目录

模块简介

CSP 压测模块是 CSP 系统中比较基础的一个模块，输入信息为各个应用的压测配置，输出信息为进行压测时各项指标（比如 qps，响应时间，load，gc 信息等）的值。目前 CSP 一共支持 5 种类型的压测：

1、 httpLoad，通过调用 http_load 进行压测。

2、 apacheSplit，通过 apache 的 mod_jk 模块进行分流模式压测。

3、 apacheProxy，通过 apache 的 mod_proxy 模块进行代理模式压测。

4、 nginxProxy，通过 nginx 的 location 转发进行 nginx 代理模式压测。

5、 hsf，通过 configServer 的 sdk 修改 configServer 目标机器的 ip 倍数进行压测。

压测原理介绍

apacheSplit、apacheProxy、nginxProxy 压测原理

压测开始

备份线上配置→获取线上配置拷贝到本地→本地修改→把修改的配置发到线上临时文件→拿临时文件覆盖线上配置→重启 apache 或者 nginx reload。

压测结束

拿线上备份的配置覆盖线上配置→重启 apache 或者 nginx reload。

http_load 压测原理

调用 http_load 进行压测，压测的用户数和压测时间在压测配置中有设定。

Hsf 压测原理

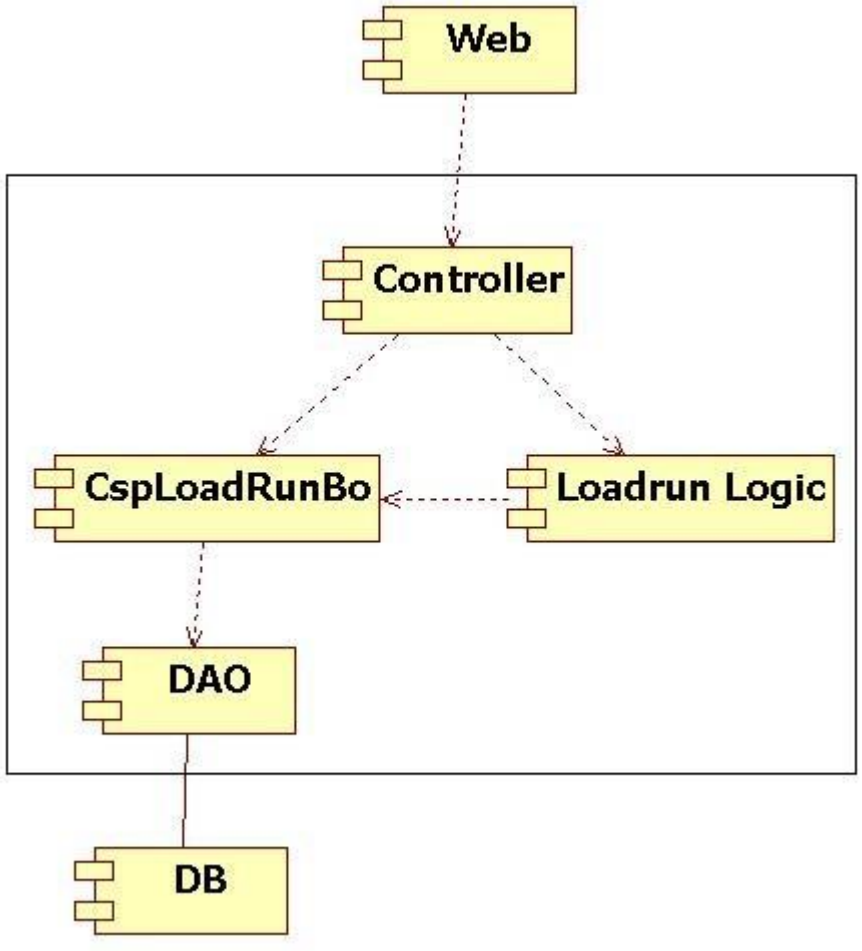
压测开始

调用 ConfigServer 的 SDK，往 configServer 发送一条 ip 权重翻倍的 rule。

压测结束

调用 ConfigServer 的 SDK，往 configServer 发送一条删除该 rule 的消息。

模块架构

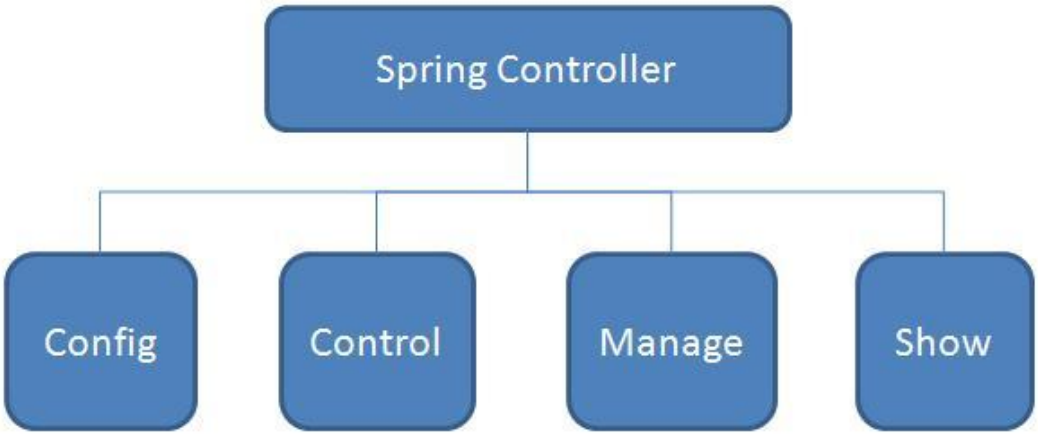


前端

web 前端采取 spring mvc 的架构，所有的 web 请求只跟 controller 进行沟通，jsp 页面中几乎没有什么 java 代码，基本做到展示与代码逻辑的分离。

