分布式配置管理平台XXL-CONF

书栈(BookStack.CN)

目　录

致谢

当前文档 《分布式配置管理平台XXL-CONF》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于2018-11-30。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/XXL-CONF

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

一、简介

1.1 概述

XXL-CONF 是一个轻量级分布式配置管理平台，拥有"轻量级、秒级动态推送、多环境、多语言、配置监听、权限控制、版本回滚"等特性。现已开放源代码，开箱即用。

1.2 特性

- 1、简单易用： 接入灵活方便，一分钟上手；
- 2、轻量级： 部署简单，不依赖第三方服务，一分钟上手；
- 3、配置中心HA：配置中心支持集群部署，提升配置中心系统容灾和可用性。
- 4、在线管理： 提供配置中心， 通过Web界面在线操作配置数据，直观高效；
- 5、多环境支持：单个配置中心集群，支持自定义多套环境，管理多个环境的的配置数据；环境之间相互隔离；
- 6、多数据类型配置：支持多种数据类型配置，如：String、Boolean、Short、Integer、Long、Float、Double 等；
- 7、跨语言：底层通过http服务（long-polling）拉取配置数据并实时感知配置变更，从而实现多语言支持。
- 8、高性能：得益于配置中心的 "磁盘配置" 与客户端的 "LocalCache"，因此配置服务性能非常高；单机可承担大量配置请求；
- 9、实时性： 秒级动态推送；配置更新后， 实时推送配置信息， 项目中配置数据会实时更新并生效， 不需要重启线上机器；
- 10、配置变更监听功能：可开发Listener逻辑，监听配置变更事件，可据此动态刷新JDBC连接池等高级功能；
- 11、最终一致性：底层借助内置广播机制，保障配置数据的最终一致性，从而保证配置数据的同步；
- 12、配置备份： 配置数据同时在磁盘与MySQL中存储和备份，并定期同步， 提高配置数据的安全性；
- 13、多种获取配置方式：支持 "API、 注解、XML占位符" 等多种方式获取配置，可灵活选择使用；
- 14、兼容Spring原生配置：兼容Spring原生配置方式 "@Value"、"${}" 加载本地配置功能；与分布式配置获取方式隔离，互不干扰；
- 15、分布式： 支持多业务线接入并统一管理配置信息，支撑分布式业务场景；
- 16、项目隔离： 以项目为维度管理配置， 方便隔离不同业务线配置；
- 17、高性能： 通过LocalCache对配置数据做缓存， 提高性能；
- 18、客户端断线重连强化：设置守护线程，周期性检测客户端连接、配置同步，提高异常情况下配置稳定性和时效性；
- 19、空配置处理：主动缓存null或不存在类型配置，避免配置请求穿透到远程配置Server引发雪崩问题；
- 20、用户管理：支持在线添加和维护用户，包括普通用户和管理员两种类型用户；
- 21、配置权限控制；以项目为维度进行配置权限控制，管理员拥有全部项目权限，普通用户只有分配才拥有项目下配置的查看和管理权限；
- 22、历史版本回滚：记录配置变更历史，方便历史配置版本回溯，默认记录10个历史版本；
- 23、配置快照：客户端从配置中心获取到的配置数据后，会周期性缓存到本地快照文件中，当从配置中心获取配置失败时，将会使用使用本地快照文件中的配置数据；提高系统可用性；
- 24、访问令牌（accessToken）：为提升系统安全性，配置中心和客户端进行安全性校验，双方AccessToken匹配才允许通讯；

1.3 发展

于2015年，我在github上创建XXL-CONF项目仓库并提交第一个commit，随之进行系统结构设计，UI选型，交互设计……

至今，XXL-CONF已接入多家公司的线上产品线，接入场景如电商业务，O2O业务和核心中间件配置动态化等，截止2018-10-24为止，XXL-CONF已接入的公司包括不限于：

```
- 1、深圳市绽放工场科技有限公司
- 2、深圳双猴科技有限公司
- 3、商智神州软件有限公司
- 4、浙江力太科技
- ……
```

> 更多接入的公司，欢迎在 登记地址 登记，登记仅仅为了产品推广。

欢迎大家的关注和使用，XXL-CONF也将拥抱变化，持续发展。

1.4 背景

> *why not properties*

常规项目开发过程中， 通常会将配置信息位于在项目resource目录下的properties文件文件中， 配置信息通常包括有： jdbc地址配置、redis地址配置、活动开关、阈值配置、黑白名单……等等。使用properties维护配置信息将会导致以下几个问题：

- 1、需要手动修改properties文件；
- 2、需要重新编译打包；
- 3、需要重启线上服务器（项目集群时,更加令人崩溃）；
- 4、配置生效不及时： 因为流程复杂， 新的配置生效需要经历比较长的时间才可以生效；
- 5、不同环境上线包不一致： 例如JDBC连接， 不同环境需要差异化配置；

