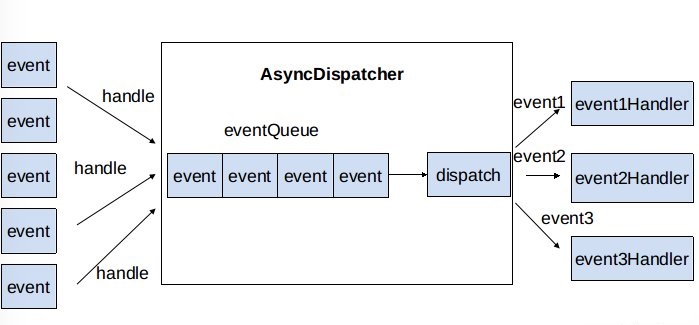
# Hadoop AsyncDispatcher

Hadoop YARN采用了基于事件驱动的并发模型，该模型能够增加并发性，从而提高系统的整体性能。在该模型中YARN将各种处理逻辑抽象成事件和对应的事件调度器，每类事件的处理过程分成多个步骤，用有限状态机表示，事件处理模型如下图所示：



处理大致过程：

1. 处理请求作为事件进入系统
2. 中央异步调度器（AsyncDispatcher）负责传递给相应事件调度器（Event Handler）
3. 事件调度器对事件进行处理，也可以交给状态机的事件处理器或者其他事件调度器

下面介绍使用的Example

# 定义事件类型和事件

//定义事件类型

*public enum AgentEventType {*

*REGISTER,*

*UNREGISTER,*

*EXPIRED,*

*HEARTBEAT,*

*JOBLAUNCH,*

*JOBKILL*

*}*

//定义事件

*public class AgentEvent extends AbstractEvent<AgentEventType> {*

*private Agent agent;*

*public AgentEvent(AgentEventType agentEventType, Agent agent) {*

*super(agentEventType);*

*this.agent = agent;*

*}*

*public Agent getAgent() {*

*return agent;*

*}*

*}*

*public class JobEvent extends AgentEvent {*

*private int jobId;*

*public JobEvent(AgentEventType agentEventType, Agent agent , int jobId) {*

*super(agentEventType, agent);*

*this.jobId = jobId;*

*}*

*public int getJobId() {*

*return jobId;*

*}*

*}*

# 定义事件处理器

*//Agent Event事件处理器*

*public class Agent implements EventHandler<AgentEvent> {*

*private final int id;*

*private Set<Long> runningJobs = new HashSet<Long>();*

*public Agent(int id) {*

*this.id = id;*

*}*

*......*

*public void handle(AgentEvent agentEvent) {*

*switch (agentEvent.getType()) {*

*case REGISTER:*

*System.out.println("Receive Register event in agent");*

*break;*

*case UNREGISTER:*

*break;*

*case EXPIRED:*

*break;*

*case JOBLAUNCH:*

*JobEvent jobEvent = (JobEvent)agentEvent;*

*System.out.println("agent:" + agentEvent.getAgent()*

*+" receive JOBLAUNCH event, jobId :" + jobEvent.getJobId());*

*break;*

*case JOBKILL:*

*break;*

*}*

*}*

*}*

# 定义事件处理服务

*public class AgentManager extends AbstractService implements EventHandler<AgentEvent> {*

