应用配置管理 ACM

最佳实践

最佳实践

配置变更风险管理

本节介绍如何通过配置管理的一些最佳实践降低配置变更在生产上可能带来的风险。

如何更好地组织配置

ACM 提供了 dataId， group，app，namespace 等不同的维度来帮助您管理好您所在组织中的所有配置。如果勤于梳理并善用这些维度，则能很好地减少在配置管理过程中发生无心的错误，降低给系统稳定性带来的伤害。

推荐配置组织方式

dataId

用来表示一组相关的 key=value 的配置项的集合。

善用 dataId 命名，给 dataId 起一些有意义的名称，例如：com.company.trade.threadpool.params，trade.p1.props

group

- 一般使用模块名或者云资源名，如果一个应用使用了 Nginx，SLB， 此处 group 可能就是 nginx，slb。

app

- 应用分组，一般是一个简单的业务单元或者微服务分组,一般一个应用是由一个小型或者中型的 team 开发并维护。

namespace

- 粗粒度隔离多个应用的配置，例如多环境。

做好配置影响分级

每个配置项的变更对系统的影响是非常不同的。例如日志级别的变更如果错了，会改变系统的日志量，此外一般不会有其它负面的影响。而连接池、线程池、限流阈值、主机配置之类的变更往往是一个 Server 级别或者一个应用服务集群级别的影响。

分布式系统的诸如全局路由规则，负载均衡策略，网络配置等之类则是非常重量级的配置，错误的变更往往会带来严重的后果。所以按照配置影响力，将配置分为 P0~P4 是一个非常好的实践方式。

以下是示例说明。

级别	影响力	例子
P0	全网	全局路由规则，网络配置，负载均衡配置
P1	应用集群	集群限流阈值，集群服务端点
P2	主机级别关键配置	主机资源配置，线程池大小
P3	进程级别非关键配置	日志级别
P4	无关紧要的配置	版本信息

尽量避免大配置

如果应用所有的配置项都放在一个配置里，则意味着所有的人都在一个配置集上修改。这不仅导致配置变更、推送变得相对频繁，也增加了互相冲突以及误操作的风险，为更好的配置授权和分级的变更流程管控设置了障碍。

重要配置变更一定要灰度

做好配置分级之后，应该考虑重要的配置，例如 P0，P1 级的配置应该考虑诸如变更审核，灰度等发布策略来降低变更的风险，这点非常重要。

如何确保配置安全

敏感配置信息请加密后存储在 ACM 里

ACM Server 在主机上以及配置存储中默认不做数据加密。对于一些敏感配置信息，如密码，Token，AccessKey之类的信息，必须加密后才可以往 ACM 上存储。

ACM 在 API 和控制台上，都提供了配置内容加密工具，帮助您将内容加密之后再存储到 ACM 里。

MQ 客户端流控设计

MQ（Message Queue），即消息队列，是一种常用的异步 RPC 技术。本文以阿里云 MQ 为例，介绍了如何使用 ACM 对 MQ 实现流量控制。

MQ 流量控制简述

对于 MQ 调用，通常的限流方式是在订阅端限流。限流方式有两种：

- 针对消息订阅者的并发流控
- 针对消息订阅者的消费延时流控

针对消息订阅者的消费延时流控的基本原理是，在客户端每次消费时增加一个延时来控制消费速度，此时理论消费并发最快速度为：

```
MaxRate = 1 / ConsumInterval * ConcurrentThreadNumber
```

例如，如果消息并发消费线程（ConcurrentThreadNumber）为 20，延时（ConsumInterval）为 100 ms，代入上述公式可得：

```
200 = 1 / 0.1 * 20
```

可知，理论上可以将并发消费控制在 200 以下。

与并发线程数流控相比，消费延时流控的优点在于其实现相对简单，对 MQ 类客户端包依赖较少，而且不需要客户端提供控制并发线程数的动态调整接口。

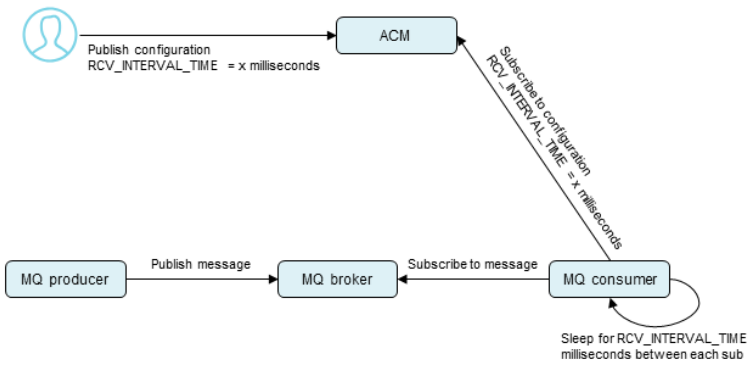
以上各种流量控制方法，如果要在分布式架构下做到全局动态控制，一个简单的技术方法是依赖配置中心，即通过配置中心来下发流控参数。

下文详细介绍了如何基于配置中心来实现异步消息消费的全局动态流控。示例中使用了阿里云的 MQ（消息队列）和 ACM（应用配置管理）产品，基于 Java 语言。

注意：以 MQ 为例是因为截至目前，MQ Consumer Client SDK 暂不支持动态调整现成并发数，而通过 ACM 来动态调整消费延迟的方法可以解决 MQ 消费流控动态的问题。

基于消费延时流控的基本原理

如图所示，管理员或应用程序通过 ACM 控制台发布消费延时配置（RCV_INTERVAL_TIME），所有 MQ 消费程序订阅该配置。理论上，该配置从发布到下发至所有客户端，可以在 1 秒内完成（取决于网络延时）。



代码示例

本章节给出了基于配置中心来实现异步消息消费的全局动态流控的代码示例。关于 SDK 的详细介绍，参见 MQ（消息队列）和 ACM（应用配置管理）产品官方文档。

创建 ACM 配置

在 ACM 上创建消费延时的参数。

设置全局消费延时变量

设置消费接收延时的全局变量。

```
// 初始化消息接收延时参数，单位为millisecond
static int RCV_INTERVAL_TIME = 10000;

// 初始化配置服务，控制台通过示例代码自动获取下面参数
ConfigService.init("acm.aliyun.com", /*租户ID*/"xxx", /*AK*/"xxx", /*SK*/"yyy");
// 主动获取配置
String content = ConfigService.getConfig("app.mq.qos", "DEFAULT_GROUP", 6000);
Properties p = new Properties();
try {
```

```
p.load(new StringReader(content));
RCV_INTERVAL_TIME = Integer.valueOf(p.getProperty("RCV_INTERVAL_TIME"));
} catch (IOException e) {
e.printStackTrace();
}
```

