**统一任务调度平台**

**部门：物流研发部-开放平台研发部-计费研发部**

**参与人：孙朋伟、张宁、李桐葳、王瑞**

**版本：1.0**

目录

[背景 3](#_Toc520748038)

[业务背景 3](#_Toc520748039)

[现有的解决方案 3](#_Toc520748040)

[现有解决方案的不足 3](#_Toc520748041)

[平台介绍 4](#_Toc520748042)

[目标 4](#_Toc520748043)

[整体介绍 4](#_Toc520748044)

[创新性 4](#_Toc520748045)

[价值与收益 5](#_Toc520748046)

[易用性 5](#_Toc520748047)

[功能介绍 6](#_Toc520748048)

[任务管理 6](#_Toc520748049)

[任务执行 6](#_Toc520748050)

[系统监控 6](#_Toc520748051)

[后台智能调度 7](#_Toc520748052)

[新节点加入 7](#_Toc520748053)

[存活节点检测 7](#_Toc520748054)

[任务动态分配 7](#_Toc520748055)

[死亡节点任务回收 7](#_Toc520748056)

[系统界面 7](#_Toc520748057)

[系统实现 9](#_Toc520748058)

[关于Go语言 9](#_Toc520748059)

[包结构及功能 10](#_Toc520748060)

[功能实现 11](#_Toc520748061)

[总结及展望 11](#_Toc520748062)

# 背景

## 业务背景

在一些计算机系统中，存在一些实时运行的计算任务，这些任务处理数据量大、处理过程复杂，如计费系统，承载了企业多条业务线的数据，需要处理数据转换、计费处理、汇总等任务，这些任务往往消耗的计算机资源多、处理时间长，需要统一管理任务的调度。

## 现有的解决方案

此类问题通常的解决办法是多节点分布式处理，如公司的clover工具，实现了多节点分别处理不同分片数据。具体实现为：

1. 在处理数据的应用中引入clover工具包，处理任务类实现公共接口并发布任务；
2. 启动应用自动注册任务到clover任务列表；
3. 登录clover管理平台配置任务执行频率、每次处理数据量大小等参数，管理任务的运行和停止；
4. clover分配任务到多节点，根据配置参数执行任务。

## 现有解决方案的不足

Clover对使用者属于非透明状态，研发不可控，只能依赖架构人员监控调度机器使用情况；

任务执行期间，节点故障死亡，需人为删除死亡节点；

不支持任务分组等批量处理，需要一个一个启动、停止；

依赖jsf服务，使用有局限性；

调用线程数量有限。

# 平台介绍

## 目标

最大限度减少任务运行期间人为对运行节点负载的监控，实现多节点的自管理、自监控，和节点间任务的动态分配；

提高调度并发量，同时触发多条任务到业务机器；

提高容错性，一个节点出错，不至于让该节点上的任务丢失或失败，其他节点可以接管它的工作；

支持节点扩容，新节点上线可根据当前集群负载自动分配任务，无需人为干预；

增强对节点运行性能和异常日志的监控和报警；

实现对任务的分组和批量管理，及友好的界面操作。

## 整体介绍

实现了通过页面管理、监控任务的功能，构建了一个可自行监控、调节和报警的平台。基于去中心化分布式应用程序的思想，利用心跳检测、缓存、网络通信等技术，实现一个可自治理的统一任务调度平台。

在平台工具的实现上，采用了在网络通信、并行编程有着极佳支持的Go语言，更好地利用分布式和多核计算机特性。

Go语言有着极好的代码风格规范和极佳的开发效率，方便代码库日后快速开发和维护。

该平台包含了前台界面和后台任务调度部分。对任务管理、监控进行了增强，易于用户创建和配置任务；增加了节点负载监控界面。后台任务实现了任务在注册中心之间的分配和互相检测功能。