> *why XXL-CONF*

- 1、不需要（手动修改properties文件）： 在配置中心提供的Web界面中， 定位到指定配置项， 输入新的配置的值， 点击更新按钮即可；
- 2、不需要（重新编译打包）： 配置更新后， 实时推送新配置信息至项目中， 不需要编译打包；
- 3、不需要（重启线上服务器）： 配置更新后， 实时推送新配置信息至项目中， 实时生效， 不需要重启线上机器；（在项目集群部署时， 将会节省大量的时间， 避免了集群机器一个一个的重启， 费时费力）
- 4、配置生效 "非常及时" ： 点击更新按钮， 新的配置信息将会即可推送到项目中， 瞬间生效， 非常及时。比如一些开关类型的配置， 配置变更后， 将会立刻推送至项目中并生效， 相对常规配置修改繁琐的流程， 及时性可谓天壤之别；
- 5、不同环境 "同一个上线包" ： 因为差异化的配置托管在配置中心， 因此一个上线包可以复用在生产、测试等各个运行环境， 提供能效；

1.5 下载

文档地址

- 中文文档

源码仓库地址

源码仓库地址	Release Download
https://github.com/xuxueli/xxl-conf	Download
http://gitee.com/xuxueli0323/xxl-conf	Download

中央仓库地址

```
<dependency>
  <groupId>com.xuxueli</groupId>
  <artifactId>xxl-conf-core</artifactId>
  <version>{最新稳定版}</version>
</dependency>
```

技术交流

- 社区交流

1.6 环境

- Maven3+
- Jdk1.7+
- Mysql5.6+

来源(书栈小编注)

> *http://www.xuxueli.com/xxl-conf/#/*

二、快速入门

2.1 环境准备

初始化“数据库”

请下载项目源码并解压，获取 "数据库初始化SQL脚本（Mysql）" 并执行即可。脚本位置如下：

```
1. xxl-conf/doc/db/xxl-conf.sql
```

2.2 编译源码

解压源码,按照maven格式将源码导入IDE， 使用maven进行编译即可，源码结构如下图所示：

```
1. - xxl-conf-admin：配置中心
2. - xxl-conf-core：公共依赖
3. - xxl-conf-samples: 接入XXl-CONF的示例项目，供用户参考学习
4.     - xxl-conf-sample-frameless：    无框架版本，main方法直接启动运行
5.     - xxl-conf-sample-spring：       spring版本
6.     - xxl-conf-sample-springboot：   springboot版本
7.     - xxl-conf-sample-jfinal：       jfinal版本
8.     - xxl-conf-sample-nutz：         nutz版本
```

2.3 “配置中心” 搭建（支持集群）

```
1. 项目：xxl-conf-admin
2. 作用：提供一个完善强大的配置管理平台，包含：环境管理、用户管理、项目管理、配置管理等功能，全部操作通过Web界面在线完成；
```

方式1：源码编译方式搭建：

- 配置文件位置：

```
1. /xxl-conf/xxl-conf-admin/src/main/resources/application.properties
```

- 配置项说明：

```
1. #  配置中心数据库配置，存储配置元数据
2. spring.datasource.url=jdbc:mysql://127.0.0.1:3306/xxl-conf?Unicode=true&characterEncoding=UTF-8
3.
4. #  配置中心配置数据磁盘路径地址，务必对该路径存在读写权限
5. xxl.conf.confdata.filepath=/data/applogs/xxl-conf/confdata
6.
7. #  配置中心接入验证TOKEN，选填，非空时启用，进行安全严重
8. xxl.conf.access.token=
```

- 配置中心启动：
  项目编译打包后，可直接通过命令行启动；

```
// 方式1：使用默认配置，mysql默认为本地地址；
java -jar xxl-conf-admin.jar

// 方式2：支持自定义 mysql 地址；
java -jar xxl-conf-admin.jar --spring.datasource.url=jdbc:mysql://127.0.0.1:3306/xxl-conf?Unicode=true&characterEncoding=UTF-8
```

方式2：Docker 镜像方式搭建：

- 下载镜像

```
// Docker地址：https://hub.docker.com/r/xuxueli/xxl-conf-admin/
docker pull xuxueli/xxl-conf-admin
```

- 创建容器并运行

```
docker run -p 8080:8080 -v /tmp:/data/applogs --name xxl-conf-admin  -d xuxueli/xxl-conf-admin

/**
* 如需自定义 mysql 等配置，可通过 "PARAMS" 指定；
* 配置项参考文件：/xxl-conf/xxl-conf-admin/src/main/resources/application.properties
*/
docker run -e PARAMS="--spring.datasource.url=jdbc:mysql://127.0.0.1:3306/xxl-conf?Unicode=true&characterEncoding=UTF-8 " -p 8080:8080 -v /tmp:/data/applogs --name xxl-conf-admin  -d xuxueli/xxl-conf-admin
```

"配置中心" 集群：

配置中心支持集群部署，提高配置中心负载能力和可用性。配置中心集群部署时，项目配置文件保持一致即可。

2.4 “接入XXL-CONF的示例项目” 项目配置

```
项目：xxl-conf-sample-springboot
作用：接入XXl-CONF的示例项目，供用户参考学习。这里以 springboot 版本进行介绍，其他版本可参考各自sample项目。
```

A、引入maven依赖

```
<!-- xxl-conf-client -->
<dependency>
    <groupId>com.xuxueli</groupId>
    <artifactId>xxl-conf-core</artifactId>
    <version>{最新稳定版}</version>
</dependency>
```