设置 ACM listener，确保当配置被修改时，RCV_INTERVAL_TIME 参数即时更新。

```
 // 初始化的时候，给配置添加监听,配置变更会回调通知
ConfigService.addListener("app.mq.qos", "DEFAULT_GROUP", new ConfigChangeListener() {
public void receiveConfigInfo(String configInfo) {
Properties p = new Properties();
try {
p.load(new StringReader(configInfo));
RCV_INTERVAL_TIME = Integer.valueOf(p.getProperty("RCV_INTERVAL_TIME"));
} catch (IOException e) {
e.printStackTrace();
}
}
});
```

设置 MQ 消费延时逻辑

完整实例如下。

注意：

- 本例中 RCV_INTERVAL_TIME 参数的访问刻意没有加锁，原因不做赘述。
- Aliyun ONS Client 不提供动态线程并发数，默认并发为 20。因此本例正好使用消费延时参数来动态调节 QoS。

```
//以下代码可直接贴在Main()函数里
Properties properties = new Properties();
properties.put(PropertyKeyConst.ConsumerId, "CID_consumer_group");
properties.put(PropertyKeyConst.AccessKey,"xxx");
properties.put(PropertyKeyConst.SecretKey, "yyy");
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
// 设置 TCP 接入域名（此处以公共云生产环境为例）
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet");

Consumer consumer = ONSFactory.createConsumer(properties);

consumer.subscribe(/*Topic*/"topic-name", /*Tag*/null, new MessageListener()
{
public Action consume(Message message, ConsumeContext context) {

// MQ Subscribe QoS logical start,
// Each consuming process will sleep for RCV_INTERVAL_TIME seconds with 100 ms sleeping cycle.
// Within each cycle, the thread will check RCV_INTERVAL_TIME in case it's set to a smaller value.
// RCV_INTERVAL_TIME <= 0 means no sleeping.
```

```
int rcvIntervalTimeLeft = RCV_INTERVAL_TIME;
while (rcvIntervalTimeLeft > 0) {
if (rcvIntervalTimeLeft > RCV_INTERVAL_TIME) {
rcvIntervalTimeLeft = RCV_INTERVAL_TIME;
}
try {
if (rcvIntervalTimeLeft >= 100) {
rcvIntervalTimeLeft -= 100;
Thread.sleep(100);
} else {
Thread.sleep(rcvIntervalTimeLeft);
rcvIntervalTimeLeft = 0;
}
} catch (InterruptedException e) {
e.printStackTrace();
}
}
// MQ Subscribe interval logical ends

System.out.println("Receive: " + message);

/*
* Put your business logic here.
*/
doSomething();

return Action.CommitMessage;
}
});
consumer.start();
```

运行结果

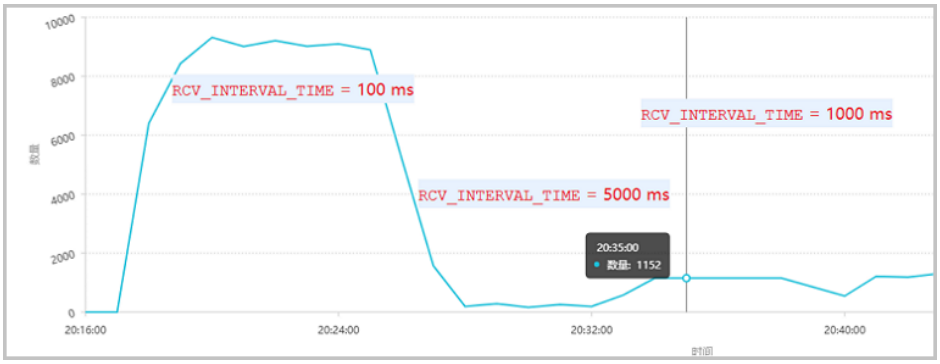
单机运行 Consumer 进行消费，假设队列内的消息无限多、不会出现消费完的情况，分三段测试，分别运行约 5 分钟，通过 ACM 配置推送来达到以下效果。

- RCV_INTERVAL_TIME = 100 ms
- RCV_INTERVAL_TIME = 5000 ms
- RCV_INTERVAL_TIME = 1000 ms

在单 MQ 消费业务处理耗时约 100 ms 情况下、单机并发 20 线程的测试结果如下。

- RCV_INTERVAL_TIME = 100 ms：平均消费性能约为 9000 tpm 左右
- RCV_INTERVAL_TIME = 5000 ms：平均消费性能被限制到 200 tpm 左右
- RCV_INTERVAL_TIME = 1000 ms：平均消费性能回升到 1100 tpm 左右

该结果基本达到消费和 tpm 成反比的预期。最重要的是，整个过程中应用不中断，流控推送结果对分布式集群秒级生效。单机性能结果如下所示。



利用配置中心规范构建 PaaS 服务配置

MQ（Message Queue），即消息队列，是一种常用的异步 RPC 技术。本文以使用 ACM 对 MQ 实现流量控制的场景为例，介绍了如何以规范的配置命名格式来进行限流设置。

