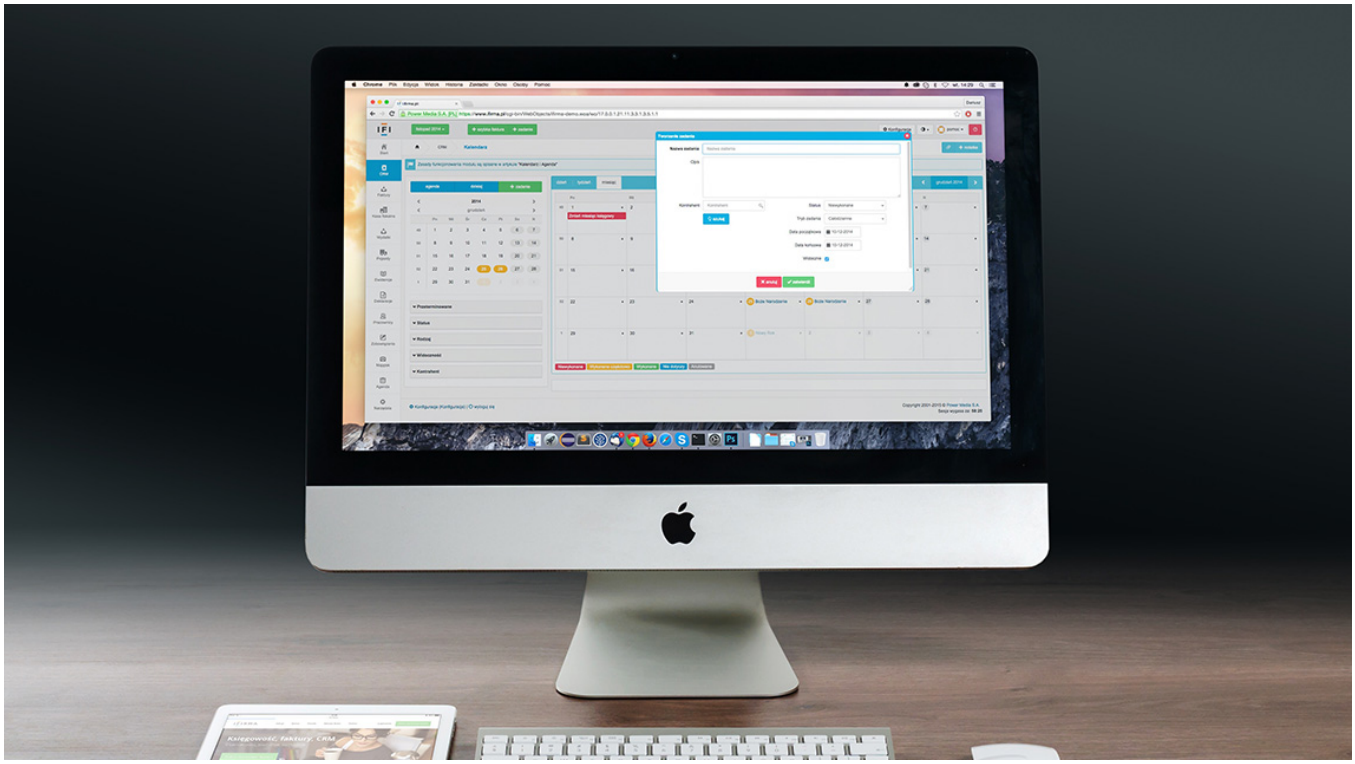
10 | 让环境自己说话，论环境自描述的重要性

2018-07-26 王潇俊



10 | 让环境自己说话，论环境自描述的重要性
朗读人：王潇俊　11′01″ | 5.05M

在前两篇文章中，我从现实需求、成本与效率的角度，分析了对环境管理者来说最重要的一个问题，即到底需要多少套环境来支撑持续交付。如果你已经从中能掌握了一些环境管理的窍门，那么你基本就可以搞定对环境管理的宏观把控了。

但是，除了宏观的把控和管理外，即使只有一套环境，你还是有可能陷入无穷无尽的细节工作中。因为在日常的环境管理过程中，环境配置才是工作的重头和难点。那么今天，我就来跟你详细说说有关环境配置的问题。

