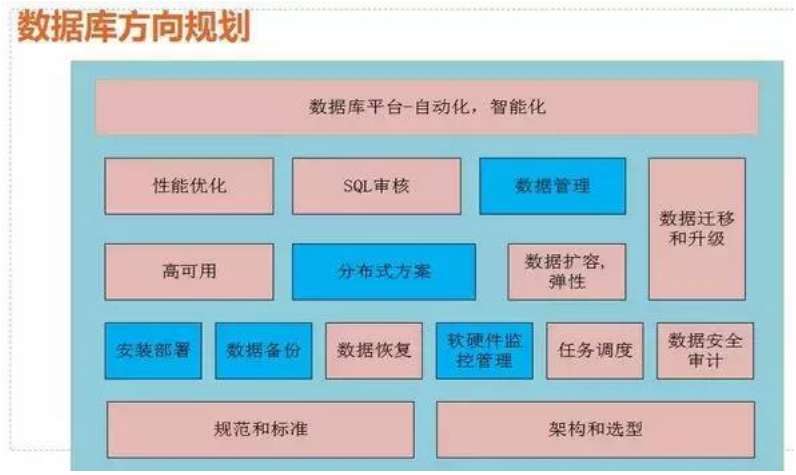
本次分享来自自动化运维群的分享，摘录整理出来。

先来一张图，这是我在去年的时候规做的一个数据库方向规划。



蓝色的部分是我们已有的部分，另外的部分是我们当时做得不好的地方。当然这个过程说起来都是辛酸泪。都是一点一滴的改进。

本次分享里说到的脚本管理和工单管理其实这是里面的两个模块，此外，我们还做了很多的模块。 所以先来简单说下目前系统的使用情况，这是我做的一个看板，已经做了前后端分离，可以统计调用 API 的次数，部署实例的次数等，这些都是一些业务属性。

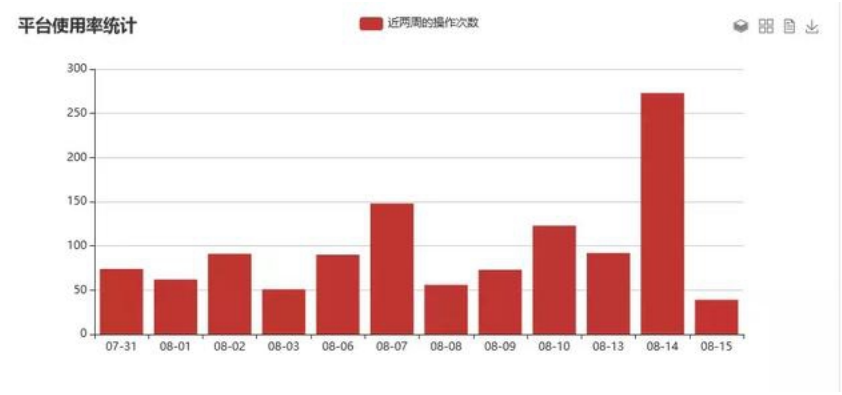
下面还有平台的一些使用频率的统计，通过这些我可以知道目前系统的使用情况。

我还做了一个功能的统计表，可以看到不同功能的使用情况，这样也能知道一些功能的使用情况，是否存在明显的问题。

操作次数

#	功能	使用次数
1	新功能实验室-SQL审核测试查询	527
2	新功能实验室-SQL审核	267
3	资产管理-资产看板	252
4	基础运维-MySQL权限开通	150
5	MySQL管理-SQL查询	134
6	基础运维-系统权限开通	114
7	资产管理-实例管理	109
8	用户中心-菜单管理	104
9	工单管理-MIS工单接入	90
10	基础运维-系统权限查看	74
11	资产管理-平台使用率看板	71
12	基础运维-MySQL实例部署	61
13	备份恢复-Redis备份配置	51
14	用户中心-菜单权限开通	42
15	资产管理-Redis实例明细	42
16	脚本管理-脚本列表	41
17	备份恢复-Redis备份列表	31
18	API管理-API接入	30
19	用户中心-用户中心	30
20	备份恢复-Redis备份看板	27