配置规范问题的产生

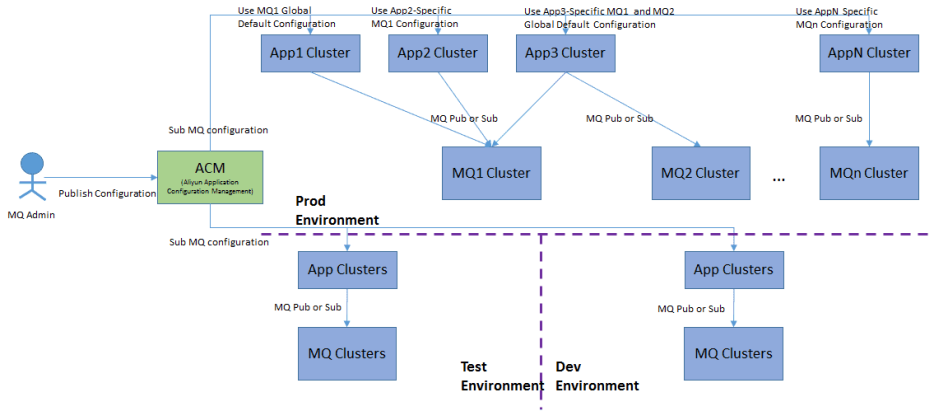
如果是单一应用的单一属性配置，使用以下配置文件即可完成，没有配置规范的问题。

```
//配置目录结构
--app
|--src
|--config
|--application.properties

//配置内容
RCV_INTERVAL_TIME=20
```

但是，如果是为分布式 PaaS 服务编写分布式规则，PaaS 服务提供方（而非应用方）在设计配置时会遇到不少问题。以 MQ 限流场景为例，可能遇到的问题有：

- 如何区分全局配置和局部应用配置：例如，PaaS 服务方在统一管控平台提供的服务时，如何既有全局的规则配置，又能针对某个应用进行特殊配置。
- 如何区分不同集群 MQ 服务：例如，在保证配置命名统一的情况下，如何区分 MQ1 Cluster 和 MQ2 Cluster 的配置。
- 如何使用同一套配置中心来隔离 Dev、Test、Staging、Prod 等不同环境。

以上 MQ 限流场景需求如图所示。



显然，不恰当的配置命名规范将影响以上配置的易用性。

接下来，本文围绕配置中心介绍这方面的最佳实践。为了说明配置命名规范，首先需要介绍一下配置中心的配置结构组织的能力。

配置中心的配置结构功能

除了集中管理配置和订阅推送等能力，配置中心还具备配置结构化能力，能帮助管理员大幅简化各种复杂应用场景下的配置管理。

配置中心的配置结构能力说明

- 租户隔离：配置中心根据用户或场景对配置进行隔离的能力。有了租户隔离，不同的配置在不同的租户可以重名，而且具有不同的鉴权机制。
- 最小配置集合：配置中心如何将若干配置组合成一个配置集合。将不同配置放在一个最小配置集合来更改和发布，可以统一发布类似的配置，以便应用统一处理。配置路径的概念类似于文件路径或者网络域名，它让不同配置集合之间具有层级关系。
- 具体配置的 Key-Value 形式：用户如何在配置中心中设置具体配置内容。

配置中心配置结构能力产品比较

为了更直观地说明，我们对比了几个配置中心产品：

- 阿里云 ACM：阿里云应用配置管理，前身为 Diamond，是国内最早的配置中心产品。目前在 Git 上有不同开源版本，在阿里云上有商业版可供使用。
- Spring Cloud Config：Spring Cloud 官方配置中心工具，主要用于 Java Spring 领域。
- ZooKeeper：虽然具备部分配置中心能力，但是由于定位于分布式协调信息管理，因此只适合在应用规模不大的情况下用作配置中心。鉴于其使用广泛，此处也将其纳入对比范围。

对比详情如下：

功能	ACM	Spring Cloud Config	ZooKeeper
租户级隔离	采用 Namespace 和 Group 双重隔离，其中不同 Namespace	一个 Git 项目为一个租户。	没有直接可用的租户隔离技术。

	需要不同 AK/SK 鉴权，Group 则不需要。		
最小配置集合	以 DataID 为标识的配置集为最小配置集合。配置集合没有路径概念，但是通过合理设计结合通配符查询，可以间接达到配置路径的效果。	Git 项目中的配置文件为最小配置集合，没有配置路径概念。	Znode 为最小配置集合，有配置路径概念。
配置 Key-Value	KV 内容由用户在 DataID 下自由组装，格式不限，可以是 properties、Json、XML 等，管理页面提供校验功能。	通过 Git 项目中 properties 形式的配置文件设置 KV。	通过设置 Znode 的 Value来设置 KV，内容格式无限制。

综上所述，ACM 在租户隔离和最小配置集合方面都有较大的灵活性。下文介绍如何利用 ACM 的 Namespace、Group、DataID 等配置功能来设计一个用于执行 QoS限流策略的合理配置结构。

基于配置中心的分布式服务配置设计最佳实践

配置结构

为了满足 MQ 配置的功能需求，结合 ACM 的特点，可采用以下配置方法。

以不同 Namespace 隔离不同环境的 MQ 配置。例如：

- ProdEnv 命名空间用于生产环境，TestEnv 用于测试环境，DevEnv 用于开发环境。
- 不同环境通过 AK/SK 天然隔离，安全性进一步加强。

以 Group 区分不同集群提供的 MQ 服务，以隔离配置和简化访问形式。

例如，对于专门为子部门核心交易部门服务的 MQ 集群，和为子部门交易类目部门服务的 MQ 集群，可通过 Group 区分不同的全局配置。这样做的优点在于，在生产系统中，所有应用采用同一（子）公司的 AK/SK（或类似认证体系密钥），因而简化了部署，有效隔离了不同集群的配置，降低了配置复杂性。

```
DataID: mq.global.qos
Group: Trading

DataID: mq.global.qos
Group: ProductCategory
```

全局配置以全局统一 DataID 命名配置项存放。

其中，配置 ID 以 mq 开头，global 表示全局配置，qos 表示 qos 方面的配置，Group 可使用默认值。

```
DataID: mq.global.qos
Group: Default_Group
```

应用局部配置以相同前缀 qos.* 来命名 ID。

配置 ID 以 mq 开头，app.[appname] 表示需要重载的 app 配置项，Group 可使用默认值。

```
DataID: mq.app.app1.qos
Group: Default_Group

DataID: mq.app.app2.qos
Group: Default_Group
```

配置具体 KV 的设置

在很多配置中心产品中，例如 Appolo、ACM System Manager Parameter Store，每个具体的配置是配置中心最小粒度的管理单元，用户需要在某个粒度上逐一设置配置的 KV。但是 ACM 并没有该限制。常用做法有两种：

- 仿照以上配置中心，将每个 Key 存在一个独特的 DataID上，例如：
  - mq.global.qos.RCV_INTERVAL_TIME 设置为 50
  - mq.global.qos.MAX_THREAD 设置为 20