从我的实践经验看，要想把环境配置这件事做好，就是要做到让环境自己能说话。

要做到这点，首先需要定义配置的范围。

从面向的目标来看，环境配置大体上可以分为两大部分：

1. 以环境中每台服务器为对象的运行时配置；
2. 以一个环境为整体目标的独立环境配置。

服务器运行时配置

以一个 Java Web 应用为例，需要哪些运行时配置呢？

1. 安装 war 包运行依赖的基础环境，比如 JDK，Tomcat 等。
2. 修改 Tomcat 的配置文件，关注点主要包括：应用的日志目录，日志的输出格式，war 包的存放位置。Tomcat 的 server.xml 配置包括：连接数、 端口、线程池等参数。
3. 配置 Java 参数，包括 JVM 堆内存的最大最小方式，GC 方式、参数，JMX 监控开启等。
4. 考虑操作系统参数，比较常见的一个配置是 Linux 的文件句柄数，如果应用对网络环境有一些特殊要求的话，还需要调整系统的 TCP 参数等配置。

经过上面这 4 步，一个简单的运行时环境的配置就算是完成了， 可以开始运行一个程序了。是不是感觉有点复杂呢？

而这，对正常的运行时配置管理来说，只不过是冰山一角而已。

我们不光要考虑单个实例初始化配置，还要考虑每次 JDK、Tomcat 等基础软件的版本升级引起的运行时配置的变更，而且这些变更都需要被清晰地记录下来，从而保证扩容出新的服务器时能取到正确的、最新的配置。

另外，对于一个集群的服务器组来说，还需要强制保证它们的运行时配置是一致的。

独立环境配置

独立环境配置的主要目的是，保证一个环境能够完整运作的同时，又保证足够的隔离性，使其成为一个内聚的整体。

所以，要让一个环境能够符合需求的正常运作，你需要考虑的内容包括：

1. 这个环境所依赖的数据库该如何配置，缓存服务器又该如何配置。
2. 如果是分布式系统，或者 SOA 架构的话，就需要考虑服务中心、配置中心等一系列中间件的配置问题。

其中，最为重要的是配置中心的配置。只有先访问到正确的配置中心，才能获取到其他相关的环境配置或者应用配置信息。也就是说，如果配置中心的配置错了，那么环境就会陷入混乱状态。

1. 要考虑访问入口问题。 这套环境的入口在哪里？是一个站点还是一个服务入口？
   如果是一个站点的话，那这个站点的访问域名就需要被特殊配置。如果这是一个内部环境的话，那么这个内部域名的 DNS 解析也需要被配置。如果这套环境中有多个 Web 应用，那么你就要考虑 7 层路由的配置问题了。

2. 还要配置环境对应的基础服务，比如监控，短信，搜索等。

读到这里，如此多的与环境有关的配置，有没有让你觉得太复杂了。

再想象一下，如果你的环境要承载多种语言栈，各类应用依赖的基础软件也不同，环境和环境之间有各种关联设置，数据库的连接分配，环境中负载均衡的设置，等等。是不是让你感觉有些焦虑？

如果每天都要和这样的工作做斗争，那简直就是一场噩梦。更别提在这样的环境下，完成持续交付了，那简直就是难如登天。

虽然环境配置有这么多糟心的待处理事项，但是环境本身也是一个非常强大的工具，本身包含非常多的信息，如果这些糟心的事情环境能和你一起来解决，那就简单了，也就是我所说的让环境自己来说话，那么接下来就看看怎么做到吧。

环境一定要标准化

解决复杂问题的办法，无非是先将其分解，再将其简单化，对环境配置这个难题来说也是同样的道理。想要解决它，首先得要想办法分解、简化它。

最好的简化方法，莫过于标准化了。所谓标准化，就是为了在一定范围内获得最佳秩序，对实际的或潜在的问题制定共同、可重复使用的规则。

标准化也就是让环境学会了一门统一的语言，是自己说话的前提。

按照这个思路，我们首先可以实现对语言栈的使用、运行时配置模板、独立环境配置的方法等的标准化：

1. 规定公司的主流语言栈；
2. 统一服务器安装镜像；
3. 提供默认的运行时配置模板；
4. 统一基础软件的版本，以及更新方式；
5. 在架构层面统一解决环境路由问题；
6. 自动化环境产生过程。

