项目申请书

项目名称
增强Volcano Agent以支持Cgroup V2与Systemd

申请人
陈时

日期
2025.06.06

邮箱
chenshiabcd@163.com

需求分析

1. 支持 Cgroup v2 的 Volcano Agent: 提供能够与 Cgroup v2 系统正常工作的 Volcano Agent 版本。
2. 支持 Systemd 的 Volcano Agent: 提供能够与 Systemd 系统良好集成的 Volcano Agent 版本。
3. Kubelet cgroup-driver 兼容性: 实现与 Kubelet 不同 cgroup-driver 配置（包括 systemd 和 cgroupfs）的兼容性，确保 Volcano Agent 在不同配置下都能正确运行和管理资源。
4. 功能测试与验证: 提供充分的单元测试、集成测试和端到端测试，验证 Volcano Agent 在 Cgroup v2 和 Systemd 环境下的核心功能（统一调度、动态资源超卖、OoS 保障等）的正确性，并验证与不同 Kubelet cgroup-driver 的兼容性。
5. 配置和部署文档更新: 更新 Volcano Agent 的配置和部署文档，详细说明如何在 Cgroup v2 和 Systemd 环境下部署和配置 Agent，以及与 Kubelet cgroup-driver 相关的配置说明。
6. 社区集成与发布: 将开发成果集成到 Volcano 社区主干分支，并按照社区流程进行版本发布。

支持Cgroup v2的Volcano Agent

技术要求:

- 熟悉linux cgroup v1和v2的架构差异 理解统一层次结构概念
- 熟悉现有agent中对cgroup和systemd的路径管理
- 理解容器资源隔离机制和OCI规范

开发路径:

- 当前CgroupManager pkg/agent/utils/cgroup/cgroup.go 定义了基础的cgroup路径管理功能，但仅支持cgroup v1
- 扩展CgroupManager接口以支持cgroup v2
- 更新cgroup v2对应的控制文件名
- 添加cgroup版本检测方法 根据版本选择相应的路径和文件操作

支持Systemd的Volcano Agent

技术要求:

- 熟悉systemd架构 了解systemd slice、scope、service概念以及层次结构
- 理解systemd的服务生命周期和依赖管理
- 了解golang使用DBus与systemd交互

开发路径:

- 当前代码已经包含了基础的systemd支持，主要为将cgroup名称转换为systemd格式的路径-cgroup.go ToSystemd()
- 扩展systemd slice检测 支持完整的cgroup路径计算
- 扩展使用DBus接口与systemd进行交互 获取服务状态与资源使用情况 监听服务状态变化 保证Agent的资源状态与kubelet同步

Kubelet cgroup-driver 兼容性

技术要求:

- 了解kubelet对cgroup资源管理的逻辑和配置
- 熟悉目前Volcano Agent中与kubelet有交互的部分
- 熟悉kubelet使用不同cgroup-driver时cgroup文件路径和资源配置信息的差异

开发路径:

- 现有方法中获得cgroup路径依赖于kubelet使用的cgroup驱动类型
- 扩展从kubelet的配置文件/kubelet进程中动态读取cgroup-driver配置 增强现有的cgroup路径处理函数
- 增加cgroup路径验证功能 agent启动时的验证和初始化功能
- 增加CgroupManager的错误处理和恢复机制

功能测试与验证

开发路径:

- **单元测试** 扩展现有cgroup测试框架 pkg/agent/utils/utils_test.go；添加针对Cgroup v2和systemd的测试用例；验证CgroupManagerImpl在不同驱动下的路径解析
- **集成测试** 利用现有的测试框架，测试用例覆盖 cgroupfs + Cgroup v1/v2 和 systemd + Cgroup v1/v2 ； 验证Agent的资源监控和超卖功能
- **E2E测试** 在现有E2E测试框架中添加新的测试套件验证Agent的功能；在不同kubelet cgroup-driver配置下进行验证

开发方案

代码结构

CgroupManager接口在 pkg/agent/utils/cgroup/cgroup.go 主要负责：获取cgroup路径；支持cgroupfs和systemd驱动

主要使用CgroupManager的组件：

- CPU QoS管理 pkg/agent/events/handlers/cpuqos
- Memory QoS 管理 pkg/agent/events/handlers/memoryqos
- CPU Burst 管理 pkg/agent/events/handlers/cpuburst
- 资源管理 pkg/agent/events/handlers/resources

添加cgroup v2支持

cgroup v2支持

在pkg/agent/utils/cgroup下添加cgroup v2的支持，实现cgroup v2路径管理

```
type CgroupManager interface {
    GetRootCgroupPath(cgroupSubsystem CgroupSubsystem) (string, error)
    GetQoSCgroupPath(qos corev1.PodQOSClass, cgroupSubsystem CgroupSubsystem)
(string, error)
    GetPodCgroupPath(qos corev1.PodQOSClass, cgroupSubsystem CgroupSubsystem,
podUID types.UID) (string, error)
}
```

添加cgroup版本检测

实现动态的cgroup版本检测

```
// GetCgroupVersion returns the cgroup version of the system
func GetCgroupVersion() (CgroupVersion, error) {

    if isCgroup2UnifiedMode() {
        return CgroupVersion2, nil
    }

    if isCgroup1Mode() {
        return CgroupVersion1, nil
    }

    return CgroupVersionUnknown, fmt.Errorf("unknown cgroup version")
}
```

修改CgroupManager的工厂函数

修改CgroupManager工厂函数，根据cgroup版本创建相应的manager

```
func NewCgroupManager(cgroupDriver, cgroupRoot, kubeCgroupRoot string)
CgroupManager {
    version := GetCgroupVersion()
```

```
    switch version {
    case CgroupVersion2:
        return NewCgroupV2Manager(cgroupDriver, cgroupRoot, kubeCgroupRoot)
    case CgroupVersion1:
        return &CgroupManagerImpl{
            cgroupDriver:   cgroupDriver,
            cgroupRoot:     cgroupRoot,
            kubeCgroupRoot: kubeCgroupRoot,
        }
    }
}
```