将常用配置聚合成一个 DataID，并编辑为一个配置文件（格式不限，例如 Properties、Json、XML 等)

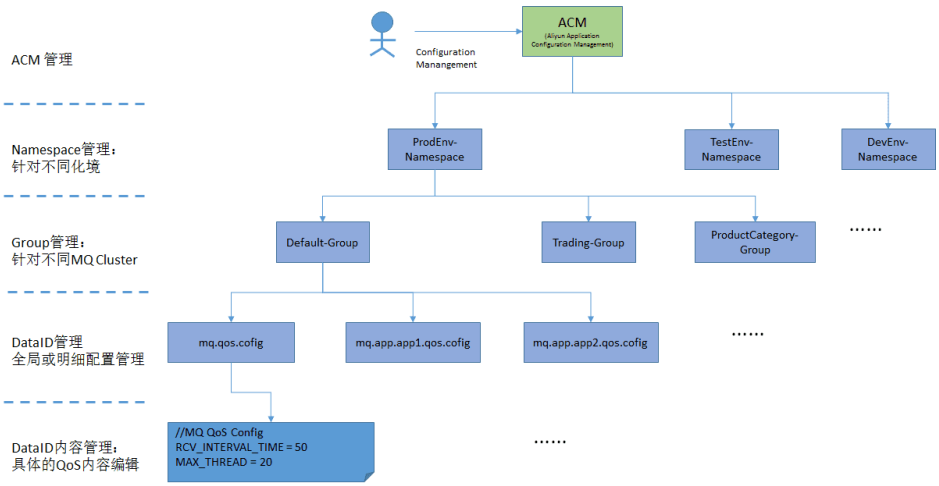
例如 mq.global.qos 设置如下：

```
//MQ 限流 QoS设置
RCV_INTERVAL_TIME = 50
MAX_THREAD = 20
```

在实践中，我们发现第二种方法更高效。除了更大的灵活性以外，还有一个优点是多个配置在一次变更中同时发布，既降低了性能开销，理论上也达到了变更批量变化的原子操作效果。

配置结构示意图

以上设计的配置结构示意图如下：

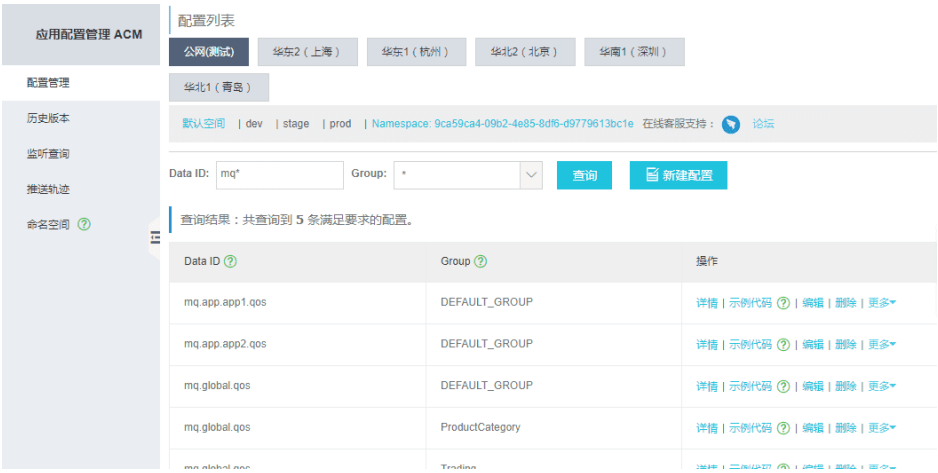


方案优点总结

以 Namespace 隔离不同环境，使 MQ 配置项在不同 Namespace 可重复。不同 Namespace 通过管理人员、程序 AK/SK 等权限设置得到隔离保护，让配置项能统一，且各个环境互不干扰。

以 Group 隔离相同环境的不同集群，既能保证不同集群下配置的统一性（如配置名不变等），使代码更加简单，又能在逻辑上隔离不同的集群配置。

通过最小配置集 DataID 的规范命名设置，各 MQ 客户端既可以方便地查找 MQ Default 全局配置，又能查找到自己的应用特殊配置。此外，管理员在 ACM Portal 上通过前缀通配查找，能方便地查找出所有 MQ 对应的所有规则，使管理变得简单。示例如下：



相关信息

关于 ACM 配置结构和命名规范的更多信息，请参考：

- ACM应用配置管理官方网站

- 阿里云 ACM 官方文档：配置变更风险管理

使用 ACM 简化 Spring Cloud 微服务环境配置管理

在本文中，我们以测试和生产环境连接不同的数据库、配置不同的数据源（包括连接池）参数为例，介绍了如何搭配使用阿里云配置中心 ACM 与 Spring Cloud，帮助您在微服务架构中简化环境配置管理。

配置的环境属性

在系统持续交付的过程中，系统最终运行环境的多样性及复杂性毫无疑问增加了我们在配置管理工作上的负担，有时候，可以说配置就是因环境而生的。

这一点在 Eugen Paraschiv的博文 Configuration Must Be Environment Specific 里有简单的阐述，在博文《现代应用架构中的配置管理面临的挑战》的容器化、调度与配置管理小节也有深入阐述。