B、添加“XXL-CONF 配置信息”

可参考配置文件：

```
/xxl-conf/xxl-conf-samples/xxl-conf-sample-springboot/src/main/resources/application.properties
```

配置项说明

```
#  配置中心跟地址，必填；
xxl.conf.admin.address=http://localhost:8080/xxl-conf-admin

#  环境配置，必填；如"test、ppe、product"等，指定配置加载环境；
xxl.conf.env=test

#  配置中心接入验证TOKEN，选填，非空时启用，进行安全严重
xxl.conf.access.token=

#  配置快照文件地址，必填；会周期性缓存到本地快照文件中，当从配置中心获取配置失败时，将会使用使用本地快照文件中的配置数据；提高系统可用性；
xxl.conf.mirrorfile=/data/applogs/xxl-conf/xxl-conf-mirror-sample.properties
```

C、设置“XXL-CONF 配置工厂”

可参考配置文件：

```
/xxl-conf/xxl-conf-samples/xxl-conf-sample-springboot/src/main/java/com/xxl/conf/sample/config/XxlConfConfig.java
```

配置项说明

```
@Bean
public XxlConfFactory xxlConfFactory() {

    XxlConfFactory xxlConf = new XxlConfFactory();
    xxlConf.setAdminAddress(adminAddress);
    xxlConf.setEnv(env);
    xxlConf.setAccessToken(accessToken);
    xxlConf.setMirrorfile(mirrorfile);

    logger.info(">>>>>>>>>>> xxl-conf config init.");
    return xxlConf;
}
```

至此，配置完成。

2.5 功能测试

a、添加和更新配置

参考章节 "4.2 配置管理" 添加或更新配置信息;

b、获取配置并接受动态推送更新

参考章节 "三、客户端配置获取" 获取配置并接受动态推送更新;

来源(书栈小编注)

> *http://www.xuxueli.com/xxl-conf/#/*

三、客户端配置获取

XXL-CONF 提供多种配置方式，包括 "API、 @XxlConf、XML" 等多种配置方式，介绍如下。

> 可参考项目 *"xxl-conf-sample-spring"*（接入*XXL-CONF*的示例项目，供用户参考学习），代码位置：
> *com.xxl.conf.sample.controller.IndexController.index()*

3.1 方式1：API方式

参考 "IndexController" 代码如下：

```
1. String paramByApi = XxlConfClient.get("default.key01", null);
```

- 用法：代码中直接调用API即可，示例代码 ""XxlConfClient.get("key", null)"";
- 优点：
    - 配置从配置中心自动加载；
    - 存在LocalCache，不用担心性能问题；
    - 支持动态推送更新；
    - 支持多数据类型；

3.2 方式2：@XxlConf 注解方式

参考 "DemoConf.paramByAnno" 属性配置；示例代码

```
1. @XxlConf("default.key02")
2. public String paramByAnno;
```

- 用法：对象Field上加注解 ""@XxlConf("key")"，支持设置默认值，支持设置是否开启动态刷新；
- 优点：
    - 配置从配置中心自动加载；
    - 存在LocalCache，不用担心性能问题；
    - 支持动态推送更新；
    - 支持设置配置默认值；
    - 可配置是否开启 "动态推送更新"；
      |“@XxlConf”注解属性|说明
      |—-
      |value|配置Key
      |defaultValue|配置为空时的默认值
      |callback|配置更新时，是否需要同步刷新配置

3.3 方式3：XML占位符方式

参考 "applicationcontext-xxl-conf.xml" 中 "DemoConf.paramByXml" 属性配置；示例代码如下：

```
1. <bean id="demoConf" class="com.xxl.conf.sample.demo.DemoConf">
```

```
    <property name="paramByXml" value="$XxlConf{default.key03}" />
</bean>
```

- 用法：占位符方式 "$XxlConf{key}";
- 优点：
    - 配置从配置中心自动加载；
    - 存在LocalCache，不用担心性能问题；
    - 支持动态推送更新；

3.4 方式4："XML + API" 混合方式

参考如下代码：

```
<bean id="demoConf" class="com.xxl.conf.sample.demo.DemoConf2">
    <constructor-arg index="0" value="# {T(com.xxl.conf.core.XxlConfClient).get('key')}" />
    <property name="paramByXml" value="# {T(com.xxl.conf.core.XxlConfClient).get('default.key03')}" />
</bean>
```

- 用法：占位符方式 "# {T(com.xxl.conf.core.XxlConfClient).get('key')}";
- 优点：
    - 配置从配置中心自动加载；
    - 存在LocalCache，不用担心性能问题；
    - 兼容性好：在一些特殊的XML配置加载场景，如 "XML构造器传参"、"自定义spring的schema/xsd"，上述几种方式不适用，此时可以考虑这种方式，兼容各种场景格式；
- 缺点：
    - 不支持动态推送更新；

3.5 其他方式：配置变更监听

可开发Listener逻辑，监听配置变更事件；可据此实现动态刷新JDBC连接池等高级功能；

参考 "IndexController" 代码如下：

```
XxlConfClient.addListener("default.key01", new XxlConfListener(){
    @Override
    public void onChange(String key, String value) throws Exception {
        logger.info("配置变更事件通知：{}={}", key, value);
    }
});
```

