文档编号：

|  |  |  |
| --- | --- | --- |
| 系统类别 |  | 共 页  （含封页） |
| 产品型号 |  |
| 产品代号 |  |
| 项目/产品名称 |  |

CloudCI Master部分概要设计

（文档版本号：V0.1）

|  |  |  |  |
| --- | --- | --- | --- |
| 拟 制： | 郭伟 | 日 期： | 2016-6-28 |
| 审 核： |  | 日 期： |  |
| 会 签： |  | 日 期： |  |
| 批 准： |  | 日 期： |  |



北京信威通信技术股份有限公司

版权所有 不得复制

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日 期 | 修订版本 | 描 述 | 作 者 |
| 2016-6-28 | V0.1 | 初稿 | 郭伟 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目 录

[1 引言 1](#_Toc455127670)

[1.1 编写目的 1](#_Toc455127671)

[1.2 术语和缩略语 1](#_Toc455127672)

[1.3 参考资料 1](#_Toc455127673)

[2 设计概述 2](#_Toc455127674)

[2.1 Cloud CI整体架构 2](#_Toc455127675)

[2.2 具体职责 2](#_Toc455127676)

[2.2.1 微服务职责 3](#_Toc455127677)

[3 接口设计 3](#_Toc455127678)

[3.1 外部接口 3](#_Toc455127679)

[3.2 内部接口 3](#_Toc455127680)

[4 业务流程 4](#_Toc455127681)

[4.1 任务管理 4](#_Toc455127682)

[4.1.1 任务创建操作 5](#_Toc455127683)

[4.1.2 任务状态操作 6](#_Toc455127684)

[4.1.3 任务结果操作 6](#_Toc455127685)

[4.2 资源管理 6](#_Toc455127686)

[4.2.1 配置整体流程 7](#_Toc455127687)

[4.2.2 配置过程接口设计 8](#_Toc455127688)

[4.3 用户管理操作 8](#_Toc455127689)

[4.3.1 用户查询操作整体 8](#_Toc455127690)

[4.3.2 告警查询与GIS数据查询 8](#_Toc455127691)

[4.4 微服务启动 8](#_Toc455127692)

[4.5 微服务升级 9](#_Toc455127693)

[4.6 日志管理与故障管理 9](#_Toc455127694)

[5 接口设计 9](#_Toc455127695)

[5.1 配置接口 9](#_Toc455127696)

[5.1.1 UE信息配置接口 9](#_Toc455127697)

[6 遗留问题 18](#_Toc455127698)

[附录A 会议纪要 19](#_Toc455127699)

[A.1 年 月 日 19](#_Toc455127700)

