作为与外部应用通信的桥梁,使在线部署模块独立于外部应用。在线部署模块使用 React 创建可视化页面,在可视化页面中管理相应的热点代码。热点代码在编辑完成后注入到在线部署模块,进而完成其热点功能在预生产环境的在线部署。实验表明,相对于重新编译部署,该部署机制对原有代码侵入影响较小,减少了反复部署应用以及不同部门协调合作造成的时间浪费,在保持系统稳定运行的同时,提升了系统的部署效率,并成功应用于新型智慧城市评估系统的热点代码模块。研究结果将为迭代开发以及高效部署提供设计思路与技术支撑。

关键词:Spring;软件部署;热点代码;虚拟容器;Docker
中图分类号:TP311.5　　**文献标识码**:A　　**文章编号**:1673-629X(2023)05-0105-05
doi:10.3969/j.issn.1673-629X.2023.05.016

Research on Hot Code Online Deployment Method for Spring

WAN Jia-long,KUANG Li-qun*,XIONG Feng-guang,XUE Hong-xin,HAN Xie
(School of Data Science and Technology,North University of China,Taiyuan 030051,China)

Abstract:With the continuous development of Spring ecology,more and more advanced deployment methods reduce the complexity of deployment and improve the deployment efficiency under different environments. However,in the preproduction environment,the deployment efficiency of frequently changed hot code is not ideal. Some simple code modifications may trigger the recompilation and deployment of all dependent services,and bring many unexpected impacts on project deployment,operation and maintenance and testing. The online deployment mechanism is improved to solve this problem. The mechanism uses custom annotations as a bridge to communicate with external applications,making the online deployment module independent of external applications. The online deployment module uses React to create a visualization page and manage the corresponding hotspot code in the visualization page. After editing,the hotspot code is injected into the online deployment module to complete the online deployment of its hotspot functions in the preproduction environment. Experiments show that compared with recompiling and deployment,such deployment mechanism has less impact on the original code intrusion,reduces the time waste caused by repeated deployment of applications and coordination and cooperation between different departments,improves the deployment efficiency of the system while maintaining the stable operation of the system,and has been successfully applied to the hot code module of the new smart city evaluation system. The research results will provide design ideas and technical support for iterative development and efficient deployment.

Key words:Spring;software deployment;hot-spot code;virtual container;Docker

0　引　言

随着业务不断变更和功能持续迭代,人们对软件更新方式的要求越来越高。个人的开发经验、团体的技术积累以及需求的不断变化,都会增加软件部署的复杂度和风险。在实际开发中,项目集成以及组件的更新会消耗大量的时间,并且存在众多潜在风险,往往等到项目开发结束的时候,才可能发现这些问题,最终可能导致项目延期。持续集成与部署可以使项目实施过程透明化,进而完善开发结构,改进开发质量,降低交付风险[1]。随着国家数字化进程的推进,越来越多

收稿日期:2022-05-09　　**修回日期**:2022-09-13
基金项目:国家重点研发计划(2018YFB2101504);山西省回国留学人员科研资助项目(2020-113);山西省科技成果转化引导专项(202104021301055)
作者简介:万嘉龙(1993-),男,硕士研究生,研究方向为数据可视化、数据挖掘;通讯作者:况立群(1976-),男,教授,CCF 会员(48959M),研究方向为人工智能与计算机视觉、软件工程。

的项目采用这种开发方式,持续集成在应用系统中发挥着越来越重要的作用[2]。

传统上通常使用人工干预的持续化部署模式,通过各种交互式手段完成自动化流程,以此实现对应的代码部署以及功能上线。Jenkins 是一款持续集成和持续交付的开源软件,常用于项目部署,不仅提供了直观友好的用户界面,而且可减少由于人工直接部署导致项目出现的系统兼容和版本冲突等问题[3]。但是这种方式不适用于部署频繁改动的热点代码,因为这将导致频繁的对整个应用程序进行重新打包以及手工部署,由此产生较高的学习成本,且效率相对较低。