来源(书栈小编注)

> *http://www.xuxueli.com/xxl-conf/#/*

四、管理中心操作指南

4.1、环境管理

进入 "环境管理" 界面，可自定义和管理环境信息。单个配置中心集群，支持自定义多套环境，管理多个环境的的配置数据；环境之间相互隔离；

新增环境：点击 "新增环境" 按钮可添加新的环境配置，环境属性说明如下：

```
- Env：每个环境拥有一个维护的Env，作为环境标识；
- 环境名称：该环境的名称；
```

环境切换：配置中心顶部菜单展示当前操作的配置中心环境，可通过该菜单切换不同配置中心环境，从而管理不同环境中的配置数据；

4.2、用户（权限）管理

进入 "用户管理" 界面，可查看配置中心中所有用户信息。

新增用户：点击 "新增用户" 按钮，可添加新用户，用户属性说明如下：

1. - 权限：
2. - 管理员：拥有配置中心所有权限，包括：用户管理、环境管理、项目管理、配置管理等；
3. - 普通用户：仅允许操作自己拥有权限的项目下的配置；
4. - 用户名：配置中心登陆账号
5. - 密码：配置中心登陆密码

系统默认提供了一个管理员用户和一个普通用户。

分配项目权限：选中普通用户，点击右侧 "分配项目权限" 按钮，可为用户分配项目权限，权限细粒度到 "环境 + 项目"。 拥有环境项目权限后，该用户可以查看和操作该环境项目下全部配置数据。

修改用户密码：配置中心右上角下拉框，点击 "修改密码" 按钮，可修改当前登录用户的登录密码（除此之外，管理员用户，可通过编辑用户信息功能来修改其他用户的登录密码）；

4.3、项目管理

系统以 "项目" 为维度进行权限控制，以及配置隔离。可进入 "配置管理界面" 操作和维护项目，项目属性说明如下：

1. - AppName：每个项目拥有唯一的AppName，作为项目标识，同时作为该项目下配置的统一前缀；
2. - 项目名称：该项目的名称；

系统默认提供了一个示例项目。

4.4 配置管理

进入"配置管理" 界面， 选择项目，然后可查看和操作该项目下配置数据。

新增配置：点击 "新增配置" 按钮可添加配置数据，配置属性说明如下：

1. - KEY：配置的KEY，创建时将会自动添加所属项目的APPName所谓前缀，生成最终的Key。可通过客户端使用最终的Key获取配置；
2. - 描述：该配置的描述信息；
3. - VALUE：配置的值；

至此， 一条配置信息已经添加完成；通过客户端可以获取该配置， 并且支持动态推送更新。

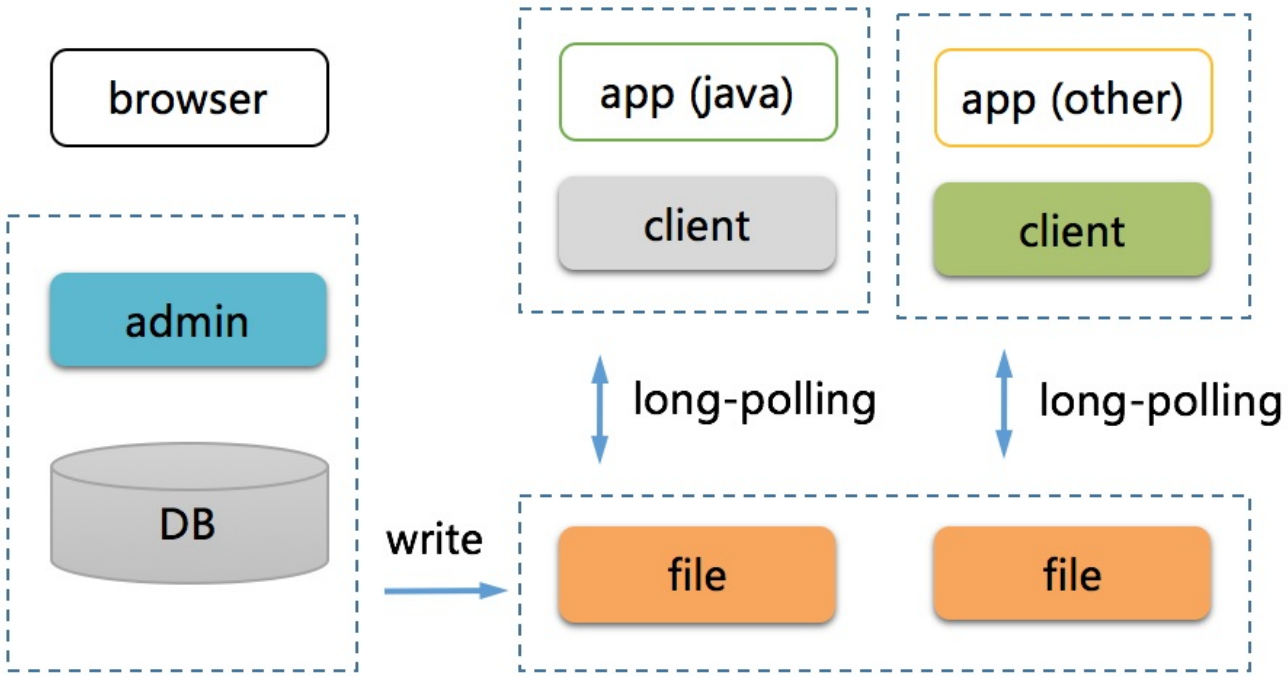
历史版本回滚：配置存在历史变更操作时，点击右侧的 "变更历史" 按钮，可查看该配置的历史变更记录。包括操作时间、操作人，设置的配置值等历史数据，因此可以根据历史数据，重新编辑配置并回滚到历史版本；