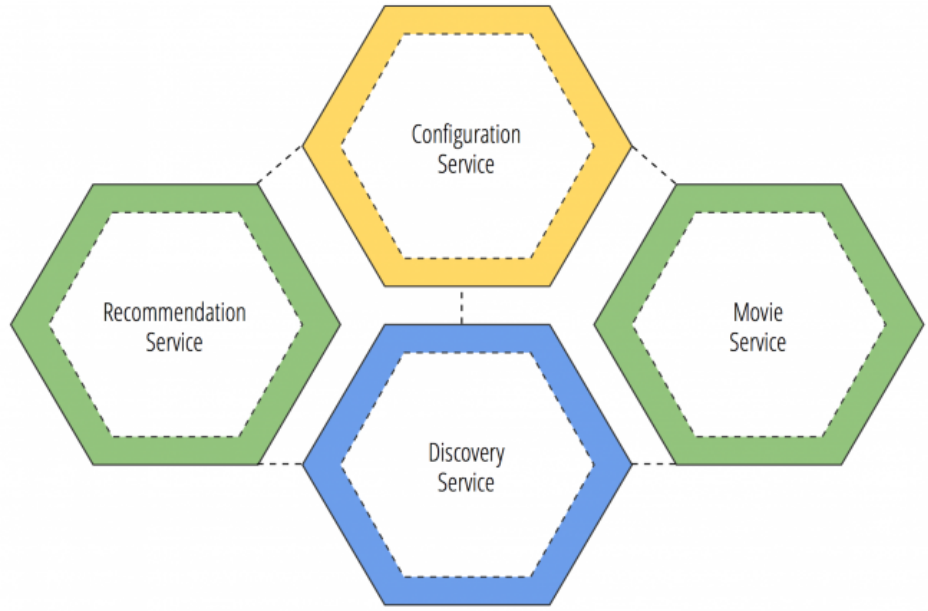
如果要问，是什么导致了我们应用的构建物（artifact）在各个环境不能保持一致？有时候 Docker 无法轻易达成“Build Once, Run Anywhere!”的承诺，原因往往就是环境配置的差异。为帮助您理解，此处列举一些简单的例子：

- 在开发环境中将 logLevel 设置为 DEBUG，在预发环境中将 logLevel 设置为 INFO，在生产环境中将 logLevel 设置为 WARNING。
- 在开发环境中使用 4 核 8G 的机器运行数据库，而在生产中用 32 核 96G 的机器运行数据库。
- 在日常环境执行线程池的最大线程数应该设置为 15，而生产环境上该值应该大一些，默认设为 150。
- 在线上环境中，中心机房内应用数据源需要连接 A 库，而深圳机房，应用应该就近连接使用 B 库。
- 只有在小淘宝环境，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杭州单元把该特性开放出来，其他的单元环境先不要开放出来。

在本文中，我们简要介绍了如何使用阿里云 ACM 配置管理产品在 Spring Cloud 中替代 Spring Cloud Config 以简化环境配置管理，帮助您理解基于 ACM 简化微服务环境配置管理的方案。此外，本文还会简单比较一下 ACM 与 Spring Cloud Config 方案的优劣。

场景故事

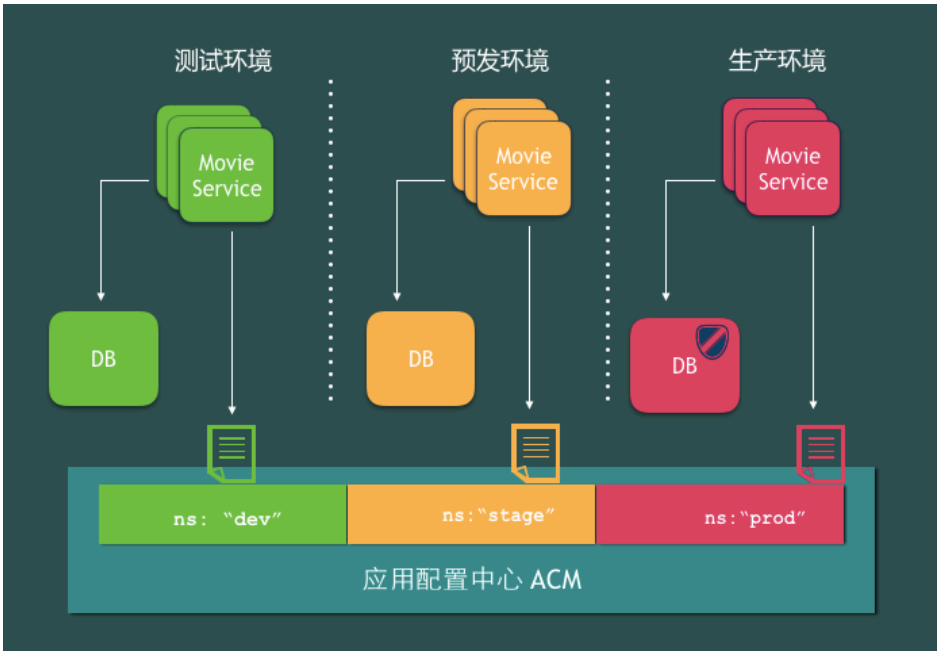
为帮助理解需求和场景，在日常工程实践中，我们一般会以用户故事（User Story）的方式预设一个简单的场景，以此来做阐释和交流。下面是一张早期的布道图：



以 Movie Service 为例，假设我们需要从关系数据库 MySQL(RDS) 检索所有电影信息列表，但是只有生产库需要顶配的机器，测试、预发和生产环境需要使用不同的数据库，因此我们的应用需要在不同的环境具备不同的数据源配置、连接池配置、数据库安全配置等等。