添加Systemd支持

设计SystemdManager

设计SystemManager通过DBus与systemd交互

```
type SystemManager {
    slice Slice
    conn *systemd.Conn
}

// 支持slice的创建、删除、属性管理等
func (sm *SystemManager) CreateSlice()
func (sm *SystemManager) DeleteSlice()
func (sm *SystemdManager) GetSliceProperties()
func (sm *SystemdManager) GetSliceStatus()
```

Kubelet cgroup-driver兼容性

从kubelet中读取cgroup-driver版本

从kubelet配置中读取cgroup驱动类型

```
func getCgroupDriverFromKubelet() string {
    kubeletConfigFile := "kube_config_path"

    config := parserConfig(kubeletConfigFile)

    return config.CgroupDriver
}
```

修改cmd/agent/app/agent.go 添加cgroup driver动态读取配置

```
func Run(ctx context.Context, opts *options.VolcanoAgentOptions) error {
    klog.InfoS("Start volcano agent")
    conf, err := NewConfiguration(opts)
    if err != nil {
        return fmt.Errorf("failed to create volcano-agent configuration: %v",
err)
    }
```

```
    // TODO: get cgroup driver dynamically
    cgroupDriver := getCgroupDriverFromKubelet()

    ...

    cgroupManager := cgroup.NewCgroupManager(cgroupDriver, path.Join(sfsFsPath,
"cgroup"), conf.GenericConfiguration.KubeCgroupRoot)
}
```

单元测试&集成测试&E2E测试

单元测试

- 测试cgroup v1/v2 manager
- 测试获取cgroup版本
- 测试从kubelet获取cgroup-driver
- 测试与systemd交互
- 测试与不同cgroup-driver的kubelet交互

集成测试

- cgroup版本兼容性测试
- cgroup-driver兼容性测试
- 资源管理功能测试-CPU内存资源限制；QoS级别资源隔离；CPU Burst功能

E2E测试

- 基础组件功能测试-Agent正常启动；与kubelet交互；配置更新与reload
- Pod生命周期
- 资源超卖场景
- QoS测试-高优先级Pod资源保障与抢占；低优先级Pod资源限制；多租户隔离
- 故障恢复-Agent异常重启
- 兼容性测试：不同kubernetes版本；不同runtime；

时间规划

cgroup v2支持

- 第1周：cgroup v1/v2测试环境 CgroupManager接口扩展

  Day1-3：阅读NPU调度插件和现有插件源码 熟悉Volcano代码架构 搭建开发环境

  Day4-7：实现cgroup版本检测 扩展CgroupManager接口
- 第2周：CgroupV2Manager实现

  Day1-5：实现统一的cgroup路径管理 & 实现cgroup v2文件控制

  Day4-7：编写单元测试

systemd支持

- 第3周：SystemManager开发

  Day1-3：实现DBus通信

  Day4-7：实现Slice管理和监控

- 第4周：systemd集成

  Day1-3：集成systemd slice创建和管理

  Day4-7：添加状态监控 & 编写单元测试

kubelet cgroup-driver兼容性

- 第5周：Cgroup Driver 适配

  Day1-3：实现kubelet配置读取 & driver切换

  Day4-7：cgroup路径计算兼容性处理

- 第6周：资源同步机制

  Day1-3：实现与kubelet的资源同步

  Day4-7：配置验证 & 编写单元测试

全面测试&编写文档

- 第7周：单元测试 & 集成测试

  Day1-2：完善单元测试

  Day3-7：编写集成测试

- 第8周：E2E测试&用户文档

  Day1-4：编写E2E测试

  Day5-7：编写用户文档和开发者文档

volcano项目结构

基本概念CRD

Queue 定义资源分配策略和优先级，根据需要动态调整资源分配，用于管理和组织作业Job。定义了资源池以及如何将这些资源分配给不同的任务。

PodGroup 一组需要一起调度的Pod集合。用于实现Gang Scheduling-全调度或无调度。用于需要协同工作的分布式应用，这些应用需要多个Pod同时启动才能正常工作，否则会导致资源浪费或死锁。

Volcano Job Volcano定义的Job资源类型。与Kubernetes的Job相比，提供了更多的配置选项。一个Job可以包含多个Task，每个Task可以有多个Pod副本；每个VCJob属于一个Queue

Queue PodGroup VCJob关系

当创建Volcano Job时，Job Controller会为该Job创建一个对应的PodGroup，并且PodGroup的Queue属性会设置为Job的Queue属性。

在scheduler cache中，PodGroup会被关联到一个Queue，如果PodGroup没有指定Queue，则使用默认队列。

Queue Controller维护每个Queue中的所有PodGroup的状态统计信息。

- Volcano Job和PodGroup是一对一的关系-Job Controller负责为每个Job创建并维护一个PodGroup
- 每个PodGroup必须属于一个Queue，但一个Queue可以包含多个PodGroup。Queue Controller会跟踪Queue中每个PodGroup的状态
- 每个Volcano Job必须属于一个Queue，但一个Queue可以包含多个Job。Job在创建时指定其所属的Queue

Volcano Job和PodGroup关系

Volcano Job关注批处理作业的定义和生命周期，包括任务定义、依赖关系和重试策略等

PodGroup关注Gang Scheduling的实现，确保一组Pod能够一起被调度

- PodGroup可以独立使用：PodGroup不一定通过VC Job创建，其它系统和用户可以直接创建PodGroup来实现Gang Scheduling
- scheduler只关注PodGroup，而不需要管理VC Job