Spring Controller

系统中有 4 个 spring controller，分别处理 4 种类型的页面请求。



Config：压测配置的维护，包含压测配置的新增、修改、删除等

Control：压测的控制，包含手动压测与自动压测的触发等

Manage：压测结果的维护，包含压测结果的删除等

Show：前端展示的取数，包含压测配置的查看展示、压测结果的查看展示等

Loadrun Logic(压测逻辑介绍)

压测的逻辑是压测模块中比较核心的一部分内容，压测分两种方式，五种类型。其中 5 中类型在前面已经提到过，压测的两种方式分别为手动压测和自动压测：

1、手动压测：在人工的参与控制下完成压测。

2、自动压测：一旦触发即自动跑完整个压测流程，自动压测可配置 cron 表达式并开启自动执行模式，则会在设置的时间点自动执行，也可人工进行触发。

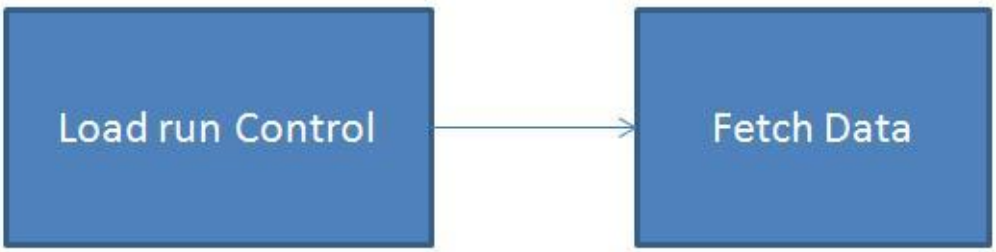
自动压测与手动压测的区别

自动压测一旦触发会按压测的配置自动执行完整个压测流程，并支持 cron 表达式的定时任务，而手动压测需要手动输入一些压测配置进行人工控制。

压测逻辑整体结构

站在一万米的高空上看 CSP 的压测逻辑，可把 CSP 的压测逻辑分成两块，一块为压测的控制，另一块为压测结果的采集。

在真正开始压测之前，获取各类压测结果指标的线程已经启动，跟压测结果的数据源进行了连接，并开始持续取数，待压测结束的时候，会先关闭这些连接，完成压测结果的采集。



压测逻辑详细介绍

从一万米的高空了解了压测的整体结构分成压测控制与压测结果采集之后，我们再回到地面来仔细探究这两块的详细结构与流程。

压测控制的结构与流程

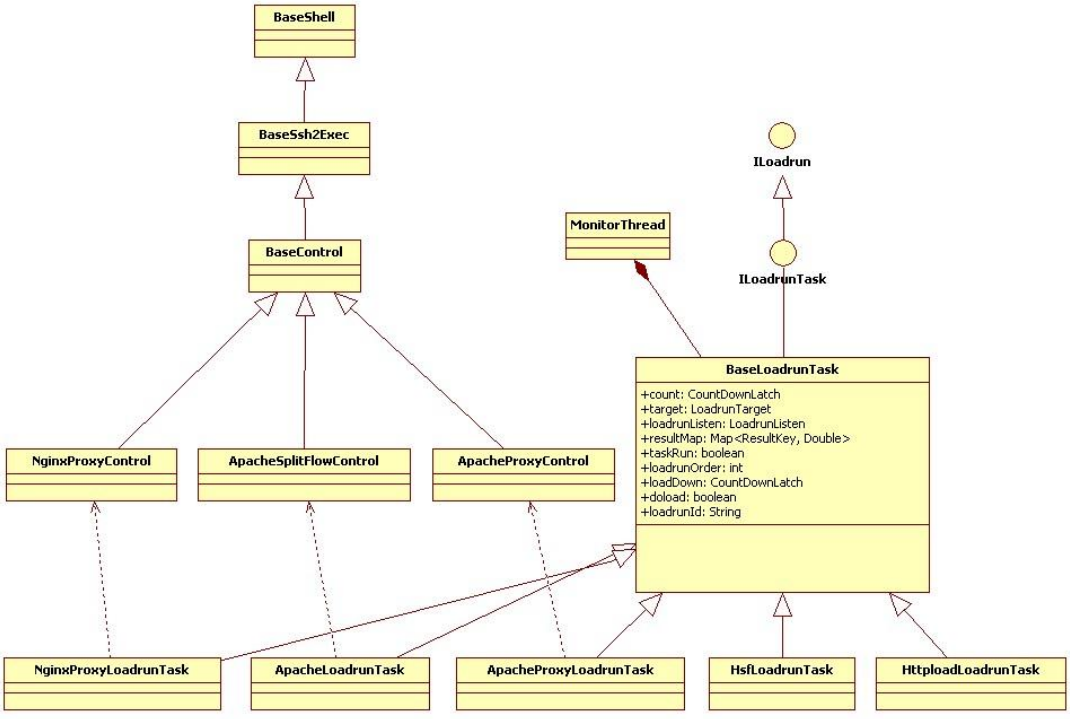
压测控制的核心组件是压测任务，在系统中体现为 Task。5 种类型的压测都分别有对应的 Task 类，比如 http_load 类型的压测对应的 Task 类为 HttploadLoadrunTask。需要修改线上配置的 Task 会对应一个与线上系统进行沟通的 BaseControl 类型的实例。