本文介绍了如何基于阿里云 ACM 的 Namespace 映射不同环境，为 Movie Service 在不同运行环境设置不同的数据源配置。

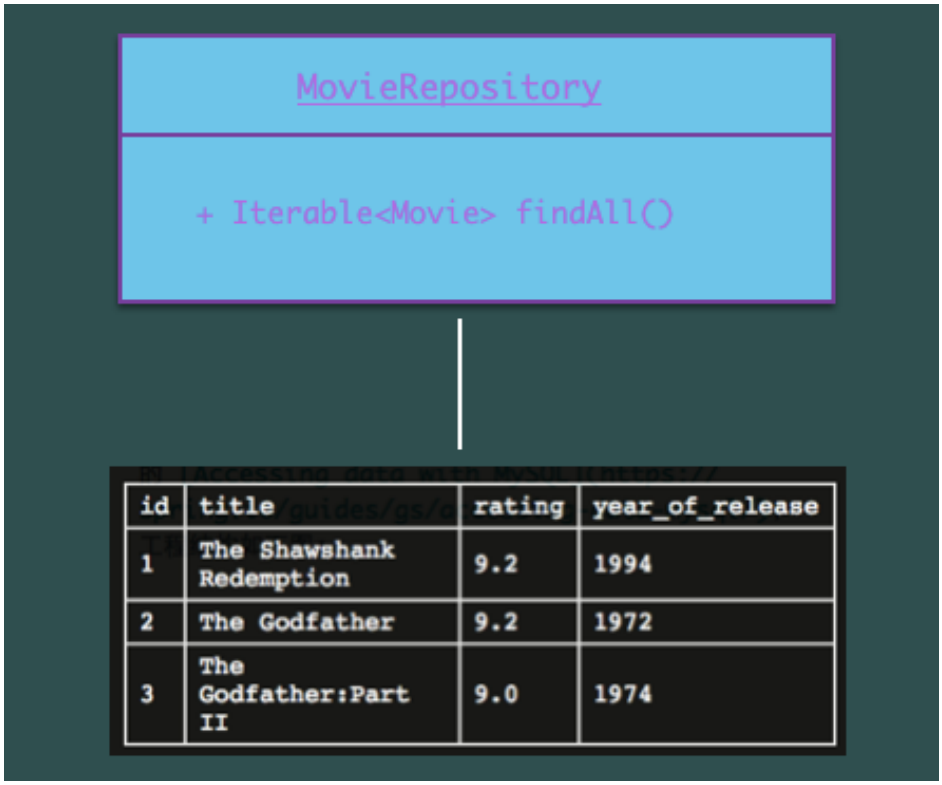
如下图所示：



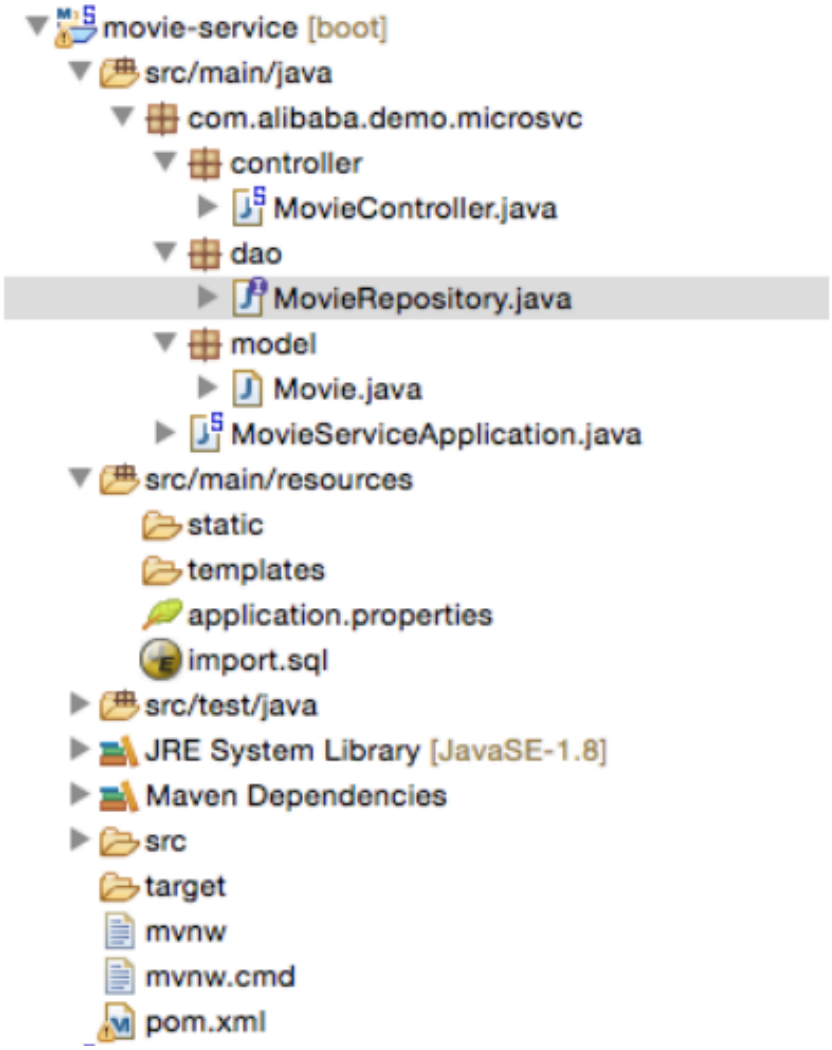
创建微服务 Movie Service

新建 Spring Boot Starter 微服务应用 movie service

movie service 的业务逻辑很简单——列出 MySQL(RDS) 中所有的 movie：



这里我们创建了一个标准的 JPA 应用（类似于 Spring 官网的样例工程 Accessing data with MySQL ）。工程结构如图所示：



引入 JPA、MySQL、连接池 HikariCP 以及 WEB 依赖

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<scope>runtime</scope>
</dependency>

<dependency>
<groupId>com.zaxxer</groupId>
```

```
<artifactId>HikariCP</artifactId>
<version>2.7.6</version>
</dependency>
```

创建 MySQL(RDS) 数据库及用户