## 创新性

**产品创意：**

将任务分布式处理、数据分片化管理与分布式系协同工作思想相结合；

通过http协议直接调用jsf服务，减少了额外的业务处理开发工作；

调度管理和业务代码解耦，现有的clover，需在代码中加入对clover的依赖，并实现clover的接口，该平台无需依赖外部工具；

同一节点对不同类型任务的调度，可以从节点的维度去查看任务调度情况和资源使用情况；

可根据整理资源使用情况，分配新节点和下线节点，同时不影响任务的运行。

**技术：**

采用了心跳检测、负载监控、缓存、分布式技术；

平台实现基于Go语言，可以以更快的速度开发软件，更好地在现代多核计算机上和网络环境下工作。

**设计思路：**

巧妙的管理新节点的加入、死亡节点的下线和运行期间根据节点的负载自行调度，无需人为干预即可实现任务在这些节点间的均衡分配。

## 价值与收益

使用场景广泛，此平台适用于数据量大、计算时间长需要分布式执行任务的系统，或需定时执行任务的场景，如一些企业尤其大型企业往往存在大量的数据处理任务。

实现资源的合理利用，能够更好的利用机器的资源，通过自行调度，实现多节点间计算机资源的合理利用，避免计算机资源的闲置。

普适性强，所有可通过http协议访问的任务都可以在平台上进行调度管理，不仅仅局限于jsf服务；因此，可推广到更多类型任务的调度，具有更广泛的社会价值。

节省研发资源和硬件资源成本，使用该平台，可以使研发人员只需开发业务代码，不用过多关心调度任务，可以将多更的时间投入到业务处理中。

## 易用性

使用该平台的用户，只需根据自己的需要对任务进行简单的新增、启动和停止，不用关心后台的任务调度是如何实现的。

节点的新增由平台运维人员进行管理，用户无需关心调度机器的运行情况。

运维人员可通过页面查看各节点的负载情况。

机器性能监控，超过阈值的机器，会发送报警邮件。

异常信息采集，对异常日志进行记录，方便查找异常原因。

# 功能介绍

## 任务管理

进入任务管理界面，可以查看已创建的任务，如需对创建的任务参数修改，可以修改任务，还可在界面管理任务的启动和停止。对任务的新增、修改、状态的变化，均会写入数据库，并刷新缓存。