压测控制的另一个重要组件是压测配置。压测配置在系统中对应了两个类：LoadRunHost(负责跟数据库打交道)和 LoadrunTarget（负责跟压测 Task 打交道，

Task 的创建以 LoadrunTarget 为基础，准确地说是以 LoadrunTarget 的成员 AutoLoadType 为基础），LoadRunHost 为了方便在存取数据库的时候方便，存的都是原始信息，而 LoadrunTarget 为了方便压测，会对 LoadRunHost 进行简单的解析。

手动压测的控制流程为直接执行 Task；自动压测会先把 Task 包装成 Job，序列化后发送给各个 Slave 去执行，而自己维护着一个 Master 来管理这些 Slave 并对 Slave 在执行过程中的消息进行处理（包括压测的 report、类信息的加载等），Slave 牵涉到 CSP 的 Assign 模块，Assign 是 CSP 系统中用于分布式处理任务的一个模块，后续会简单进行介绍。Master 与 Slave 的通讯系统采用的是 Mina 框架。

Task 的结构



最下面的为具体各种类型压测的 Task。其中 apache 分流、apache 代理、nginx 代理都是需要修改线上的配置，因为都依赖了对应的 BaseControl。

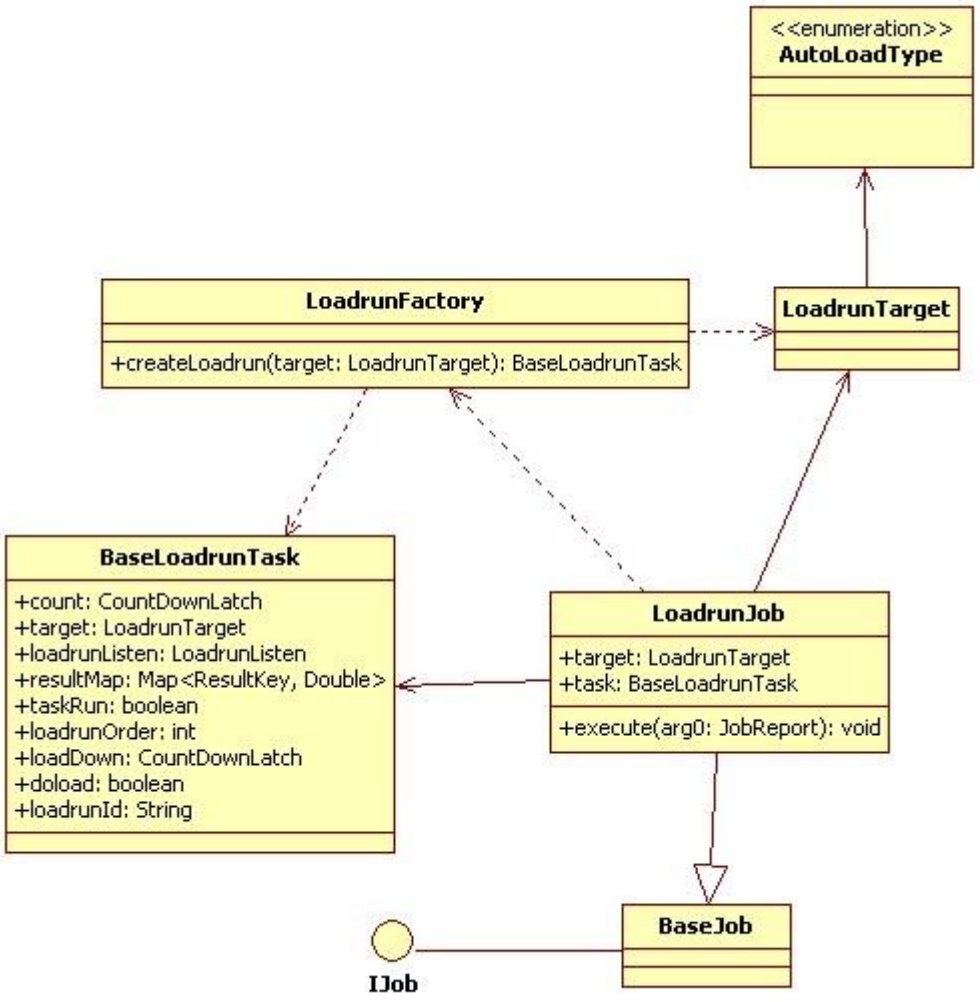
BaseControl 可跟线上系统进行操作，用到的是三方包 ganymed-ssh2。BaseLoadrunTask 作为压测任务的基类包含了压测任务的各种通用操作，同时有

