ALIBABA CLOUD

阿里云

应用配置管理 ACM

使用场景

文档版本：20210923

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.消息队列 RocketMQ 客户端流控设计 ---- 05
2.利用配置中心规范构建 PaaS 服务配置 ---- 10
3.使用ACM简化Spring Cloud微服务环境配置管理 ---- 14
4.其他场景化文档 ---- 24

1.消息队列 RocketMQ 客户端流控设计

消息队列 RocketMQ 是一种常用的异步 RPC 技术。本文以阿里云消息队列 RocketMQ 为例，介绍了如何使用 ACM 对消息队列 RocketMQ 实现流量控制。

迁移到MSE Nacos

ACM进入下线状态，所有配置管理相关的需求由MSE中的Nacos承接（ACM独享版，更好的安全和稳定性）。您需要在ACM控制台导出配置，然后在MSE控制台导入之前导出的配置即可完成迁移。具体操作，请参见将应用配置从ACM迁移到MSE Nacos。

消息队列 RocketMQ 流量控制简述

对于消息队列 RocketMQ 调用，常用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，每次消费时在客户端增加一个延时来控制消费速度，此时理论上消费并发最快速度为：

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

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
```

由上可知，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，对消息队列 RocketMQ 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。

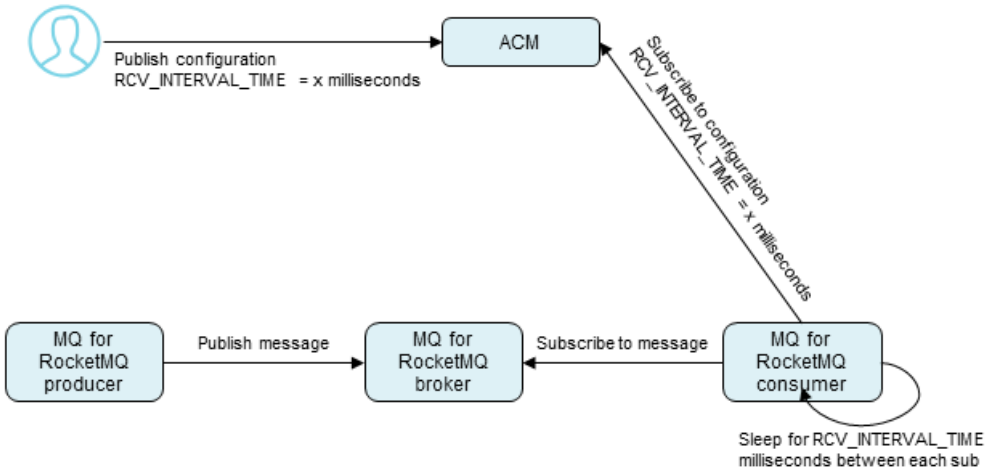
若使用以上的流量控制方法在分布式架构下做到全局动态控制，可通过配置中心下发流控参数来实现。

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消息队列 RocketMQ（消息队列）和 ACM（应用配置管理）产品，语言为 Java。

> **说明** 以消息队列 RocketMQ 为例是因为：截至目前，消息队列 RocketMQ 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消息队列 RocketMQ 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），所有消息队列 RocketMQ 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

以下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，参见消息队列 RocketMQ 和应用配置管理 ACM 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

1. 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();
}
```

2. 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();
    }
  }
});
```

设置消息队列 RocketMQ 消费延时逻辑

完整实例如下。

> **说明**
>
> - 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) {
    //消息队列 RocketMQ 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();
      }
    }
    //消息队列 RocketMQ Subscribe interval logical ends
    System.out.println("Receive: " + message);
    /*
     * Put your business logic here.
     */
    doSomething();
    return Action.CommitMessage;
  }
});
consumer.start();
```

