自研配置中心微服务
需求分析及产品设计方案

（第一版）

郑琳

2023 年 8 月 5 日

目录

一、项目背景

1.1 关于参数配置

从接触电脑和手机开始，各种参数配置就伴随着日常操作出现，从操作系统到应用软件，参数配置是灵活性的一个重要体现。

同样，在一个业务系统中，参数配置也尤为重要，不管从上下游系统对接、数据库访问等外部对接，还是服务提供、日志记录等内部实现，使用参数配置都可以大大提升系统的应用场景和业务兼容性。

而传统的单应用配置形式，有着存在一些潜在缺陷，如随着规模的扩大，部署效率降低，团队协作效率差，系统可靠性变差，维护困难，新功能上线周期长等，所以迫切需要一种新的架构去解决这些问题。

1.2 配置中心

面对着业务系统越来越复杂，对吞吐量、并发量要求越来越高，使用微服务架构可以很好的满足系统模块化和自由部署的目的。

而参数配置，则以配置中心这一微服务的方式，从各个系统中剥离，配置中心可进行各类参数配置的独立存储，通过 API 的方式与为给其他微服务或者用户提供支持，是一个非常实用的基础服务型微服务。

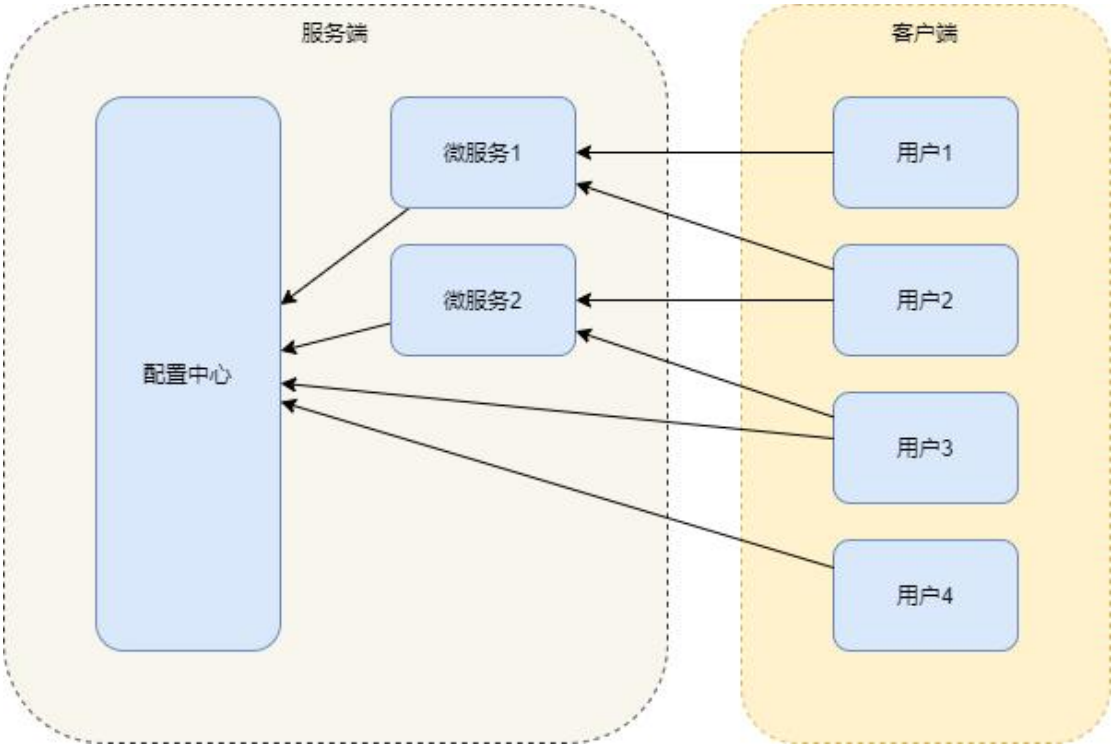

图 1-1 配置中心示意图

1.3 自研配置中心

当下，市面上已经有不少成熟的配置中心产品，有开源的也有闭源的，各有各的特色和优缺点，有些已经被广泛应用。

既然有这么多的配置中心产品，那自研的是不是意味着重复造轮子，是一个劳民伤财的事情？其实不然，使用现成产品，固然能带来很多便利，但是自研产品，也有着自己的优势。

1.3.1 自主可控

自主可控，才可以高枕无忧。

虽然市面上已经有不少成熟的开源配置中心产品，也可以在一定程度上做到风险可控，但大多是基于 Java 开发的，开发语言不熟悉则无法进行二次开发，比如 Apollo 默认只支持 MySql 数据库，想要使用其他数据库，就必须要安排一个 Java 程序员进行二次开发。

另外，开源项目的不确定性较差，比如 XDiamond 目前已经停止维护，之前使用该产品的项目，可能需要被迫迁移到其他产品中。

而自研产品，在很大程度上，可以进行自由的定制化开发，同时也可以最大程度的规避产品更替风险。

