# 前言微服务要素-十二要素（The Twelve Factors）

今天看“如何实现现代应用的快速落地”公开课，提到十二要素，之前文章也提到多次，这里统一汇总下：

目录

[1. 基准代码 一份基准代码，多份部署 1](#_Toc5413)

[2. 依赖 显式声明依赖关系 1](#_Toc11839)

[3. 配置 在环境中存储配置 2](#_Toc749)

[4. 后端服务 把后端服务当作附加资源 2](#_Toc28768)

[5. 构建，发布，运行 2](#_Toc17219)

[6. 进程 以一个或多个无状态进程运行应用 2](#_Toc8147)

[7. 端口绑定通过端口绑定提供服务 3](#_Toc21928)

[8. 并发 通过进程模型进行扩展 3](#_Toc12299)

[9. 易处理 快速启动和优雅终止可最大化健壮性 3](#_Toc20974)

[10. 开发环境与线上环境等价 3](#_Toc11026)

[11. 日志 把日志当作事件流 3](#_Toc11631)

[12. 管理进程 后台管理任务当作一次性进程运行 3](#_Toc24172)

十二要素

如今，软件通常会作为一种服务来交付，它们被称为网络应用程序，或“软件即服务”（SaaS）。“十二要素应用程序”（12-Factor App）为构建如下的SaaS应用提供了方法论：

使用标准化流程自动配置，从而使新的开发者花费最少的学习成本加入这个项目；

和操作系统之间尽可能的划清界限，在各个系统中提供最大的可移植性；

适合部署在现代的云计算平台，从而在服务器和系统管理方面节省资源；

将开发环境和生产环境的差异降至最低，并使用持续交付实施敏捷开发；

可以在工具、架构和开发流程不发生明显变化的前提下实现扩展；

这套理论适用于任意语言和后端服务（数据库、消息队列、缓存等）开发的应用程序。

## 1. 基准代码 一份基准代码，多份部署

基准代码和应用之间总是保持一一对应的关系：

一旦有多个基准代码，就不能称为一个应用，而是一个分布式系统。分布式系统中的每一个组件都是一个应用，每一个应用可以分别使用12-Factor进行开发。

多个应用共享一份基准代码是有悖于12-Factor原则的。解决方案是将共享的代码拆分为独立的类库，然后使用依赖管理策略去加载它们。尽管每个应用只对应一份基准代码，但可以同时存在多份部署。所有部署的基准代码相同，但每份部署可以使用其不同的版本。

## 2. 依赖 显式声明依赖关系

12-Factor规则下的应用程序不会隐式依赖系统级的类库。 它一定通过依赖清单 ，确切地声明所有依赖项。此外，在运行过程中通过 依赖隔离 工具来确保程序不会调用系统中存在但清单中未声明的依赖项。这一做法会统一应用到生产和开发环境。

## 3. 配置 在环境中存储配置

12-Factor推荐将应用的配置存储于环境变量 中 (env vars, env) 。环境变量可以非常方便地在不同的部署间做修改，却不动一行代码；与配置文件不同，不小心把它们签入代码库的概率微乎其微；与一些传统的解决配置问题的机制（比如Java的属性配置文件）相比，环境变量与语言和系统无关。

12-Factor应用中，环境变量的粒度要足够小，且相对独立。它们永远也不会组合成一个所谓的“环境”，而是独立存在于每个部署之中。当应用程序不断扩展，需要更多种类的部署时，这种配置管理方式能够做到平滑过渡。

## 4. 后端服务 把后端服务当作附加资源

12-Factor应用不会区别对待本地或第三方服务。 对应用程序而言，两种都是附加资源，通过一个url或是其他存储在 配置 中的服务定位/服务证书来获取数据。12-Factor应用的任意 部署 ，都应该可以在不进行任何代码改动的情况下，将本地MySQL数据库换成第三方服务(例如 Amazon RDS)。类似的，本地SMTP服务应该也可以和第三方SMTP服务(例如Postmark)互换。

## 5. 构建，发布，运行

严格分离构建和运行

12-facfor应用严格区分构建，发布，运行这三个步骤。每一个发布版本必须对应一个唯一的发布ID。

新的代码在部署之前，需要开发人员触发构建操作。但是，运行阶段不一定需要人为触发，而是可以自动进行。

## 6. 进程 以一个或多个无状态进程运行应用

12-factor应用的进程必须无状态且无共享 。任何需要持久化的数据都要存储在后端服务内，比如数据库。粘性Session是twelve-factor极力反对的。Session中的数据应该保存在诸如Memcached 或 Redis 这样的带有过期时间的缓存中。

## 7. 端口绑定通过端口绑定提供服务

12-factor应用完全自我加载而不依赖于任何网络服务器就可以创建一个面向网络的服务。互联网应用 通过端口绑定来提供服务，并监听发送至该端口的请求。

## 8. 并发 通过进程模型进行扩展

在12-factor应用中，进程是一等公民。 12-factor应用的进程主要借鉴于 unix守护进程模型 。开发人员可以运用这个模型去设计应用架构，将不同的工作分配给不同的进程类型 。

## 9. 易处理 快速启动和优雅终止可最大化健壮性

12-factor应用的进程是可支配的，意思是说它们可以瞬间开启或停止。 这有利于快速、弹性的伸缩应用，迅速部署变化的代码或配置，稳健地部署应用。进程应当追求最小启动时间；进程一旦接收终止信号(SIGTERM) 就会优雅的终止 。进程还应当在面对突然死亡时保持健壮 。

## 10. 开发环境与线上环境等价

尽可能的保持开发、预发布、线上环境相同

12-factor应用想要做到持续部署就必须缩小本地与线上差异。12-factor应用的开发人员应该反对在不同环境间使用不同的后端服务 ，即使适配器已经可以几乎消除使用上的差异。

## 11. 日志 把日志当作事件流

12-factor应用本身从不考虑存储自己的输出流。 不应该试图去写或者管理日志文件。相反，每一个运行的进程都会直接的标准输出(stdout)事件流。开发环境中，开发人员可以通过这些数据流，实时在终端看到应用的活动。

## 12. 管理进程 后台管理任务当作一次性进程运行

一次性管理进程应该和正常的 常驻进程 使用同样的环境。这些管理进程和任何其他的进程一样使用相同的代码和配置，基于某个发布版本运行。后台管理代码应该随其他应用程序代码一起发布，从而避免同步问题。所有进程类型应该使用同样的依赖隔离技术。12-factor尤其青睐那些提供了REPL shell的语言，因为那会让运行一次性脚本变得简单。

---------------------

作者：zeb\_perfect

来源：CSDN

原文：https://blog.csdn.net/zeb\_perfect/article/details/52536411

版权声明：本文为博主原创文章，转载请附上博文链接！