这是近一段时间的统计数据，如果前端发起了一次请求，就会记录下来。



简单介绍完，我来说说脚本管理和工单模块的建设思路。

运维平台的发展逃不过几个步骤，脚本化，工具化，可视化和自动化，注意在自动化的阶段前，有一个阶段是可视化。

但是显然在很多时候我们的脚本化做得不够好，比如代码里可能会有这样的实现的代码：

```
    elif task_info =='showprivs':
        sqltext = 'show grants for '+db_username
        #sqltext = 'show grants for '+db_username.replace("'","'\\'")
    print(sqltext)

syscmd="cd /home/dba_mysql/scripts/tmp/test; sh agent_db.sh "+vm_ip_addr+" "+vm_db_port+ ' "'+sqltext.replac

print(syscmd)
sqlresult = []
```

这种虽然能够实现功能，但是对于平台的维护来说，就是无数的钩子，不可控的钩子。

如果脚本的路径发生变化或者要对脚本做变动，几乎是一件不可能完成的事情

所以随着脚本的引入，发现了更多的问题，最大的问题就是现有的脚本没法直接用。要用起来，比如要适应一定的规则，然后最大的隔阂就来了，开发不懂运维，运维不懂开发，这样的情况会让问题白热化。

所以对于脚本的管理很重要，但是缺少一些规范可行的方式。

我做了如下的总结：

1）为了能够快速平滑的接入，脚本管理中的脚本语言其实不是瓶颈，都应该全面支持，比如使用 perl,使用 shell,SQL 等，如果脚本本身很稳定，那么完全可以接入进来，总之就是这个环节要开放，不一定要完全是 python 脚本。平台的开发功能是 python,但是脚本管理不一定是 python。

2）在脚本管理中，脚本和菜单如何映射，这是个关键，我们可以把脚本属性参数化，比如脚本名，脚本的类型等这些也是作为一种元数据来管理。这样就会是一个统一的接口的方式，至于具体的连接方式，比如树形结构或者其他可行的方式。

3）平台方向上可以提前规划，但是对于开发和业务同学来说，无需配置大量的 url，就可完成一些基础或者复杂功能的扩展。

4）现有的基础架构和功能，脚本化对于它来说也是起到促进作用。需要提前规划和已有的基础功能是否有可衔接的地方。

5）脚本管理支持文件的上传和脚本内容编辑。这个就是偏具体技术的实现了，比如 ACE 编辑器。

6）脚本的参数管理，有的脚本是1个参数，有的是2个，其实对于后台来说，就是拿到脚本来处理，怎么做标识和匹配。

7）脚本管理中，有些脚本是通用的，如果希望能够持续使用，必须要提前规划好范围和类别。有些脚本是具体的一些业务场景需要的，需要明确需要的参数和权限。

脚本不光用通用和私有的范围，而且还需要细化到具体的作用域范围。

上面的内容看起来篇幅比较大，我们再做一层提炼。

脚本管理模块主要做这些工作：

目标：初步实现脚本的提交，脚本审核和脚本查看功能

任务细则：

l 实现脚本信息的可配置化管理

l 实现脚本的信息查看

l 脚本类别和信息的管理

l 脚本信息提交后由脚本管理员审批

l 脚本审核后可以统一发送通知邮件

所以这就是我们工作中需求和设计之间的一个转化，我们可以想出很多要做的事情，但是我们还要做减法，哪些是优先要做的，哪些是锦上添花的。

这是一个脚本信息的列表。 这里需要注意的是我们在数据库中会维护这个数据结构，数据库中也会存储对应的脚本内容，同时在文件系统中也会存在对应的文件，那么我们所做的变更就会是两个层面，数据库层面和文件层面。