1.3.2 性价比高

最优的方案，就是花最少的钱，带来最大的收益。

相比与一些大型项目的构建，配置中心的服务较为单一，开发周期较短，后期维护的成本也不高。

同时，现有的很多成熟开源产品及源代码，可以为项目提供很好的参考，大幅度缩短项目开发周期，也将开发的试错成本降到最低。

虽然配置中心不能产生直接价值，但相比较于能提供一个长期稳定的配置服务来说，稳定运行的时间越长，所产生的价值越高。

1.3.3 技术沉淀

一分耕耘，一份收获。

与自研一个大型系统相比，配置中心的自主研发，开发的复杂度较低，但是作为独立产品，其整个产品的研发流程是完整的，对于开发人员来说，就是一场极简的研发流程体验，不管是项目中使用的技术点的巩固，还是产品研发流程中的各个细节，都是宝贵的财富。

1.3.4 商业价值

产品，就应该是为市场服务的。

配置中心如果作为一个独立的产品进行销售，那确实会很不尽人意，但是如果他作为一个产品系列中的一个附加产品，不仅可以让产品线看上去更加丰富，还可以比较隐性的体现出研发实力，获得客户认可。

另外，以产品簇的形式展示，不管从报价层面还是用户心里层面，都能获得不少优势。

虽然配置中心很难单独进行销售，但只要搭配得当，他也能发挥不少作用。

二、配置中心产品解构

2.1 开源产品

开源的配置中心产品目前还是比较多的，我们以目前国内比较主流的产品，分析其产品功能加以借鉴。

2.1.1 Apollo

Star 28K+

Apollo（阿波罗）是携程框架部门研发的分布式配置中心，能够集中化管理应用的不同环境、不同集群的配置，配置修改后能够实时推送到应用端，并且具备规范的权限、流程治理等特性，适用于微服务配置管理场景。

2.1.2 XDiamond

Star 500+ 已停止维护

全局配置中心，存储应用的配置项，解决配置混乱分散的问题。名字来源于淘宝的开源项目 Diamond，前面加上一个字母 X 以示区别。

2.1.3 Disconf

Star 5K+

专注于各种「分布式系统配置管理」的「通用组件」和「通用平台」，提供统一的「配置管理服务」包括 百度、滴滴出行、银联、网易、拉勾网、苏宁易购、顺丰科技 等知名互联网公司正在使用。

