

# last day

---

## 今日概要

---

- 关于小程序的疑问
- 补充代码发布系统 & 项目总结

## 今日详细

---

### 1. 小程序疑问

<https://docs.qq.com/sheet/DUkZLb3RTSnZISGNn?tab=m0uww0&c=C69A0AY0>

- 项目开发周期 / 人员配比 / 如何开发？

第一期：

...

- 之前没做过，让你直接负责小程序？

- 小项目前后端都自己写 、 大型项目要进度才会专业前端
- vue.js 和 小程序相似
- 做完自己总结，小程序起就是html、css、js只不过多了一个微信的规则。

- 小程序端的API都用过哪些 & 特点？

API

异步 + 闭包 + COS上传文件的功能

- 小程序印象深刻？

- 安全性考虑，全部的调用接口在后台配置 + https
- API比较方便 + 组件和HTML、CSS、JS相似（官方文档）。

- 自定义tabbar？
- 事件冒泡？
- 页面生命周期？
- 小程序支付环节？
- 支付完成时，服务器宕机如何解决订单状态未更新的问题？
- 了解的restful规范？

https / 名词

- drf都继承过那些视图类？在视图类中 你都写过那些自定义方法？

序列化  
认证  
保存之前

- drf序列化如何操作？

source  
自定义  
嵌套  
钩子: get    validate

- 集成腾讯对象存储COS的好处？且 授权
- 短信实现流程？频率限制？
- ajax的FormData上传
- 自定义ModelForm的钩子，实现 上传本地 -> 上传到COS
- Celery实现定时订单和状态处理
- Datetimepicker去选择时间

## 2.代码发布系统

### 2.1 关于代码发布

- 人肉

开发，写完代码把他拷贝/git上传到某个地方。  
运维，  
    下载代码  
    把代码通过SCP发到指定的服务器 c1.com/c2.com/c3.com  
    连接上每台服务器，执行服务的重启。

- 工具

开发，写完代码把他拷贝/git上传到某个地方。  
运维，  
    下载代码  
    工具：把代码发到指定的服务器 c1.com/c2.com/c3.com  
    工具：连接上每台服务器，执行服务的重启。

工具: saltstack、ansible、puppet

- jenkins软件

开发，写完代码把他拷贝/git上传到某个地方。

运维用jenkins,

- 配置：代码目录 / git地址 / 开发语言 / 定制
- 点击构建
  - git中下载代码
  - SCP/工具 把代码发到指定的服务器 c1.com/c2.com/c3.com
  - 定制脚本，连接上每台服务器，执行服务的重启等命令。

工具：saltstack、ansible、puppet

- 开发代码发布平台

## 2.2 我们的发布系统

系统主要用于实现减少人工干预及成本，基于python的框架和gitlab实现发布任务的处理。项目中对于密钥进行了统一管理，避免权限冲突。为了方面项目中对于项目和服务器之间的管理和隔离，给项目分配指定服务器，分配之后再通过“全量发布”和“灰度发布”进行更小粒度的管控，通过django channels组件使项目支持websocket，以实现发布、回滚实时数据和日志的展示，另外通过 gojs 动态绘制发布任务步骤的流程图，支持发布和同时多审阅。

### 功能

- 摒弃原本地ssh文件的方式，使用StringIO实现对于密钥文本处理。
- 为项目创建多环境管理，并根据环境去做发布任务的配置。
- 对于不同发布任务，通过配置remote path 进行远程项目路径的管理。
- 支持自定义Hook脚本并能够根据用户选择实现模板的二次利用。
- 直接使用paramiko进行远程服务器的操作，避免对于ansible、saltstack等工具的依赖。
- 通过 with上下文管理和定制Transport对象 实现同用 SSHProxy组件，以提升远程服务器操作性能。
- 基于gitpython模块封装常见远程仓库操作，并结合ModelForm Hook 实现选择版本、分支、commit等，避免用户在页面重复输入导致发布过程异常。
- 引入前端组件GoJS，内部基于canvas实现动态绘制流程图。
- 研究 Websocket 实现原理，并基于 django-channels 组件实现日志、图表实时动态展示(channels layer 实现群组概念)。
- 支持定义钩子实现发布任务的高度定制化，以应对复杂的发布业务流程的处理。