看到这里，你可能感觉需要标准化的内容也是多种多样的，而且每个公司的具体情况也不同，那么标准化实施起来也必定困难重重。

从我的实践经验来看，建议你在实施持续交付的同时，去推动形成以下几个方面的规范：

1. 代码及依赖规范；
2. 命名规范；
3. 开发规范；
4. 配置规范；
5. 部署规范；
6. 安全规范；
7. 测试规范。

其实，不管是持续交付还是架构改造，标准先行都是技术实施的前提条件。

约定大于配置

讲到这里，你可能也会疑惑了，和环境有关的内容实在是太多了，即使有了标准化，怎么可能都通过配置实现呢？

举个例子，代码的部署路径，标准化后所有服务器的路径都应该遵循这个标准，但是不可能在每台服务器上都去定义一个配置文件或环境变量来标示它，也没有这个必要。

实际上，你也从来都没有疑惑过部署路径的问题，因为从你来到公司起，它就已经是约定俗成了。而且，每家公司都是这样的，难道不是吗？

像代码的部署路径这种情况，我们就把它叫作“约定大于配置”，在实际工作中，还有很多类似的场景，你完全可以利用这套方法，简化环境配置。

比如，每个环境的域名定义，可以遵循以环境名作为区分的泛域名实现；又比如，可以用 FAT，UAT 这样的关键词来表示环境的作用；又比如，可以约定单机单应用；再比如，可以约定所有服务的端口都是 8080。

“约定大于配置”的好处是，除了简化配置工作外，还可以提高沟通效率。团队成员一旦对某项内容形成认知，他们的沟通将不再容易产生歧义。

“约定大于配置”相当于赋予了环境天生的本能，进一步加强了环境的自我描述能力。

让环境自己能开口说话

有了环境标准化，以及约定大于配置的基础，你就可以顺利地让环境自己开口说话了。

也就是说，通过环境的自描述文件，让环境能讲清楚自己的作用、依赖，以及状态，而不是由外部配置来解释这些内容。

以一台服务器为例，一旦生成，除了不能控制自己的生死外，其他运行过程中的配置，都应该根据它自身的描述来决定。

那么，如何让服务器自己说话呢？

首先，需要定义 Server Spec。

这是重中之重，在服务器生成时，写入它自己的描述文件。我们通常把这个文件命名为“Server Spec”。在这个文件里，记录了这台服务器的所有身份信息，包括：IDC，型号，归属环境，作用，所属应用，服务类型，访问路径等。

其次，解决配置中心寻址。

中间件根据 Server Spec 的描述，寻找到它所在环境对应的配置中心，从而进一步获取其他配置，如数据库连接字符串，短信服务地址等等。

最后，完成服务自发现。

其实这就是一个服务自发现的过程。根据服务类型，访问路径等，还可以自动生成对应的路由配置，负载均衡配置等。

总结来说，我们是在尝试把环境配置的方向调个个儿：由原来外部通过配置告知环境应该干什么，转变成环境根据自身的能力和属性，决定自己应该去干什么。

这种尝试，标志着环境配置能力的质的飞跃。一台服务器可以实现自描述，你同样就可以把这个方法推广到所有服务器中。同理，一个环境可以实现自描述，你就可以把自描述的方式扩展到所有环境中。

从此，环境配置将变得不再艰难。

总结

我主要围绕环境配置的问题，讲了它的内容和一些特性，以及简化和优化的一些方案。

一定要意识到，环境配置是非常复杂的，直接影响你的环境治理能力，而环境治理能力又直接影响着持续交付的能力。但是我们还是可以通过：标准化、约定、自描述等方式去简化和优化环境配置工作。

我们的目标是，让环境自己能说话。

思考题

在你的公司，这些环境配置相关的工作由谁来完成？又由谁来为他们制造工具和提高工作效率？

欢迎你给我留言。

通过留言可与作者互动