来源(书栈小编注)

> http://www.xuxueli.com/xxl-conf/#/

五、总体设计

5.1 架构图



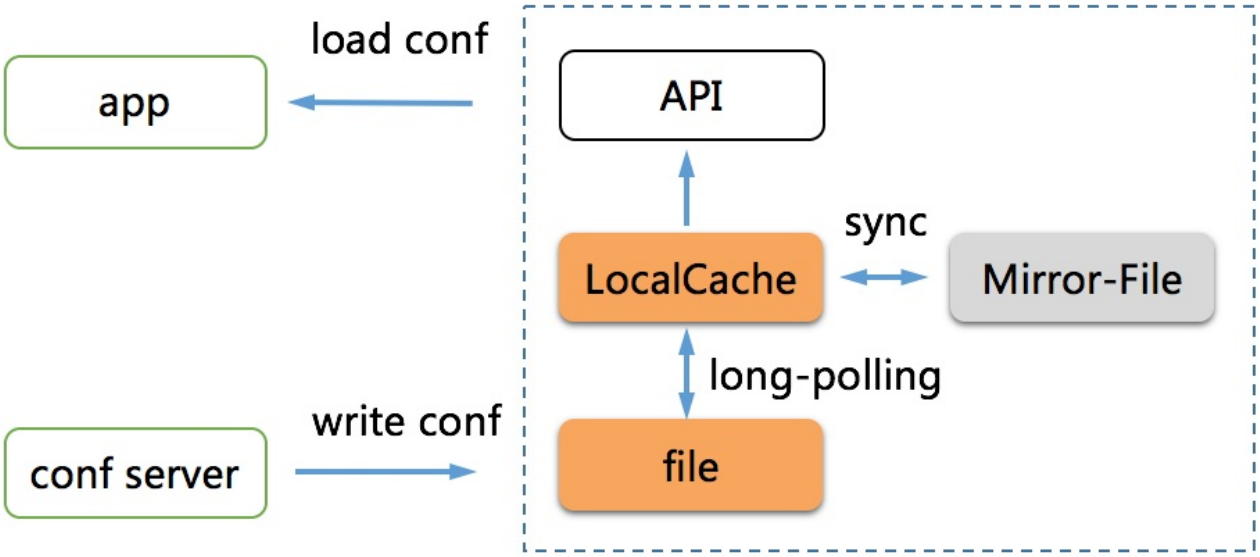
XXL-CONF架构图 v1.6

5.2 "配置中心" 设计

配置中心由以下几个核心部分组成：

- 1、管理平台：提供一个完善强大的配置管理平台，包含：环境管理、用户管理、项目管理、配置管理等功能，全部操作通过Web界面在线完成；
- 2、管理平台DB：存储配置信息备份、配置的版本变更信息等，进一步保证数据的安全性；同时也存储"管理平台"中多个模块的底层数据；
- 3、磁盘配置数据：配置中心在每个配置中心集群节点磁盘中维护一份镜像数据，当配置新增、更新等操作时，将会广播通知并实时刷新每个集群节点磁盘中的配置数据， 最终实时通知接入方客户端；
- 4、客户端：可参考章节 "5.3 客户端 设计" ；

5.3 "客户端" 设计



XXL-CONF 客户端设计 v1.6

客户端基于多层设计，核心四层设计如下：

- 1、API层：提供业务方可直接使用的上层API，简单易用，一行代码获取配置信息；同时保证配置的实时性、高性能；
- 2、LocalCache层：客户端的Local Cache，极大提升API层的性能，降低对配置中心集群的压力；首次加载配置、监听配置变更、底层异步周期性同步配置时，将会写入或更新缓存；
- 4、Mirror-File层：配置数据的本地快照文件，会周期性同步 "LocalCache层" 中的配置数据写入到 "Mirror-File" 中；当无法从配置中心获取配置，如配置中心宕机时，将会使用 "Mirror-File" 中的配置数据，提高系统的可用性；
- 3、Remote层：配置中心远程客户端的封装，用于加载远程配置、实时监听配置变更，提高配置时效性；
  得益于客户端的多层设计，以及 LocalCache 和 Mirror-File 等特性，因此业务方可以在高QPS、高并发场景下使用XXL-CONF的客户端，不必担心并发压力或配置中心宕机导致系统问题。