目前主流的部署方式大多采用 Docker 等虚拟容器,其本质上是一个内部封装 Jenkins 或与 Jenkins 同类插件的系统[4-5]。虚拟化的持续化部署方式可以减少系统开销,同时增加容器化的可移植性[6-8]。如果直接将应用部署到系统中,一旦系统环境发生变化,会引发应用程序的运行风险。虚拟容器的部署方式降低了部署难度和部署风险,并且在代码频繁改动的环境下,可以提高部署效率。但是,该方式仍然需要打包部署全部应用程序,无法仅针对局部热点代码进行部分编译部署,因此没有从根本上解决热点代码部署效率低下的问题。

针对热点代码部署效率低下的问题,该文借鉴低代码设计思想,提出一种面向 Spring 的热点代码在线部署方式,为专业计算机从业人员提供新颖的软件开发部署模式,提升代码质量及部署效率[9-10]。该方法根据 Bean 容器找到对应的算法代码段进行数据抽取,利用在线部署的方式对代码进行编译、注册及替换等操作,实现局部业务功能的在线更新。该方法应用于新型智慧城市评估系统,通过测试热点代码的频繁更新迭代,验证了在线部署模式的高效性。

1 在线编译的容器化加载机制

Java 的编译分两个部分,首先将源文件编译成字节码文件,然后字节码文件被虚拟机加载以后变成机器码[11]。对于开发者来讲,主要的关注点在第一部分。在 Spring 的在线编译中,应用内设置锚点,应用外编译代码,再将指定的功能加入到应用程序,这种在线编译的加载机制可将代码信息按照自定义格式加载到对应的环境中。而容器化的机制是将编译好的对应代码加入到 Spring 容器管理目录,其关键思想是将新增的功能代码直接加载到应用,无需拉取所有代码进行编译发布[12]。

在线编译的容器化加载机制如图 1 所示。前台将代码与类名称传入服务端,构成输入参数,将其连同程序构建的编译参数、输出参数以及程序错误输出参数一起传入到 JavaCompiler 编译工具类。待 JavaCompiler 执行成功输出对应类的 class 文件,如果执行失败将在程序错误输出中写入失败信息。容器化过程将类名称转化成相应的 Bean 名称并构成 Bean 定义,连同 class 文件一起传入 Bean 工厂,由 Bean 工厂生成相应的 Bean 容器并添加至 Spring 容器列表。

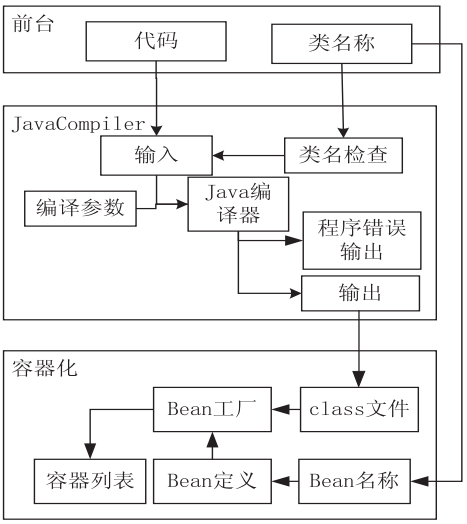

图 1 在线编译的容器化加载机制

2 热点代码在线部署方法

2.1 在线部署流程

在线部署模块是由容器配置模块、热点代码编译模块和容器装载模块组成。用户按照定义好的格式在容器化管理列表的代码编辑功能页面下进行代码编辑,并通过代码检查。前端将代码信息传输至在线部署模块的服务端,服务端对获取的信息进行解析,确定符合容器定义规范后,再对代码进行编译并生成编译文件。在线部署模块将对应的文件传输到系统指定的目录下,通过 Spring 将文件注册到 Bean 列表,并交由 Spring 进行管理,具体流程如图 2 所示。

当应用系统新增功能时,应用系统首先要调用 API 实例化容器,然后提取相应的属性信息,最后由在线部署模块将应用数据装入执行单元,完成功能的调用。系统维护的过程中,如果目标执行功能有变动,需要对目标执行功能进行卸载再由在线部署模块完成功能的新增。而在执行功能卸载时,应用系统需要通过容器化管理列表查询 Bean 容器的信息,在 Spring 中将该容器卸载,在线部署模块通过查找元信息中存储的类名来删除对应的编译文件,同时删除存放的 Bean 信息和类文件。

