



阿里中间件团队博客

致力于成为中国第一，世界一流的中间件技术团队

一篇好TM长的关于配置中心的文章

📅 2016-09-28 | 坤宇 | 📁 [配置中心](#)

配置 (Configuration)

配置(Configuration) 这个概念每个技术人都不陌生，可以说一个不提供几个配置参数的系统都不好意思上线跟别的系统打招呼。那么为什么会是这个样子呢，究其本质是我们人类无法掌控和预知一切，映射到软件领域上，我们总是需要对系统的某些功能特性预留出一些控制的线头，以便我们在未来需要的时候，可以人为的拨弄这些线头从而控制系统的行为特征，我把它叫做“*系统运行时(runtime) 飞行姿态的动态调整*”。

举个简单的例子，

```
logLevel = INFO
```

系统正常飞行的时候，我们希望其只输出INFO级别的日志信息，在生产环境中我们甚至希望只输出WARNING/ERROR级别的日志，系统出毛病了，再将日志输出动态的调整成包含诊断信息的DEBUG级别或者TRACE级别。

配置

配置是什么？

key=value?

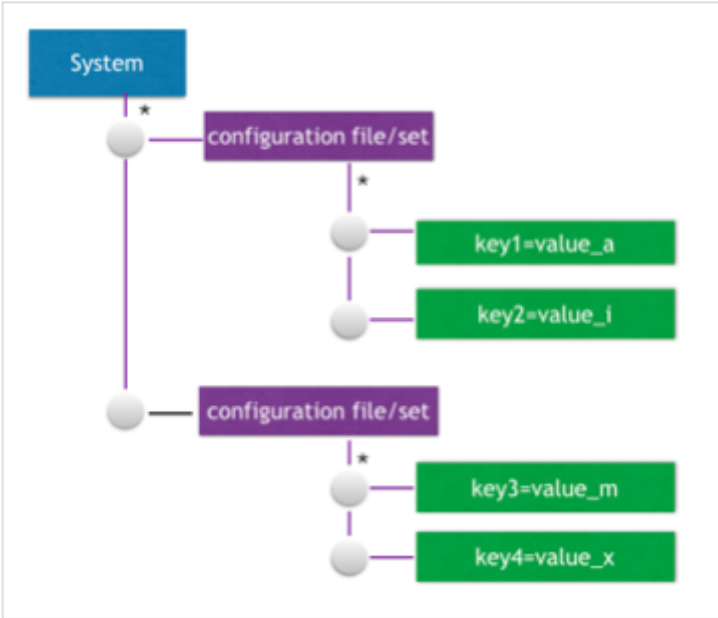
程序 运行时 动态 调整行为 的能力！

软件的老友 - 配置文件

在那个单机即系统的时代，我们基本都是在用配置文件来存储配置项，一个配置项，就是如上面的logLevel那样的一个含有 = 表达式，如下：

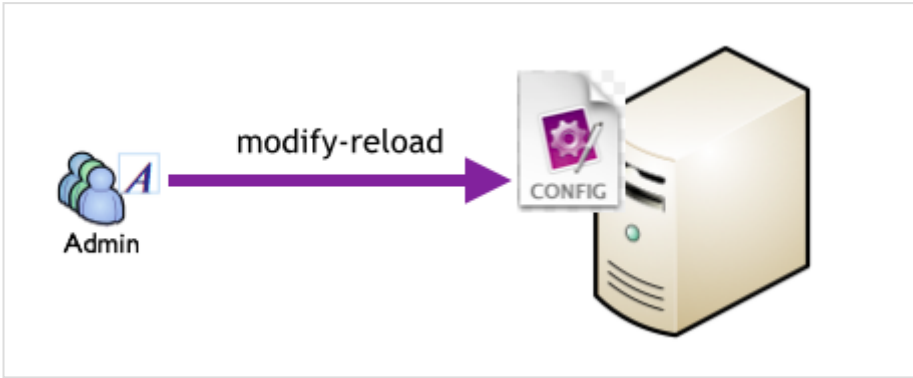
```
config_key = config_value           // value1 or value2, you can choose one from the config_v
```

一般来说 `config_value` 应该是一个有限空间的值集合，应该是有选择余地的，如果 `config_value` 没得选择，那么我只能认为这个配置项一定特么在逗我。而一个配置文件一般是一组配置项的集合或者叫配置集，一个系统根据逻辑模块划分，可以有1到多个配置文件。如下图：