```
mysql> create database db_example; -- Create the new database
mysql> create user 'springuser'@'localhost' identified by 'ThePassword'; -- Creates the user
mysql> grant all on db_example.* to 'springuser'@'localhost'; -- Gives all the privileges to the new user
on the newly created database
```

详细步骤可参考 Accessing data with MySQL 中的“Create the database”小节。

创建 WEB Controller

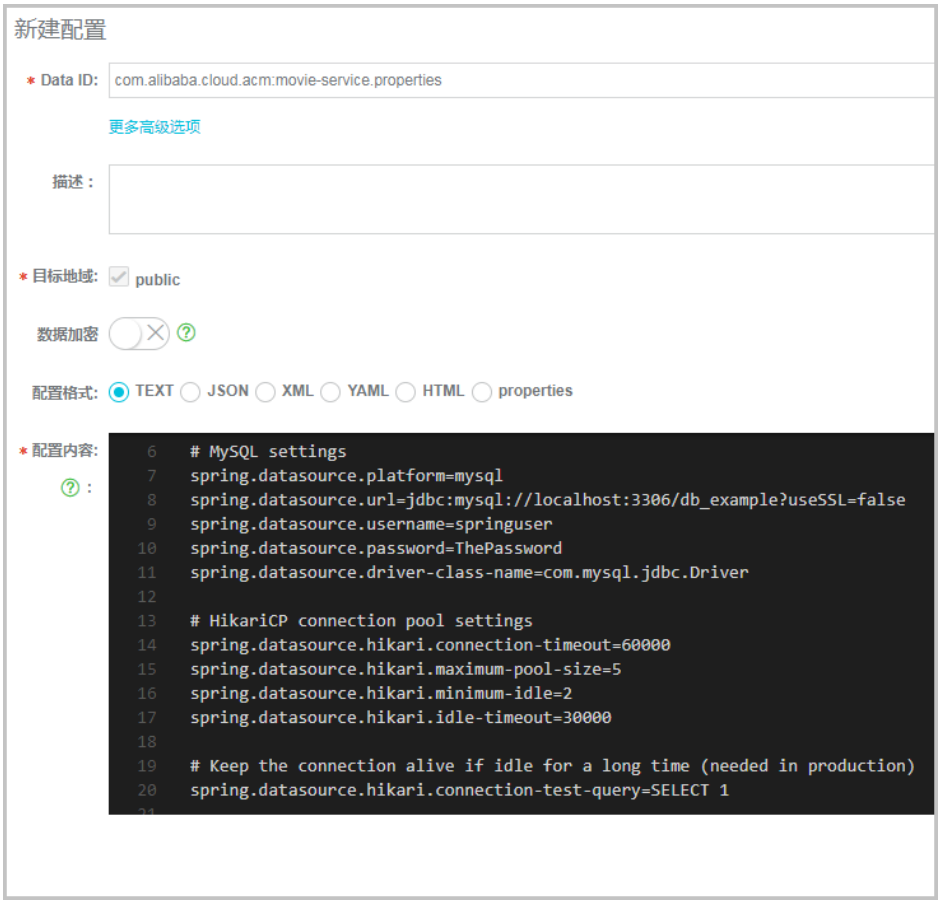
```
 package com.alibaba.demo.microsvc.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;
import com.alibaba.demo.microsvc.dao.MovieRepository;
import com.alibaba.demo.microsvc.model.Movie;

@RestController
public class MovieController {

@Autowired
MovieRepository movieRepository;

@RequestMapping("/list-movies")
public @ResponseBody Iterable<Movie> listMovies() {
return movieRepository.findAll();
}

}
```

在 ACM 中使用 Namespace 创建隔离的环境配置

注意：在阿里云上使用 ACM 的前提是开通该项服务，开通流程可参考文档快速入门。开通服务并登录后，即可进入 ACM 控制台创建命名空间及配置。

在 ACM 中创建 3 个环境（dev、stage、prod）

为 dev、stage、prod 环境分别创建配置



注意：我们完成了什么？

在上一步中，我们为相同配置项针对不同环境设置了不同的值，例如spring.datasource.url这个配置项，我们通过设置不同的 url 来为各环境连接不同的数据库，并且仅在生产环境开启 SSL (useSSL=true)。

```
dev:
spring.datasource.url=jdbc:mysql://localhost:3306/db_example?useSSL=false

prod:
spring.datasource.url=jdbc:mysql://30.5.101.169:3306/db_example?useSSL=true
```

同时，我们也为生产环境（prod）设置了更大的数据库连接池和更小的连接超时时间。

```
dev:
spring.datasource.hikari.connection-timeout=60000
spring.datasource.hikari.maximum-pool-size=10

prod:
spring.datasource.hikari.connection-timeout=15000
spring.datasource.hikari.maximum-pool-size=200
```

而为了方便开发调试，我们仅在开发环境打开了 SQL Trace。

```
dev:
spring.jpa.show-sql=true
```

Movie Service 与配置中心 ACM 集成

现在我们将集成 Movie Service 与 ACM，以便从 ACM 中获取对应环境的配置。关于如何在 Spring Cloud 中使用 ACM，请参考 Spring Cloud ACM。

为 movie service 引入 ACM 依赖

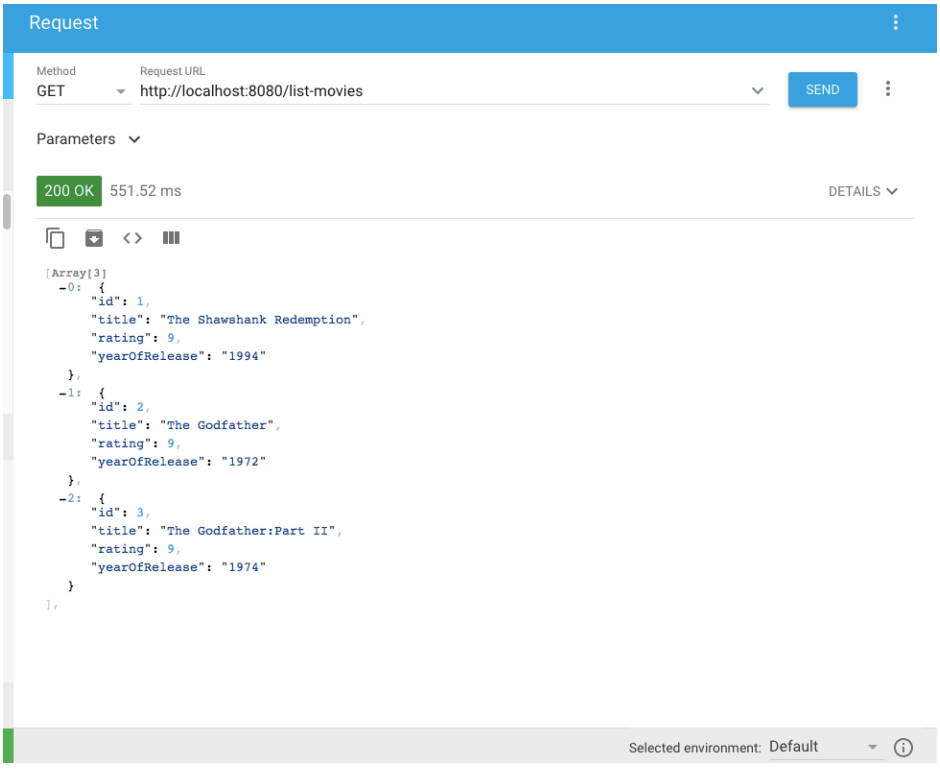
```
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-acm</artifactId>
<version>1.0.1</version>
</dependency>
```

在 application.properties 配置 ACM 连接信息、namespace、accessKey、secretKey 等信息