2.2 容器配置模块

容器配置模块是基于当前流行的 React 框架并使用 NodeJs 进行编译的前端平台。该平台使用 React 语言主要基于两方面的原因,一方面是 React 使用虚拟

Dom 对页面进行渲染,这种方法比直接操作 Dom 树速度更快,另一方面 React 可以对功能细粒度进行封装,从而提升组件的复用,且手动优化性能更加便利,所以大型项目往往都会选择 React 作为基础框架[13]。

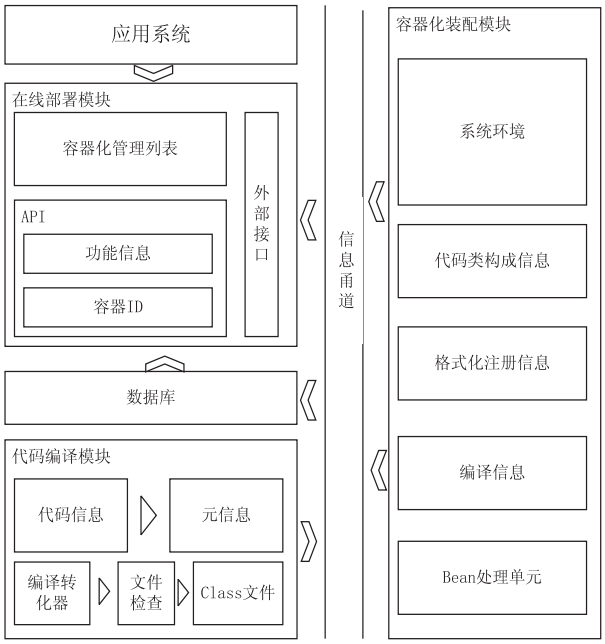

图 2　热点代码在线部署方法

如图 3 所示,容器配置模块是由容器化管理列表与对外提供服务两个部分组成,其功能是为系统提供应用管理、代码编辑以及对外服务的 API。其中应用管理的功能是为在线部署模块展示注入的应用功能状态,以及对应用功能进行基本操作。而代码编辑模块为容器化管理列表提供代码编辑和代码检查的功能,方便使用者能够摒弃繁琐的配置,专注于指定功能的代码编辑。为了使系统更加方便地调用容器配置模块所构建的功能,在容器配置模块中加入了对外服务的 API,使应用系统能够依靠简单的配置就可以完成对注入功能的测试,同时也能为容器化管理列表提供对应的功能索引,方便进行应用功能的基本操作。

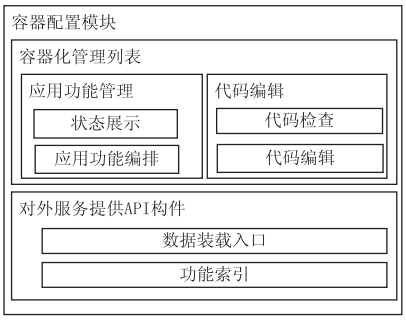

图 3　容器配置模块结构

2.3　代码编译模块

代码编译模块使用了 Java 软件开发包中自带的 java-tools 动态编译模块,可以对字符串构成的代码进行编译,并能提供丰富的编译接口。该模块将从前端传输来的代码和元信息存储到数据库,并将数据库中的对应状态标记为传输完成。随后,代码在该模块进行编译,如果编译成功,模块生成编译文件,并将生成的文件放到指定位置,然后把编译成功的状态传输到容器装载模块,接着执行后续操作。反之将编译失败的信息,连同对应的元信息一起传输到数据库中,并将数据库中的对应状态标记为编译失败。