一个内部类 MinitorThread 随时监控压测指标是否到达设置的限，一旦到达限制则立即停止压测。

BashShell 维护跟线上机器的连接，BashSsh2Exec 用于维护 shell，打开 shell，并对 shell 命令输入及 shell 输出的获取，BaseControl 包含一些共用的方法。

自动压测

自动压测会先把 Task 包装成 Job，并发到 Slave 端去执行（CSP 系统有一个分布式的任务执行框架 Assign）。LoadrunFactory 用于根据压测类型（AutoLoadType）来生成对应的压测 Task。AutoLoadType 为一个枚举类型，代表前面提到的 5 种压测类型。LoadrunTarget 维持着压测的配置信息(包括压测类型),下图是 Job 的结构。



自动压测的入口类是 LoadrunTaskMaster，LoadrunTaskMaster 在 spring 容器中被初始化。

LoadrunTaskMaster 管理整个自动压测的流程：

1、 属性 Map<String,TaskResource> needRunMap 存放所有的压测任务。

2、 属性 LinkedBlockingQueue<TaskResource> taskQueue 存放需要执行的任务。有两种途径会往 taskQueue 放入任务：一种是人工触发自动压测，另一种是 cron 表达式的时间到了（开启了自动模式），系统自动放入该任务。

3、 属性 Master master，分布式任务的管理者，用于分配 job。

4、 属性 LoadrunListen loadrunListen，Job 的监听者，用于处理一系列的事务。

5、 内部类 TaskResource，是对 LoadrunTarget 的包装，增加了 Date 类型的字段 nextFireDate（表示自动压测的任务下一次的执行时间），布尔类型的 isRun（表示压测任务是否处于运行状态）。

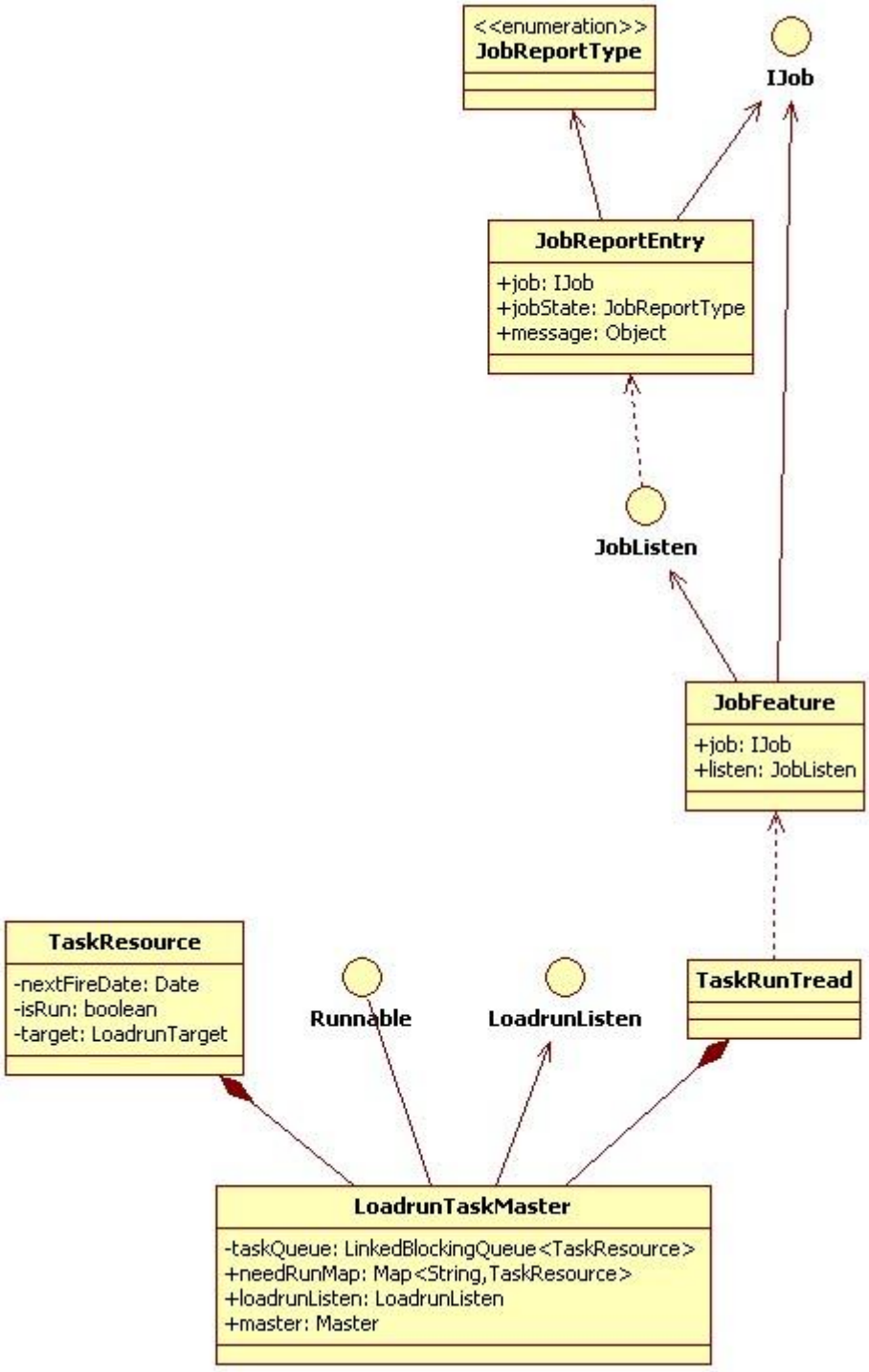
6、 LoadrunTaskMaster 本身也是一个线程，会一直扫描 needRunMap 中的任务，判断 cron 表达式是否到执行时间，如果是则把该任务的 isRun 标志设置成 true，放入 taskQueue。