```
spring.application.name=movie-service
spring.application.group=com.alibaba.cloud.acm

alibaba.acm.endpoint=acm.aliyun.com
alibaba.acm.namespace=<your_namespace_id>
alibaba.acm.accessKey=<your_ak>
alibaba.acm.secretKey=<your_sk>
```

注意：您可以在 ACM 的“命名空间详情”或者“配置的示例代码”里找到您的 namespace_id、accessKey、secretKey 等信息。

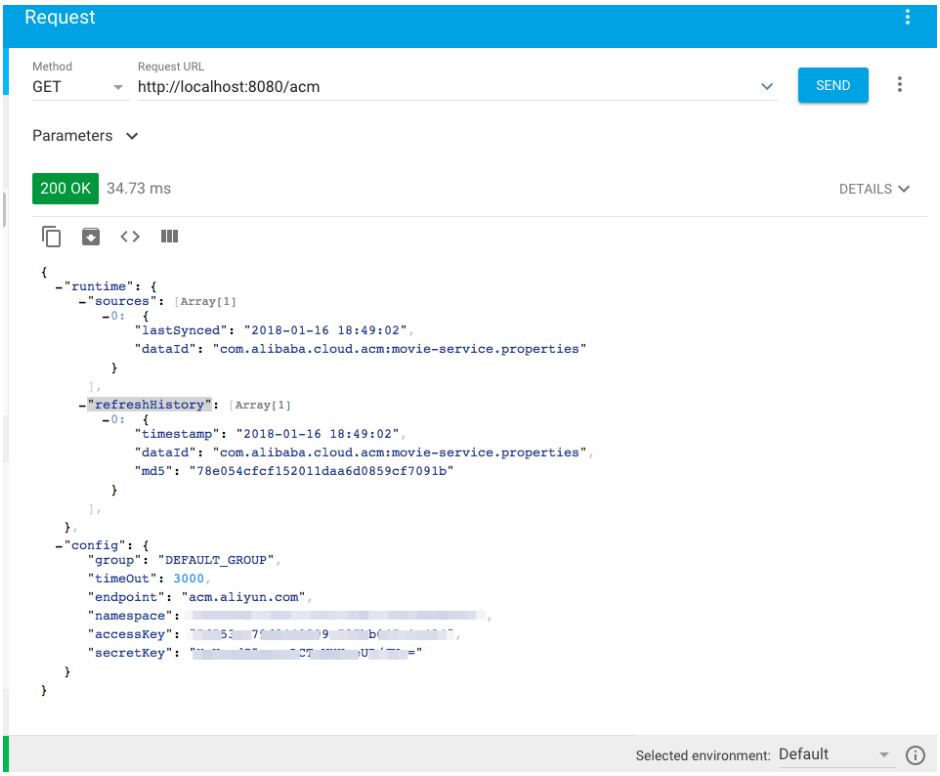
在浏览器里访问 Movie Service



查看 ACM 配置推送刷新信息

如果在 movie service 引入了 spring-boot-starter-actuator 依赖，并且在 application.properties 设置了

management.security.enabled=false，则可以通过端点http://<<ip:port>>/acm看到应用的配置消费及刷新情况。



也可以在 ACM 控制台上查看配置的推送轨迹、配置版本等信息。具体使用方法可参考 ACM官方文档。

ACM 与 Spring Cloud Config 的简单对比

对比项	Spring Cloud Config	阿里云 ACM
Spring Cloud 无缝集成	支持	支持
源码分发方式	开源	即将开源
收费模式	免费	免费
大规模（超 10 万配置）生产验证	无公开的大规模生产验证案例	阿里巴巴数据中心生产环境超百万级配置，每天超亿级配置变更推送，双 11 等严苛场景验证
配置管控 UI 控制台	无控制台，依赖 IDE、GIT 等第三方工具	专业的配置管理 UI 控制台
多语言支持	主要支持 Java 生态，无其他语言的原生客户端	支持 nodejs、c++ 等原生多语言客户端
多机房、同城双活、异地多活、多可用区等架构	依赖 GIT、ZooKeeper 等能力支持，官方无明确说明	支持
配置变更推送	依赖 RabbitMQ/KAFKA	内置的推送机制，无外部依赖
大规模推送时效	依赖 GIT Web Hook 等 SLA、WEB HOOK 在企业级大规模生产能力待验证	工业级、毫秒级

配置变更审计能力	弱	内置的审计机制（审计能力符合国家安全等保三级标准）
推送轨迹	无法查看配置推送到客户端的实时监测	有配置变更推送轨迹帮助监控配置变更推送状况
数据隔离	application、profile、label、git repo 等隔离策略	除 Spring Cloud 提供的隔离级别，还提供多租户、app、data_id、group 等多级隔离策略
生产运维成本	高（必须对 GIT/RabbitMQ 等有足够的知识储备和人才储备）	低（无三方组件依赖)
高可用	N/A（客户自行承担风险）	99.99%（阿里云承担风险）
安全通信	支持 SSL	支持 SSL
容灾	2 级（存储，服务器缓存)	3 级，另有客户端本地容灾能力

总结

在本文中，我们以测试和生产环境连接不同的数据库、配置不同的数据源（包括连接池）参数为例，介绍了如何搭配使用阿里云配置中心 ACM 与 Spring Cloud，帮助您在微服务架构中简化环境配置管理。

工程下载

本文的样例工程可以从 movie-service.tar.gz 下载。

该工程在以下版本环境测试通过：

- Spring Cloud Edgware.RELEASE
- Spring Boot 1.5.9.RELEASE
- HikariCP 2.7.6
- MySQL 5.7.11
- ACM 4.2.0
- ACM Spring Cloud SDK 1.0.1

注意：在本地运行该工程前，请务必在 application.properties 里设置您自己的 ACM accessKey 和 secretKey。

其他最佳实践

以下是 ACM 的其他最佳实践：

- CTO 指南——为什么说超过两台 ECS 就要考虑如何避免“配置飘移”问题
- 如何在云上安全高效地存放您的配置——代码示例
- 如何在阿里云上安全的存放您的配置——续
- 如何在阿里云上安全的存放您的配置
- 微服务与配置中心：别让您的微服务被配置管理“绊”一跤
- 现代应用架构中的配置管理面临的挑战
- 使用阿里云配置管理 ACM 实现 Zookeeper 依赖服务的透明 Failover 迁移