十二要素 APP 开发法——通往美好的分布式 Web APP

作者：Harvey Jones 译者：杨志昂 策划：王文婧
阅读数：358 2019 年 8 月 29 日 14:09

> 随着 Web APP 的生态系统的巨变，开发现代复杂的 Web APP 需要兼顾可移植性、可扩展性、部署灵活性以及降低开发成本等多方面目标。本文介绍了一个名为“十二要素 APP 开发法”的流程，这一开发方法为构建分布式 Web APP 和微服务提供了良好的基础，让开发过程变得更顺畅，还可以提高生产效率。

随着云计算的出现，Web APP 的生态系统也发生了翻天覆地的巨变。但实现这一过渡的过程也并非唾手可得。以前习惯于把 APP 部署到单独一台服务器的开发人员现在必须对多台服务器进行处理，并且还要保证负载的均衡分配。开发人员还必须考虑不同文件之间的组合，而且这些文件可能来自于不同的服务器。

为了解决开发者产生的种种困惑， Heroku 联合创始人 Adam Wiggins 和他的团队提出了一个名为“十二要素 APP 开发法”的流程。由于该方法建立在开发大量 APP 的经验之上，这些指导原则确保了开发活动的生产力和可扩展性。这种方法的形式主要受 Martin Fowler 的《企业 APP 体系结构的模式》（Patterns of Enterprise Application Architecture）和《重构》（Refactoring）两本著作的影响。

有人可能会问，为什么要遵循这个方法呢?

不遵循这个方法，就意味着你将会重蹈这种方法的创建者和其他高级开发人员的覆辙，在解决系统性问题上浪费时间，走太多的弯路。相反地，遵循这种方法可以让你利用现有的、经过实践锤炼的经验，从长远来看，这可以节省你不少时间。

考虑到这些原因，让我们先来回答一个最明显的问题：具体而言，到底什么是“十二要素 APP 开发法”呢?

什么是十二要素 APP 开发法?

十二要素 APP 开发法是一组指导原则，为开发现代复杂 Web APP 提供了最佳实践和条理分明的方法。该方法所建议的原则并不局限于任何特定的编程语言或数据库，这些指导原则也有足够的灵活性，可以用于任何编程语言或数据库。

在讨论具体的十二要素之前，我们先来重点关注一下目的，这也很重要。这一套指导方针旨在实现两个特定的目标，它们是：

- 十二要素 APP 开发法具体想要实现什么?
- 它是如何实现这一目标的?

让我们看看十二要素 APP 开发法的官网对这些目标有什么说法。

十二要素 APP 开发法的目标

相关阅读
前端之巅
2019 年 6 月 20 日
使用自选择创建团队
2017 年 11 月 20 日
通过卓越生产实现复杂系统中的可持续运营
2019 年 7 月 14 日
打造工业级推荐系统（九）：从零开始入门推荐算法工程师
2019 年 7 月 19 日
Java 发明者 Sun 公司的出生与死亡
2018 年 12 月 27 日
WebAssembly — 技术变革，未来已来
2019 年 8 月 12 日
通过 Lisp 语言理解编程算法：数据结构篇
2019 年 8 月 27 日

1.“使用声明式格式进行自动化设置，让新加入项目的开发人员付出的时间和成本最小化”

通过声明式语法来自动化配置开发环境，可以让新加入的开发人员配置项目初始环境和加入项目生态系统的时间和成本最小化。尤其是当 APP 所提供的服务规模扩大时，这种做法非常有用。

2.“与底层操作系统保持干净、简单的交互，在多种执行环境之间提供最大的可移植性”

通过将软件元素与底层操作系统解耦，在不同的执行环境之间提供最大的可移植性。由于软件已经变得相对独立，与平台无关，因此它就能在多种执行环境中发挥更大作用。

3.“（这些 APP 应该）适合部署在现代化的云平台上，不需要服务器和系统管理活动”

开发易于在现代化的云平台（如 AWS 、 Azure 、 GCP 等）上部署的软件，就不需要你再去对自己搭建的服务器进行配置或管理，这也为 APP 提供了极大的灵活性。如果有人想在自己非云端的基础设施上设置 APP，部署过程仍可得到大大地简化。

4.“最小化开发和生产之间的差异，使用持续部署以获得获得最大的灵活性”

通过应用持续部署（CD）的概念，最小化生产环境和开发环境之间的差异。这样做也能让开发人员的调试过程变得更容易。

5.“（这些 APP）可以在不需要对工具、架构或开发实践进行重大更改的情况下进行扩展。”

这允许软件规模毫无困难地扩大或缩小，这也是当今软件世界的一个基本特性。

现在，我们来讨论这些目标是如何通过以下十二个要素来实现的，而这些要素是这个方法的核心内容。

十二要素 APP 开发法

此方法中的每个要素在其建议的实践和体系架构中都扮演着独特而重要的角色。让我们先来简单地介绍下每个要素。

1. 代码库

> “用版本控制管理一套代码库，并在此之上进行多次部署”

