链滴

处理 k8s 中 java 应用 OOM 时的 dump 文件 (非 preStop)

作者：fish2018

原文链接：https://ld246.com/article/1600340869911

来源网站：链滴

背景：

公司新项目在进行容器化工作，有开发提出他们的java应用存在OOM的情况，通过配置参数 -XX:+HapDumpOnOutOfMemoryError -XX:HeapDumpPath=/dumps/oom可以将jvm信息dump下来但是在K8s中出现OOM会直接重启容器，无法查看/获取dump文件。并且dump的文件通常比较大(发估计2G左右)

解决方案

实践得知OOM时并不会触发preStop，所以需要其他方式解决。

通过jvm参数 -XX:OnOutOfMemoryError=./dump-handler -k \$HOSTNAME -e \$ENV在发生OM时触发工具执行，将dump文件传到oss，并发送钉钉告警(直接把dump文件链接一同发送出来)

工具项目地址：

https://github.com/fish2018/dump-handler.git

jvm关键参数：

```
-Xms1024m
-Xmx1024m
-Xss512k
-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap
-XX:MetaspaceSize=512m
-XX:MaxMetaspaceSize=512m
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=/dumps/oom
-XX:+ExitOnOutOfMemoryError
-XX:OnOutOfMemoryError=./dump-handler -k \$HOSTNAME -e \$ENV
```

注意：

k8s的资源限额limit值要大于jvm参数配置的内存(50~100M)