概要设计

【摘要】：本文档主要介绍Cloud CI中的Master部分与其他部分之间的接口和网络架构，以及Master内部之间的接口和Cloud CI内部组织方式。

【关键词】：Cloud CI，Master

# 引言

## 编写目的

本文规定了Cloud CI中的Master的实现方案，包括系统架构和接口实现等

## 术语和缩略语

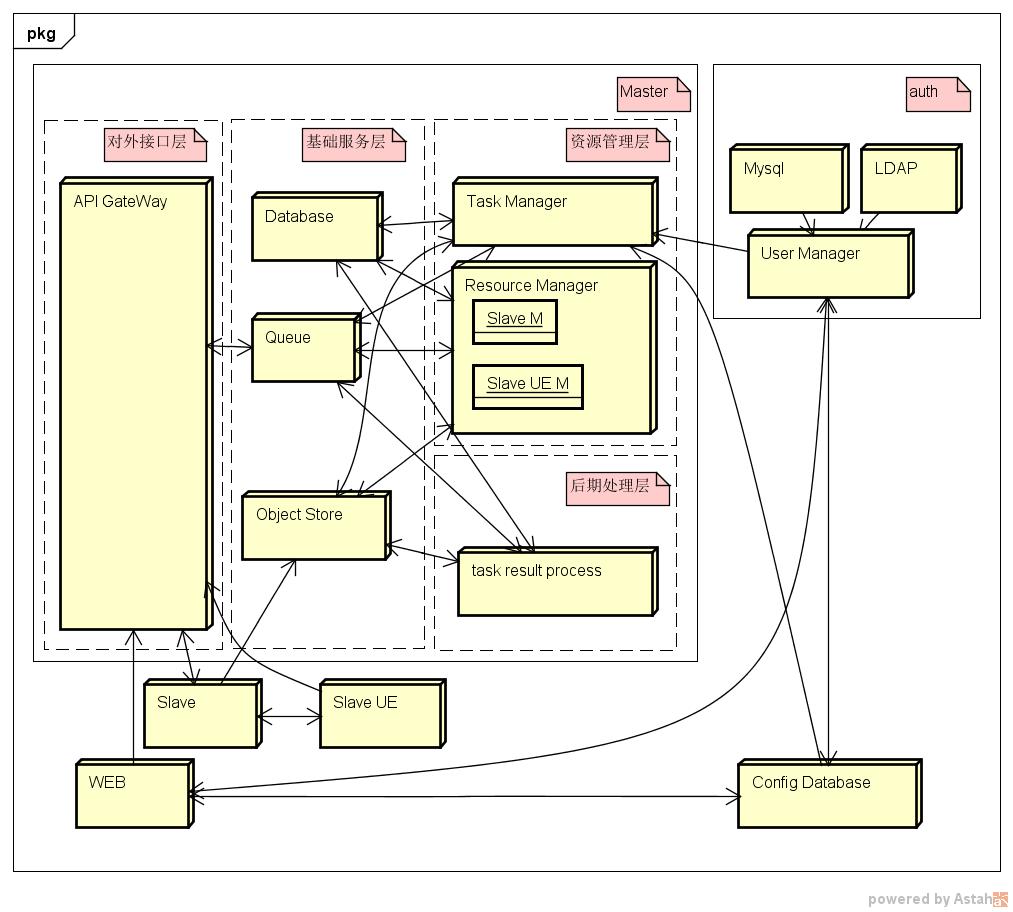
|  |  |  |
| --- | --- | --- |
| 简写 | 全拼 | 中文 |
| RMS | remote management call | 远程终端管理 |
| RPC | Remote Procedure Call | 远程过程调用 |
| SOAP | Simple object access protocol | 简单对象操作协议 |
| MQ | Message Queue | 消息队列中间件 |
| Memcache | Memory Cache | 内存数据缓存 |
| MS | Microservice | 微服务 |

## 参考资料

* 1. 《环境组网》
  2. 《云测试平台 接口定义》
  3. 《Test Frame Work UML》

# 设计概述

## Cloud CI整体架构



主要设计思路是由资源负责制去区分微服务。以微服务的形式独立进行部署，以方便管理，升级等。上图中为Cloud CI的整体图，Master部分详细展示，而其他部分均为示意。

## 具体职责

图中展示了Cloud CI的整体结构，现阶段Cloud CI由WEB，Master，Slave，SlaveUE（多UE控制），Auth组成。其中，WEB部分负责展示与配置，Master负责Slave与任务管理，Slave负责具体执行任务，Slave UE负责多UE控制部分工作，Auth负责认证，用户信息查询等工作。因Master性能问题现将Master分为多个微服务以提高Master处理性能。这里主要介绍Master和Auth中划分的各个微服务的具体职责。

### 微服务职责

* API Gateway微服务

向外部呈现统一的外部接口，方便其他模块调用。

负责所有Master收到的消息的内容校验部分。

* 任务管理微服务

负责任务的统一管理。

负责申请空闲资源用于任务运行。

负责任务结果检查，模拟CI发信等操作。

* 资源管理微服务

负责资源的统一管理。

负责处理任务管理的资源申请请求。

负责资源状态与数据维护。

* 任务结果处理微服务

负责在任务运行完成之后处理任务结果。

负责在真实环境任务结束之后生成流量图，bler图等。

* 用户管理微服务

负责用户信息的认证与查询。

负责加载用户信息，以及用于信息的基本组织形式维护。

# 接口设计

## 外部接口

Master部分向外提供接口还是以Restful的方式提供，Auth部分向外部提供接口也以Restful的方式提供。

## 内部接口

在Master内部微服务之间的交互使用MQ进行通信，不同微服务之间使用特定通道（channel）进行通信，通道中传输JSON数据。这样考虑而不是直接传输Java对象是为了适应微服务异构的情况。Python，Java都可以使用MQ，都可以解析JSON数据。规定向某个微服务发送数据就是用微服务的名称作为通道名。

在微服务之间使用Memcache的方式进行数据共享。这里的Memcache使用Redis实现。所有的Redis与Mysql之间的同步由数据操作模块同时操作两个数据库完成。使用Redis的0号数据库作为任务相关信息，包括任务输出，任务步骤，~~CI任务缓存~~。使用1号数据库作为在线UE信息存储。使用2号数据库作为任务EI信息存储使用。

# 业务流程

接下来使用业务划分的方式将整体的软件架构分功能进行描述，以便进一步设计，开发时方便了解软件整体架构在网管中的具体意义。

## 任务管理

任务管理中又设计多个use case，这里说明整体任务创建，任务状态，任务结果的过程的管理。



### 任务创建操作

任务创建按照自动化程度可以分为：手动测试与自动化CI测试两种。按照运行方式还可以分为新创建任务和重新执行任务两种。上图中所有的流程即所有的流程的合并，一下详细描述任务创建流程：