5.4 配置中心 http 服务（多语言支持）

Java语言应用，可以直接通过依赖提供的Client包的方式，方便快速的接入和使用配置中心；可参考章节 "二、快速入门"：

非Java语言，可借助 XXL-CONF 提供的 "配置中心http服务"，获取配置、实时感知配置更新，从而实现多语言支持。

配置中心提供的 "配置中心http服务" 只会读磁盘配置数据，因此性能极高，而且配置中心支持通过集群无线横向扩展；

"配置中心http服务" 接口文档如下：

a、配置批量获取接口：

```
说明：用于批量查询配置数据；

// 接口地址格式
{配置中心跟地址}/conf/find?env={环境}&keys={配置Key}&keys={配置Key02}

// 示例
http://localhost:8080/xxl-conf-admin/conf/find?env=test&keys=default.key01&keys=default.key02

// 请求参数：get/post方式均可
accessToken ：   配置中心接入验证TOKEN，选填，非空时启用，进行安全严重
env         ：    环境配置，必填；如"test、ppe、product"等，指定配置加载环境；
keys        ：    配置Key，支持传递多个，

// 响应数据格式：
{
  "code": 200,      // 200 表示正常、其他失败
  "msg": null,      // 错误提示消息
  "data": {         // 配置信息，KV格式
    "default.key02": "22",
    "default.key01": "111"
  }
}
```

b、配置实时监控接口：

```
说明：用于实时监控配置数据更新，为 long-polling 接口，请求后将会立即阻塞，期间如若参数中配置Key有变动则立即响应通知请求方，否则将会一直阻塞，默认阻塞30s；

// 接口地址格式
{配置中心跟地址}/conf/monitor?env={环境}&keys={配置Key}&keys={配置Key02}

// 示例
http://localhost:8080/xxl-conf-admin/conf/monitor?env=test&keys=default.key01&keys=default.key02

// 请求参数：get/post方式均可
accessToken ：   配置中心接入验证TOKEN，选填，非空时启用，进行安全严重
env         ：    环境配置，必填；如"test、ppe、product"等，指定配置加载环境；
keys        ：    配置Key，支持传递多个，

// 响应数据格式：
{
  "code": 501,                    // 200 表示正常，一直阻塞到结束，说明配置数据没变动；501 表示配置数据有变化；其他标示请求失败
  "msg": "Monitor key update."    // 错误提示消息
}
```

接入方可以借助上面两个接口，获取配置、实时感知配置更新；

5.5 配置快照功能

客户端从配置中心获取到的配置数据后，会周期性缓存到本地快照文件中，当从配置中心获取配置失败时，将会使用使用本地快照文件中的配置数据；提高系统可用性；

5.6 多环境支持

单个配置中心集群，支持自定义多套环境，管理多个环境的的配置数据；环境之间相互隔离；

此处给出一些多环境配置的建议：

- 机器资源紧缺、系统规模较小时：建议部署单个配置中心集群，比如部署 "配置中心集群"，通过定义多套环境，如 "dev、test、ppe、product" 隔离不同环境配置数据；优点是，可以同享配置中心资源；
- 机器资源充足、系统规模较大时：建议部署多个配置中心集群，比如部署 "配置中心集群A"，定义环境 "ppe、product"；部署 "配置中心集群B"，定义环境 "dev、test"等；优点是，可以避免多个集群相互影响；

5.7 对象代理情况下配置获取

在配置所属对象存在代理(JDK、CGLib)的特殊情况下，推荐使用以下方式获取配置：（非代理情况下，可以忽略本章节）

- 1、采用“API方式”获取配置：最稳定的配置获取方式，API方式底层存在Local Cache不必担心性能问题；
- 2、为配置属性添加 get、set 方法，不要直接访问配置属性，而是通过配置属性相应的 get 方法获取；

5.8 容灾性

XXL-CONF拥有极高的容灾性，首先配置数据进行多级存储， 可分为以下几层：

- DB：完整的配置数据存储在数据库中，极大的方便配置数据的备份与迁移；
- 配置中心磁盘：配置中心在每个配置中心集群节点磁盘中维护一份镜像数据，并实时同步更新；
- Client-镜像文件：接入配置中心的客户端应用会自动对使用的配置生成镜像文件，远程配置中心故障时降级实用镜像文件；
- Client-LocalCache：接入配置中心的客户端引用，优先使用LocalCache内存中的配置数据，提高性能的同时，降低对底层配置服务的压力；

- Client-Api：最后暴露给业务的API，用户可具体加载配置数据，完成业务；
  鉴于以上基础，在配置服务故障时，可以快速进行配置服务降级与恢复：

- 配置中心宕机时：对业务系统无影响，业务系统从配置中心磁盘与Client端镜像文件中获取配置数据；

- DB宕机：对业务系统无影响，业务系统从配置中心磁盘与Client端镜像文件中获取配置数据；
- 配置中心宕机 + DB宕机 + Client端镜像文件被删除：此时，只需要手动创建一份配置镜像文件，上传到Client端应用指定位置即可，业务无影响；

来源(书栈小编注)

> *http://www.xuxueli.com/xxl-conf/#/*

六、历史版本

6.1 版本 v1.0.0 特性[2015-11-13]

- 初始版本导入;

6.2 版本 v1.1.0 特性[2016-08-17]