实施

jenkins pipeline

```
pipeline {
    agent any
    options {
        ansiColor('xterm')
        buildDiscarder(logRotator(daysToKeepStr: '1', numToKeepStr: '3'))
    }
    tools {
        maven 'apache_maven_3.5.0'
        jdk 'jdk_1.8_202'
        git 'git_2.19.1'
        dockerTool 'docker_19.03.12'
    }
    parameters{
        booleanParam(name: 'CHECK_CODE_QUALITY', defaultValue: false, description: '静态代
质量检查【勾选为检查，不勾选为不检查】')
    }
    environment {
        GIT = 'http://172.19.76.212/opu/opu-im-order-web.git'
        IMAGE_GROUP = "opu" //对应harbor镜像分组
        REPLICAS = 1
        TEMPLATE="deployment.yml"
        JVM="""  ,"-Xms1024m","-Xmx1024m","-Xss512k","-XX:+UnlockExperimentalVMOptions"
"-XX:+UseCGroupMemoryLimitForHeap","-XX:MetaspaceSize=512m","-XX:MaxMetaspaceSiz
=512m","-XX:+HeapDumpOnOutOfMemoryError","-XX:+ExitOnOutOfMemoryError","-XX:Hea
DumpPath=/dumps/oom","-XX:OnOutOfMemoryError=./dump-handler -k \$HOSTNAME -e \
ENV" """
        XDIAMOND=" "
        ARGS="""["-jar"${SKYWALKING}${XDIAMOND},"-server"${JVM},"-Dprofile.active=${ENV}"
"-Dspring.profiles.active=${ENV}","-Dserver.port=8888","-Dport=8888","${PROJECT}.jar"]"""
        K8S_NAMESPACE = "${ENV}-${IMAGE_GROUP}"
        PROJECT = sh(script: "echo ${GIT} | awk -F '/' '{print \$NF}' | awk -F '.' '{print \$1}'", return
tdout: true).trim()
        ENV = sh(script: "echo ${JOB_BASE_NAME} | awk -F '-' '{print \$1}'", returnStdout: true).tr
m()
        SKYWALKING_SERVER="ops-system.demo.com:38080"
        SKYWALKING="""  ,"-javaagent:agent/skywalking-agent.jar","-Dskywalking.collector.back
nd_service=${SKYWALKING_SERVER}","-Dskywalking.agent.namespace=${ENV}","-Dskywalkin
.agent.service_name=${ENV}-${PROJECT}" """
        HARBOR_HOST = 'test-devops-harbor.demo.com'
        DOCKER_IMAGE = "${IMAGE_GROUP}/${JOB_BASE_NAME}:${VERSION_VALUE}"
        MAIL_TO = "admin@demo.com"
        CHECK_TAG = sh(script: "echo ${BRANCH_OR_TAG} | awk -F '/' '{if (\$3) print \$3; else pri
t \$1}'", returnStdout: true).trim()  // 分支或tag
        VERSION_VALUE = "${CHECK_TAG}-${TIME}" // 分支或tag
        TIME = sh(script: "date '+%Y%m%d%H%M%S'", returnStdout: true).trim()
    }
```

```
    stages {
        stage ('代码获取') {
            steps {
              echo "\033[46;30m********************************************** 拉取代码开始 *******
**************************************\033[0m"
              deleteDir() // 清理工作目录
              git credentialsId: 'gitlab_username_password_credential', url: "${GIT}"
              sh '[ -n "${CHECK_TAG}" ] &&  git checkout ${CHECK_TAG} ||  { echo -e "切换至指定的
ag的版本，tag：${CHECK_TAG} 不存在或为空，请检查输入的tag!" && exit 111; }'
              echo "\033[46;30m********************************************** 拉取代码结束 *******
**************************************\033[0m"
            }
        }

        stage('代码静态检查') {
            when{
                expression {
                    params.CHECK_CODE_QUALITY == true
                }
            }
            steps {
                echo "\033[46;30m********************************************** 代码静态检查开始 *
*****************************************\033[0m"
                withSonarQubeEnv("sonar_server") {
                    sh "mvn sonar:sonar \
                        -Dsonar.projectKey=sonar-check \
                        -Dsonar.host.url=http://172.19.88.0:9000 \
                        -Dsonar.login=32d06d4d9b19cedb892b3abbafdd2a4dd15170a"
                }
                echo "\033[46;30m********************************************** 代码静态检查结束 *
*****************************************\033[0m"
            }
        }

        stage('检查结果分析') {
            when{
                expression {
                    params.CHECK_CODE_QUALITY == true
                }
            }
            steps {
                echo "\033[46;30m********************************************** 检查结果分析开始 *
*****************************************\033[0m"
                script {
                    timeout(10) {
                        def qg = waitForQualityGate()
                            if (qg.status != 'OK') {
                                echo "\033[0;37;41m ========== 未通过代码质量阈检查，请及时修改!
查失败: ${qg.status}  ==========\033[0m"
                            }
                        }
                }
                echo "\033[46;30m********************************************** 检查结果分析结束 *
*****************************************\033[0m"
```

```
        }
      }

      stage ('代码编译') {
        steps {
          echo "\033[46;30m********************************************** 编译打包开始 ********
**************************************\033[0m"
          sh 'mvn -version'
          sh 'mvn -U clean install -DskipTests'
          echo "\033[46;30m********************************************** 编译打包结束 ********
**************************************\033[0m"
        }
      }

      stage('镜像构建') {
        steps {
          echo "\033[46;30m********************************************** 镜像构建开始 ********
**************************************\033[0m"
          script {
            sh "/usr/bin/cp -f /data/template/docker/Dockerfile ."
            sh "/usr/bin/cp -r -f /data/template/skyagent/agent ."
            sh "/usr/bin/cp -r -f /data/template/preStop/devops ."
            sh "sed -i -e 's#{SW_AGENT_NAME:Your_ApplicationName}#${JOB_BASE_NAME}#g
 agent/config/agent.config"
            sh "sed -i 's/###PROJECT###/${PROJECT}/g' ./Dockerfile"
            sh "docker build -t ${HARBOR_HOST}/${DOCKER_IMAGE} ."
            sh "docker push ${HARBOR_HOST}/${DOCKER_IMAGE}"
            sh "docker rmi ${HARBOR_HOST}/${DOCKER_IMAGE}"
          }
          echo "\033[46;30m********************************************** 镜像构建结束 ********
**************************************\033[0m"
        }
      }

      stage('发布服务至kubernetes集群') {
        steps {
          script {
          echo "\033[46;30m********************************************** 发布服务至kuberne
es集群开始 **********************************************\033[0m"
            sh "cp /data/template/k8s/${TEMPLATE} ${TEMPLATE}"
            sh "sed -i -e 's#{IMAGE_URL}#${HARBOR_HOST}/${DOCKER_IMAGE}#g;s#{ENV}#$
ENV}#g;s#{PROJECT}#${PROJECT}#g;s#{ARGS}#${ARGS}#g;s#{IMAGE_GROUP}#${IMAGE_GR
UP}#g;s#{K8S_NAMESPACE}#${K8S_NAMESPACE}#g;s#{REPLICAS}#${REPLICAS}#g;' ${TEMPL
TE}"
            sh "kubectl --kubeconfig /data/kubecfg/test-cluster cluster-info && kubectl --ku
econfig /data/kubecfg/test-cluster get nodes"
            sh "kubectl --kubeconfig /data/kubecfg/test-cluster apply -f ${TEMPLATE} --nam
space=${K8S_NAMESPACE}"
          echo "\033[46;30m********************************************** 发布服务至kuberne
es集群结束 **********************************************\033[0m"
          }
        }
      }
    }
```

```
}
```