启动状态的任务会被调度的节点抓取，并根据配置的参数，运行任务。

任务的属性包括：任务名称、任务分组、并发个数、任务类型、初始状态、CRON、是否写日志、负责人邮箱、任务描述、任务参数、是否分区、调度请求方式；

其中：

**并发个数**：表示执行任务该任务时，并发协程的个数；

**任务类型**：分为定时任务和手动任务，处于启动状态的定时任务可自动定时执行，手动任务需人为触发执行；

**初始状态**：任务创建后的状态，分为启动、停止、默认，如果任务类型为自动任务，初始状态不能为默认；

**CRON**：CRON表达式，任务的执行频次，当选择任务类型为手动时，此项可忽略。

**任务参数**：执行任务时，触发业务方法时的入参。

**是否分区**：支持数据分区

**调度请求方式**：GET、POST等。

## 任务执行

各节点获取启用且未被执行的任务，根据任务的cron表达式，通过采用http协议通过url调用任务，触发任务的执行。或手动触发任务。

系统任务执行信息界面，可查看任务的执行状态、错误信息，可根据任务编号、分组和状态进行搜索。方便用户查看具体任务的执行情况。

## 系统监控

包含节点性能监控和任务监控。

节点性能监控，可按ip、时间段查看节点的CPU、内存使用情况。

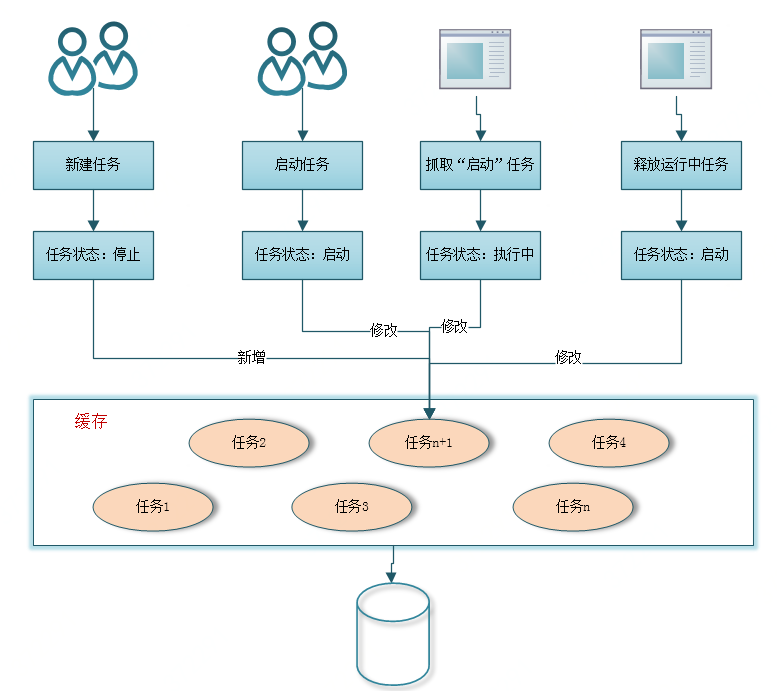
任务监控，可查看当前总任务量、执行中的任务、定时任务和手动任务。

## 后台智能调度

平台后台各节点对任务的调度功能，包含：新节点加入操作、存活节点检测、节点间任务自动分配、死亡节点任务回收等。如图：



系统运行期间任务的生命周期包含：启动、停止、运行中等状态。这些状态之间的转化，如图：



### 新节点加入

在新机器上部署应用，即实现新节点的加入。新节点加入后，可以抓取未被抓取的任务或分摊其他高负荷节点的任务，作为新成员检测节点是否存活。

### 存活节点检测

节点每隔一定时间，查询其他节点心跳信息，判断节点是否存活。如检测到该节点死亡，会再一次发送网络心跳检测，经过两次检测均为死亡的节点，判定为死亡节点。

### 任务动态分配

各节点执行任务期间，会检测自身的负载情况，检测指标包括：CPU和内存情况，这些指标一旦超过临界值，会把自身部分任务迁移到新加入的节点。

### 死亡节点任务回收

对于死亡的节点，其他节点可根据自身负载抓取该死亡节点的任务。

## 系统界面

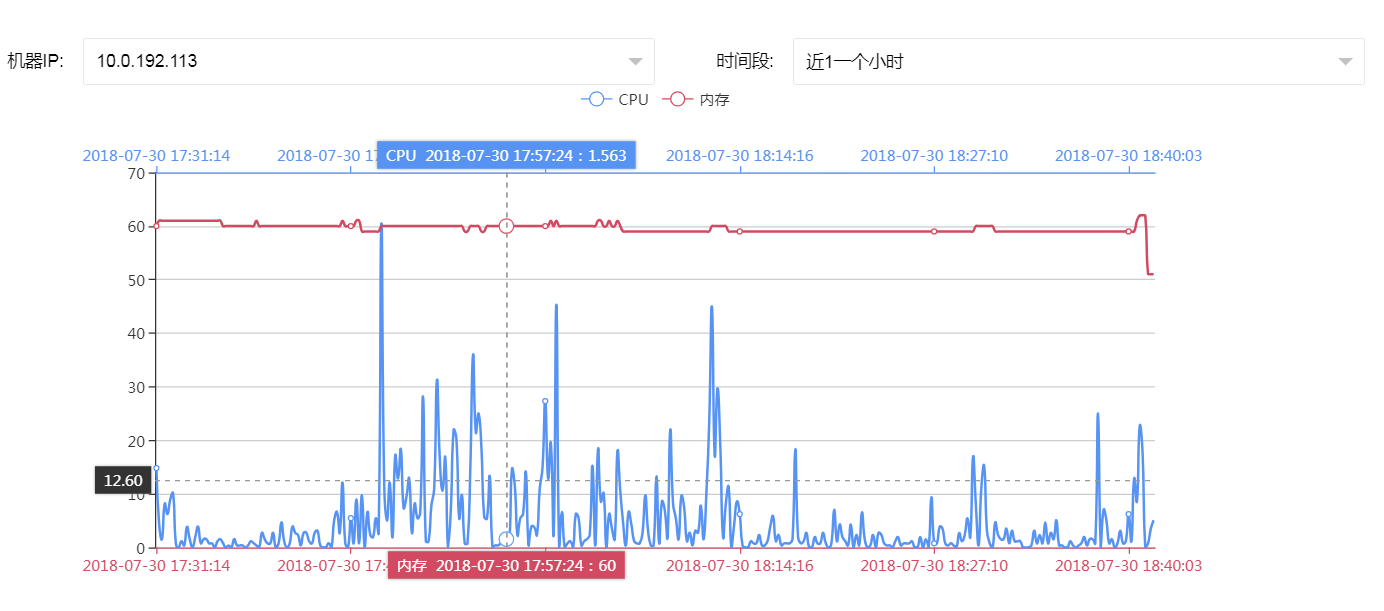
**注册中心**：

**任务管理界面**：

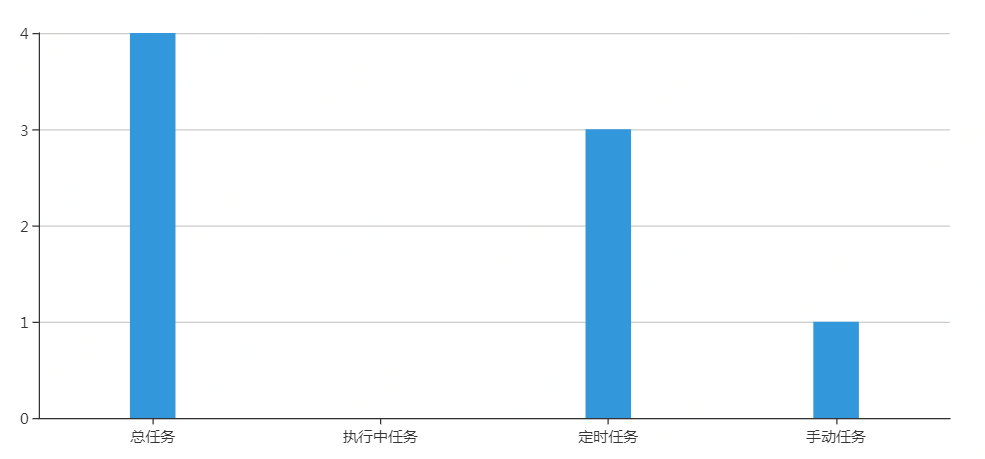