- 1、简单易用：上手非常简单，只需要引入maven依赖和一行配置即可;
- 2、在线管理：提供配置中心，支持在线管理配置信息;
- 3、实时推送：配置信息更新后，Zookeeper实时推送配置信息，项目中配置数据会实时更新并生效，不需要重启线上机器;
- 4、高性能：系统会对Zookeeper推送的配置信息，在Encache中做本地缓存，在接受推送更新或者缓存失效时会及时更新缓存数据，因此业务中对配置数据的查询并不存在性能问题;
- 5、配置备份：配置数据首先会保存在Zookeeper中，同时，在MySQL中会对配置信息做备份，保证配置数据的安全性;
- 6、HA：配置中心基于Zookeeper集群，只要集群节点保证存活数量大于N/2+1，就可保证服务稳定，避免单点风险;
- 7、分布式：可方便的接入线上分布式部署的各个业务线，统一管理配置信息;
- 8、配置共享：平台中的配置信息针对各个业务线是平等的，各个业务线可以共享配置中心的配置信息，当然也可以配置业务内专属配置信息;

6.3 版本 v1.2.0 新特性[2016-10-08]

- 1、配置分组：支持对配置进行分组管理，每条配置将会生成全局唯一标示GroupKey,在client端使用时,需要通过该值匹配对应的配置信息;

6.4 版本 v1.3.0 新特性[2016-10-08]

- 1、支持在线维护配置分组;
- 2、项目groupId从com.xxl迁移至com.xuxueli，为推送maven中央仓库做准备;
- 3、v1.3.0版本开始，推送公共依赖至中央仓库;

6.5 版本 v1.3.1-beta 新特性[2017-08-10]

- 1、本地配置优先加载逻辑调整;
- 2、zookeeper地址方式从磁盘迁移至项目内;

6.6 版本 v1.3.1-beta2 新特性[2017-08-19]

- 1、配置文件统一问题fix;

6.7 版本 v1.4.0 新特性[2018-03-02]

- 1、支持通过 "@XxlConf" 注解获取配置；
- 2、动态推送更新：目前支持 "XML、 @XxlConf、API" 三种配置方式，均支持配置动态刷新；
- 3、配置变更监听功能：可开发Listener逻辑，监听配置变更事件，可据此动态刷新JDBC连接池等高级功能；
- 4、用户管理：支持在线添加和维护用户，包括普通用户和管理员两种类型用户；
- 5、配置权限控制；以项目为维度进行配置权限控制，管理员拥有全部项目权限，普通用户只有分配才拥有项目下配置的查看和管理权限；
- 6、配置变更版本记录：记录配置变更历史，方便历史配置版本回溯，默认记录10个历史版本；
- 7、客户端断线重连强化，除了依赖ZK之外，新增守护线程，周期性刷新Local Cache中配置数据并watch，进一步提高配置时效性；
- 8、ZK过期重连时，主动刷新LocalCache中配置数据，提高异常情况下配置时效性；
- 9、ZK重入锁做二次校验，防止并发冲突；
- 10、主动缓存null或不存在类型配置，避免配置请求穿透到ZK引发雪崩问题；
- 11、Local Cache缓存长度固定为1000，采用LRU策略移除。
- 12、表结构优化；
- 13、重构核心代码，规范代码结构；
- 14、环境配置文件，支持自定义存放位置，项目resource下或磁盘目录下均可；
- 15、支持设置ZK中配置存储路径，方便实现多环境复用ZK集群；
- 16、用户在线修改密码；
- 17、升级依赖版本，如Ehcache、Spring等；
- 18、弹框插件改为使用Layui；
- 19、AdminLTE版本升级；
- 20、Sample项目目录结构规范；
- 21、新增SpringBoot类型Sample项目；

6.8 版本 v1.4.1 新特性[2018-04-12]

- 1、Ehcache缓存对象CacheNode序列化优化；
- 2、XML配置方式，Bean初始化时配置加载逻辑优化；
- 3、升级多项依赖至较新版本：spring、spring-boot、jackson、freemarker、mybatis等；

6.9 版本 v1.4.2 新特性[2018-05-30]

- 1、多环境支持：单个配置中心集群，支持自定义多套环境，管理多个环境的的配置数据；环境之间相互隔离；
- 2、多数据类型配置：支持多种数据类型配置，如：String、Boolean、Short、Integer、Long、Float、Double 等；
- 3、多语言支持：提供配置Agent服务，可据此通过Http获取配置数据，从而实现多语言支持。Agent存在Ehcache缓存性能极高，并且支持集群横向扩展；
- 4、新增 "Jfinal" 类型Sample项目；
- 5、新增 "Nutz" 类型Sample项目；
- 6、支持ZK鉴权信息配置；
- 7、Local Cache缓存长度扩充为100000，采用LRU过期策略。
- 8、配置数据强制编码 UTF-8，解决因操作系统编码格式不一致导致的配置乱码问题；
- 9、XxlConf与原生配置加载方式("@Value"、"${…}")兼容，相互隔离，互不影响；替代原LocalConf层；
- 10、移除Spring强制依赖。在保持对Spring良好支持情况下，提高对非Spring环境的兼容性；
- 11、容器组件初始化顺序调整，修复@PostConstruct无法识别问题；