常规采用的重新编译部署方式如图 4 的上部分所示,重新编译部署方式在功能修改时,由用户开发好相应的功能代码,并将代码上传至代码管理平台。用户要完成应用的部署还需使用持续集成工具 Jenkins,将代码拉取至服务端,Jenkins 在服务端对代码进行编译,然后编译生成 Jenkins 构件,启动构件即完成部署。该文采用基于在线部署的编译方式(见图 4 下半部分),用户是在在线部署模块中进行编码,服务端仅负责接收和编译由在线部署模块传输过来的代码,之后将生成的文件通过 Spring 的环境直接注入到系统中,进而完成项目的部署。与重新编译部署方法相比,在线部署编译方式减少了代码拉取以及构件启动的时间。在应用功能未确定时,功能的改变会引发频繁部署,采用在线部署方法可以节省很多的时间。

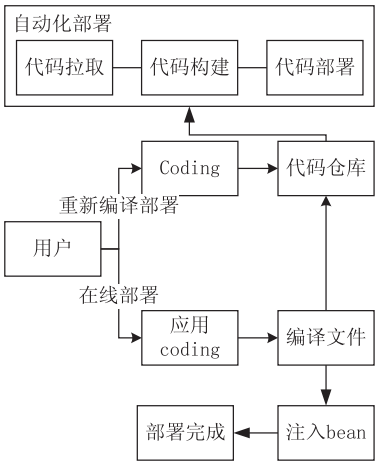

图 4　重新编译部署与在线部署的编译方式对比

2.4　容器装载模块

容器装载流程如图 5 所示。

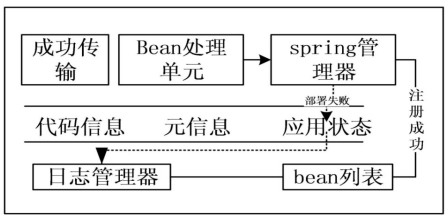

图 5　容器装载流程

容器装载模块将 Spring 应用环境的上下文实例化,通过加载具有复合注解的代码,将应用功能注入到 Spring 环境。该模块从编译模块获取编译成功的文件和元信息并加入到 Bean 处理单元,然后将新生成的

Bean 容器注册到 Spring 环境。如果注册成功,则依据元信息在数据库中找到指定记录,并将其状态更新为部署成功,然后把对应的状态写入到日志管理器,反之,注册失败则标记部署失败。

容器注册的核心是将热点代码转化成 Bean 容器。对所需功能注册时,将类名与 Bean 的 ID 进行绑定,保证在 Bean 注册机制下,注册的容器是唯一有效的。容器是整个 Spring 环境的重要组成部分,在应用系统中,Bean 的实例化获取与销毁都由 Spring 进行管理。如图 6 所示,当在线部署某一应用程序时,采用不停机的工作模式,事先定义好该程序的计算接口和数据信息接口,然后在代码编辑模块中实现上述两个接口。其中,计算接口用于实现具体的应用功能,而数据信息接口用于实现容器配置模块对应用功能的管理。接口代码完成编译后,Spring 调用 Bean 管理器将转化好的容器注册到容器列表,为代码无侵入做准备。应用在停机部署时,卸载在线部署模块不会影响系统的使用。依据不同开发阶段的要求,在线部署模块可以灵活制定相应的部署策略,降低开发与运维的工作难度,保障系统的稳定,降低项目管理成本。

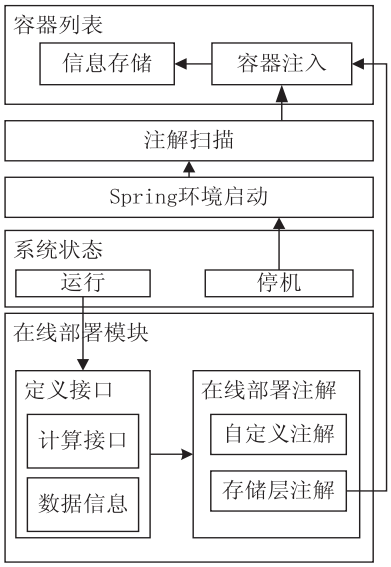

图 6 运行与停机状态下的容器注册方法

3 性能测试与分析

3.1 实验环境

实验的硬件环境是 Intel(R)Corn(TM)i7-7700HQ CPU @ 2.8 Hz 2.81 GHz,内存为 8 GB,软件环境为 windows tomcat idea2022。实验在 Spring 官网下生成空白的项目,添加相应的功能构建不同复杂度的系统,以便对比不同部署方式的部署效率。