这意味着应该由一个版本控制系统集中地维护和管理项目的所有源代码，这会让开发人员更容易访问代码。虽然代码库只有一套，但部署可以有多个不同的环境。

第一个要素中包含的建议提出了仅有一套代码库的原则，不要构建另一套代码库。为了在不同的环境中进行部署只需要做出不同的环境配置。例如，一个代码库不应该为开发和生产维护两个存储库。这种做法并不推荐。

不同的环境代表不同的部署状态。而无论这些状态如何不同，它们都共享相同的代码库。从技术上讲，可以将这些不同的部署状态看成 Github 这样的版本控制系统为不同执行环境拉出多个不同分支。

一起来看另一个例子，帮助你更好地理解这一点：假如你正在与 3 个团队成员构建一个名为 Todo 的 APP。一个下下策是在每个成员的系统中各自创建三个文件夹，名为 todo-app-development、todo-app-staging 和 todo-app-production。这样的版本管理方式，其实很难跟踪每个团队成员的进度和整个项目的进展。

而一个好的方法是选用任何一个集中版本控制系统，如 Github、Gitlab 等，创建一个名为 todo-app 的存储库，然后为之创建两个分支，分别是开发和展示。然后将“主分支”命名为生产 / 发布分支，当然每个

团队对分支名称的意见可能有所区别，这无所谓。

2. 依赖项

> “显式声明和隔离依赖项”

这个要素的主要意思是，不应该假设基于代码库的目标系统已经完全安装了所有依赖项（即运行时所需要的外部库）。

将依赖项连同代码库一起上传到线上并不是一个推荐的实践，因为它可能会产生诸如底层平台依赖项之类的问题。例如，如果上传的模块是从 Windows PC 角度而言的依赖项，而使用 Mac OS 的开发人员下载了该代码库。毫无疑问，他们会在运行这个项目时遇到麻烦。

根据 APP 开发所使用的技术栈，通过读取表示依赖项名称和版本的特定文件，使用诸如 npm 、 YARN 或类似的包管理器去下载各自系统上的依赖项，这种方法始终是一个值得推荐的优秀实践。

3. 配置

> “在环境中存储配置”

配置信息永远不应该存储在代码源文件中。我们肯定不希望像数据库连接信息或其他机密数据这样的敏感信息被公开。此外，我们的配置主要取决于它所处的运行环境。

例如，如果你正在开发一个医院管理系统，该系统在实际投入使用时会有诸如 AWS DYnamo DB 这样的云数据库服务凭证，但在开发过程中你会在本地的 MongoDB 服务器上对这个 APP 进行测试，你会选择一次又一次地更改源代码，还是选择仅仅修改一组配置变量去改变其运行环境（就像 node.js 中的 dotenv 那样），让你的代码运行得完美顺畅呢?

如果是我，当然会选择后一种方法。

4. 支持服务

> “把所有支持性服务都视为附加资源”

首先，支持性服务的定义是任何依赖于网络连接才能成功运行的资源。它可以是像 PostgreSQL 这样的数据库或其他任何资源。

为了方便起见，假设我们正在使用 Nodejs 和 PostgreSQL 开发一个项目。在开发环境中，我们在自己的机器上运行一个本地的 PostgreSQL 服务器。现在，我们想通过在另一台服务器上运行这个数据库来进入生产环境。我们该怎么办呢?

我们不应该更改代码，而应该去更改可用配置信息来切换环境。于是，两种环境唯一的区别应该是生产环境的 URL 不同于开发环境的 URL。

5. 构建、发布和运行

> “严格地隔离开构建和运行阶段”

这一要素通过为每个 APP 组划分阶段来隔离各个时期不同的关注重点。它分为三个阶段，描述如下:

- 第一，构建阶段。开发人员对这个阶段有完全的控制，标记新版本并修复任何 bug。最好只在构建阶段更改代码，而不要在其他阶段对代码产生干扰。

- 第二，发布阶段。在这个阶段，构建阶段的代码会在目标环境上执行，这提供了一种真实运行环境的模拟。执行测试以检查 APP 在真实环境下是否一切正常。
- 第三，运行阶段。这是运行 APP 的最后一个阶段。它不应受到任何其他阶段的干预。

假设你正在为一家餐厅构建一个管理系统。如果忽略这些阶段而直接开发，则会浪费很多时间去解决本地系统和目标餐厅系统之间的大量问题。

6. 进程

> “将 APP 作为一个或多个无状态进程来执行”