volcano组件&调度流程

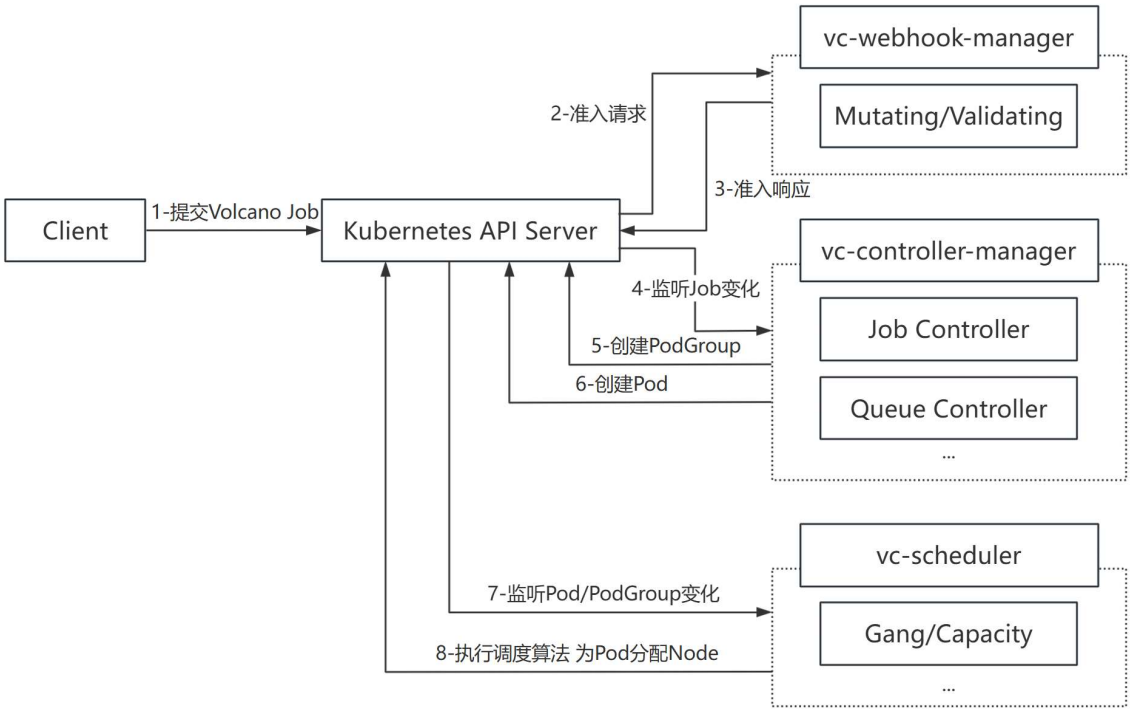
vc-scheduler 负责deployment或job的调度，扩展了Kubernetes原生调度器的功能，专为批处理工作负载提供高级调度能力

vc-controller-manager 负责管理和维护Volcano自定义资源的生命周期。它是一个控制器集合，包含多个子控制器，每个子控制器负责不同类型资源的管理。

vc-webhook-manager 负责验证提交资源合法性和处理资源变更

1. 用户创建新Job，由API Server向vc-webhook-manager准入请求，vc-webhook-manager负责验证准入请求并返回响应；
2. vc-controller-manager监听到Job等资源变化会开始执行相应操作，如创建Job对应的PodGroup/Pod；
3. vc-scheduler监听到Pod/PodGroup的变化，根据注册插件的相应调度操作，为Pod分配Node

Volcano各组件间不相互通信



vc-scheduler

执行流程

初始化调度器后，在调度器的主循环中按照配置顺序执行Actions，每个Actions都会接收一个Session对象，Session包含当前集群的状态和由Plugins注册的各种函数

scheduler/plugin

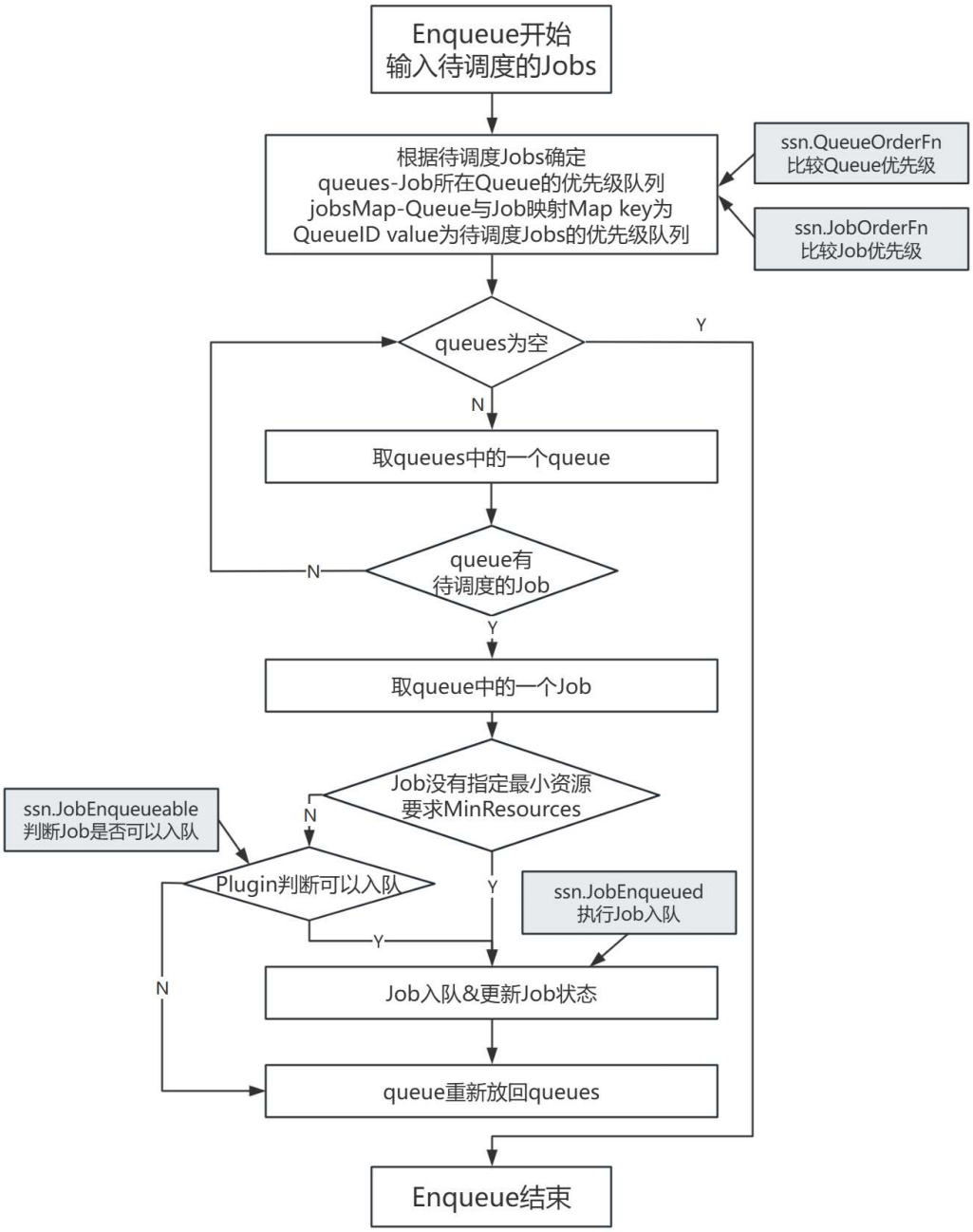
调度器采用插件架构，各个插件负责不同的调度功能，通过Session对象进行交互和协作