3.2 评价标准

为了评估应用在不同软件复杂度下的在线部署方法与重新编译部署方法的效率,采用不同维度下的软件复杂度进行实验,衡量软件复杂度的常用度量方法,包括代码行数度量法和 Halstead 度量法。其具体构建如下:

代码行数度量法在空白项目中写入 100 行代码形成一个请求单元,标识系统拥有 100 的复杂度,将代码请求单元扩增至 10 个请求单元,可以使代码行数复杂度增加 10 倍,变为 1 000 的复杂度,以此类推,构建 10 000、100 000 的代码行数复杂度的测试项目。同理,使用 Halstead 度量法构建不同复杂度的项目,向空白项目中写入定义好的度量单位为 100 的模块,以此为模板扩增,与代码行数复杂度一致,分别增加至 10 000 和 100 000 的复杂度。度量单个单元复杂度的 Halstead 公式如下所示:

$$Mcount = 0.05(N + N_1)$$

其中,N 为程序词汇表的长度,N_1 为程序长度,为了与代码行数度量法的对比更加直观,设定了 0.05 的系数。

3.3 实验结果

对重新编译部署与在线部署两种方法在时间效率上进行对比,从程序开始编译计时,到实际应用部署为止,得出各自所需的时间。实验中选择不同度量法下的不同系统复杂度,在进行 10 次相对独立的实验后取平均值。其中 T_1 表示在线部署的时间,T_2 表示重新编译部署的时间,T_1/T_2 是两种部署模式下的时间比值。

表 1 两种部署方式在代码行数复杂度下的对比

代码行数度量	文中实验方法(T_1)/ms	重新编译部署方法(T_2)/ms	T_1/T_2
100	469	4 489	0.104 478
1 000	532	5 302	0.100 339
10 000	519	6 963	0.076 547
100 000	420	7 259	0.073 564

表 2 两种部署方式在 Halstead 复杂度下的对比

Halstead 度量法	文中实验方法(T_1)/ms	重新编译部署方法(T_2)/ms	T_1/T_2
100	434	2 249	0.192 975
1 000	457	3 029	0.150 875
10 000	526	4 530	0.116 115
100 000	493	7 210	0.068 377

如表 1 和图 7 所示,随着代码行数复杂度的增加,在线部署的时间明显小于重新编译部署的时间,并随着代码行数复杂度的增加,时间比值在逐渐下降。如表 2 和图 8 所示,Halstead 复杂度和代码行数复杂度的结果基本一致。结果表明,在线部署比重新编译部署效果更好,推广到现在大规模、复杂性高的系统中,运用该热部署方法部署代码更有利于节约时间和资源。

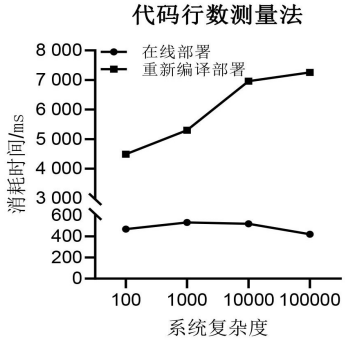

图 7　两种部署方式在代码行数复杂度下的对比

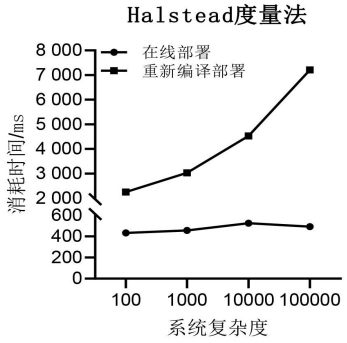

图 8　两种部署方式在 Halstead 复杂度下的对比

4　系统应用

该文提出的在线部署方法应用于新型智慧城市评估系统的评估体系构建模块。随着城市的不断发展,城市评价的指标体系也在不断地进步,指标计算也越来越复杂。在系统业务开发的过程中,对城市评估的评估体系往往考虑得不够全面,所以在系统运行初期,会不断地调整指标中的评估算法,与之相对应的测试环境以及预生产环境的应用也需要频繁地部署。针对这种情况,系统为其设计了可在线编辑的指标计算模块,该模块通过在线操作完成新指标的部署。