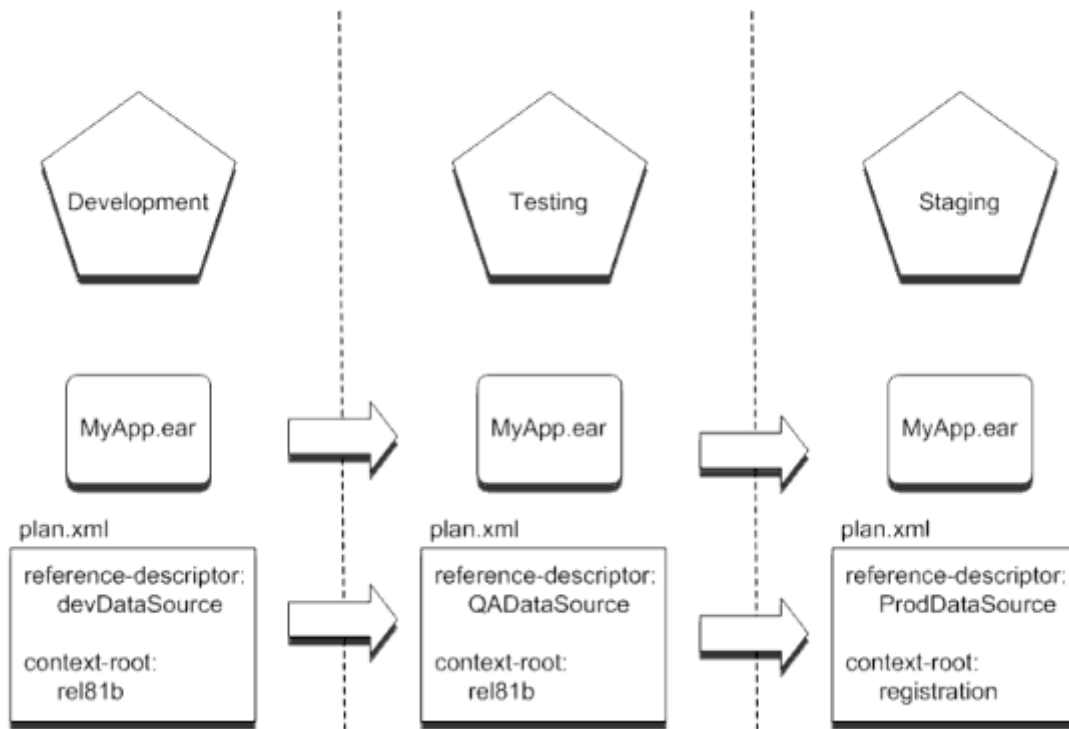


在集中式开发时代，配置文件基本足够用了，因为那时配置的管理通常不会成为一个很大的问题，简单一点来说，系统上了生产之后，如果需要修改一个配置，登录到这台生产机器上，vi修改这个配置文件，然后reload一下并不是什么很大的负担。

如下图:



所以曾经的企业级应用架构标准J2EE里并没有制定关于配置管理这一块的任何标准。当然一些Follow J2EE标准的厂商，如以前Oracle的中间件WebLogic在实际实践时，因为很多大企业客户的环境也挺复杂的，所以WebLogic在这一块还是做了一些工作，其支持一个叫做Deployment Plan的特性，如下图:

Figure 4-2 Single Deployment Plan Workflow

其背后的本质就是开发人员(dev)打出来的应用war包里面的配置文件都是一些Placeholder, 在部署人员(ops)部署war包的时候, 为目标环境提供与之匹配的的depoloy plan xml文件, WebLogic Server本身在deploy这个stage将war包配置项的placeholder值替换成plan里面的值。

分布式系统给系统配置管理带来的挑战

关于什么是分布式系统, 本文不再赘述, 毫无疑问今天阿里的系统就是一个大型的、服务化的、复杂的、分布式系统实现之一。在这个领域有3本书值得反复阅读<<分布式系统概念与设计>> <<分布式系统原理与泛型>> 以及 Distributed Systems For System Architects, 有意思的是这三本书只有最后一本在21.3小节简单的提了一下 Configuration Of Distributed Systems, 里面简单的说了一下静态配置和动态配置的概念和区别 "...System configuration may be static or dynamic..." 这说明什么? 这说明我们阿里技术人包括我们中间件今天面临的很多问题和领域已经进入深水区, 已经没有人会直接给你提供这个领域清晰的解决方案, 我们自己正站在前沿, 而我们的成功的或者失败的探索, 其经验和成果都应该总结并分享给整个业界。

在过去的大约15年左右, 软件工程在如何持续演进软件以适应一直要变的需求的方法论上有了很多的突破和大量的实践, 在这个领域, 从面向对象设计方法论, 到极限编程, 到敏捷开发、持续集成、单元测试等等, 理论和实践都已经比较成熟了, 关键是配合这一套方法论的配套的工具和软件集都变得非常的成熟。

这里面的一个非常有意思的东西是关于系统的演进或者进化论(Software Evolution), 一个系统或者说软件从被创造出来之后会经历研发、测试、到最后的go live, 上了生产系统。那么这个之后, 如何持续并且无痛的为其添加新行为或者调整已有行为的表现特征? 这确实非常复杂, 尤其是要达到无痛的这个目标, 毕竟线上系统, 调

整即意味着可能出故障。系统的动态配置管理毫无疑问是其中的一个小部分，如果每一个系统行为的任何一个微调都需要将整个系统停机，重启或者甚至重新构建、发布部署来实现，那要达到无痛这个目标恐怕难度更高。

在分布式系统中，一次构建、发布、上线是非常非常重的一个过程，它不像单机时代那样重启一台机器、一个进程就可以了，在分布式系统中，它涉及到将软件包(例如war)分发到可能超过几千台机器，然后将几千台机器上的应用进程——重启这么一个过程，超过2000台机器的一个应用一次完整的发布过程需要多长时间，相信很多核心系统的小二都深有体会。

那么如何在不停应用集群的情况下，调整整个集群的运行时的行为特征（即系统运行时的飞行姿态），是一个分布式系统必须回答的一个问题。从这个角度讲,我们认为:

每一个大型分布式系统都应该有一个配置中心!