- **插件注册**：所有Plugins都需要实现Plugin接口，并通过调度器的插件注册机制进行注册
- **分层结构**：插件被组织在不同的层级Tiers中，高层级的插件优先于低层级的插件执行
- **共享Session**：所有Plugins共享同一个Session对象，通过这个对象注册回调函数并访问共同的调度状态

Action

- **Enqueue-入队**

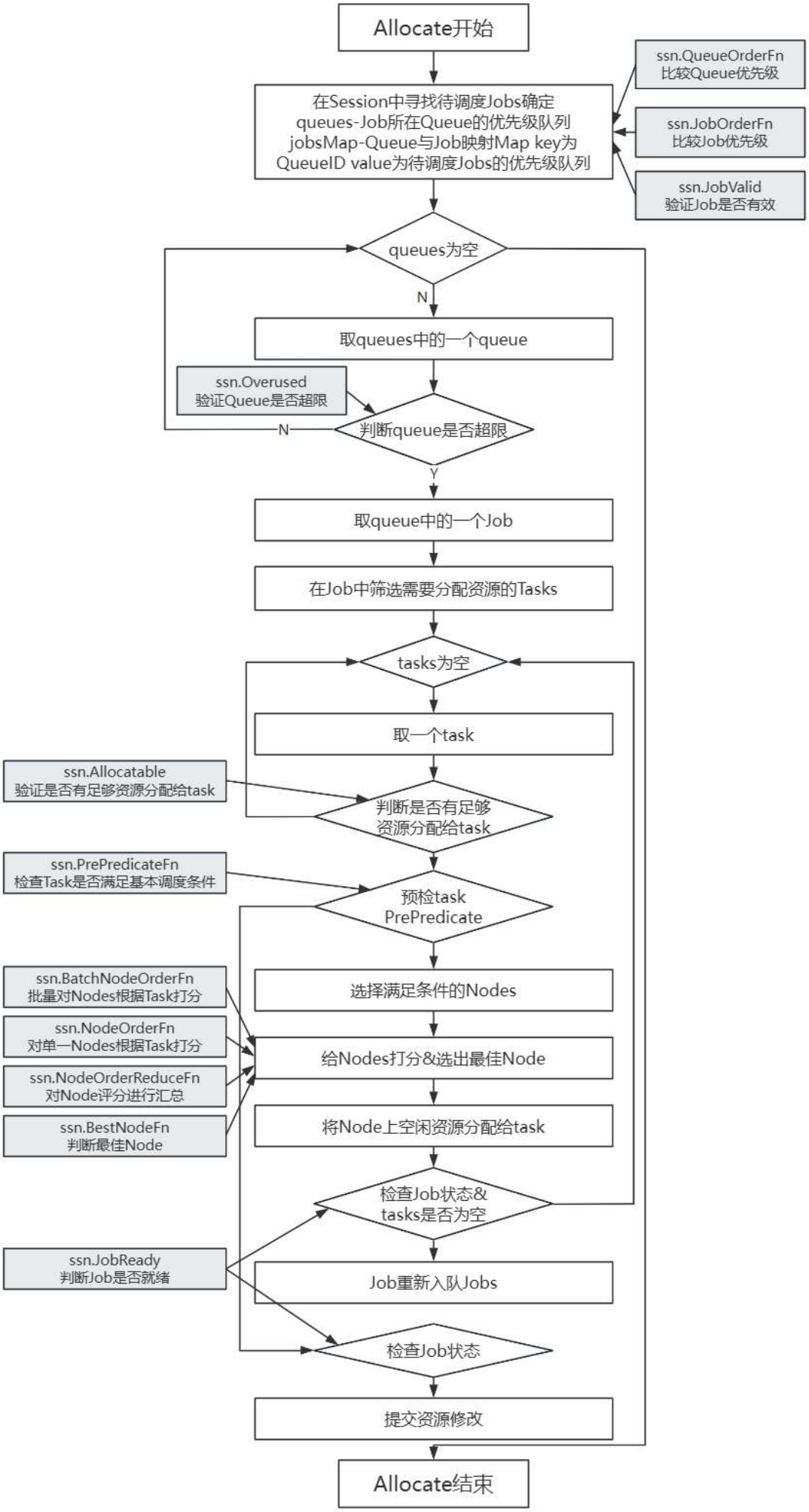
  确定哪些Job准备好被调度，基于资源可用性和队列策略将Job从Pending状态移动到Inqueue状态