7、 内部 TaskRunTread，一个单独运行的线程，负责从 taskQueue poll 出任务，交给分布式框架的 Master 去分配执行，这里才是压测真正开始的地方。

8、 LoadrunTaskMaster 包含一系列方法，如：提交、新增、修改、删除等供 spring 的 controller 调用。

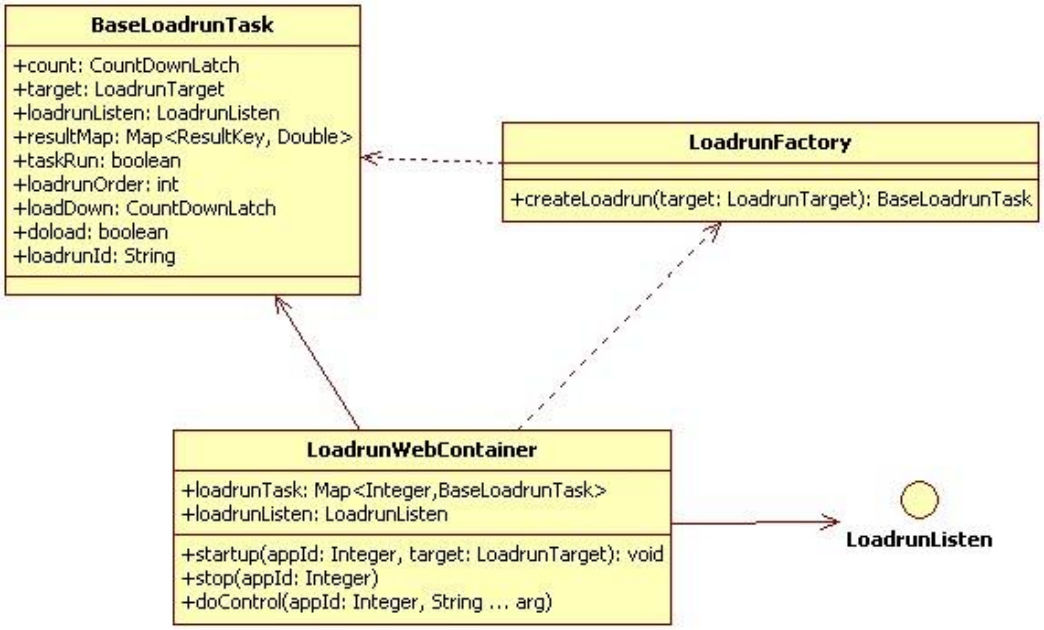
9、 内部类 TaskRunTread 轮询访问 taskQueue，把 Task 包装成 Job，然后交给 Master。

10、 JobFuture 是对分配给 Slave 的 Job 的包装，增加了 JobListen 用户处理事务。JobReportEntry 是 Slave 端在执行 Job 时给 Master 汇报的工作内容之一，Slave 与 Master 的通讯都是基于 BasePacket，Packet 好比是信封，RequstPacket 代表发信，ResponsePacket 代表回信，而通讯的内容存在于信封之中（Object 类型），Packet 有类型 PacketType，Master 与 Slave 接收到 Packet 后都会先判断 Packet 的类型来做相应的响应。有关 Assign 后面会有更详细的介绍。



手动压测

手动压测没有自动压测那么复杂，入口类 LoadrunWebContainer 同样是在 sprint 容器中被初始化。仅维持着一个需要进行手动压测的集合，直接自己执行 task。