当用户对指标计算模块编辑时,先在指标编辑的模块中对指标计算的算法进行编辑,之后系统根据算法的名称和配置构建元信息,将代码连同元信息一起传输到系统后台,并在后台对算法进行编译,然后将编译后的文件通过 Spring 注入到容器列表,即可完成对应指标的部署。在对城市评估的时候,评估系统使用元信息查找到对应的算法,完成指标的计算。该在线部署方法为在线评估系统的指标拓展计算提供了思路,同时很好地解决了频繁部署热点代码的问题。

5　结束语

研究了基于 Spring 的在线热点代码部署方法,并在新型智慧城市评估系统进行了验证。在系统部署的过程中,运维人员会将所有的代码拉取到服务器中,对应用重新部署。功能应用在频繁改动的时候,重复拉取代码后进行部署,很容易导致一些问题出现。该方法能够使应用系统集成的在线部署模块在不停机的状态下,稳定地增加功能,并以较快的速度部署,同时在应用必须停机时,卸载在线部署模块,零配置重新部署功能 。该文设计的在线部署方法有望在紧急上线、预生产环境调试等环境下发挥作用[14-15]。

同时,该部署方案还有进一步完善的空间。在应用过程中热点的代码只能修改单独的类,只能在线单独改变其中一个功能,这种方法免去了代码的整包构建,提升了速率。但是这种方案仅仅只能在热点代码中使用,不适合大范围的修改代码,此外对于复杂环境的分布式系统,还需要考虑各个层级的依赖问题。

参考文献:

[1] 何宏伟. 基于 Web 的 Docker 持续集成部署设计与实现[D]. 西安:西安电子科技大学,2020.

[2] JEONG J, HA Y, HONG C. Application hot deploy method to guarantee application version consistency and computer program stored in computer readable medium therefor: US10545754B2[P]. 2020-01-28.

[3] 张力文. 基于 Jenkins 的项目持续集成方案研究与实现[D]. 成都:西南交通大学,2017.

[4] 胡　鹏,常朝稳,祝现威,等. 面向持续集成的回归测试优化方法[J]. 计算机应用研究,2021,38(12):3709-3714.

[5] DE FRANÇA B B N, SANTOS P S M, MATALONGA S. Towards a theory on architecting for continuous deployment[J]. arXiv:2108.09571,2021.

[6] 付琳琳,邹素雯. 微服务容器化部署的研究[J]. 计算技术与自动化,2019,38(4):151-155.

[7] 钟炜达. 一种基于 Docker 的持续集成平台的设计与实现[D]. 广州:华南理工大学,2016.

[8] 李恩洲,况立群,张　元,等. 智慧供热大数据监测平台研究及应用[J]. 计算机技术与发展,2021,31(11):176-182.

[9] BOCK A C, FRANK U. Low-code platform[J]. Business & Information Systems Engineering,2021,63(6):733-740.

[10] WASZKOWSKI R. Low-code platform for automating business processes in manufacturing[J]. IFAC-PapersOnLine, 2019,52(10):376-381.

[11] 谢伟增,金振乾. Java 语言在线编译器的设计与实现[J]. 电子技术与软件工程,2019(20):241-243.

[12] SONI R K, SONI N. Deploy a spring boot application talking to MySQL in AWS[M]//Spring Boot with react and AWS. Berkeley:Apress,2021:103-141.

[13] 巢晟盛. 基于 SpringBoot 微服务架构下前后端分离的 MVVM 模型浅析[J]. 电脑知识与技术,2021,17(23):128-129.

[14] 周　莹,欧中红,李　俊. 基于 Jenkins 的持续集成自动部署研究[J]. 计算机与数字工程,2016,44(2):267-270.

[15] 黄　磊. 基于微服务的运维平台设计与异常检测研究[D]. 成都:电子科技大学,2021.