- **Allocate-分配**

  负责将作业分配给节点，根据优先级选择作业，为他们找到合适的节点

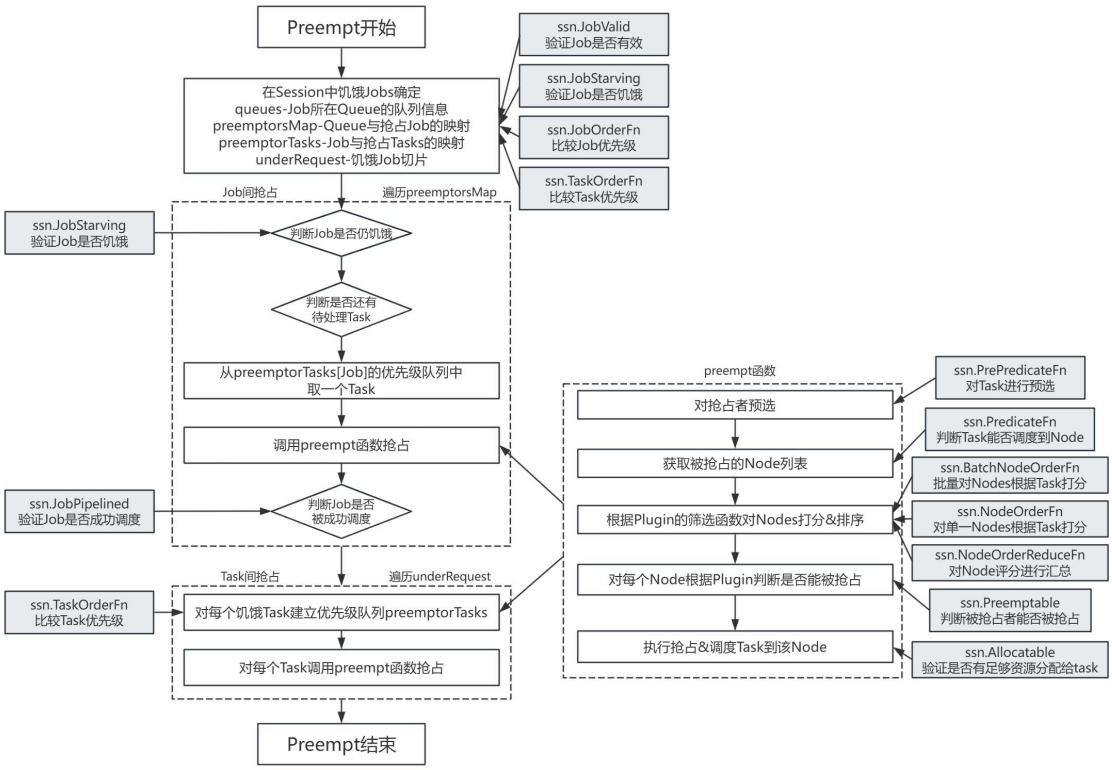
  主要流程：收集队列和作业信息pickUpQueuesAndJobs；为Job分配资源allocateResources


Allocate开始
在Session中寻找待调度Jobs确定
queues-Job所在Queue的优先级队列
jobsMap-Queue与Job映射Map key为
QueueID value为待调度Jobs的优先级队列
ssn.QueueOrderFn
比较Queue优先级
ssn.JobOrderFn
比较Job优先级
ssn.JobValid
验证Job是否有效
queues为空
N
取queues中的一个queue
ssn.Overused
验证Queue是否超限
判断queue是否超限
N
Y
取queue中的一个Job
在Job中筛选需要分配资源的Tasks
tasks为空
取一个task
ssn.Allocatable
验证是否有足够资源分配给task
判断是否有足够
资源分配给task
ssn.PrePredicateFn
检查Task是否满足基本调度条件
预检task
PrePredicate
ssn.BatchNodeOrderFn
批量对Nodes根据Task打分
选择满足条件的Nodes
ssn.NodeOrderFn
对单一Nodes根据Task打分
给Nodes打分&选出最佳Node
ssn.NodeOrderReduceFn
对Node评分进行汇总
将Node上空闲资源分配给task
ssn.BestNodeFn
判断最佳Node
检查Job状态&
tasks是否为空
ssn.JobReady
判断Job是否就绪
Job重新入队Jobs
检查Job状态
提交资源修改
Allocate结束

- **Preempt-抢占**

当资源不足时，允许高优先级任务抢占低优先级任务

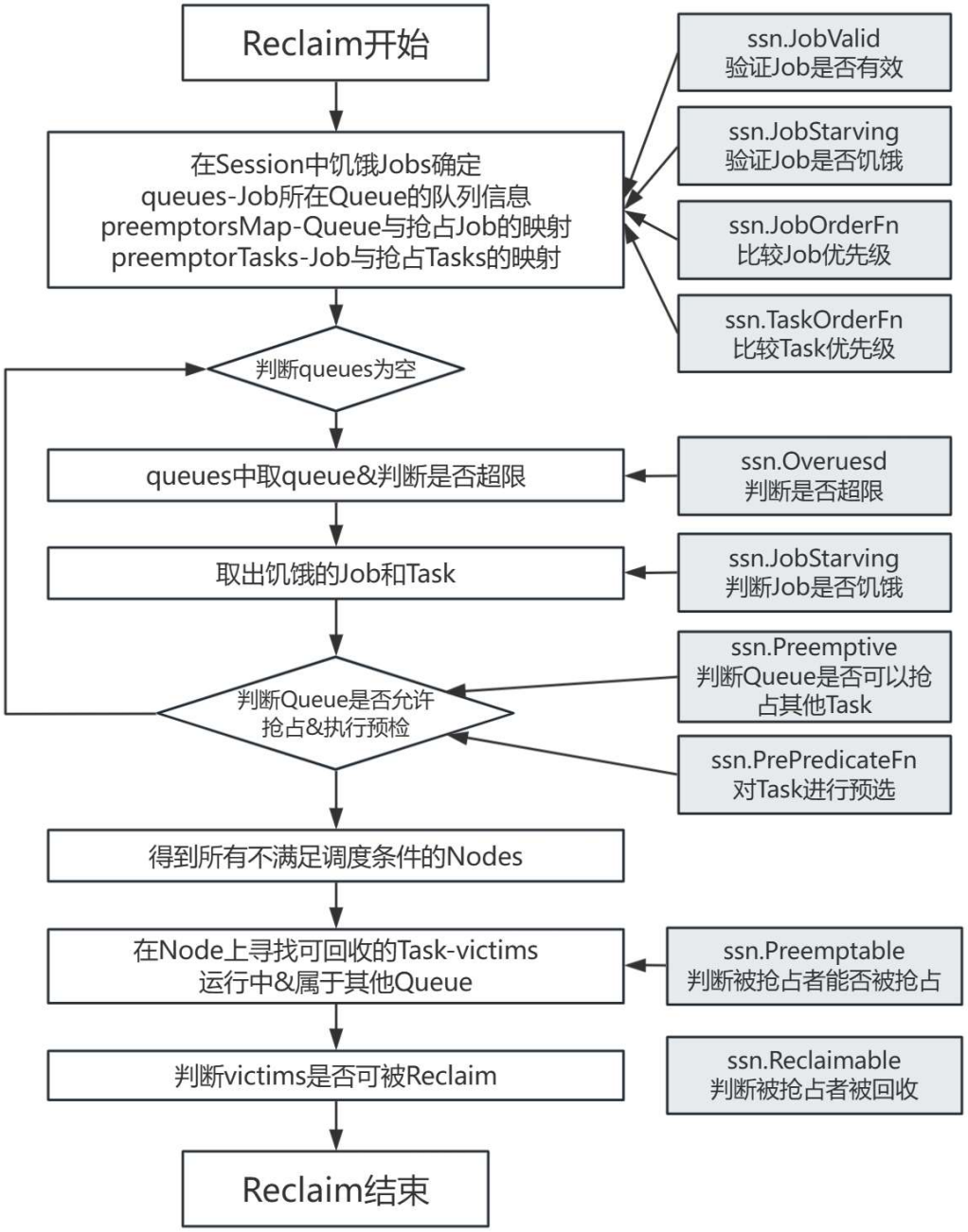
主要流程：识别解饿作业starving Jobs；对每个Queue进行处理；队列内抢占；作业内抢占



