链滴

云平台之 SaaS 随想

作者：88250

原文链接：https://ld246.com/article/1405994194928

来源网站：链滴

<h3>SaaS 平台</h3>
<h4>以应用为中心</h4>
“平台”本来就比较泛，再加上“SaaS”的话就更飘渺了。
我们先从一个简单的场景来看：
<blockquote>
开发者开发应用后在市场上线
用户购买应用使用
开发者通过市场反馈调整运维，为后续版本计划提供依据
新版本上线，用户升级使用
</blockquote>
这是以应用为中心的一个闭环（市场-开发-运维-市场），实现了应用的整个生命周期，我们可把平台看成是这个场景的支撑，场景中的所有活动都是在平台上完成的，整个场景就是一个 SaaS 生系统。
<h4>组成元素</h4>
在这个 SaaS 生态系统中，我们可以简单总结出以下几个必须的组成元素：

开发者：个人/组织，要做的事情是开发应用、运维应用
运行环境：应用程序实际运行的环境，要解决的是如何接入/部署应用
运维控制：应用运行情况监控，要解决的是动态监控
用户：使用应用的个人/组织，要做的事情是（购买）使用应用
社区：开发者社区、用户社区，供开发者/用户进行分享、反馈
应用市场：应用上架展示，供开发者应用上线，用户（购买）使用

按参与者角色（开发者、平台、用户）把这几个组成元素分类后，SaaS 平台部分包括了：运行境、运维控制、社区、应用市场。这里的“平台”是个广义概念，是多个具体平台的综合。例如“Android 平台”，包括了开发平台、应用市场、硬件平台、开发者社区等。
<h4>面向应用用户</h4>
如果把 SaaS 应用比作是一个游戏，那么，

SaaS 平台制订了基本的游戏规则，并提供了游戏道具
开发者制订了游戏的细节规则，形成游戏玩法
用户选择游戏，玩游戏

最终，用户的体验是该游戏是否好玩，这是由平台和开发者共同决定的。也就是说，SaaS 平台开发者是利益共同体，并且平台的基本规则决定了游戏的质量的起点。
<h4>PaaS 与 SaaS</h4>

从用户角度看：PaaS 面向的是开发者用户，SaaS 面向的是应用用户
从应用角度看：PaaS 侧重应用的 Runtime，SaaS 侧重应用的接入与集成
PaaS 不关注应用业务领域，SaaS 则是某业务领域
PaaS 成功与否看的是开发者的反馈，SaaS 成功与否是应用用户的反馈

另外，SaaS 不是必然包含 PaaS。开发者在选定了 SaaS 后应该也可以选择 PaaS。但这个方式要开发者熟悉多种平台，提高了运维的难度。
<h3>应用引擎</h3>
应用引擎（App Engine）是目前业界实现 PaaS 的主流方式。它至少需要为开发提供以下几个功能：

部署：上传部署包部署
实例管理：启停实例
日志：查看日志，分实例

内部至少需要实现以下几个功能：

请求路由：请求分发到应用进程实例
状态采集：基础设施、实例状态采集
应用隔离：应用之间不能相互影响
实例管理：按需启停
资源控制：应用对资源的使用是受控的
耗用统计：API 调用次数、IO/存储大小
配额模型：量化应用对资源的使用

目前我们熟悉的几个 XAE 都是这样做的，并且从传统的沙箱模型（API 受控）迁移到基于 LXC 容器（Docker）已经是趋势，因为这样对应用开发的限制更小，开发者更容易接受。
<h4>服务</h4>
目前业界主流的 PaaS 中都提供了基础服务，这些服务都是属于技术服务：

缓存
消息队列
消息推送
文件存储
定时任务
...

这一块相对比较固定，调用方式一般都是基于 SDK API，有的也有 RESTful 接口。
<h4>部署包</h4>
以 Java 为例，部署包一般都是 war 包，但除了满足标准 war 结构外，还需要加入一些平台特定配置规则。比如通过配置文件描述 appid（或是通过 war 包名），用于部署时对应到平台上的应用配。也就是说所有的应用配置都是可以做成非包内配置文件的，好比应用上某些地方需要抉择使用数据或配置文件，这是平台设计时需要仔细考虑的。
<h3>应用集成</h3>
应用集成主要是针对 SaaS 而言，用户选择多个应用后可以在一个视图中使用它们，这些应用之也可能存在调用交互。
<h4>视图</h4>
最简单的视图集成方式就是通过导航（图标），用户安装了某应用后，该应用图标就出现在这个户的“首页”视图中。由平台给出集成规则，应用开发时遵循这些规则就可以集成进来。
对于平台来说，这一块很有难度，或者说很难把握：

如果集成规则太复杂，那会对开发者造成很多困扰和不便，但对用户来说就更透明、无缝，用户验会更好
如果集成规则太简单，那对开发者约束较低，但集成度也更低，用户体验可能很难提升

目前业界的大多数 SaaS 在做这一块时都选择了简单的集成方式：接入图标，用户使用时点击图并跳转到对应的应用。深度的集成（样式、交互模式统一）的方式很少见（互联网 SaaS 基本不可行除非已经是业界标准）。
<h4>RPC</h4>
服务端的调用也存在集成，应用之间互调也是 SaaS 需要考虑的场景。这部分可选的做法是 SaaS 提供 RPC 协议实现，这样应用间可以通过统一的调用协议进行互调。当然，也可以通过 HTTP 来实，这样限制更少一些，但平台对应用的管控也会更弱。
<h3>多租户</h3>
多租户支持主要目的是简化应用开发，让应用可以全心全意关注业务逻辑而不是关注租户相关逻，具体细节请参考<a href="http://88250.b3log.org/cloud-app-platform-multitenancy">这里</>。
<h3>开放平台</h3>
前面我们提到过 SaaS 应该是可以接入其他 PaaS 应用的，这类应用我们可以认为是“外部应用。既然是外部应用，那肯定是需要特定的接入规则，可以考虑参考行业标准规范。
<h4>OAuth</h4>
通过该协议，SaaS 可以将服务（甚至是一些应用）暴露为 RESTful 接口给外部应用使用，这样

以充分利用平台的资源，吸引更多的应用进入到 SaaS 这个生态系统中。
<h3>分润 </h3>
最终买单的是用户，SaaS 平台和开发者是利益共同体，所以平台在指定规则时需要考虑好与开者的分润。对于部署在 SaaS 内的应用和外部应用，分润规则应该是不一样的。下面是两种简单的方：

内部应用：通过平台提供的支付接口进行分润，应用好卖，平台跟着受益
外部应用：应用支付给平台配额耗用费用，应用固定支付给平台其资源使用费用

<h3>总结 </h3>
SaaS 需要从至少两方面进行设计：基于特定的 PaaS；可以接入外部应用。自下而上、自上而下包罗万象、井井有条。
平台最终比拼的是应用资源而不是平台本身，尽可能吸引开发者是很重要的成功提。要做到这一点，我们需要：

垂直领域（例如协同办公）
降低开发门槛（减少配置，轻量化 SDK）
概念具象化（例如多租户）
实现采用业界主流技术（Golang/Docker）
支持多种编程语言
活跃开发者社区