脚本信息列表

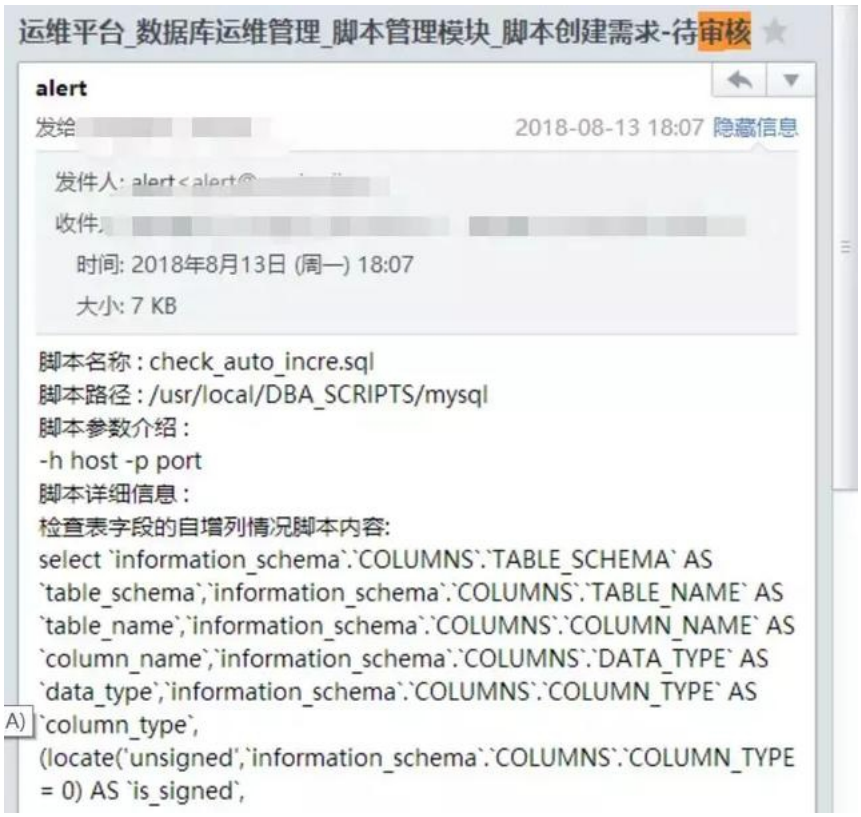
数据库层面的就是脚本的提交，基本通过前端的输入，提供了脚本内容，脚本的状态就是“待审核“