2.2 分发模式

不难看出，所有的配置中心，都提供了一个中心化配置界面，这也符合配置中心这一名称的含义。

不过，不同的配置中心，分发模式其实是有所区别的，有些是集中化配置后提供接口进行分发，有些则是提供分布式同步的方式直接下载配置文件。

2.2.1 全局模式

Apollo/XDiamond

全局模式，即所有配置信息只存在配置中心，所有服务或者用户需要配置信息时，实时通过 Api 获取配置信息。

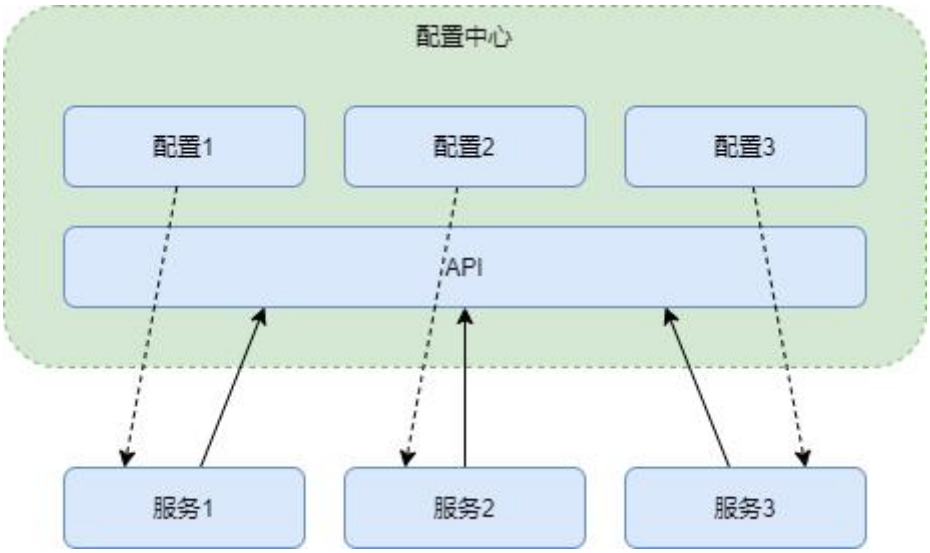

图 2-1 基于 API 的中心全局配置

全局模式的优势：

- 数据高度统一，服务每次加载获取到的都是最新的配置；
- 无须存储，可适应一些无法进行存储的极端应用环境；
- 配置安全，使用端没有真实的配置文件，就没有泄露的风险。