运行结果

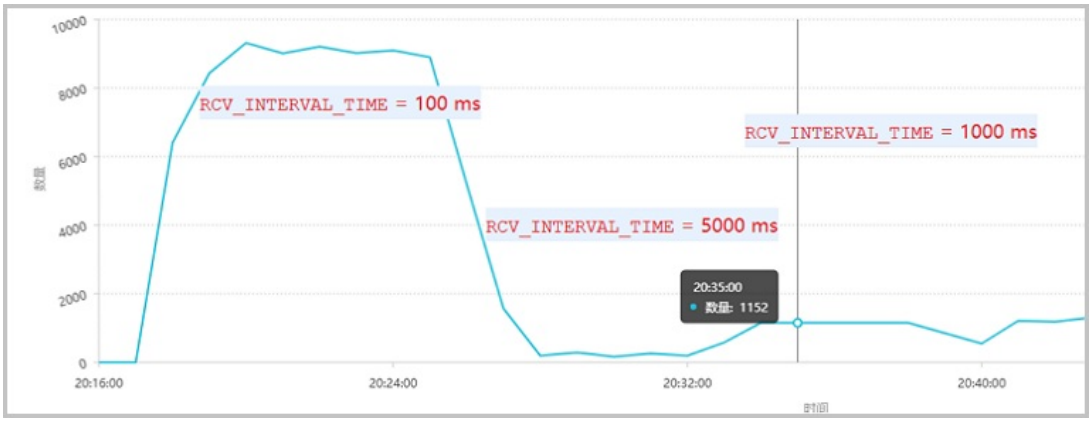
在队列内的消息无限多的情况下，单机运行 Consumer 进行消费，分三段测试，分别运行约 5 分钟，通过 ACM 配置推送来达到以下效果。

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

在单消息队列 RocketMQ 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

从以上结果中可以得出：消费和 tpm 成反比，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。与预期结果相符。单机性能结果如下所示。



2.利用配置中心规范构建 PaaS 服务配置

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

迁移到MSE Nacos

ACM进入下线状态，所有配置管理相关的需求由MSE中的Nacos承接（ACM独享版，更好的安全和稳定性）。您需要在ACM控制台导出配置，然后在MSE控制台导入之前导出的配置即可完成迁移。具体操作，请参见将应用配置从ACM迁移到MSE Nacos。

配置规范问题的产生

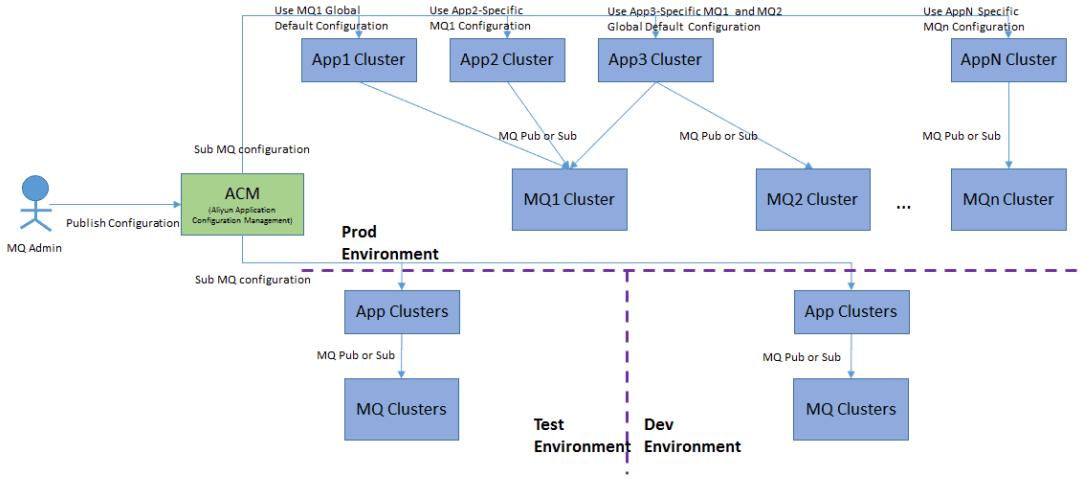
如果是单一应用的单一属性配置，使用以下配置文件即可完成，没有配置规范的问题。

```
//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许多问题。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。

下文详细介绍了以规范的配置命名格式来进行限流设置的方法。为了说明配置命名规范，首先需要介绍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。

1. 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。使用租户隔离，不同的配置在不同的租户可以重名，且具有不同的鉴权机制。
- 最小配置集合：若干配置通过配置中心组合成一个配置集合。使用最小配置集合来更改和发布不同配置，可实现统一发布和统一处理。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：在配置中心中设置具体配置内容。

2. 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，是中国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 需要不同 AK/SK 鉴权，Group 则不需要。	一个 Git 项目为一个租户。	没有直接可用的租户隔离技术。
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，可以实现间接配置路径的目标。	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

1. 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。

- 用 Group 来区分不同集群提供的 MQ 服务，可以达到隔离配置和简化访问形式的效果。
  - 例如，对于专门为子部门的核心交易部门服务的 MQ 集群，和为子部门的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。这种方法使所有应用采用同一（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后缀 .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
```