1. 从WEB侧发送过来任务创建请求之后，由API Gateway进行接收并根据请求URL进行转发。
2. 在Task服务收到请求之后，判断以上四种执行方式，进行不同的处理。
3. 在Task中根据任务要求向Resource申请资源。创建自动化测试用例等操作。
4. 发送任务开始信息给后部处理服务，以方便后部处理服务进行策略维护。
5. 发送任务给Slave，由Slave开始处理任务。

### 任务状态操作

任务状态维护主要关注任务的状态，在一些异常情况下:任务异常退出，slave异常退出的情况下用来维护任务状态。

1. 在任务开始后，每隔两秒slave上报一次任务状态，上报任务进度。
2. 任务进度将被存储到redis中，由WEB侧用户展示时使用。
3. 在3次没有收到slave的心跳后认为任务异常结束，并进入任务结束流程。

### 任务结果操作

在任务结果操作中，需要检查测试用例最终实行结果，这些结果需要检查任务输出中最终的测试用例组的执行结果。

1. 在任务结果处理中需要考虑任务的几种形式，根据不同的形式完成不同的操作。
2. 自动化模拟任务需要发信给响应的提交人以及组内领导，通知某个服务进行任务结果记录。自动化真实环境任务需要通知wiki进行任务结果记录。
3. 真实环境任务将任务结束结果发送给后部处理服务，由后部处理服务完成图片的生成。
4. 通知资源管理服务释放资源，由资源管理模块完成资源状态管理。

## 资源管理

资源管理模块主要负责资源状态以及资源动态信息的维护。其中可能涉及到资源的自动上线时配置信息自动填充部分。



### 资源管理流程

具体操作过程以及各模块接口定义：

1. Slave上线后需要向Master发送上线消息。
2. Gateway在收到上线之后，转发给Resource，由Resource处理具体内容。
3. Resource在收到上线之后检查配置，如果配置存在直接上线并记录动态信息到数据库中。
4. 如果配置不存在则新建一个Slave配置，并在配置上线。
5. 用户上线时检查为任务Slave时需要下发EI配置信息到Slave上，因为在采集基站EI信息时需要EI配置信息进行解析。
6. 在Slave上线后需要与Master保持心跳以便进行状态管理。没过30秒进行一次心跳，三次没有收到心跳则认为Slave已经下线，需要进行下线处理。
7. 在任务执行过程中Resource不在修改Slave状态，在Slave被释放之后进行修改。

## 用户管理操作

在Cloud CI中有多处需要使用到用户信息。这里将用户管理特意的分为一个独立的微服务进行管理，方便各处对用户信息的使用，屏蔽用户管理底层细节。



### 用户管理流程

用户管理主要负责用户信息整合与认证的工作，以下具体介绍流程：

1. 在WEB上用户登录时需要进行用户鉴权操作。在WEB侧将用户名，密码提交给用户管理之后，由用户管理部分进行用户鉴权，并将鉴权结果返回给WEB。
2. 在任务管理服务中需要CI发信时需要查询用户所在组以及组内领导的，需要向用户管理模块完成服务发送用户名，有用户管理模块完成用户信息的查询并返回。
3. 在WEB侧可能会修改用户管理数据库的配置，在修改完配置后由WEB通知用户管理服务。在用户管理服务中完成用户信息的重新加载与维护。
4. 在各模块使用用户管理服务时，如遇到发生异常，超长时间未返回等问题时，微服务需要完成自己的工作，不要让故障传播。
5. 在数据库配置未完成的情况下，要求用户管理模块设置默认管理员账号，进行使用。在配置完成之后自动取消默认账号的使用权限。

## 微服务启动

为了整体架构的稳定性与健壮性，所有服务启动顺序不做规定。所有微服务都必须适应在某个相关微服务掉线之后的操作与处理。

微服务具体启动过程：

1. 检查是否携带升级脚本作为启动参数。如果携带则执行升级脚本。
2. 根据配置参数连接数据库，MQ，Memcache。
3. 检查数据库中是否有表Schema，如果没有则创建。

## 微服务升级

微服务升级启动方式使用单服务退服，然后重新部署单服务并启动。

在升级启动时需要指定微服务启动脚本，然后根据启动过程进行启动。

## 日志管理与故障管理

现阶段使用各个微服务自行管理日志的方式进行日志管理。故障管理使用维护人员手动维护来完成。

# 接口设计

参见《云测试平台 接口定义》

# 遗留问题

1.自动化任务结果通知目的未知。

1. 会议纪要
   1. 年 月 日

地点：

与会人员：

主要结论：