全局模式的缺点：

✧ 高度依赖网络，配置中心必须 7*24 小时不间断提供服务，一旦配置中心服务出现问题，那其他服务也将面临无法正常启动的风险；

✧ 使用者无法离线调试，使用时必须连接配置中心获取信息。

2.2.2 分布模式

Disconf

与全局模式不同的是分布模式，分布模式最大的特点是配置中心与使用者同时都会存储配置文件，使用者使用同步手段保持与配置中心之间的数据一致性。

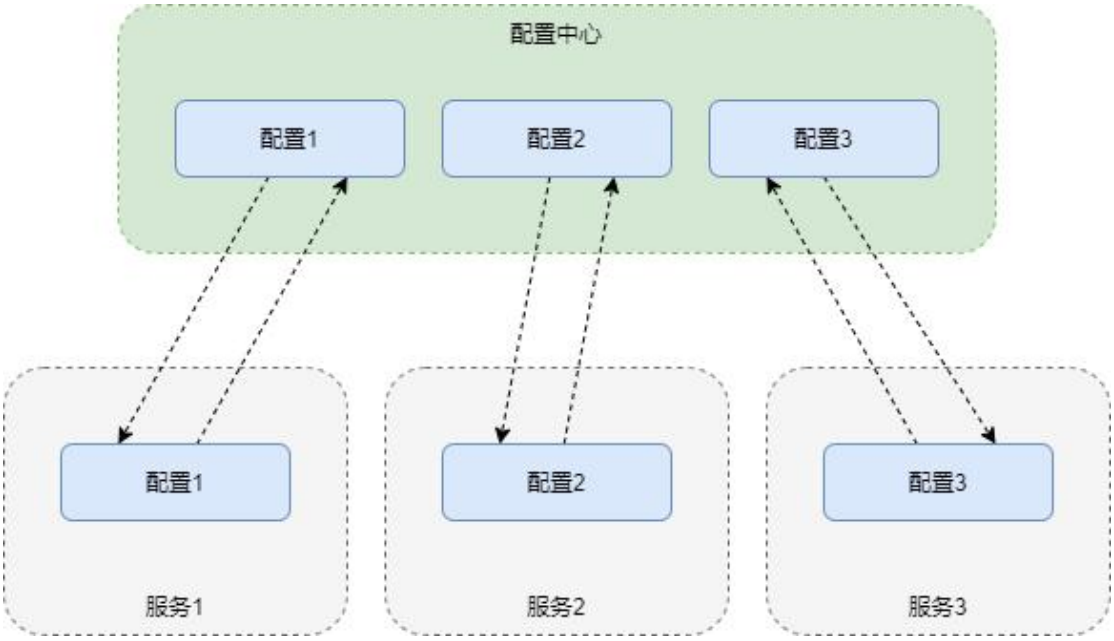

图 2-2 基于同步的配置分发

分布模式的优势：

➢ 网络依赖程度低，使用者有独立的配置文件，也就意味着当使用者可以无法连接配置中心时也可以独自运行，只在需要更新配置时触发同步即可；

➢ 自由度高，配合版本管理，可以在使用者端修改配置再同步给配置中心，再由配置中心分发给其他使用者

➢ 测试环境搭建简单，可以完全先自建配置文件测试，测试完成后再对接配置中心。

分布模式的缺点：

✧ 配置文件管理复杂度高，带来的项目复杂度也相应提升，开发成本更高；

✧ 存在配置文件同步时间差，使用者的配置信息也有可能因为网络或其他原因与配置中心不同步，因为不影响使用会增加发现问题所在的难度，所以变相增加了运维难度。

2.3 回滚与灰度的支持

回滚与灰度是 Apollo 特有的功能支持。

2.3.1 回滚

Apollo 的配置存在版本控制，也就是可以进行版本的回溯，一旦发现新的配置有问题，可以在第一时间进行版本回调，恢复原来的配置信息，这一功能，对处理时效敏感的项目，其实是很有效的，可以在短时间内避免因配置造成的系统

停摆；但在对处理时效不敏感的项目，此功能则没有太大的作用。

2.3.2 灰度发布

Apollo 的配置是需要发布后才能生效的，不过此功能比较适用于权限分离的场景，也就是编辑配置和发布者并非同一人时，才有比较明显的作用，对于项目实施复杂度不高的系统，此功能能起到的作用也不大。