3. 配置具体 KV 的设置

在很多配置中心产品（例如 Appolo、ACM System Manager Parameter Store）中，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

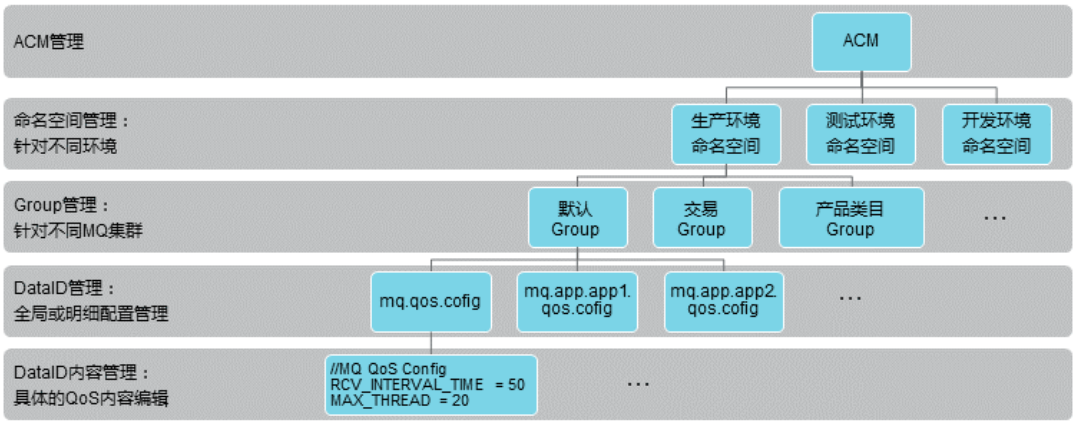
- 将常用配置聚合成一个 DataID，并编辑为一个配置文件（格式不限，例如 Properties、JSON、XML 等）
  - 例如 mq.global.qos 设置如下：

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

在实践中发现第二种方法不但具有更大的灵活性，而且支持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。

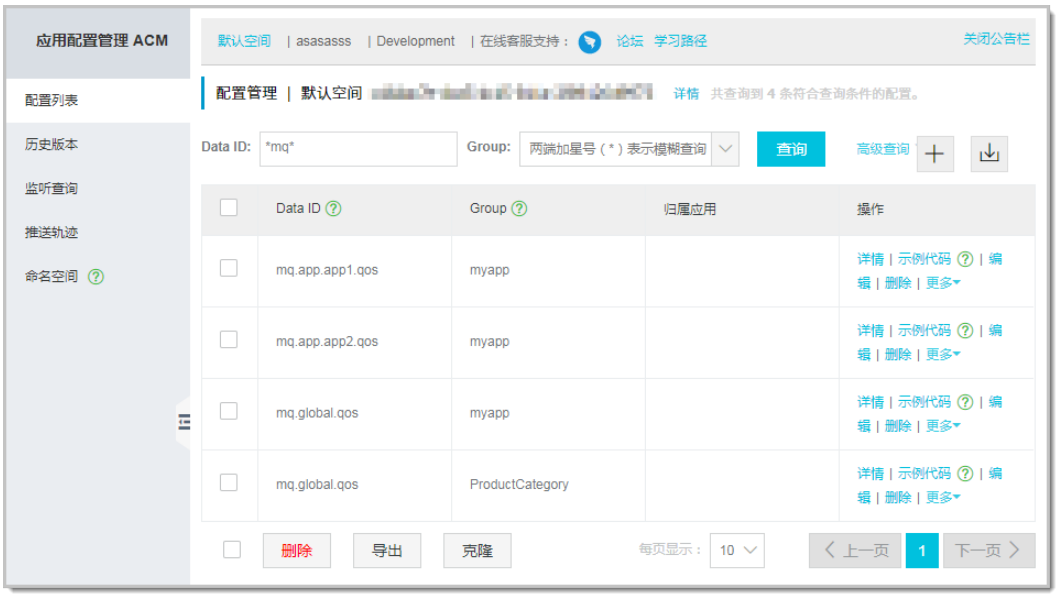
4. 配置结构示意图

以上设计的配置结构示意图如下：



5. 方案优点总结

- 通过 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，达到配置项统一且各个环境互不干扰的效果。
- 通过 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 客户端即可方便地查找 MQ Default 全局配置和应用特殊配置。此外，在**配置列表**页面上，管理员可通过在关键字两端各加一个星号（*），方便地找出所有 MQ 规则。示例如下：