- YES!
 - 业界正在跟随我们，逐渐走向配置中心解决方案
 - [spring cloud config server](http://springcloud.org/)
 - <http://scottfrederick.cfapps.io/blog/2012/05/22/Custom-PropertySource-in-Spring-3.1---Part-1>
 - <http://potocki.io/post/141230472743/configuration-management-for-distributed-systems>
 - [cfg4j](#)

现在我们很容易理解，其实我们平时常见的分布式系统的配置变更，诸如:

- 线程池、连接池大小
- 开关、预案、限流配置
- toggleFeature
- 数据源主备容灾切换
- 路由规则
- 我是等等等等

背后的本质都是在做分布式系统运行时行为特征（飞行姿态）的调整。

每一个分布式系统都应该有一个动态配置管理系统

是的，你一定注意到了，这里的说法跟图片上的有点不一样，这里想强调的是，未必都需要配置中心，在一个分布式系统规模还较小时，比如一个公司就二个应用集群，那无论你是用配置文件+auto-reload还是用redis，zookeeper什么都可以，这个动态配置管理系统不一定要是一个独立存在的，可以跟其它的系统,例如注册中

心，甚至作为消息中间件的一个子系统都没有关系。但是重要的是知道一定要有这么一个东西，它给自己的系统提供了动态调整行为的能力，而配置管理系统基本固有的特性一定要实现。

动态配置和静态配置的区别

曾经我也傻傻分不清楚其区别是啥，这很正常。动态和静态这是一个相对的概念，海枯石烂，永远不变的那不叫配置，可能是撩妹的鬼话，即使这个配置可能是放在一个看起来很像配置文件的文本里，配置一定是可能修改其值的，而是否是动态配置主要是看这个配置是不是跟应用的版本构建发布(build-deploy lifetime)强绑定的。如果一个配置项，跟软件的版本构建是不耦合的，在应用进程运行时，可能需要变更配置值的就是动态配置，哪怕是变更频率可能非常低，也许你设计了一个配置项，发现最后下来3年也没变更过一次，那也是动态配置，相反，配置变更只发生在软件版本构建和发布的那个点，那么就是静态配置，哪怕你构建很频繁，1个小时就来一回，那也是静态配置，举个简单的例子：

```
build-version = 3.4.6-1569965
```

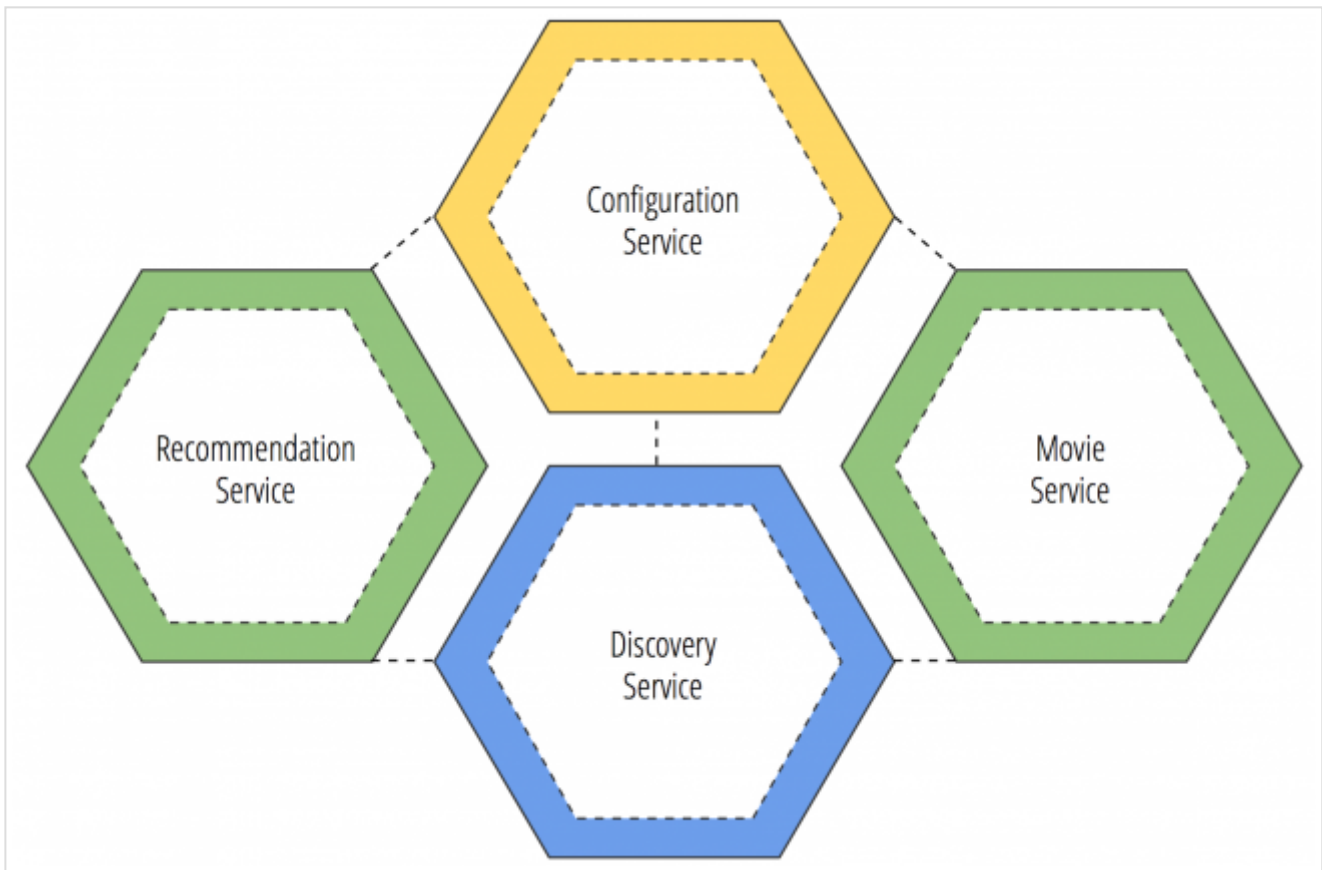
这个配置项，永远只在某个软件版本被构建出来时会变更其值，一旦这个版本被构建出来，并且在程序运行时，是一定没有变更诉求的，这就是一个跟构建绑定的静态配置。而文章开始时举得logLevel的例子，则是一个动态配置的例子。