Dockerfile

```
FROM test-devops-harbor.demo.com/devops/jdk-8u202-baseimage:2.0.0_ubuntu
USER root
RUN ["mkdir","/im-svc"]
ADD ./target/###PROJECT###.jar /im-svc
ADD agent /im-svc/agent
ADD dump-handler /im-svc/dump-handler
RUN ["chmod","755","/im-svc/###PROJECT###.jar"]
ENV arg1 ""
WORKDIR "/im-svc"
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-Duser.timezone=Asia/Shang
ai","-jar","###PROJECT###.jar","$arg1"]
```

deploy.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {PROJECT}
  namespace: {K8S_NAMESPACE}
  labels:
    app: {PROJECT}
spec:
  replicas: {REPLICAS}
  selector:
    matchLabels:
      app: {PROJECT}
  template:
    metadata:
      labels:
        app: {PROJECT}
    spec:
      imagePullSecrets:
        - name: harbor-registry
      terminationGracePeriodSeconds: 90
      volumes:
      - name: heap-dumps
        emptyDir: {}
      containers:
      - name: {PROJECT}
        image: {IMAGE_URL}
        imagePullPolicy: Always
        volumeMounts:
        - name: heap-dumps
          mountPath: /dumps
        command: ["java"]
        args: {ARGS}
        ports:
        - containerPort: 8888
        env:
```

```
        - name: ENV
          value: {ENV}
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "1200Mi"
            cpu: "500m"
        readinessProbe:
          httpGet:
            path: /actuator/health
            port: 8888
            scheme: HTTP
          initialDelaySeconds: 10
          timeoutSeconds: 2
          periodSeconds: 10
---
apiVersion: v1
kind: Service
metadata:
  name: {PROJECT}
  namespace: {K8S_NAMESPACE}
spec:
  type: NodePort
  ports:
  - port: 8888
    protocol: TCP
    targetPort: 8888
  selector:
    app: {PROJECT}
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: {ENV}-{PROJECT}
  namespace: {K8S_NAMESPACE}
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: {ENV}-{PROJECT}.demo.cn
    http:
      paths:
      - backend:
          serviceName: {PROJECT}
          servicePort: 8888
        path: /
        pathType: ImplementationSpecific
```

验证

演示OOM demo项目：

https://github.com/fish2018/eureka-client-demo.git

核心代码

```
package com.elvin.example.eurekaclientdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.List;

@SpringBootApplication
@EnableEurekaClient
@RestController
public class EurekaClientDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaClientDemoApplication.class, args);
    }

    @RequestMapping("oom")
    public String helloDemoEurekaClientOOM(){
        List<Object> demoList = new ArrayList<Object>();
        while(true) {
            demoList.add(new Object());
        }
    }