参考

- 配置变更风险管理

3.使用ACM简化Spring Cloud微服务环境配置管理

在本文中，以测试和生产环境下连接不同的数据库、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。

迁移到MSE Nacos

ACM进入下线状态，所有配置管理相关的需求由MSE中的Nacos承接（ACM独享版，更好的安全和稳定性）。您需要在ACM控制台导出配置，然后在MSE控制台导入之前导出的配置即可完成迁移。具体操作，请参见将应用配置从ACM迁移到MSE Nacos。

配置的环境属性

在系统持续交付的过程中，运行环境的多样性和复杂性增加了配置管理工作的负担。这一点在Eugen Paraschiv的博文Configuration Must Be Environment Specific里有简单的阐述。

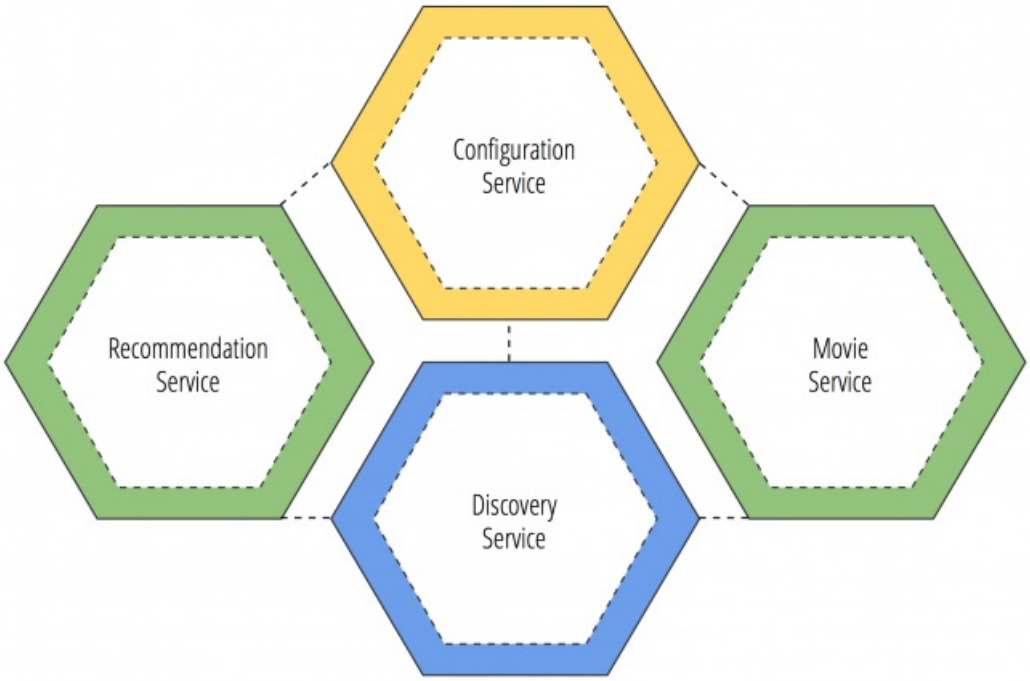
环境配置的差异导致了应用的构建物（artifact）在各个环境不能保持一致，而且有时候Docker无法轻易达成只搭建一次即可在所有环境运行的效果。以下列举一些简单的例子来帮助您理解：

- 在开发环境中logLevel应设置为DEBUG，在预发环境中为INFO，在生产环境中为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B库。
- 新的特性仅在线上的杭州单元开放，其他单元环境暂缓。

下文中，简要介绍了使用阿里ACM配置管理产品在Spring Cloud中替代Spring Cloud Config的方法，帮助您理解基于ACM简化微服务环境配置管理的方案。此外，下文还将简要比较说明ACM与Spring Cloud Config方案的优劣。