2.3 配置隔离

所有的配置中心，都支持不同环境的配置隔离，Apollo 甚至支持跨集群的不同环境配置隔离。

对于不同的环境提供配置隔离，应该时一个配置中心的基础功能。

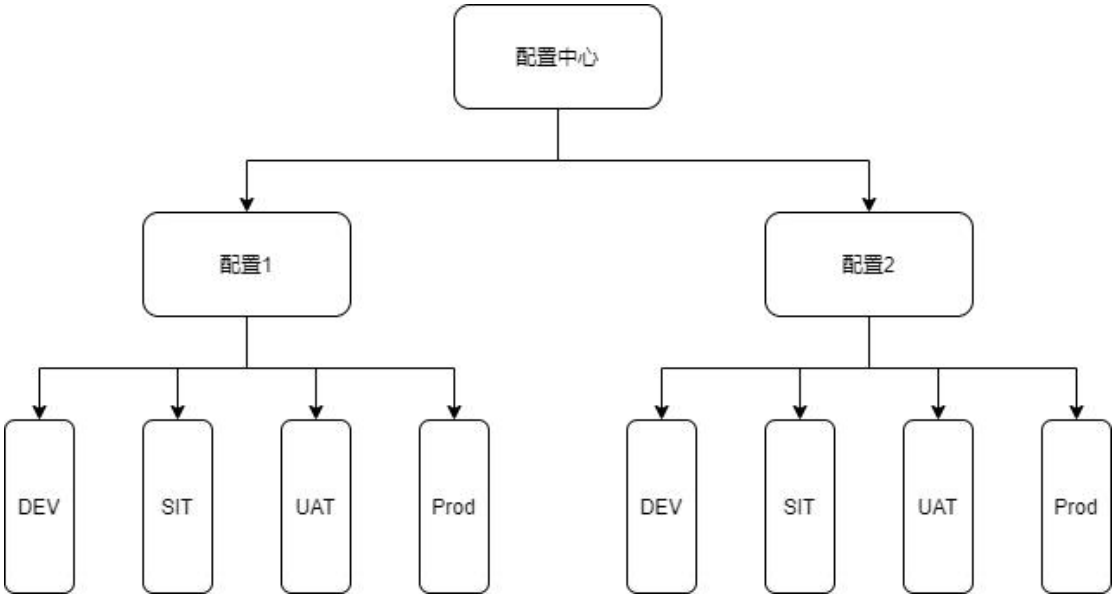

图 2-3 配置隔离

2.4 热更新

热更新就是针对运行中的服务变更配置，应用于某些运行时敏感的配置项，比如上下游系统对接的变更等对实时操作影响比较大的参数配置。

而热更新则分为轮询与推送两种方式。

2.4.1 轮询

Apollo

轮询，就是使用者每隔一段时间，向配置中心发起一次请求，配置中心返回

最新的配置信息，使用者接收到新的配置信息后实时更新配置。

轮询的优势：

- 实现成本低，轮询接口可以和常规的获取接口相同，只需在 SDK 中加入定时触发即可；
- 兼容性较强，一般能正常访问配置中心，即可支持轮询。

轮询的缺点：

- 使用者运算资源浪费较大，因为每隔一段时间需要做一次完整的数据交互，因此不可避免会产生多余的资源开销；
- 实时性较低，轮询是有间隔时间的，所以并不是真正意义上的实时，特别是需要考虑运算压力时，可能将间隔时间设置的比较大。

2.4.2 推送

XDiamond/Disconf

推送，就是由配置中心主动做推送，当配置修改时，配置中心主动向 Zookeeper 推送变更消息，使用者接收到变更消息后，再向配置中心获取最新的配置信息。

推送的优势：

- 实时性非常高，Zookeeper 的消息推送往往能在 1 秒内到达使用者端，整个更新流程一般在秒级或者更短时间内完成；
- 使用者资源占用较低，没有更新时，几乎不占用使用者的运算资源。