所以看一下你的系统的配置项，你会发现动态配置其实更多，而跟行为演进相关的几乎都是动态配置。

为什么是淘宝

我在进来阿里做Diamond之后，思考过一个有意思的事情，为什么独立的配置中心这个东西会首先出现在淘宝？你去国内著名的竞价排名搜索引擎百度上搜“配置中心”，你会发现信息不是特别多，但是排在前面的都是X Diamond, SuperDiamond这种Diamond一族，反正我们没有为让配置中心跟Diamond这个名字关联给百度付过1毛钱，所以这个搜索结果应该是个自然结果，侧面也反应了在国内说起配置中心在生产上大规模运用是独此一家，别无分号。

而在业界，如下图，Spring Boot/Cloud微服务将注册中心(discovery service)和配置中心(configuration service)提出来还处在布道阶段，




而且从Spring的实现方式的技术局限性来看，应该是还没有哪个公司基于这个配置中心的方案在生产上实际支持大规模分布式系统，毕竟，翻遍所有blog和文章，还没有哪个老外开始提到，这个基于git的方案应该怎么去做多数据中心以及容灾相关的非功能性需求。

回到那个问题，为什么是淘宝，我们都知道在国内业界淘宝率先开启了去IOE,全面采用MYSQL，在这个过程中，在国内，大规模的系统性的解决分库分表这个命题的毫无疑问是阿里以及阿里中间件，在这个过程中，诞生了业内著名的TDDL.而与TDDL关联的一个核心问题是，分库分表之后，这多个库的数据源的配置信息存放在哪里，并对应用屏蔽多库这个事实? 好吧后面的事情也许你知道了，放在配置中心里! 但还是要提的是曾经并没有Diamond,都在注册中心（ConfigServer）里，后来Diamond从ConfigServer分了出来，这个过程，很多人看到的是数据是持久化和非持久化的区别以及当时产品的稳定性方面的考量，直到今天也还是如此，但是，通过这么多年实践和演进下来，才恍然大悟，拆分的背后其实是服务发现(Service Discovery)和动态配置管理服务(Dynamic Configuration Management)根本就是两个不同的东西，而在当时可能仅是一种直觉。

莫道君行早，更有早行人

有时候我们会因为在某个领域我们走的早一点而产生一点点的“优越感”和“虚荣感”，曾经我们以为Dynamic Configuration For Distributed System这个领域的实践我们不能说在业界独占鳌头，但是绝对位居前列。但是下面这两个老哥再次提醒我们，技术人踏实前行，不要有不必要的想法：




Jeff Kramer received the B.Sc. (Eng.) degree in electrical engineering from the University of Natal, South Africa, in 1970. In 1972 he received the M.Sc. degree in computing, and in 1979 he received the Ph.D. degree for an approach to the design and verification of distributed systems, both from Imperial College of Science and Technology, London, England.

He is currently a Lecturer in the Department of Computing at Imperial College, teaching courses in program design, software engineering, and distributed systems. His current research interests include the specification and design of distributed systems, and tools for the production of verified software. A special interest of his is in the production of large systems which are expected to evolve as requirements change.

Dr. Kramer is a member of the Association for Computing Machinery, and he is a founder member of the EWICS (European Workshop

on Industrial Computer Systems) TC11 on Application Oriented Specifications.



Jeff Magee received the B.Sc. degree in electrical engineering from Queens University, Belfast, Ireland in 1973. In 1978 he received the M.Sc. degree in computing, and in 1984 he received the Ph.D. degree for work aimed at providing flexibility in distributed systems, both from Imperial College of Science and Technology, London, England.

He is currently a Lecturer at Imperial College, teaching courses on operating systems and the programming and design of embedded systems. His current research interests include distributed operating systems and development of support environments for large distributed systems.