*private static final Log LOG = LogFactory.getLog(AgentManager.class);*

*private int expireInterval;*

*private int monitorInterval;*

*private Thread checkerThread;*

*private boolean stopped = false;*

*private Map<Agent, Long> running = new HashMap<Agent, Long>();*

*public AgentManager() {*

*super("AgentManager");*

*}*

*protected void serviceInit(Configuration conf) throws Exception {*

*super.serviceInit(conf);*

*int expireIntvl = conf.getInt("com.baidu.dscheduler.agent.expire.ms",*

*1000 \* 60 \* 10);*

*//need some sanity check*

*this.expireInterval = expireIntvl;*

*this.monitorInterval = expireInterval / 3;*

*this.checkerThread = new Thread(new PingChecker());*

*this.checkerThread.setName("Ping Checker");*

*}*

*protected void doStart() throws Exception {*

*super.serviceStart();*

*this.checkerThread.start();*

*}*

*protected void doStop() throws Exception {*

*stopped = true;*

*super.serviceStop();*

*checkerThread.interrupt();*

*}*

*public void handle(AgentEvent agentEvent) {*

*switch (agentEvent.getType()) {*

*case REGISTER:*

*System.out.println("receive register request in agent manager.");*

*if (!running.containsKey(agentEvent.getAgent())) {*

*running.putIfAbsent(agentEvent.getAgent(), System.currentTimeMillis());*

*agentEvent.getAgent().handle(agentEvent);*

*}*

*break;*

*case UNREGISTER:*

*if (running.containsKey(agentEvent.getAgent())) {*

*agentEvent.getAgent().handle(agentEvent);*

*running.remove(agentEvent.getAgent());*

*}*

*break;*

*case EXPIRED:*

*break;*

*case HEARTBEAT:*

*if (running.containsKey(agentEvent.getAgent())) {*

*Agent agent = agentEvent.getAgent();*

*running.put(agent, System.currentTimeMillis());*

*agentEvent.getAgent().handle(agentEvent);*

*}*

*break;*

*case JOBLAUNCH:*

*case JOBKILL:*

*JobEvent jobEvent = (JobEvent)agentEvent;*

*System.out.println(" agent master receive "*

*+" receive "+ jobEvent.getType()*

*+", agent: "+ agentEvent.getAgent()*

*+ ", jobId :" + jobEvent.getJobId());*

*if (running.containsKey(agentEvent.getAgent())) {*

*Agent agent = agentEvent.getAgent();*

*running.put(agent, System.currentTimeMillis());*

*agentEvent.getAgent().handle(jobEvent);*

*}*

*}*

*}*

*private class PingChecker implements Runnable {*

*private PingChecker() {*

*}*

*public void run() {*

*while (!AgentManager.this.stopped && !Thread.currentThread().isInterrupted()) {*

*synchronized (AgentManager.this) {*

*Iterator<Map.Entry<Agent, Long>> iterator = AgentManager.this.running.entrySet().iterator();*

*long currentTime = System.currentTimeMillis();*

*while (true) {*

*if (!iterator.hasNext()) {*

*break;*

*}*

*Map.Entry<Agent, Long> entry = (Map.Entry) iterator.next();*

*if (currentTime > ((Long) entry.getValue()).longValue() + (long) AgentManager.this.expireInterval) {*

*iterator.remove();*

*//AgentManager.this.expire(entry.getKey());*

*AgentManager.LOG.info("Expired:" + entry.getKey().toString() + " Timed out after " + AgentManager.this.expireInterval / 1000 + " secs");*

*}*

*}*

*}*

*try {*

*Thread.sleep((long) AgentManager.this.monitorInterval);*

*continue;*

*} catch (InterruptedException e) {*

*AgentManager.this.LOG.info( AgentManager.this.getName() + " thread interrupted");*