- 12、配置优化，移除冗余配置项；
- 13、小概率情况下BeanRefresh重复刷新问题修复；
- 14、升级pom依赖至较新版本，如Spring、Zookeeper等；

6.10 版本 v1.5.0 新特性[2018-06-15]

- 1、配置中心Agent服务增强：针对非Java应用提供Agent服务获取配置，提供同步、异步两种Http请求方式，原生支持 long-polling（Http） 的方式获取配置数据、并实时感知配置变更。同时，强化请求权限校验；
- 2、配置同步功能：将会检测对应项目下的全部未同步配置项，使用DB中配置数据覆盖ZK中配置数据并推送更新；在配置中心异常恢复、新配置中心集群初始化等场景中十分有效；
- 3、配置快照：客户端从配置中心获取到的配置数据后，会周期性缓存到本地快照文件中，当从配置中心获取配置失败时，将会使用使用本地快照文件中的配置数据；提高系统可用性；
- 4、配置中心，迁移为spring boot项目；
- 5、配置中心，提供官方docker镜像；
- 6、Cglib代理情况下，如 "@Configuration" 注解，Bean无法注入配置问题修复；
- 7、springboot项目加载prop失败的问题修复；
- 8、升级多项maven依赖至较新版本，如spring等；

6.11 版本 v1.5.1 新特性[2018-10-24]

- 1、ftl变量判空问题修复；
- 2、配置快照文件生成时自动创建多层父目录；
- 3、移除ehcache依赖，取消local cache容量限制；
- 4、ZK初始化逻辑优化，避免并发初始化，阻塞至TCP连接创建成功才允许后续操作；
- 5、升级多项maven依赖至较新版本，如spring等；

6.12 版本 v1.5.2 Release Notes[2018-11-13]

- 1、ZK节点watch逻辑优化，配置中心取消冗余的watch操作；
- 2、ZK初始化时unlock逻辑调整，优化断线重连特性；
- 3、Client端ZK初始化逻辑调整，取消对ZK状态的强依赖，连接失败也允许启动，此时使用镜像配置文件；
- 4、修复配置监听首次无效的问题，监听前先get一次该配置；
- 5、新增无框架接入配置中心Sample示例项目 "xxl-conf-sample-frameless"。不依赖第三方框架，快速接入配置中心，只需main方法即可启动运行；
- 6、权限控制增强，细粒度到环境权限校验；

6.13 版本 v1.6.0 Release Notes[2018-11-29]

- 1、轻量级改造：废弃ZK，改为 "DB + 磁盘 + long polling" 方案，部署更轻量，学习更简单；集群部署更方便，与单机一致；
- 2、pom依赖清理、升级；客户端唯一依赖组件为 "slf4j-api"，彻底的零依赖。配置中心升级部分依赖；
- 3、Docker基础镜像切换，精简镜像；
- 4、高性能：得益于配置中心的 "磁盘配置" 与客户端的 "LocalCache"，因此配置服务性能非常高；单机可承担大量配置请求；
- 5、跨语言：底层通过http服务（long-polling）拉取配置数据并实时感知配置变更，从而实现多语言支

持。

- 6、访问令牌（accessToken）：为提升系统安全性，配置中心和客户端进行安全性校验，双方AccessToken匹配才允许通讯；
- 7、启动时，优先全量加载镜像数据到registry层，避免逐个请求耗时；

6.14 版本 v1.6.1 Release Notes[迭代中]

TODO LIST

- 本地优先配置：优先加载该配置中数据，常用于本地调试。早期版本功能实用性低，现已移除，考虑是否完全移除；
- 注册中心特性：原生支持注册中心功能，强一致性推送注册信息；
- 分布式锁特性：原生支持分布式锁功能；
- 支持托管配置文件，properties或yml，待考虑，不利于配置复用与细粒度管理；
- 配置中心告警功能；
- 灰度发布：将配置推送到指定环境上的指定ip或者指定模块进程；
- 配置的发布也可以考虑增加审核功能；
- XxlConfClient 更名为 XxlConf。
- 配置告警：底层DB异常时，主动推送告警信息；推送粒度为管理员还是配置影响用户，未定；
- 配置列表只展示有权限的项目列表，无有权限项目时限制不允许登陆；
- 配置日志优化，支持一件回滚与对比：
- 配置锁：项目粒度，管理员锁定后，禁止普通用户直接操作；
- 锁定Key变更，需要申请和审核；
- 灰度发布
- 配置关注：关注Key变更发送邮件通知；

来源(书栈小编注)

> *http://www.xuxueli.com/xxl-conf/#/*

七、其他

7.1 项目贡献

欢迎参与项目贡献！比如提交PR修一个bug，或者新建 Issue 讨论新特性或者变更。

7.2 用户接入登记

更多接入的公司，欢迎在 登记地址 登记，登记仅仅为了产品推广。

7.3 开源协议和版权

产品开源免费，并且将持续提供免费的社区技术支持。个人或企业内部可自由的接入和使用。

- Licensed under the GNU General Public License (GPL) v3.
- Copyright (c) 2015-present, xuxueli.

捐赠

无论捐赠金额多少都足够表达您这份心意，非常感谢 ： ） 前往捐赠

> 原文: *http://www.xuxueli.com/xxl-conf/# /*