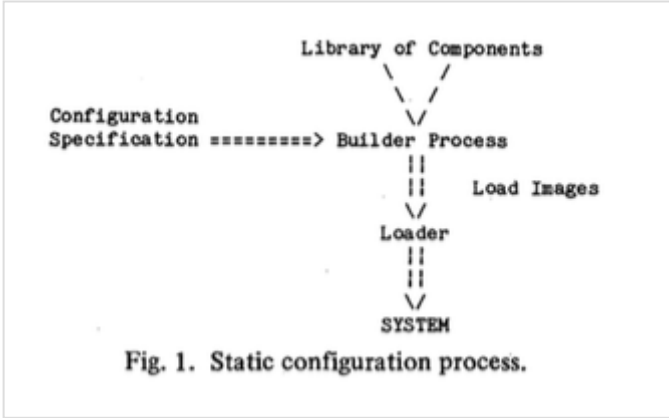
Dr. Magee is a member of the Institution of Electrical Engineers.

这两位老哥在1985年4月，在IEEE TRANSACTIONS ON SOFTWARE ENGINEERING 上发表了一篇名为 Dynamic Configuration for Distributed Systems的论文(Paper)，奶奶的，1985年！！什么概念，当时我4岁，还在玩泥巴,穿开裆裤。而Diamond现在的主力技术架构研发同学都还没出生呢！！我们能做的,只能是向这两位前辈再次致敬！

在这篇论文中，虽然从现在的观点来看，当时人们对分布式系统的认识跟现在有很大的区别，但是其中的问题识别的非常的准确:

“....Dynamic systemc onfiguration is the ability to modify and extend a system while it is running. The facility is a requirement in large distributed systems where it may not be possible or economic to stop the entire system to allow modification to part of its hardware or software. It is also useful during production of the system to aid incremental integration of component parts, and during operation to aid system evolution.The paper introduces a model of the configuration process which permits dynamic incremental modification and extension. Using this model we determine the properties required by languages and their execution environments to support dynamic configuration...”

并且很清晰的区分了静态配置和动态配置的基本模型:



配置与环境

另一个值得从技术上玩味的是关于“配置”的“环境”属性，这个表达可能有点抽象，比较难理解，这有点像技术上常提的一个叫Context的概念，很多Component会关联一个Context，Component+Context才是一个完整的运行时故事。环境恰恰也是热帖中大家可能会产生的疑问之一，为何Diamond会暴露这么多的环境让应用去选择？而进一步对中间件更熟稔一点的人会问，在单元化场景中，为何注册中心是一个大集群模式，而配置中心又是一个小集群模式？

另一方面，多个环境恰恰也是加重分布式系统需要依赖一个独立的配置管理系统的要素之一，可以说哪个公司的环境越复杂，分布式应用和服务越多，哪个公司诞生出独立的配置中心系统的可能性也就越大。

举几个容易理解的表述，来帮助理解配置的环境属性，

“在开发环境中将logLevel设置为DEBUG,在预发环境logLevel设置为INFO,生产环境里logLevel设置为WARNING”

“在日常环境执行线程池的最大线程数应该设置为15，而生产环境上这个值应该大一点，默认设为150”

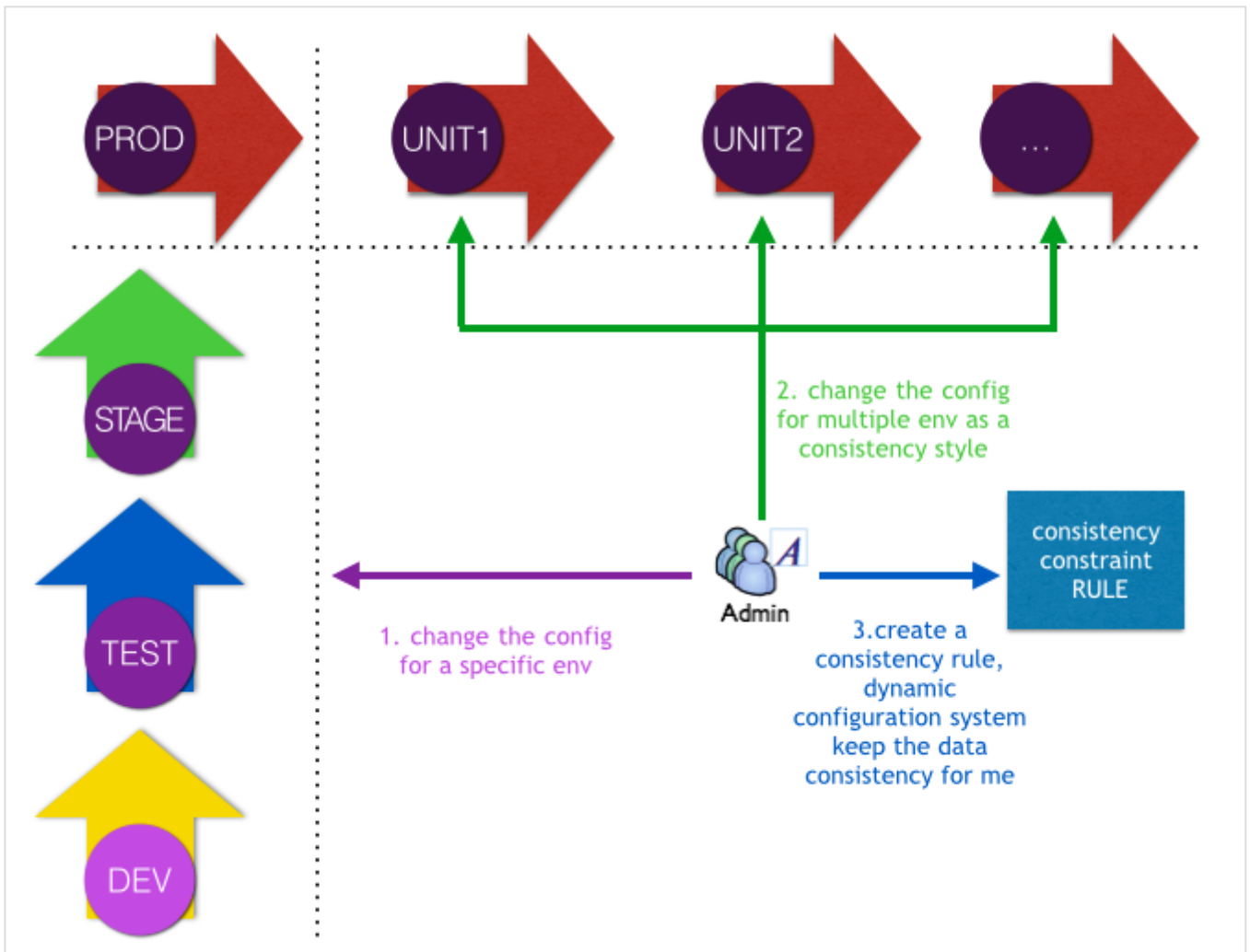
“在线上环境中，中心机房，应用数据源需要连接A库，而S机房，应用应该就近连接使用B库”