### 技术栈

django / MySQL / Paramiko / SaltStack / Gojs / websocket / channel-layers / redis / celery / 上下文管理 / git

## 2.3 发布系统细节

### 2.3.1 密钥管理

目前只是完成密钥的基本CURD管理。

密钥管理他可以集成权限&用户，来实现每个用户维护自己的密钥（github/码云）。

知识点：

- 自己本地生成 一对公钥和私钥
- 公钥上传到服务器、私钥保留在A电脑
- 以后A电脑访问服务器就不需要密码。

发布系统 秘钥 服务器 项目 环境 Link Dropdown				
添加				
ID	用户名	私钥	状态	操作
1	root	-----BEGIN RSA PR...	启用	编辑 删除

2.3.2 服务器管理

目前对公司所有的服务器进行一个管理和分配。

服务器、部门、负责人、运维产生联系，一旦发生问题即使找到负责人和部门（预算）。

发布系统 秘钥 服务器 项目 环境 Link Dropdown				
添加				
ID	主机名	操作		
1	10.211.55.25	编辑 删除		

2.3.3 项目（应用管理）

发布系统 秘钥 服务器 项目 环境 Link Dropdown				
添加				
ID	项目名	仓库地址	操作	
1	luffycity	https://gitee.com/wupeiqi/xxoo.git	编辑 删除	

2.3.4 环境管理

发布系统 秘钥 服务器 项目 环境 Link Dropdown						
添加						
ID	项目	环境	线上路径	服务器列表	发布任务	操作
6	luffycity	正式	/data/prod/	10.211.55.25	发布任务 (0/2)	编辑 删除
7	luffycity	测试	/data/test/	10.211.55.25	发布任务 (0/0)	编辑 删除

2.3.5 发布任务管理

为每个环境提供多次发布任务，创建发布任务时需要指定：Git版本（tag/branc+commit） / 全量灰度 / 自定义钩子。

发布系统

密钥

服务器

项目

环境

Link

Dropdown

创建发布任务单

项目名称: luffycity

环境: 测试

仓库地址: https://gitee.com/wupeiqi/xxoo.git

仓库地址: /data/test/

目标服务器:

- 10.211.55.25

基础配置

描述

请输入描述

分支

请选择分支

版本

请选择分支

提交记录

请选择提交记录

发布类型

全量主机发布

选择主机

10.211.55.25

发布流程 & 钩子

01 开始

02 下载前

03 下载代码

04 下载后

05 本地打包

06 上传代码

07 发布前

08 发布

09 发布后

02 下载前

请选择模板

请输入下载前脚本

☐ 保存为模板

请输入模板名称

04 下载后

请选择模板

请输入下载后脚本

☐ 保存为模板

请输入模板名称

07 发布前

请选择模板

请输入发布前脚本

☐ 保存为模板

请输入模板名称

09 发布后

请选择模板

请输入发布后脚本

☐ 保存为模板

请输入模板名称

保存

知识点：

- ModelForm进行表单的展示

- 封装git-python模块实现tag/branch/commit的获取，再将数据集成到ModelForm中。
- 钩子部门通过ModelForm展示。

```
class TaskModelForm(forms.ModelForm):
    f1 = forms.CharField(...)
    f2 = forms.CharField(...)
    ...
    f16 = forms.CharField(...)
    class Meta:
        model = models.Task
```

- 数据动态展示

内部Form初始化时候进行 init\_git / init\_hook

- 下拉框绑定change事件

```
class HookScript(models.Model):
    """
    钩子脚本
    """
    title = models.CharField(verbose_name='标题', max_length=32)
    hook_type_choices = (
        (2, '下载前'),
        (4, '下载后'),
        (7, '发布前'),
        (9, '发布后'),
    )
    hook_type = models.IntegerField(verbose_name='钩子类型',
    choices=hook_type_choices)
    script = models.TextField(verbose_name='脚本内容')

    def get_script_template(request, template_id):
        """
        获取脚本模板
        :param request:
        :param template_id:
        :return:
        """
        response = BaseResponse()
        try:
            script_object =
models.HookScript.objects.filter(id=template_id).first()
            response.data = script_object.script
        except Exception:
            response.status = False
            response.error = '获取模板失败'
        return JsonResponse(response.dict)
```