这个阶段的意义就是提交脚本的时候压根不需要声明脚本的路径，这个工作是在审核的时候来做的。

而重要的工作就是脚本审核阶段，脚本审核主要是完成两件事情，一个是脚本的路径规划，另外一个是脚本在中控服务器上生成。

这样一来我们就有了一个初版的脚本管理结构，目录都是统一规划的，不是所有的脚本都要融入进来。

我们做了一个初版的脚本提示，如果创建了一个脚本会发送相应的邮件，这样一来这就是一个正式的过程。



有了消息提醒，这就是一个相对完整的审核过程了。

然后来说下**工单管理模块**的建设。

运维工作其实也是一种服务，所以对于运维提供的服务来说，甭管你是使用了高大上的方式或者规范的流程还是手工处理，如果高效完成，那对于应用来说就是大大的赞。

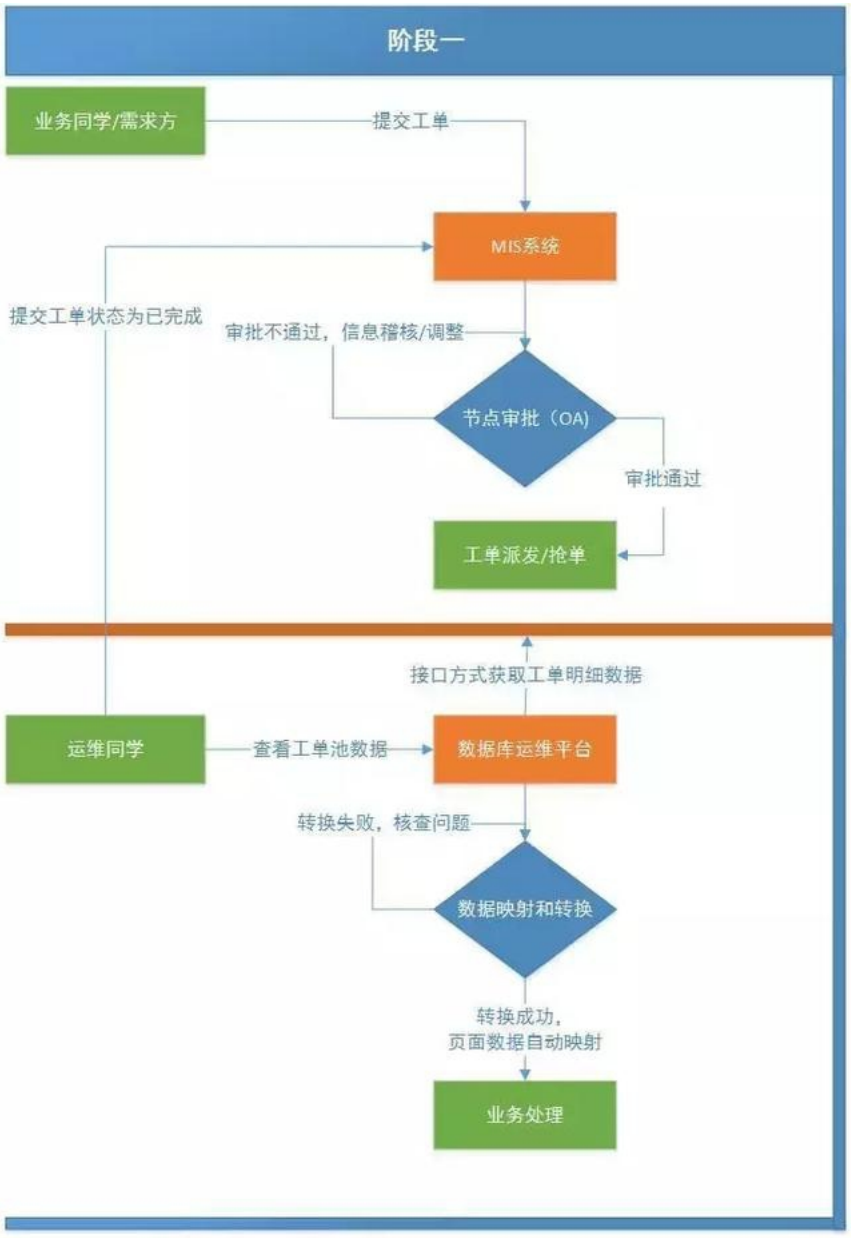
实际上，很多公司里面的工单基本会和两类属性挂钩，一类属性是部门预算，或者说是工时，比如处理一个问题，需要花费2个小时，那么这个服务就可以通过工时的方式来评估服务的结算费用，第二类属性是服务质量，比如工单的处理结果是否满意，是否有一些规范的操作流程等，这些是需要做工单反馈的。

至于工单模块和运维模块的建设，哪个在先？其实这是一个互相促进的过程。早期的工单肯定没有自动化运维的辅助，所以肯定是有工单模块，但是早期的工单模块建设肯定不够完善，基本操作和审批是脱节的，那就需要完成工单的自动化处理。互相促进之后，这就是一个完善的链条了。