*}*

*}*

*}*

*}*

*}*

# 测试程序

*AgentManager agentManager = new AgentManager();*

*Configuration conf = new Configuration();*

*AsyncDispatcher dispatcher = new AsyncDispatcher();*

*dispatcher.init(conf);*

*//注册事件及事件处理器*

*dispatcher.register(AgentEventType.class, agentManager );*

*dispatcher.start();*

*//发送事件*

*dispatcher.getEventHandler().handle(*

*new AgentEvent(AgentEventType.REGISTER, new Agent(1)));*

*System.out.println("send register request in asyncDispatcherTest1");*

*//发送事件*

*dispatcher.getEventHandler().handle(*

*new JobEvent(AgentEventType.JOBLAUNCH, new Agent(1), 2));*

*System.out.println("send job launch request in asyncDispatcherTest1");*

*try {*

*TimeUnit.SECONDS.sleep(3);*

*} catch (InterruptedException e) {*

*e.printStackTrace();*

*}*

*dispatcher.stop();*

输出如下：

*send register request in asyncDispatcherTest1*

*send job launch request in asyncDispatcherTest1*

*receive register request in agent manager.*

*Receive Register event in agent*

*agent master receive receive JOBLAUNCH, agent: Agent: 1, jobId :2*

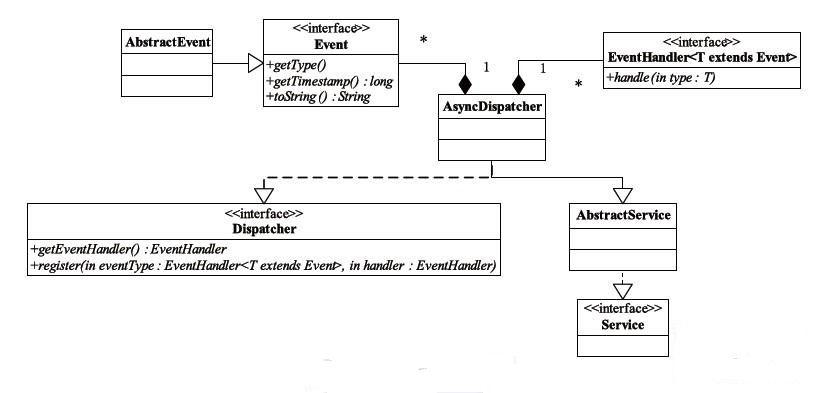
*agent:Agent: 1 receive JOBLAUNCH event, jobId :2*

https://blog.csdn.net/houzhizhen/article/details/79425946

http://www.cnblogs.com/smartloli/p/4669968.html

# AsyncDispatcher源码分析

事件与事件处理器类的关系如下图所示：



当使用YARN事件库时，要先定义一个中央异步调度器AsyncDispatcher，负责事件的处理与转发，然后根据业务需求定义一系列事件Event与事件处理器EventHandler，并注册到中央处理器中以实现事件统一管理和调度。在AsyncDispatcher中，核心的成员变量为：

* eventQueue: BlockingQueue<Event>，阻塞队列，存放全部等待调度处理的事件
* eventDispatchers: Map<Class<? extends Enum>, EventHandler>，存储事件类型枚举类Enum与对应事件处理器EventHandler实例的映射关系，其存放着事件的分发逻辑
* handlerInstance: GenericEventHandler，接受到事件后由handlerInstance将事件放入队列
* eventHandlerThread，单线程，从事件队列中取出时间，并从eventDispatcher中查找事件处理器，然后转交给EventHandler进行事件的处理

下面介绍具体的执行过程。

1. Event的注册，AsyncDispatcher初始化后向其注册事件，示例中执行如下：

*//注册事件及事件处理器*

*dispatcher.register(AgentEventType.class, agentManager );*

注册过程即将其放入eventDispatcher中

*eventDispatchers.put(eventType, handler);*

注册后即可接受事件并处理

1. Event的处理逻辑，事件的调用处理如下：

*//发送事件*

*dispatcher.getEventHandler().handle(*

*new JobEvent(AgentEventType.JOBLAUNCH, new Agent(1), 2));*

* 由GenericEventHandler将事件加入到EventQueue中

*class GenericEventHandler implements EventHandler<Event> {*

*public void handle(Event event) {*

*...*

*eventQueue.put(event);*

*...*

*};*

*}*

* eventHandlingThread，不停的遍历queue，从中取出事件交给对应的EventHandler，执行如下：

*public void run() {*

*while (!stopped && !Thread.currentThread().isInterrupted()) {*

*...*

*Event event;*

*try {*

*event = eventQueue.take();*

*} .....*

*if (event != null) {*

*dispatch(event);*

*}*

*}}*

在dispatch(event)是事件处理的核心逻辑，

*protected void dispatch(Event event) {*

*...*

*//获取事件的类型*

*Class<? extends Enum> type = event.getType().getDeclaringClass();*

*try{*

*//获取该类型事件对应的EventHandler*

*EventHandler handler = eventDispatchers.get(type);*

*if(handler != null) {*

*//由EventHandler进行处理*

*handler.handle(event);*

*}....}*

* EventHandler进行Event的处理，其继承EventHandler<Event>接口，核心处理逻辑定义子handle方法中，如上例中对AgentEvent的EventHandler

*public class Agent implements EventHandler<AgentEvent> {*

*public void handle(AgentEvent agentEvent) {*

*switch (agentEvent.getType()) {*

*case REGISTER:*

*System.out.println("Receive Register event in agent");*

*break;*

*.....}*

至此AsyncDispatcher的执行逻辑已经分析完毕。