推送的缺点：

- 需要 Zookeeper 的支持，相当于要多一个微服务来支持整个系统；
- 开发成本较高，需要额外针对 Zookeeper 做对接开发。

三、总体方案

在分析现有产品的基础上，我们可以大胆的去设计更加符合我们自身实际的配置中心产品。

3.1 分发模式

全局模式 + 配置缓存

在对比全局模式和分布模式的优缺点后，有几点时需要我们迫切解决的：

- ✧ 以最低的成本来实现最高的可用性
- ✧ 允许使用者独立进行调试测试，降低服务间的相互依赖程度
- ✧ 避免出现数据一致性问题

从成本与一致性问题出发，那分布模式显然不太适合我们的实际情况，那全局模式就是我们唯一的选择，那就只剩下了怎么允许独立调试的问题了。

重新整理分布模式的优势，发现其能够脱离配置中心工作的最大因素，是分布模式将配置文件同步到了使用者本地。

那这样的话，如果我们在获取了全局配置之后，在使用者端缓存一份配置的备份，作为无法连接连接配置中心时的一个备用信息，那问题也就得到了解决。

3.2 轮询热更新

成本为王

轮询方式，无论从开发成本，还是后期的维护成本，都是推送方式无法匹敌的，在项目的商业化尚未明朗的当下，轮询无疑是第一选择。

同样，对于性价比不高的回滚和灰度功能，则直接先不予考虑。

3.3 基于项目和环境的配置隔离

站在巨人的肩膀上

配置隔离这一块，现有的产品基本给我们提供了一个标准答案，在不考虑集群一类过于复杂的场景时，基于项目和环境的配置隔离，已经足够满足使用。

四、详细设计

4.1 总体业务流程

明确了总体方案后，大致的业务流程呼之欲出：

- 配置中心先进行基础数据的建立
- 使用者向配置中心发起配置信息获取请求
- 配置中心收到请求后，先进行身份校验
- 身份通过校验后，查询对应项目和环境的配置信息
- 配置中心输出配置信息给使用者
- 使用者接收配置信息，并进行缓存
- 使用者读取缓存中的配置信息，并应用配置