压测控制逻辑小结

1、 前台配置的压测信息直接映射成 LoadrunHost 与数据库打交道。

2、 进行压测的时候把 loadrunHost 简单解析成 LoadrunTartet 与压测的 Tast 打交道。

3、 LoadrunFactory 根据 LoadrunTartet 的 AutoLoadType 来生产对应的 LoadrunTask 对象。

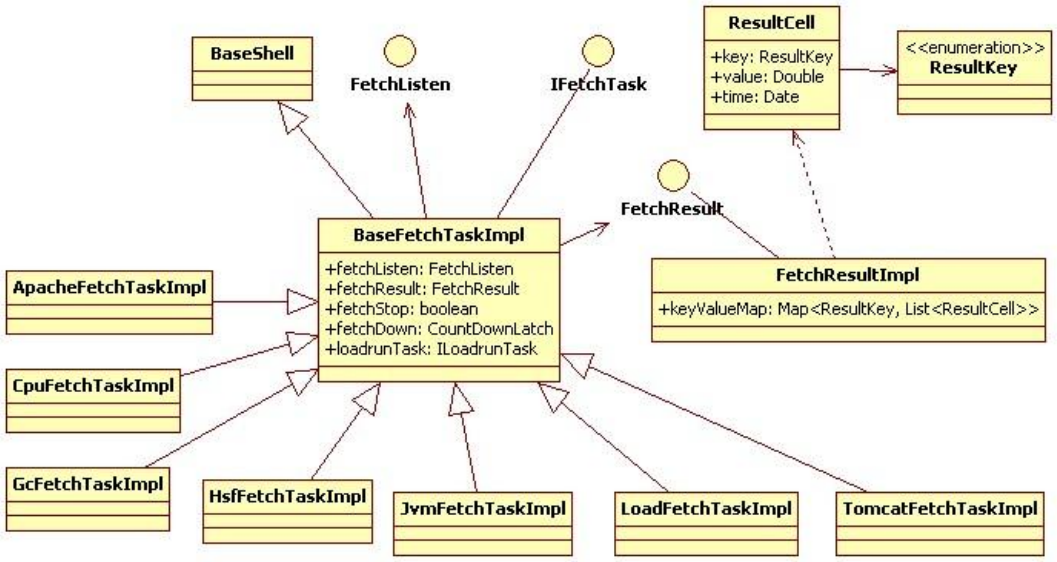
4、 手动压测方式，系统直接执行该 Task。

5、 自动压测方式，先把 Task 包装成 Job 交给 Master 来管理。

6、 Task 根据类型，可能需要 Control 来操作线上机器的配置。

Fetch Data（压测结果采集）介绍

前面介绍了压测的控制部分，压测控制的目的是什么？是为了得出在压测进行时，压测机器的各项指标的值，而这些指标可能需要从不同的途径来获取，可能要对不同的日志文件进行解释或者通过 shell 命令直接查看线上机器的运行情况等等。这就引入了下面要介绍的内容：压测结果指标的获取。



压测结果的采集基于 IFetchTask，所有的 Fetch 都继承自 BaseFetchTaskImpl 这个基类，BaseFetchTaskImpl 包含一些压测结果采集的基本操作，具体的 Fetch 类最大的不同在于对取到的信息提取压测结果的分析规则不一样。FetchResult 用于存取到的结果,ResultKey 是结果的 key，ResultCell 是对结果的简单封装（包含 ResultKey，value，及时间）。压测执行完需要记录数据，而这个数据就是保存 FetchResult 中。因为结果的获取基本上都要上线上机器取信息，所以 BaseFetchTaskImpl 继承自 BashShell。BashShell 在前面压测控制的时候有介绍到，通过 ssh 与压测机器进行沟通。每一个 Fetch 都会有自己的数据源，分析数据源传过来的数据，并提取自己关注的指标数据。对信息的分析规则用到了正则表达式。

Fetch 之 ApacheFetchTaskImpl

数据源

tail -f /home/admin/cai/logs/cronolog/yyyy/MM/yyyy-MM-dd-taobao-access_log

提取的指标

Apache pv：ResultKey.Apache_Pv

Apache 响应时间：ResultKey.Apache_Rest

Apache 返回 200 的次数：ResultKey.Apache_State_200

Apache 页面大小：ResultKey.Apache_PageSize

Fetch 之 CpuFetchTaskImpl

数据源

mpstat 1

提取的指标

CPU 使用率：ResultKey.CPU

Fetch 之 GcFetchTaskImpl

数据源

tail -f /home/admin/logs/gc.log

提取的指标

CPU 使用率：ResultKey.CPU

GC 消耗的时间：ResultKey.GC_Min_Time

单次请求内存消耗：ResultKey.GC_Memory

GC 次数：ResultKey.GC_Min

Full GC 的时间：ResultKey.GC_Full_Time

Full GC 的次数：ResultKey.GC_Full

CMS 方式 GC 次数：ResultKey.GC_CMS

Fetch 之 HsfFetchTaskImpl

数据源：

tail –c n

/home/admin/logs/monitor/monitor-app-org.eclipse.osgi.internal.baseadaptor.DefaultClassLoader.log

提取的指标

Hsf 接口 QPS：ResultKey.Hsf_pv

Hsf 接口响应时间：ResultKey.Hsf_Rest

Fetch 之 JvmFetchTaskImpl

数据源

tail -f /home/admin/logs/mbean.log

提取的指标

Jvm 内存使用率：ResultKey.Jvm_Memeory

AJP 线程处于阻赛状态的个数：ResultKey.AJP_BLOCKED

AJP 线程处于运行状态的个数：ResultKey.AJP_RUNNABLE

AJP 线程处于等待状态的个数：ResultKey.AJP_WAITING

Fetch 之 LoadFetchTaskImpl

数据源

top -bi -d 1

提取的指标

CPU Load：ResultKey.Load

Fetch 之 TomcatFetchTaskImpl

数据源

tail -f /home/admin/logs/tomcataccess/localhost_access_log. yyyy-MM-dd.log

提取的指标

Tomcat 每秒 pv：ResultKey.Tomcat_Pv

Tomcat 返回 200 的次数：ResultKey.Tomcat_State_200

Tomcat 相应时间：ResultKey.Tomcat_Res

Tomcat 页面大小：ResultKey.Tomcat_PageSize

Assign 简介

Assign 设计的初衷一个分布式的任务执行框架。CSP 自动压测的 Job 是交给 Assign 的 Slave 去处理的，在 CSP 压测模块中自己维护着一个 Master 来对 Slave 进行管理，存在于自动压测的控制类 LoadrunTaskMaster。