所以简单来说，工单系统和运维系统需要对接起来，对接之后就能够互相关注自己特定的业务范围，把每一个环节都做到了可控和可考核。

我补个图来说明一下。

这个是一个初版的工单处理流程图，这个阶段是完成一个初步的系统对接。



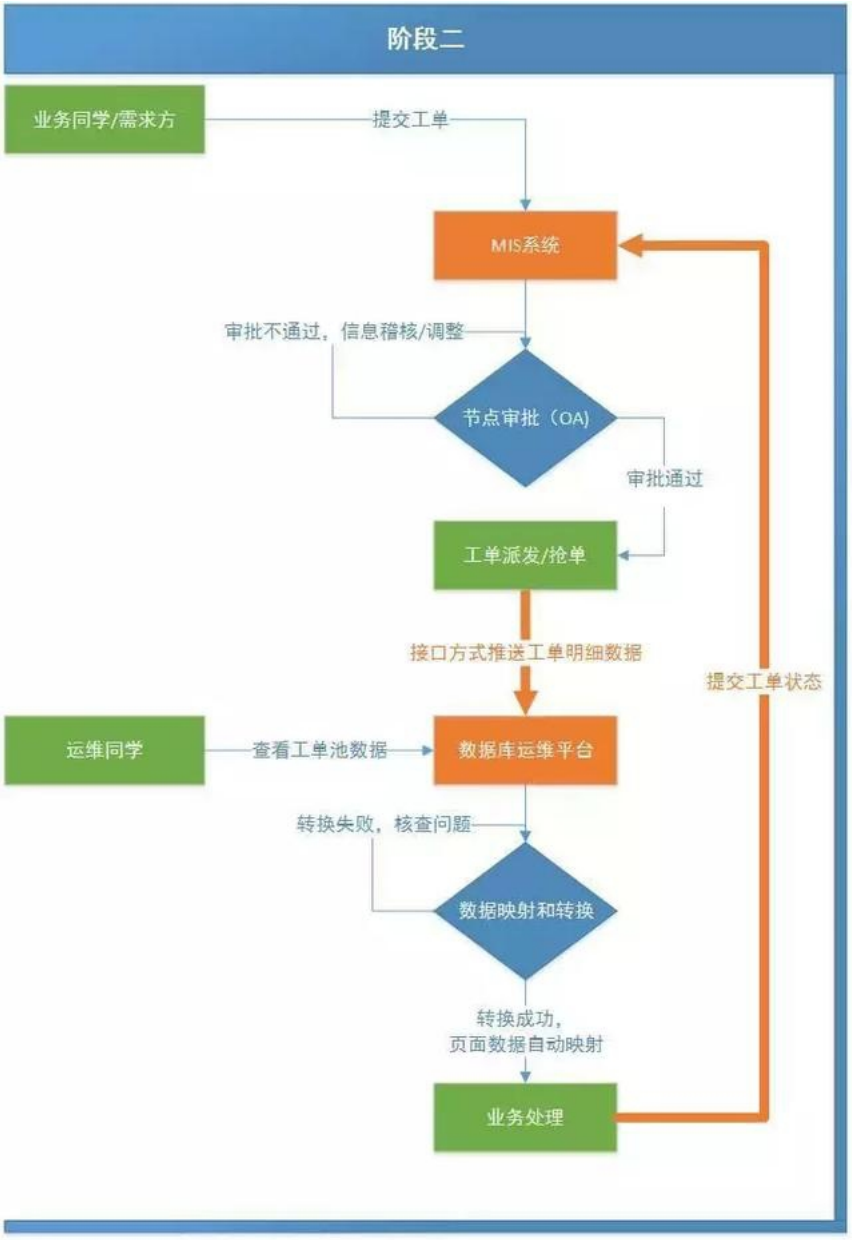
第一步是申请工单系统的接口权限，即工单的审批还是在已有的工单系统中完成，而工单的信息一定有一个流水编号，是一个唯一的 ID 值，我需要的就是根据这个唯一的编号能够从工单系统中得到一个 JSON 数据串，得到这个数据串之后我来解析它把它拆分为符合业务属性的工单。所以所做的工作会分为以下几个步骤：