- 提交发布任务时，需要对钩子进行处理。选择作为模板，则必须写模板名称。

### 2.3.6 发布

通过gojs / channels / channel-layer 实现动态图表展示 及 实时日志输入。

发布系统

密钥

服务器

项目

环境

Link

Dropdown

Channels发布

返回

发布

重新发布

项目名称: luffycity	环境: 测试 (待发布)
仓库地址: https://gitee.com/wupeiqi/xxoo.git	上线版本: luffycity-v1-20200221105803
线上项目地址: /data/test/	线上上传代码地址: /data/codes/luffycity/luffycity-v1-20200221105803
目标服务器: <ul style="list-style-type: none"><li>10.211.55.25</li></ul>	

开始

下载

下载后

打包

10.211.55.25

发布前

发布

发布后

```
>>> 【初始化图表】成功
>>> 【开始发布】成功
>>> 【下载代码】成功
>>> 【下载后钩子】成功。执行日志:
>>> 【打包】成功
>>> 【10.211.55.25 上传代码】成功
>>> 【10.211.55.25 发布后钩子】成功。
详细信息如下: anaconda-ks.cfg
original-ks.cfg
>>> 【10.211.55.25 发布】成功
>>> 【10.211.55.25 发布后钩子】成功。
详细信息如下: /root
```

- 打开页面
  - 展示发布基本信息。
  - 已发布，则显示图表信息。
- 发布按钮 (gojs+websocket)
  - 按照流程逐一实现，成功则显示绿色；失败则显示红色。
- 重新发布 (gojs+websocket)
  - 重新执行未成功的节点。
- channels一个重要的点 (bug)

默认无法试试动态的展示，因为源码内部维护了队列+单线程，队列中实时数据无法展示。  
通过创建子线程去执行任务。  
主线程去队列中获取数据，做实时响应。

```
def channels_deploy(request, task_id):
    """
    websocket发布
    :param request:
    :param task_id:
    :return:
    """
    task_object = models.DeployTask.objects.filter(id=task_id).first()

    deploy_server_list =
models.DeployServer.objects.filter(deploy=task_object)
    return render(request, 'web/channels_deploy.html',
                    {'deploy_server_list': deploy_server_list, 'task_id':
task_id, 'task_object': task_object})
```

- 展示 & 发送websocket请求初始化图和日志

```
{% extends 'layout.html' %}
{% load staticfiles %}
{% block css %}
    <style>
        .log {
            min-height: 300px;
            border-radius: 0;
        }

        .log .item {
            margin: 5px 0;
        }

        .log .item:before {
            content: ">>>";
        }
    </style>
{% endblock %}
{% block content %}
    <div class="container">
        <div class="btn-group" role="group" style="float: right;">
            <a href="{% url 'deploy_task_list' env_id=task_object.env_id
%}" class="btn btn-primary">
                <i class="fa fa-reply" aria-hidden="true"></i> 返回</a>
            <button id="deploy" type="button" class="btn btn-success">发
布</button>
            <button id="retryDeploy" type="button" class="btn btn-
success">重新发布</button>
```



```

        <!--
        {% if task_object.status == 1 %}
            <button id="deploy" type="button" class="btn btn-success">
发布</button>
        {% endif %}
        {% if task_object.status == 4 %}
            <button id="retryDeploy" type="button" class="btn btn-
success">重新发布</button>
        {% endif %}
        -->
    </div>
    <h1 style="margin-top:0">Channels发布</h1>
    <div>
        <table class="table table-bordered">
            <tbody>
                <tr>
                    <td>项目名称: {{ task_object.env.project.title }}</td>
                    <td>环境: {{ task_object.env.get_env_display }} ({{
task_object.get_status_display }}) </td>
                </tr>
                <tr>
                    <td>
                        仓库地址: {{ task_object.env.project.repo }}
                    </td>
                    <td>
                        上线版本: {{ task_object.uid }}
                    </td>
                </tr>
                <tr>
                    <td>
                        线上项目地址: {{ task_object.env.path }}
                    </td>
                    <td>
                        线上上传代码地址: /data/codes/{{
task_object.env.project.title }}/{{ task_object.uid }}
                    </td>
                </tr>
                <tr>
                    <td colspan="2">
                        目标服务器:
                        <ul>
                            {% for item in deploy_server_list %}
                                <li>{{ item.server.hostname }}</li>
                            {% endfor %}
                        </ul>
                    </td>
                </tr>
            </tbody>
        </table>