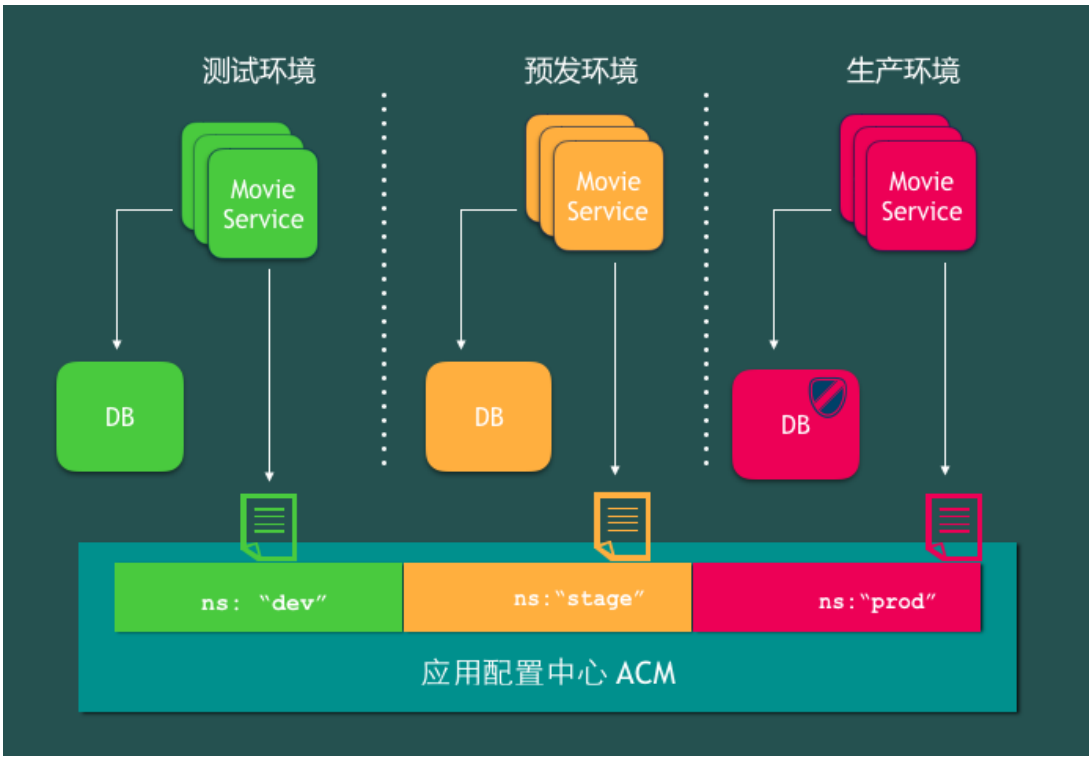
场景故事

为帮助您理解需求和场景，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，因此应用需要在不同的环境下具备不同的数据源配置、连接池配置、数据库安全配置等。

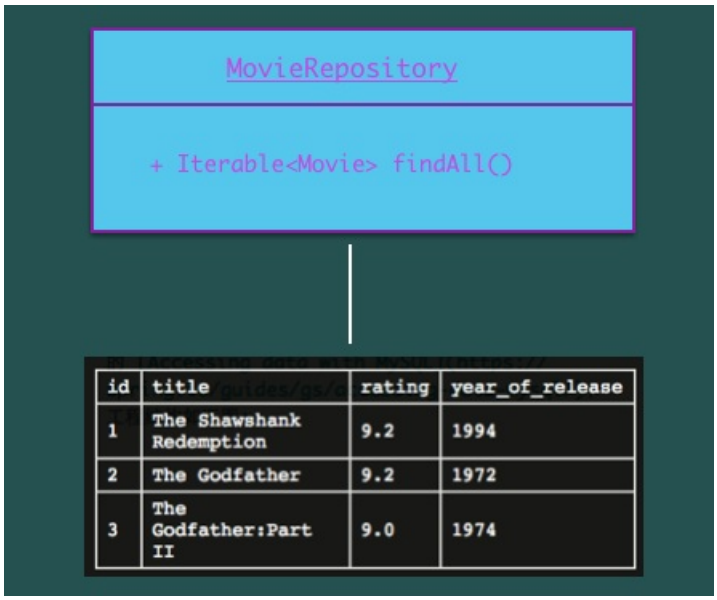
下图介绍了如何基于阿里云ACM的Namespace映射不同环境，为Movie Service在不同运行环境下设置不同的数据源配置。