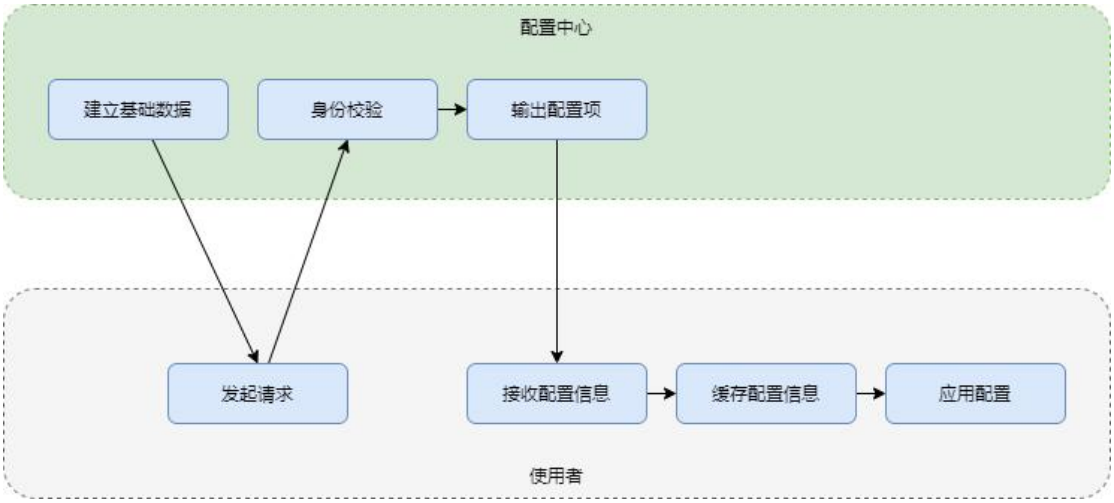

图 3-1 总体业务流程

4.2 总体模块设计

从业务流程中，基本可以将整个系统划分为基础数据、配置存储、配置缓存和配置应用四大模块。

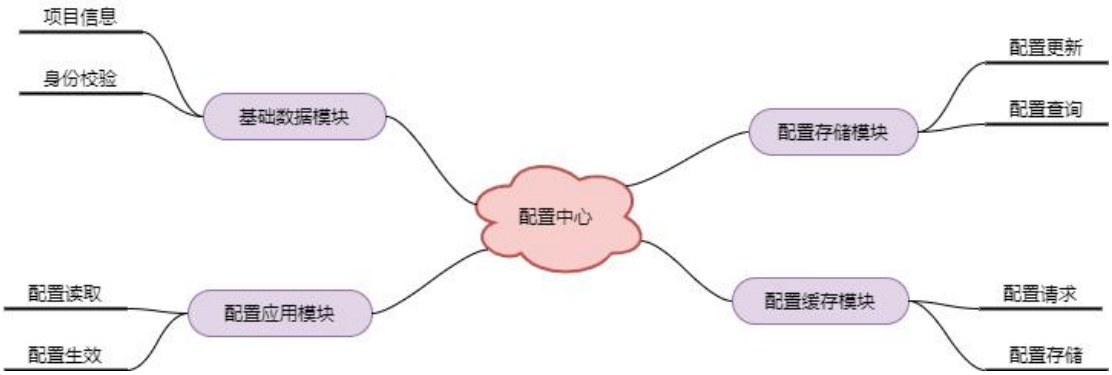

图 4-1 总体模块设计

4.3 详细模块说明

4.3.1 基础数据模块

基础数据模块负责整个配置中心的基础架构，主要提供基础信息及身份校验。

图 4-2 基础数据模块

4.3.2 配置存储模块

配置存储模块主要是提供配置信息的管理及配置查询接口。

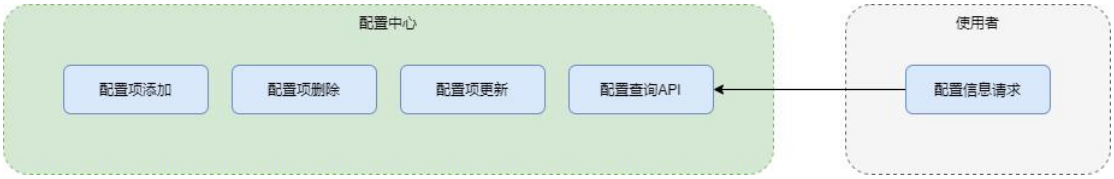

图 4-3 存储配置模块

4.3.3 配置缓存模块

配置缓存模块则是负责使用者向配置中心发起配置查询并建立配置缓存。

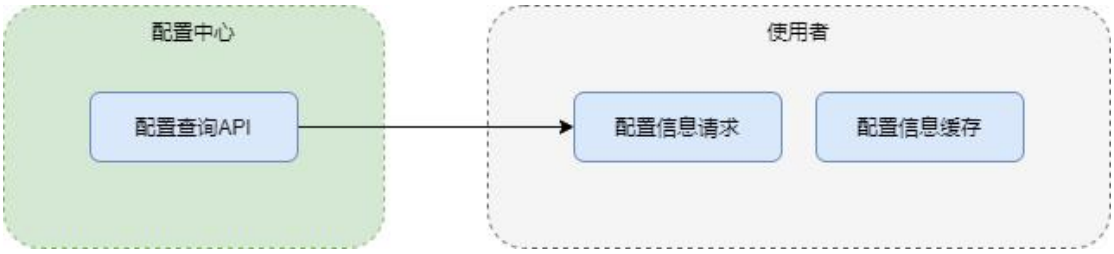

图 4-4 配置缓存模块

4.3.4 配置应用模块

配置应用模块则是负责使用者读取缓存配置并应用到实际项目中。

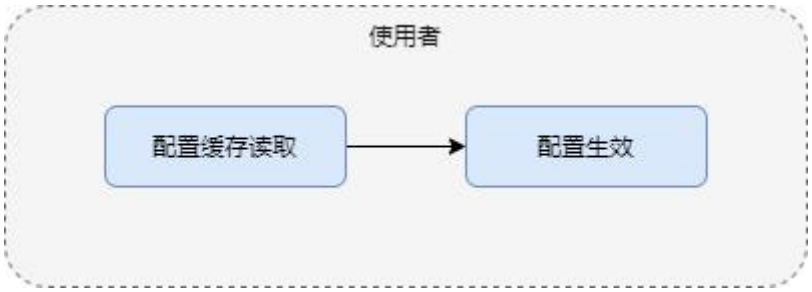

图 4-5 配置应用模块

五、方案实施

整个方案分为两块内容，可分开实施：

5.1 配置中心

即配置中心服务端主体

配置中心需要实施的模块如下：

模块	说明
基础数据模块	用户配置与登录、退出 项目分类信息管理 项目信息管理
配置存储模块	配置项管理 配置内容管理 配置查询接口
配置缓存模块	无
配置应用模块	无

5.2 开发 SDK

即使用者端的开发组件

开发 SDK 需要实施的模块如下：

模块	说明
基础数据模块	无
配置存储模块	无
配置缓存模块	配置查询接口对接 本地缓存写入
配置应用模块	本地缓存读取 配置应用