代码库的状态控制部分应该与进程或 APP 的实例进行分离。状态控制部分应该只存在于数据库和共享存储中。

当在云平台上进行多节点部署以实现该 APP 的可扩展性时，这个要素非常有用。数据不会持久地存储在云平台上，因为如果其中任何一个云节点崩溃，数据就会丢失。例如，会话相关的数据就不应该存储在 APP 的进程中，因为如果它是基于云进行存储的，并且 / 或者某个节点以某种方式崩溃，该 APP 将要求用户再次登录才能继续使用。我始终认为，在实践中将会话数据存储在诸如 Redis 之类的数据存储中会是一种不错的选择。

7. 端口绑定

> “通过端口绑定输出服务”

你的 APP 服务应该可以通过 URL 被其他服务访问。通过这种方式，该 APP 的服务可以在需要时充当其他服务的资源。这个要素促进了 APP 的自包含的独立性。你可以使用这个概念为其他 APP 构建 REST 风格的网络接口，即 REST API。例如，你希望自己开发的社交媒体 Web APP 中的帖子能够被其他第三方 APP 访问，你就可以为它们提供类似于 https://www.xyz/api/posts:5000 的 API URL，以及一些请求令牌，以便其他第三方访问你的数据。这是最好的方法，使用之后你的 Web APP 就已经成为另一方的支持服务。

8. 并发性

> “通过进程模型向外扩展”

APP 中的每个进程都应该能够根据需求进行扩展、重启或自我克隆。这样做可以提高 APP 的可扩展性。

使用上面提到的方法，你可以通过将每个工作负载分配给某个类型的进程（PID）来构建能够处理不同工作负载量的 APP。例如，HTTP 请求可以委托给 Web 进程处理，而长时间执行的后台任务则可以委托给 Worker 进程。

9. 善后处理

> “用快速启动和优雅退出，来最大化 APP 的健壮性”

这个原则表明，APP 的进程应该花费更少的时间，从而实现快速启动和退出。除此之外，APP 还应该能够顺畅地处理各种故障。如今像 Docker 这样的容器非常有助于执行和实现这样的功能。例如，可以使用健壮的队列后端系统，如 RabbitMQ 来处理进程的突然死亡和关闭。在这种情况下，当客户机断开连接或关闭时，正在处理的任务将返回到队列。

10. 开发 - 生产的等价性

> “尽可能保持开发、展示和生产环境的相似性”

参与该 APP 开发项目的团队应该使用相同的操作系统、支持服务和外部依赖项，以保证开发和生产之间环境的差异最小。这样做的一个好处是，开发阶段所需的时间会更少。这同时也印证了快速应用程序开发（RAD）的概念。通过减少开发和生产阶段之间的差异，持续部署的过程也会变得更加顺畅。

11. 日志

> “将日志视为事件流”

你的 APP 不应该负责日志的存储和管理，它应该随着 APP 的执行相应地打印以检查 APP 的处理流程是否正确。像 Node.js 中的 Bunyan 这样的程序可以帮助我们在各种环境中检查 APP 的执行流程，这里各种环境指诸如测试和开发这样的环境。这样的程序可比简单的“console.log()”功能要强大得多。

12. 管理流程

> “将监管 / 管理任务作为一次性进程”

最后一个因素，建议你的管理任务应该从类似生产服务器的环境中执行。管理任务可以执行数据库迁移，并从 APP 中收集分析数据，以从中获得有用的信息。这些任务通过发布的 APP 代码在 APP 上运行。

十二要素 APP 开发法的好处

我们已经看到了“十二要素 APP 开发法“如何使 APP 的开发过程变得更顺畅，还可以提高生产效率。此外，花在理解和实现这些指导方针上的时间有助于我们在软件开发上节省可观的成本。

在某些情况下，在执行中偏离上面的几个要素（例如，日志）也可能是行得通的，但是最好尽可能采用以上所有的十二个要素。

如果你正在设计微服务架构，请务必考虑这些要素并严肃认真地对待它们。当你在 5 个不同的执行环境中运行 10 个以上的服务时，现在看起来琐碎的事情可能就会变得非常重要。如果你已经在运行微服务，请对照检查看看是否有遗漏的要素，也许它们可以帮助你解决之前没有注意到的问题。

总之，这些开发指南为构建分布式 Web APP 和微服务提供了良好的基础。这种方法可以帮助你在很长时间内平稳地扩展和维护你的 APP。

祝你好运，设计愉快!

作者介绍：

Harvey Jones，在编写优美的代码方面，他绝对是同行中的佼佼者。他目前在 CarbonTeq 公司工作。

原文链接：

The Twelve-Factor App Methodology — Key to Amazing Distributed Web Applications

文化 & 方法　技术管理　微服务　Web框架　部署

1 人喜欢

收藏 评论 微信 微博

写下你的想法，一起交流

发表评论

注册/登录 InfoQ 发表评论

注册/登录