**任务执行信息**：



**性能监控**：

**任务监控**：



# 系统实现

## 关于Go语言

**Go 语言特色**

简洁、快速、安全

并行、有趣、开源

内存管理、v数组安全

编译迅速：Go 语言中另一个非常重要的特性就是它的构建速度（编译和链接到机器代码的速度），一般情况下构建一个程序的时间只需要数百毫秒到几秒。

**Go 语言用途**

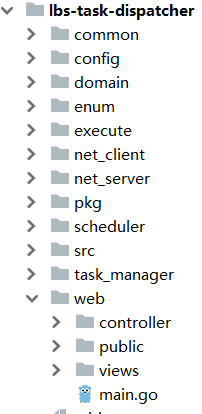
Go 语言被设计成一门应用于搭载 Web 服务器，存储集群或类似用途的巨型中央服务器的系统编程语言。

对于高性能分布式系统领域而言，Go 语言无疑比大多数其它语言有着更高的开发效率。它提供了海量并行的支持，这对于游戏服务端的开发而言是再好不过了。

Go 语言的另一个目标是对于网络通信、并发和并行编程的极佳支持，从而更好地利用大量的分布式和多核的计算机。

这个特性显然是 Go 语言最强有力的部分，不仅支持了日益重要的多核与多处理器计算机，也弥补了现存编程语言在这方面所存在的不足。

## 包结构及功能

工具核心包结构如图

common 公共类库  
config 配置信息  
domain 业务属性和方法  
enum 枚举  
execute 任务执行类  
net\_client 集群中各个节点网络通信客户端  
net\_server 集群中各个节点网络通信服务端  
scheduler 任务调度模块  
task\_manager 任务管理模块  
web 前端页面模块  
 controller 控制层  
 public js和css库  
 views 视图层

## 功能实现

Main启动后工作Main.Go的main函数为启动程序入口。

程序启动后会执行reids、mysql、task\_manager、register\_center，scheduler等init函数，初始化并开始运行程序。

代码可参考git地址：http://git.jd.com/sunpengwei/lbs-task-dispatcher.git

# 总结及展望

通过参与这次开发者大会，从开始到完成，经历了一个月的时间。小组成员齐心协力，完成了本作品的构思和实现。在技术上，将对Go语言的学习应用到实际项目中，对自身技术有了一定的提升；另一方面，经历了，作品从无到有，到慢慢完善，也获取了很多关于作品的规划和设计思想的知识；同时，也感受到团队成员完成一件作品的喜悦和成就感。

最后，由于时间有限，本平台还有许多待优化和改进的地方，如增加错误日志查询功能、参数配置管理等。后期，将系统逐渐完善。希望，能更早的将本作品应用于实际的业务生产过程中。