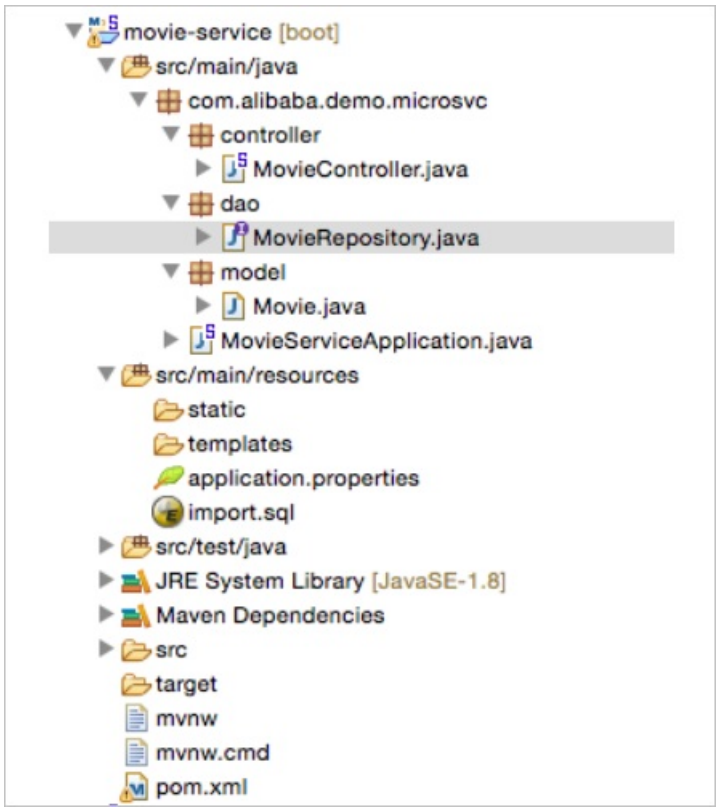
创建微服务Movie Service

- 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
```

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

- 创建WEB Controller

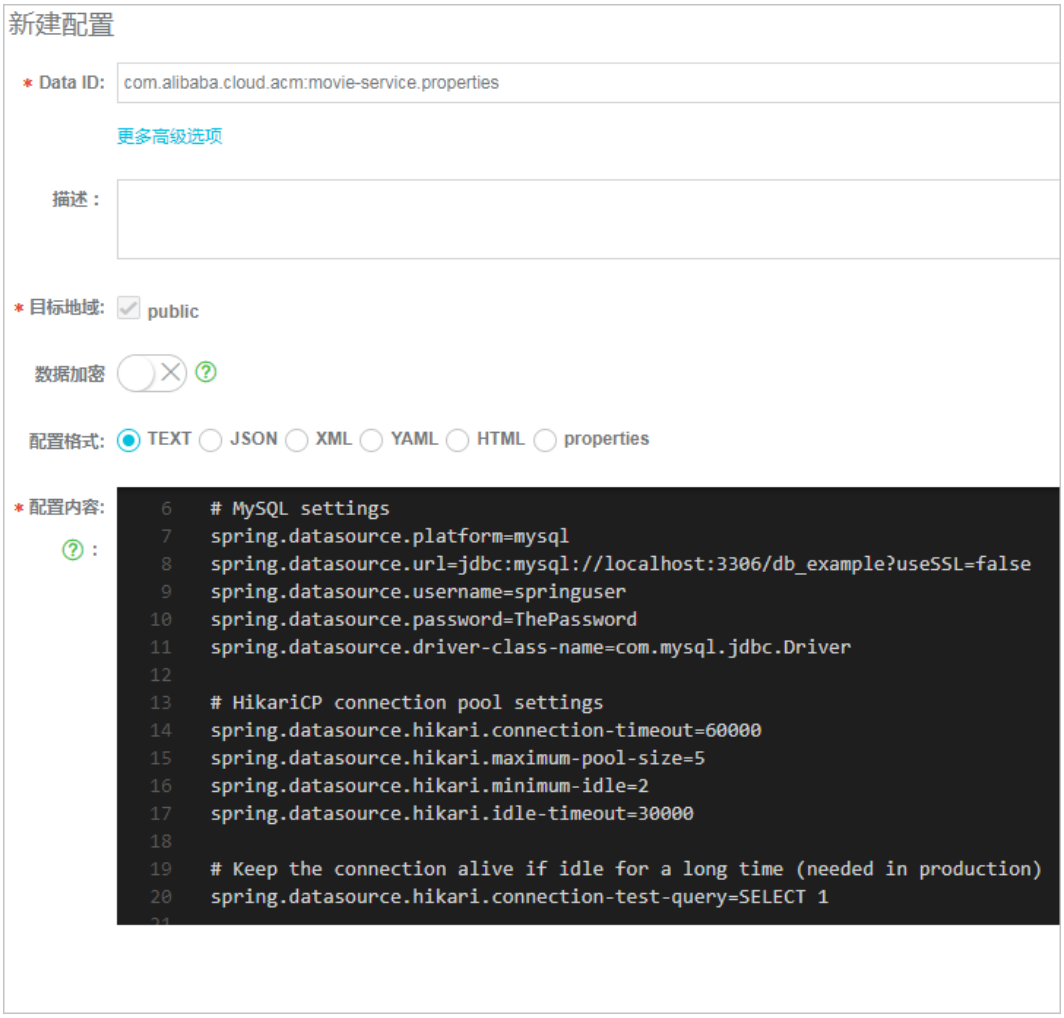
```
package com.alibaba.demo.microsvc.controller;
 import org.springframework.beans.factory.annotation.Autowired;
 import org.springframework.web.bind.annotation.RequestMapping;
 import org.springframework.web.bind.annotation.ResponseBody;
 import org.springframework.web.bind.annotation.RestController;
 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