Master 介绍

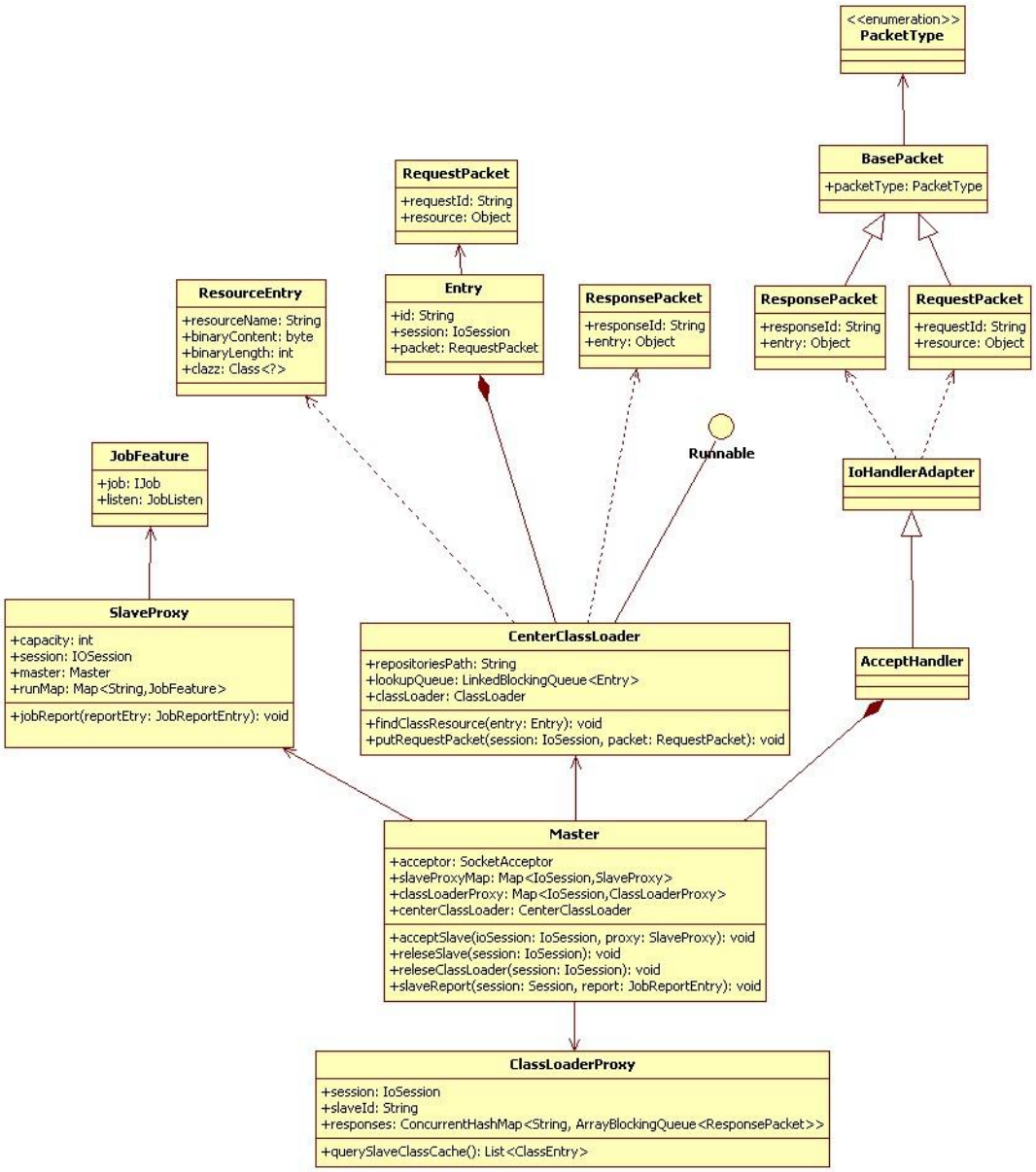
Assign 中的 Master 维护着一个 Mina 的 Acceptor 用于接收 Slave 端的连接。内部类 AcceptHandler 作为 Mina 的 handler 用来处理从 Slave 端接收到的各种请求。Master 的主要成员包含：

1、 Slave（SlaveProxy），每一个 Slave 在启动时都会向 Master 进行汇报，Master 会随之生成一个 SlaveProxy 对象，保存在 Map 中。

2、 Slave 端的 ClassLorder（ClassLoaderProxy），Slave 端有自己的 ClassLoarder，Job 相关的类在 Slave 端是不存在的，需要从 Master 端发送过去，每一个 Slave 端的 ClassLoader 也维持着一个 Mina 的 Connector，一旦连接，Master 会生成一个 ClassLoaderProxy，保存在 Map 中。