- **Reclaim-回收**

  当作业的资源被过度使用时，允许作业从其他队列请求资源

  工作流程：获取所有Queue并按优先级排序；对每个Queue检查是否Overused，在没有Overused的Queue中找到饥饿Job；对每个饥饿Job中待处理的Task，尝试在Cluster中找到可以回收资源的Node；在Node上寻找可以被回收的来自其他Queue的Task；驱逐这些被回收的Task，并将饥饿Task调度到该Node上



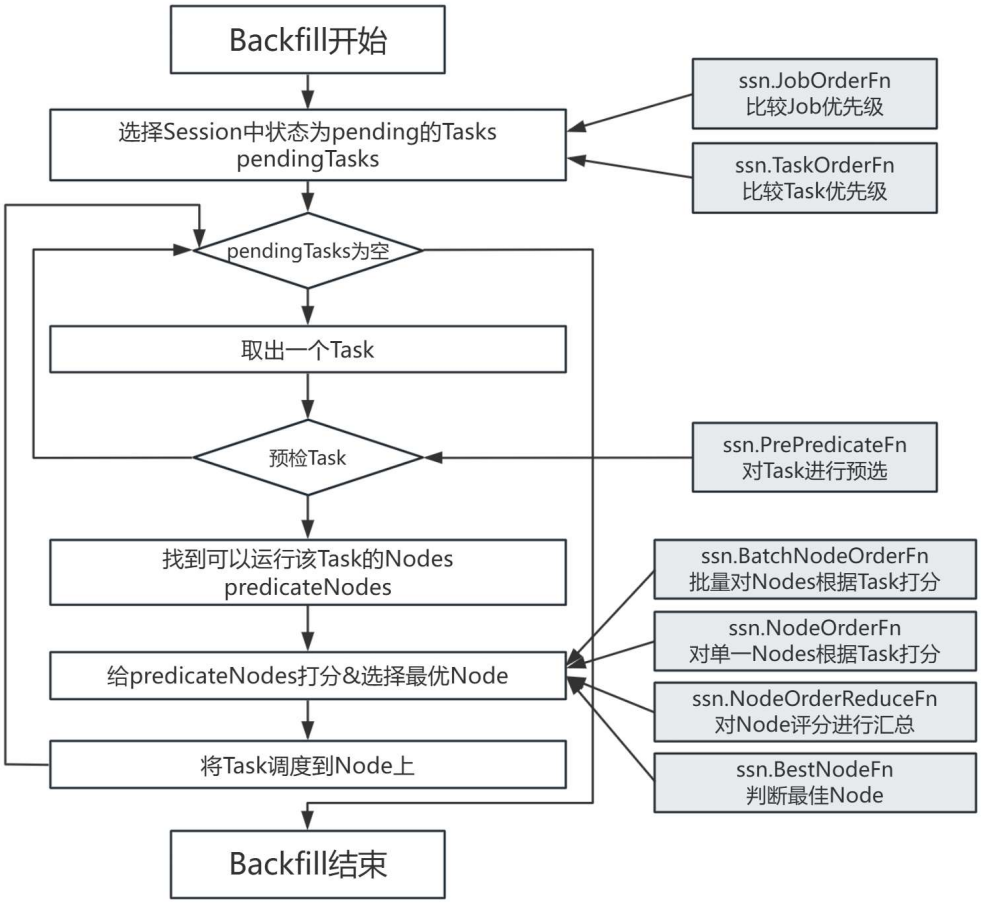
- **Backfill-回填**

  调度BestEffort的任务，以利用集群中的空闲资源

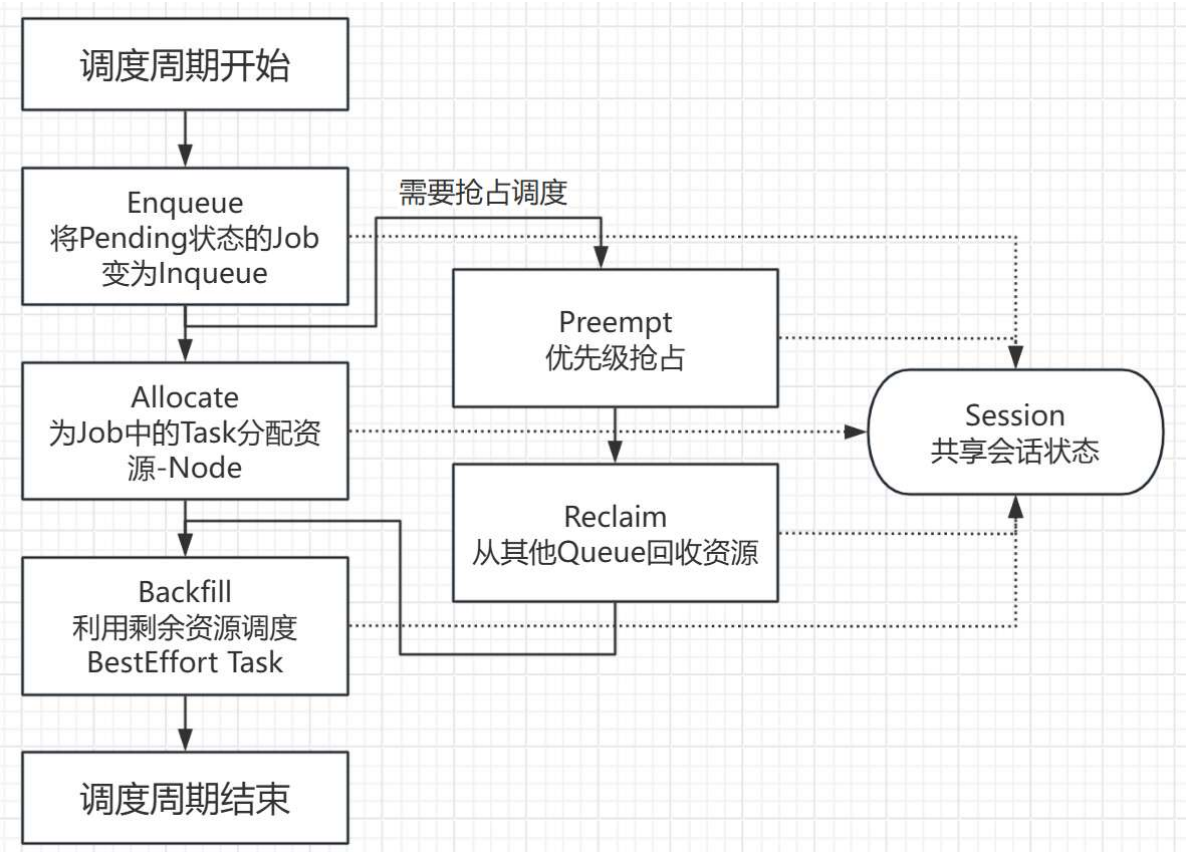
  主要流程：收集处于Pending或Pipeline状态且标记为BestEffort的Task；按照优先级队列对其进行排序；对每个Task执行预选，找到可以运行该Task的Node；将Task调度到最优Node上

  BestEffot

  - TaskInfo的一个字段 “尽力而为” 当创建新Task时，如果没有明确资源请求，它会被标记为BestEffort
  - Allocate过程中会跳过BestEffort任务，因为它们被认为是可延迟调度的
  - Backfill阶段会专门处理BestEffort任务，利用Cluster剩余资源来调度这些Task
  - Preempt过程中不能抢占非BestEffort任务，保证了资源需求明确的Task不会被资源需求不明确的Task抢占