> **说明** 在阿里云上使用ACM的前提是开通该项服务，开通流程可参见开通ACM服务。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



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，请参见Spring Cloud ACM：

- 为movie service引入ACM依赖

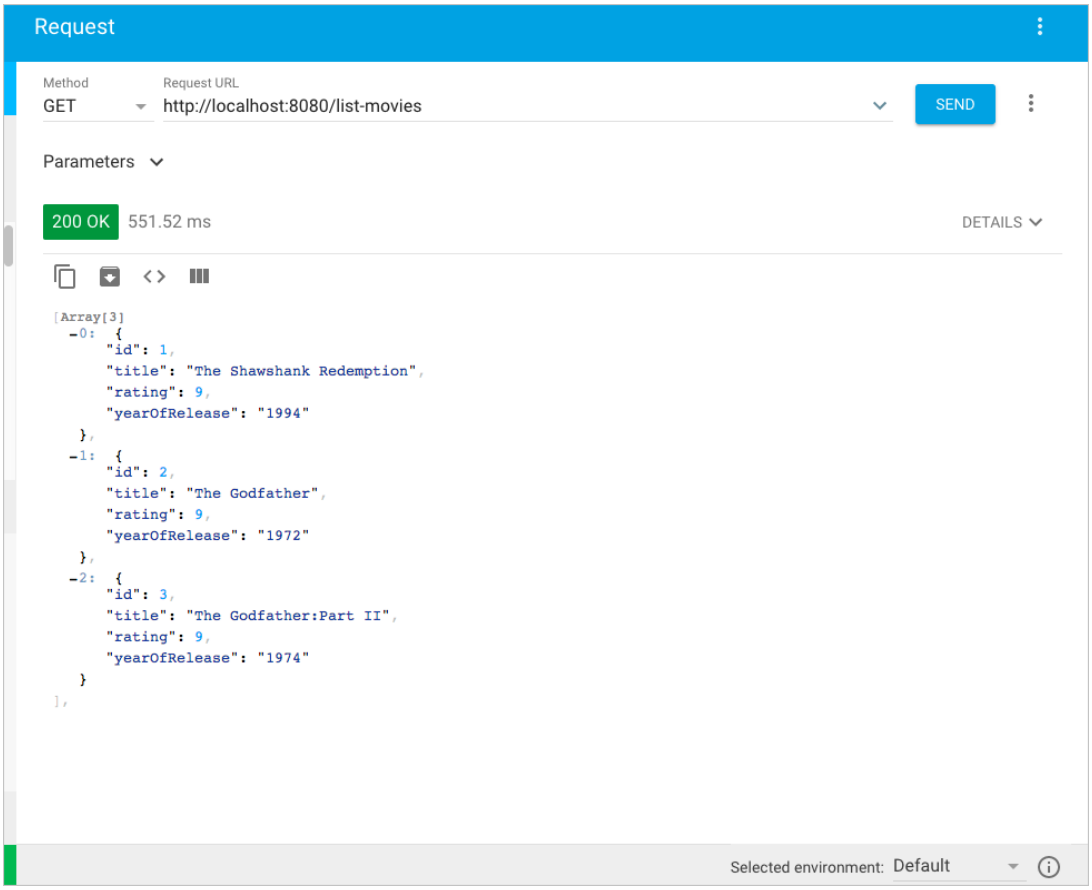
```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-acm</artifactId>
  <version>1.0.1</version>
</dependency>
```