“只有在T环境，双向同步开关才应该关闭”

“这次的改动有点大，新的特性仅在线上的H单元把该特性开放出来，其它的单元环境先不要开放出来”

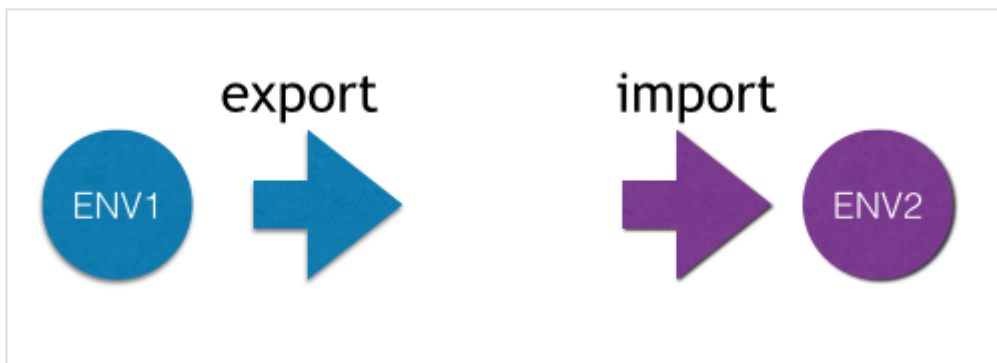
是的，相信你一定发现了，我们的某个配置项，其具体的值域的定义往往跟具体的环境相关联，现实中相当一部分配置在不同的环境必须设定不同的值，但是也有相当的另一部分配置在不同的环境要设定为完全一致的值。所以从某个应用的视角看，其100个配置项，可能有50个是每个环境要不同的值的，而另50个是不区分环境，所有环境其配置值都是需要完全一致的。这种异化给配置管理系统的设计带来了复杂性，而且这个最终语义的解释，很显然不应该在配置中心系统本身，应该交给应用，配置管理系统应该做的是提供方便的交互方式保证这两种不同的一致性诉求同时得到很好的满足，这种诉求分为3个方面，如下示意图：

是的，相信你一定发现了，我们的某个配置项，其具体的值域的定义往往跟具体的环境相关联，现实中相当一部分配置在不同的环境必须设定不同的值，但是也有相当的另一部分配置在不同的环境要设定为完全一致的值。所以从某个应用的视角看，其100个配置项，可能有50个是每个环境要不同的值的，而另50个是不区分环境，所有环境其配置值都是需要完全一致的。这种异化给配置管理系统的设计带来了复杂性，而且这个最终语义的解释，很显然不应该在配置中心系统本身，应该交给应用，配置管理系统应该做的是提供方便的交互方式保证这两种不同的一致性诉求同时得到很好的满足，这种诉求分为3个方面，如下示意图：



Diamond 这三个能力都有提供，其中1，2一般都是在配置中心的提供的客户端类库和OPS中体现。其中3这个实现是比较困难的，因为多个环境之间一般来说，都是有一些诸如远距离网络或者网络隔离之类的物理约束的，要做分布式一致性以及诸如分区容忍性之类的考量，目前Diamond只支持线上多单元一致性约束规则，但是因为历史原因大家没有真正用起来，规则本身的抽象度也不够好，不够通用。现在正在做改造，未来会将一致性规则做的更通用，后面会引导大家用起来，这样对于那些多环境保持一致的那些配置项，环境复杂性就可以对应用屏蔽掉，如果一个应用的配置项全是要各环境一致的，那么你就有福了，天空飘来五个字，“这么多环境就不是个事”。