1）解析工单信息，根据流水号信息解析工单的格式

2）工单拆分，把原来的一个工单拆分为多个业务工单，这个过程对应用同学来说是透明的。比如数据库权限开通的工单，会自动拆分为两个工单，数据库权限工单和系统权限工单。

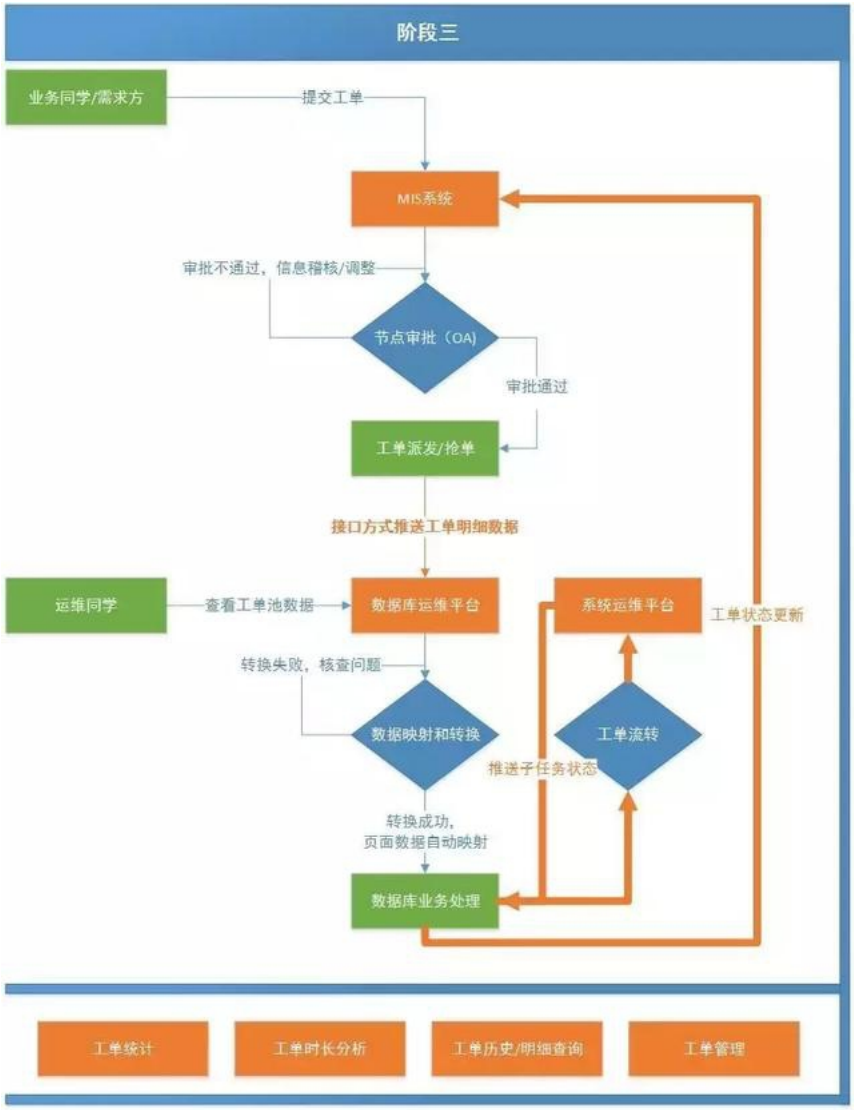
这个阶段的意义在于，两个系统开始对接起来了，虽然不是一种很自然的对接方式，但是彼此打开了一扇窗。

第二个阶段就可以更近一步，这个时候我们可以对工单系统开放接口，让他们把数据推送过来，就不用我们去拉取了。这将是一个自动的推送过程，可以省去很多的检查和反复确认环节。



这个阶段的工作的一大亮点就在于我们可以在工单拆分为业务工单，处理完成之后，确认工单完成，让工单系统开放一个写入接口，我们把工单的状态回传过去。这样业务操作就形成了一个闭环。形成闭环是这个阶段最重要的一件事情，这样审批的关注审批环节，业务操作的关注业务操作环节，各司其职。

第三阶段是更大的一个阶段，比如我们的工单可以和外部的系统通过接口交互，那么我们就可以和其他系统开始打通这个链路。 这是一个更大更全面的过程，会有更多的事情和接口需要对接。



这个阶段的意义就在于，这是一个全链条的过程，我们可以在这个阶段更多的挖掘运维数据的价值，比如工单的处理效率，工单的数据统计分析，工单的指派，业务工单拆分 逻辑等。 这些都可以逐步的细化和改进。

所以逐步迭代，完成系统的对接，完成业务支持，不知不觉，运维工作已然发生了很大的变化。

这也是我给大家分享的初衷所在。