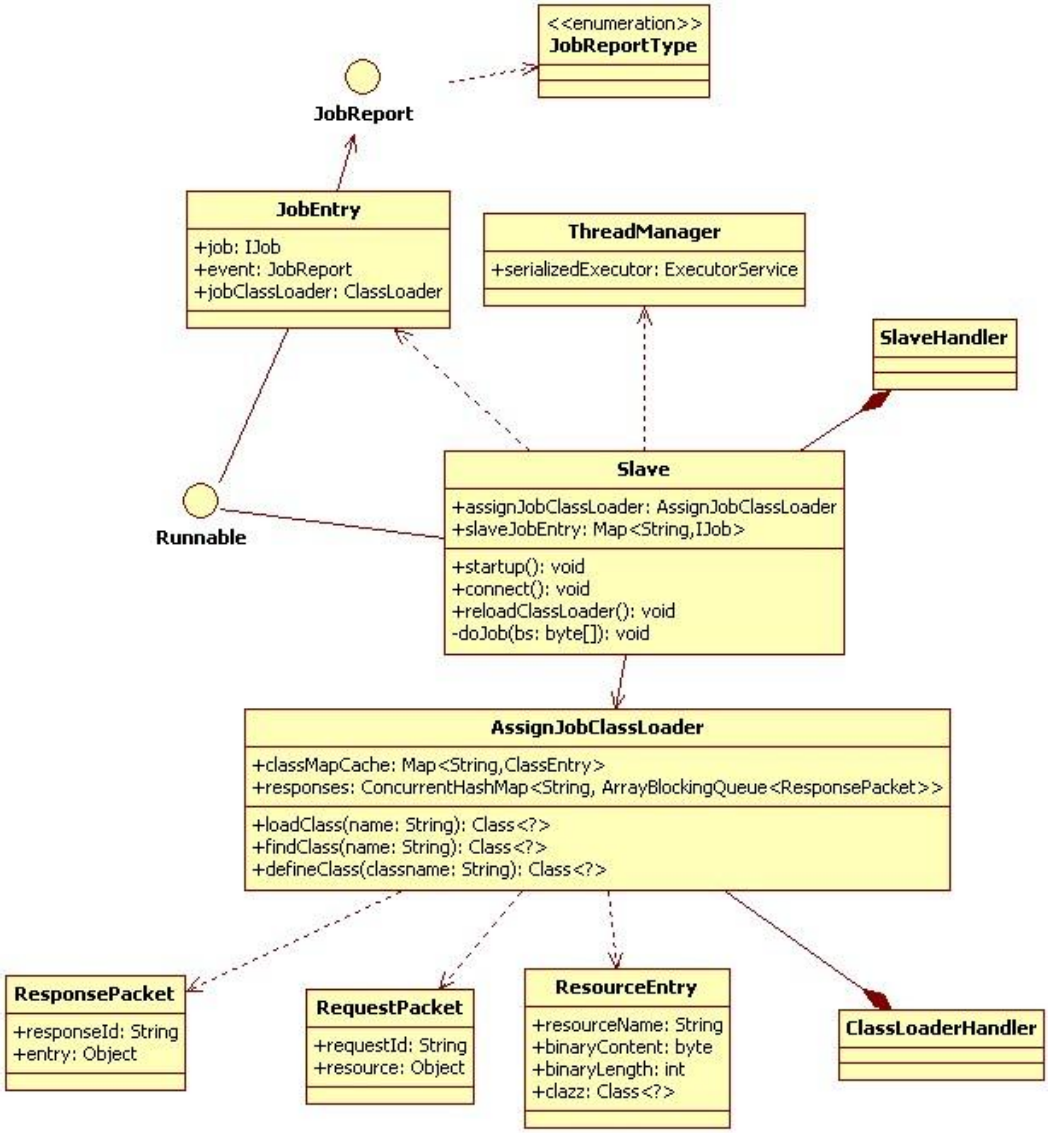
3、 Master 自己的 ClassLoarder（CenterClassLoader），当 Master 接收到 Slave 发送过来的类加载或文件加载请求时，CenterClassLoader 就发挥作用了，完成加载并把加载的结果发送给 Slave。

Master 在分配 Job 的时候会先在自己维护的 SlaveProxy 中找到空闲的 Slave，然后把 job 序列化后发生给相应的 Slave 去处理。



Slave 介绍

Slave 被 Master 所管理，单独部署在独立的机器上，作为 Mina 的 connector 与 Master 保持连接。Slave 维持着一个 Job 列表，保存从 Master 发过来的 Job。Slave 端有自己的 ClassLoader，用于加载位于 Master 的类信息，ClassLoader 同样维持着一个 Mina connector，负责与 Master 进行通讯。Slave 一个重要的工作就是执行 job，job 的具体执行其实跟 Assign 没什么关联，因为 Job 是压测模块通过 Master 发过来的，job 的相关逻辑都在压测模块，而需要的类都是 Master 从压测模块加载后发送到 Slave 端，这也保证了 Assign 作为一个分布式任务框架的通用性。



1、 Slave 管理着自己的 ClassLoader 与从 Master 端发来的 job。

2、 内部类 SlaveHandler 作为 Mina 的 Handler 对接收到的信息进行处理。

3、 AssignJobClassLoader 作为 Slave 端的 ClassLoader 复杂加载 Job 执行需要的 Class 信息，因 Class 信息可能在 Master 端，所以 AssignJobClassLoader 自身维持一个 Mina 的 Connector，与 Master 通讯来请求相应的 Class 信息。

4、 ClassLoaderHandler 作为 AssignJobClassLoader 的 MinaHandler 还处理接收到的信息。

5、 JobEntry 实现了 Runable 接口，用于执行 job，并封装了 JobEvnet 用于处理 Job 执行过程中的事件。

6、 ThreadManager 维持着一个线程池来执行 JobEntry。