所有调度动作Action都实现了Action接口pkg/scheduler/framework/interface.go



Plugins

Plugins注册和清除等管理代码在pkg/scheduler/framework/plugins.go中。

```
type Plugin interface {
    Name() string
    OnSessionOpen(ssn *Session) # 会话开启时回调函数
    OnSessionClose(ssn *Session) # 会话结束时回调函数
}
```

Binpack

“资源装箱”，尽可能地将工作负载紧密地打包到节点上，以提高集群资源的利用率。

通过评估节点上剩余资源的情况，优先选择那些剩余资源较少的节点来部署新的任务，从而实现资源的紧凑分配。

适用于资源紧张的集群环境、批处理作业、GPU管理

CPU得分：`CPU.weight*(request+used)/allocatable`

总得分：`binpack.weight*(CPU.score+Memory.score+GPU.score)/(CPU.weight+Memory.weight+GPU.weight)*100`

Capacity

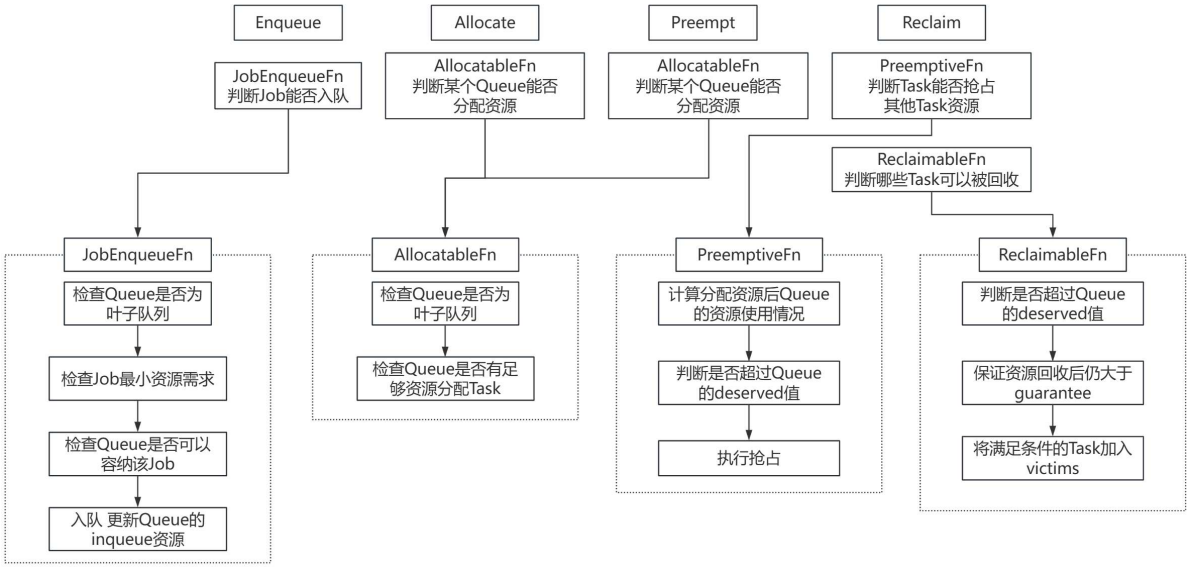
Capacity主要负责管理队列的资源容量和弹性资源分配，通过为队列的每个维度资源指定确切的资源量来实现弹性队列容量管理，而不是通过权重来划分队列的资源。