}
```

手动验证：

进入容器/dumps/目录，创建文件

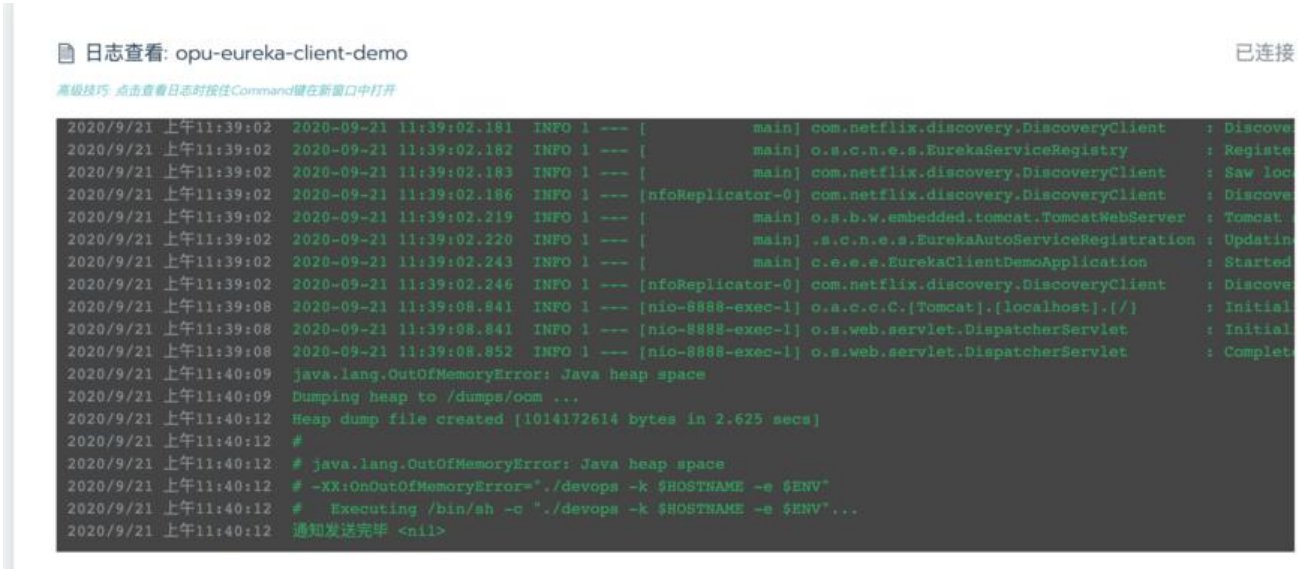
```
cd /dumps/
dd if=/dev/zero of=oom bs=1M count=20
```

手动执行工具

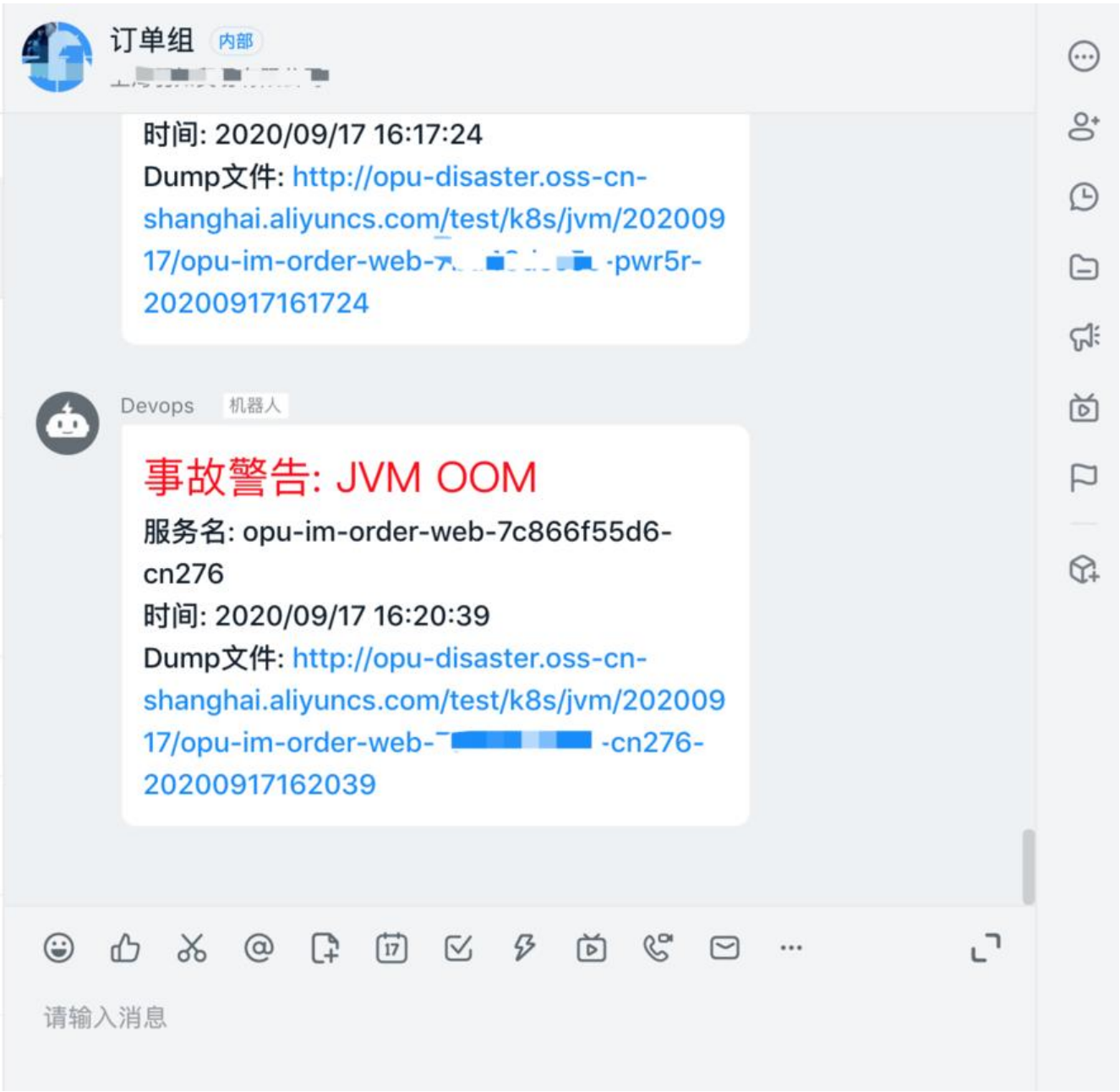
```
/im-svc/dump-handler -k ops-demo -e test
```

实战验证：

调用demo的/oom接口，让应用发生OOM



自动发送到钉钉对应群



原文链接：处理 k8s 中 java 应用 OOM 时的 dump 文件 (非 preStop)

自动上传至oss对应bucket

原文链接：处理 k8s 中 java 应用 OOM 时的 dump 文件 (非 preStop)