- 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>
```

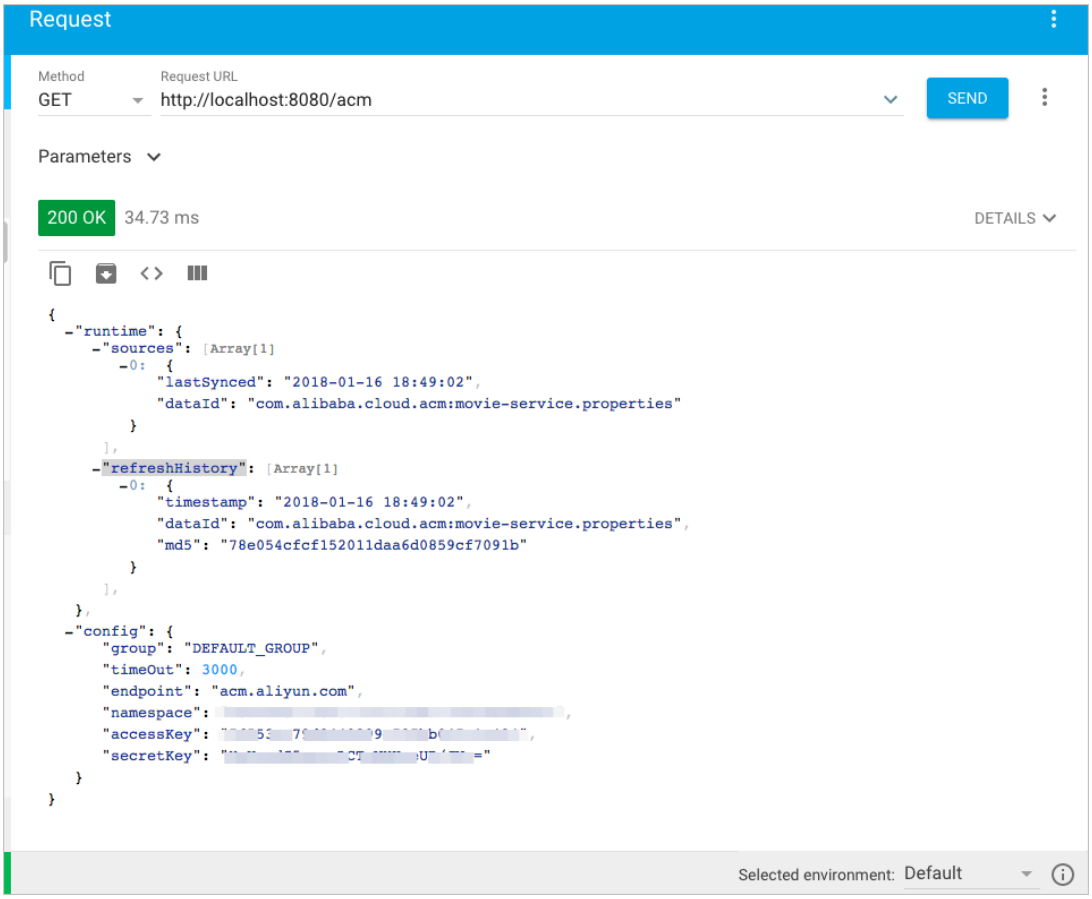
> **说明**
>
> 您可以在ACM的“命名空间详情”或者“配置的示例代码”里找到您的namespace_id、accessKey、secretKey等信息。

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，详情请参见查看历史版本和回滚配置以及查询推送轨迹。

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

对比项	Spring Cloud Config	阿里云ACM
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

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

> **说明** 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。

4.其他场景化文档

本文罗列了 ACM 的其他场景化文档。

> **说明** ACM进入下线状态，所有配置管理相关的需求由MSE中的Nacos承接（ACM独享版，更好的安全和稳定性）。您需要在ACM控制台导出配置，然后在MSE控制台导入之前导出的配置即可完成迁移。具体操作，请参见将应用配置从ACM迁移到MSE Nacos。

- ACM 配置中心实战：Spring + MyBatis + Druid + ACM
- 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
- 使用阿里云配置管理 ACM 实现 Zookeeper 依赖服务的透明 Failover 迁移