主要概念：

- **Capability** Queue的资源上限，任何情况下Queue使用的资源不能超过这个限制
- **Guarantee** Queue的资源下限，这部分资源是保留给Queue的，即使Queue中没有Job也不能借给其他Queue
- **Deserved** Queue应得的资源配额，可以在空闲时调度到其他Queue，也可以在需要时从其他Queue回收

Capacity支持层级队列结构。启动层级队列后，Volcano会创建一个根队列root queue，作为所有队列的父队列。

- 资源继承过程中，子队列的资源变化会自下而上的更新父队列的资源状态
- 只有叶子队列可以作为资源调度的Queue
- 在分配资源时，会检查整个队列层级结构，确保从叶子队列到根队列的资源分配都是合理的
- **层级队列结构和Queue资源的区别**：在Capacity插件中层级队列是一个逻辑概念，是在调度器内部构建的，而Volcano的Queue是k8s的一个CRD。在层级队列中的队列代表可分配的资源，只有叶子队列才能对应到Queue中



Gang

Gang实现了Gang Scheduling成组调度功能，确保Job中所有任务能够被同时调度，避免资源死锁和低效利用的问题。Gang Plugin保证要么所有必需的Pod都能被调度，要么一个都不调度。

Gang Scheduling解决了：当一个分布式作业需要多个Pod同时运行才能正常工作时，如果只有部分Pod被调度，这些Pod可能会占用资源但无法开始实际工作，造成资源浪费。

deviceshare

用于管理可共享设备-GPU NPU FPGA等-的插件 提供一个统一的接口来实现设备共享功能

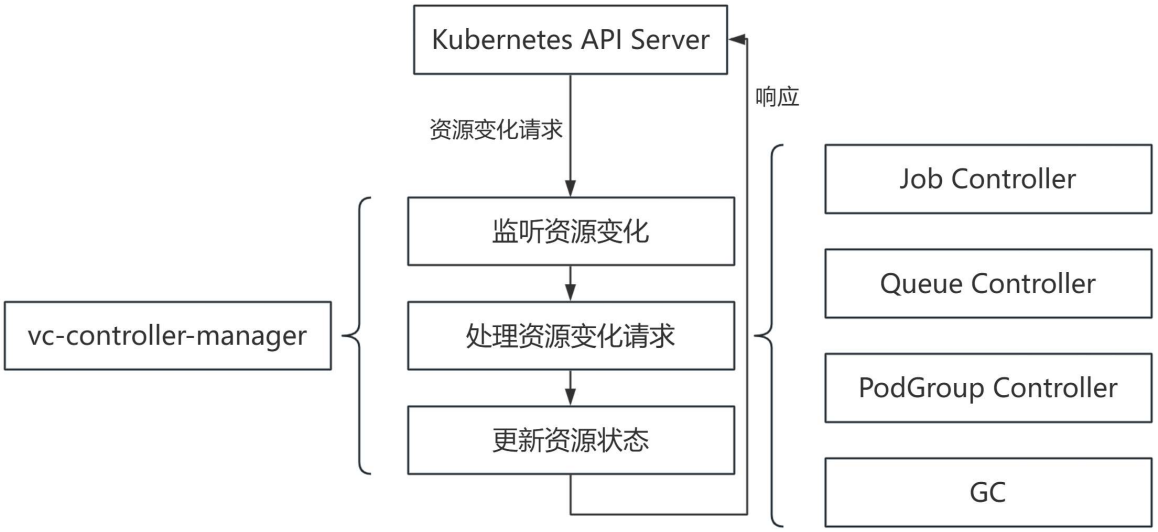
OnSessionOpen中的PredicateFn函数 用于检查task能否被调度到node上

vc-controller-manager

负责管理和维护Volcano自定义资源的生命周期。它是一个控制器集合，包含多个子控制器，每个子控制器负责不同类型资源的管理。

管理对象主要包括Volcano Job、Queue、PodGroup、JobFlow和JobTemplate、GC。

所有controller都实现了framework.Controller接口，并通过framework.RegisterController注册。



Job Controller

Job Controller通过多个informer来监听k8s和Volcano的资源变化，并将这些变化转换为事件添加到工作队列中

jobcontroller监控Job Pod PVC PodGroup Service等k8s&Volcano资源

工作流程

使用各个Informer调用k8s和Volcano的接口来监听资源变化->将变化传入jobcontroller的工作队列queue->主进程不断监听工作队列 来处理新的任务->同时管理cache&重试错误任务

PodGroup Controller

负责管理PodGroup资源 主要管理Pod PodGroup RS StatefulSet等资源

Job Controller Plugin

允许用户为Volcano Job添加额外功能-类似中间件？ 这些Plugin可以在Job生命周期的不同阶段被触发从而实现各种自定义功能

自定义Plugin需要实现PluginInterface接口 主要可以在四个地方添加中间件-创建Pod；添加Job；删除Job；更新Job

vc-webhook-manager

负责k8s资源的准入控制Admission Control。使用k8s的WebHook机制，对创建或更新的资源进行验证Validation或变更Mutation，确保这些资源符合Volcano的要求。

主要功能：

- **资源验证** 验证用户提交的资源是否符合Volcano规范
- **资源变更** 自动修改或补充资源定义使其符合系统要求
- **准入控制** 决定是否允许资源创建或更新

所有WebHook组件通过RegisterAdmission注册到webhook-manager的路由表中。

Webhook组件分为两种：变更性WebHook-Mutating WebHook；验证性WebHook-Validating WebHook

Mutating WebHook：主要负责修改资源对象，例如添加默认值或修改字段；执行验证操作，验证是否为CREATE请求

Validating WebHook：主要负责验证资源对象是否符合规则；只能验证不能修改

工作流程：API-Server发送准入请求到WebHook服务->WebHook服务根据请求路由找到对应的处理函数->验证或变更WebHook处理请求->返回准入相应给API-Server

k8s的准入请求有CREATE UPDATE DELETE

处理CREATE请求时需要依次调用Mutating和Validating；处理UPDATE请求时只调用Validating

Job Webhook

Job WebHook分为验证性WebHook和变更性WebHook。分别负责验证Job资源的合法性和在创建Job资源时添加默认值