另外，一个配置中心也应该具备的能力是配置集的导出\导入功能，可以让应用将A环境中的配置集方便的导出和导入到环境B中的能力，这对3个场景都有重大意义，一个场景是线上的配置值是经过实践验证的，现在在日常或者预发新建了应用环境，希望将线上的配置copy到线下用起来，第二个是，线下的应用终于调通要发预发或上线了，调较好的配置希望一下子发到线上去，当然这个根据我们的经验，一定要慎重，第三个是新建站或者机房，应用需要在新的机房将所有配置迁移过去，Diamond很快就会将这种能力开放出来。



理解了上面讲的这些，相信你就更能理解Spring Framework 里的两把刷子(抽象) Environment 和 PropertySource 是咋回事了,详细参见:

Spring 3.1 M1: Unified Property Management

<https://spring.io/blog/2011/02/15/spring-3-1-m1-unified-property-management/>

配置的三个属性

- 环境属性
- 稀疏变更属性
- 快速传播

环境属性我们上文已经讨论过。

稀疏变更讲的是配置的变更基本上都是稀疏的，因为系统的行为不可能非常频繁的需要动态调整，你每100毫秒调整一次系统的行为，估计系统要对你骂娘了。

而快速传播讲的是配置不变则已，一变往往要求目标集群的所有节点要几乎同时收到变更，然后几乎整齐划一的统一调整行为。切库的场景来讲，主备切换之后，应用集群写新的主库这个行为切换的稀稀拉拉，整个收敛了一天，应用肯定是受不了的，而这个属性决定了对配置中心对配置变更推送SLA的高要求。

配置中心(Diamond)和注册中心(ConfigServer)的不同

在我们的眼里，用土话讲，这就是乌龟与王八的区别，想用拜金一点的话讲，这就是金条和钻石的区别，说的洋气一点这叫Apple and Pear，但是当初起的名字的给我们带来了大麻烦，现在我们的服务注册中心现在叫Config Server, 你说坑不坑。不过很多中间件产品的名字同时承载了一段8年的历史，这名字也体现中间件持续做技术产品，坚持就是一种力量的信念在里面，n代人持续发展一个产品，说实话，只管生不管养的现象在中间件不能说没有，但确实是很少的。1个8年的产品就像八岁的孩子，已经上小学了，硬要给它改个名字，从朱屎山改成朱宝山，他的同学认不认还要打个非常大的问号，而且还要跑到派出所重新上户口，也是个麻烦事。

所以产品的取名字有大学问和大恐怖，大家取名之前一定要找算命先生给好好算一算，本来我在成为码农之前那也是仙风道骨，江湖人送外号郭半仙，那时候可以钉钉发个红包找我算一下。但成为码农之后嘛，就特么不说了，说起来全是泪，以前get到的很多技能点全丢了，现在就瞅着电脑和代码最顺眼。

2者具体的不同，参见未来会写的<<应用配置中心和服务注册中心究竟有什么不一样?>>

关于配置，业界最新动态

业界著名的专职配置中心产品几乎没有（Diamond傲娇ing~），基本都是在用git, redis, zookeeper, consul 这些凑活着搞一下，所以要了解配置中心的同学，直接了解Diamond就可以了。:-)

但是针对于配置管理的客户端编程类库这一块有一些类库牛吹得是很大的，感兴趣的同学可以了解一下：

- Apache Commons Configuration (<https://commons.apache.org/proper/commons-configuration/>)

这个类库是在是太繁琐了，用起来总感觉有点杀鸡用牛刀，力用的太大的感觉。

- owner (<http://owner.aeonbits.org/>)

简单易上手，特性看起来很多，但是在很多关键常用的特性反倒没有。

- cfg4j (<http://www.cfg4j.org/>)

简单易上手，cfg4j 支持跟多种后端集成，做配置中心的解决方案，api设计也非常的不错，我们正在设计的新diamond annotation api的时候借鉴了不少其想法。

- Spring Framework (http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#_property_source)

Spring的东西一般都还是很不错的，如果你的应用本身在用Spring，毫无疑问，就这个了，用上面的那些类库实在没有必要。

- Spring Cloud Config Server

Spring Cloud Config



Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system. With the Config Server you have a central place to manage external properties for applications across all environments. The concepts on both client and server map identically to the Spring `Environment` and `PropertySource` abstractions, so they fit very well with Spring applications, but can be used with any application running in any language. As an application moves through the deployment pipeline from dev to test and into production you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate. The default implementation of the server storage backend uses git so it easily supports labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content. It is easy to add alternative implementations and plug them in with Spring configuration.

如果你仔细读一下其内容，而你又了解Diamond的话，你会发现这就是Diamond这些年在解决的问题啊，Spring 终于从大量配置文件逐渐走向了配置中心（externalized configuration in a distributed system），而这一次我们走在了前面。