```

```

</div>

<div id="myDiagramDiv" style="width:100%; height:350px;
background-color: #DAE4E4;"></div>
<pre id="log" class="log">
</pre>
</div>
{% endblock %}

{% block js %}
<script src="{% static 'web/go-no-logo.js' %}"></script>
<script>
    myDiagram = null;
    ws = null;
    $(function () {
        initGoJS();
        initWebSocket();
        bindDeployEvent();
        bindRetryDeployEvent();
    });

    function initGoJS() {
        var $ = go.GraphObject.make;
        myDiagram = $(go.Diagram, "myDiagramDiv", {
            layout: $(go.TreeLayout, {
                angle: 0,
                nodeSpacing: 50,
                layerSpacing: 70
            })
        }); // 创建图表，用于在页面上画图

        myDiagram.nodeTemplate = $(go.Node, "Auto",
            $(go.Shape, {
                figure: "RoundedRectangle",
                fill: 'lightgray',
                stroke: 'lightgray'
            }, new go.Binding("figure", "figure"),
            new go.Binding("fill", "color"),
            new go.Binding("stroke", "color")),
            $(go.TextBlock, {margin: 8}, new go.Binding("text",
"text"))
        );

        myDiagram.linkTemplate = $(go.Link,
            {routing: go.Link.Orthogonal},
            $(go.Shape),
            $(go.Shape, {toArrow: "OpenTriangle"})
        );
    }

```

```

function initWebSocket() {
    ws = new WebSocket('ws://127.0.0.1:8000/deploy/{ task_id
}}/');

    ws.onmessage = function (event) {
        let info = JSON.parse(event.data);

        if (info.code === 'init') {
            // 初始化图表节点信息

            myDiagram.model = new go.TreeModel(info.data);
        } else if (info.code === 'log') {
            // 实时输出日志

            $('#log').append($('
```

```
class DeployConsumer(websocketConsumer):  
    pass
```



A terminal window titled '1. root@10:/data/test' with tabs for 'Default (bash)', 'root@10:/data/test (ssh)', and 'root@10:/data/test (ssh)'. The terminal shows the following commands and output:

```
[root@10 data]# sl  
-bash: sl: command not found  
[root@10 data]# \ls  
codes test  
[root@10 data]# cd test/  
[root@10 test]# ls  
luffycity  
[root@10 test]# ll  
total 4  
lrwxrwxrwx. 1 root root 64 Feb 21 07:01 luffycity -> /data/codes/luffycity/test/luffycity-v1-20200221120119/luffycity  
[root@10 test]# cd /data/codes/luffycity/  
[root@10 luffycity]# ls  
test  
[root@10 luffycity]# cd test/  
[root@10 test]# ls  
luffycity-v1-20200221120119 luffycity-v1-20200221120119.zip luffycity-v2-20200221120342 luffycity-v2-20200221120342.zip  
[root@10 test]# cd /data/  
[root@10 data]# ls  
codes test  
[root@10 data]# cd test/  
[root@10 test]# ls  
luffycity  
[root@10 test]# ll  
total 4  
lrwxrwxrwx. 1 root root 64 Feb 21 07:03 luffycity -> /data/codes/luffycity/test/luffycity-v2-20200221120342/luffycity  
[root@10 test]#
```

扩展：还可以支持celery做定时发布（小程序后端定时任务相似）

### 3.建议

- 危机感：线上面试机会，不要一直等到线下。
- 应该做什么？
  - 参考代码发布流程复习项目：微信小程序、呼啦圈、发布系统、CMDB、CRM。
  - 刷面试题：4套就业考试题
  - 小绿本前58页面 + 简历
- 面试过程：
  - 提前找老师帮助，需求给老师。
  - 线上面试（录音 + 录屏），总结面试问题，将不会的都记录并解决。
  - 心态，面试题中成长。
- 以后
  - 计划和目标：2年之内要进入BAT（倒推）
  - 永远不要混写代码的初级阶段。
  - 提升自己：Linux、代码、高并发架构