看看owner 的宣传，

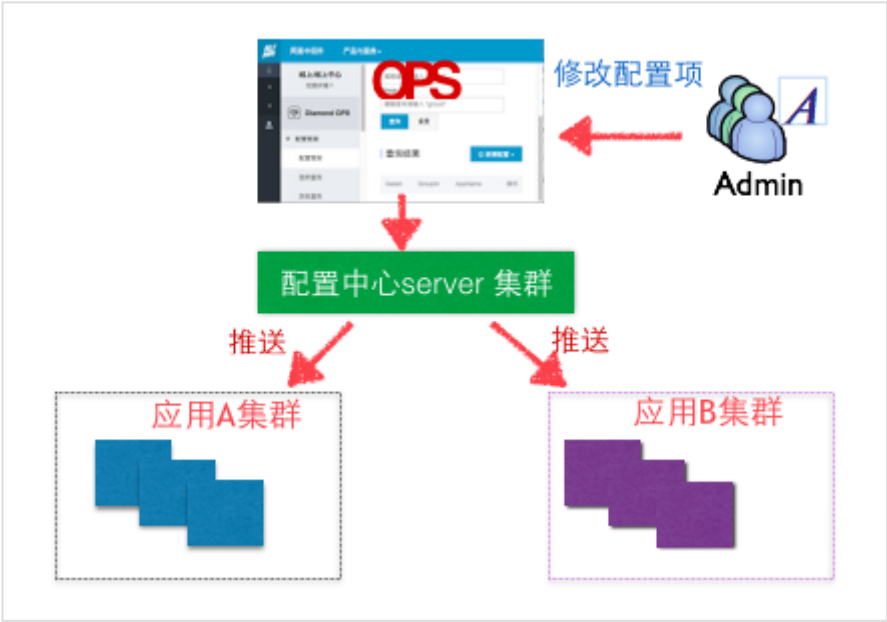


Java 他妈 properties reinvented！Java Properties的重新发明！屌不屌?! 我僧僧的觉得，我们中国码农有时候就是差了这种提炼和升华的能力，刚入行时觉得维护和修改前人留下的烂代码实在是个很苦逼，很Low的事情，但是Martin Fowler把这叫做“重构”，然后还写了一本书，然后我们读了之后，居然还觉得他妈的，讲的真的非常的有道理！把改烂代码变成叫重构之后，有了理论指导，突然觉得这真是高逼格的事情！

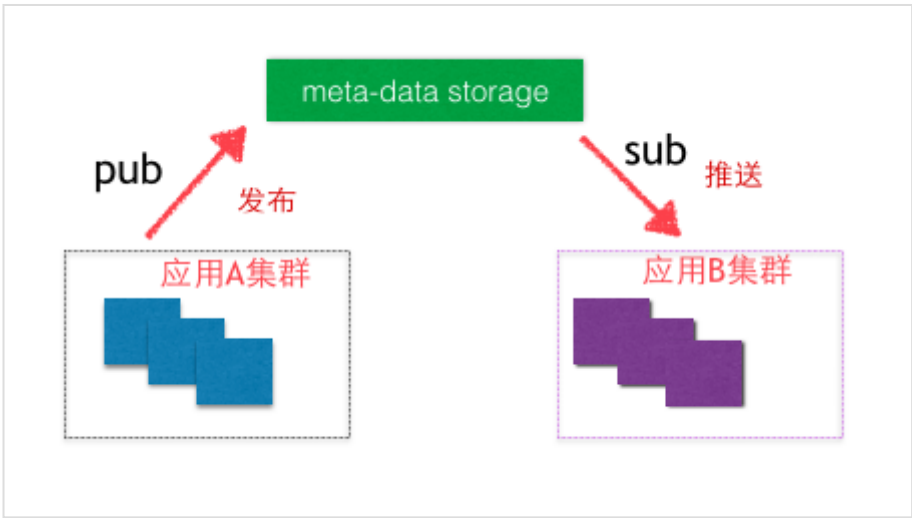
配置(configuration)与元数据(metadata)

很多人没有这种这两个概念的分，但是对于配置中心，二者其实是有微妙的差别的。

配置如前文有阐述，配置的修改基本上都是由人来驱动，并且在ops上实现变更。



而元数据的本质是一小段程序元数据，它很多时候是程序产生，程序消费，由程序通过调用Diamond的客户端api来实现变更，中间不会有ops 或者人的介入。



知道这个有什么意义？毫无疑问，配置这种需求选型比较明确，而元数据这种，可以选的pub-sub系统太多了，诸如消息队列产品，分布式coordinator产品如zookeeper, 带pub-sub 能力的k-v store 如Redis等等都可以，所以如果是元数据这种，选什么需要慎重，其间运用之妙，存乎一心，要充分评估和把握自己的需求。

Diamond 不光是应用配置存储，其目前存储的数据，很大一部分是metadata，所以Diamond 其实也是一个元数据存储中心。

结束语

这么长的文章，你居然能坚持看到这里，说明你是真正的、脱离了低级趣味的、纯粹的码农，猿类中的精英！配置中心，它没有高精尖的技术，难懂的算法，海量的数据，做这个东西只需要一个精神就够了。

Do One Thing, Do It Well!

owner之前一直在搞配置文件的支持，现在owner也开始转型搞跟zookeeper集成之类的，做配置中心的解决方案了，所以本文开头说业界正在走向配置中心解决方案不是在忽悠，是确实是这个趋势。

企业级互联网架构Aliware，让您的业务能力云化：<https://www.aliyun.com/aliware>

#配置中心

◀ 企业传统IT架构将何去何从？

开发你的第一个SchedulerX任务 ▶

分享到：

微博

微信

QQ空间

腾讯微博

© 2017 ♥ 阿里中间件

由 Hexo 强力驱动 | 主题 - NexT.Muse

