# 统一配置中心概要设计

## 功能特点与设计理念

统一配置中心是一套完整的基于zookeeper的分布式配置统一解决方案。

**它的功能特点是**

* 支持配置（配置项+配置文件）的分布式化管理
  + 配置发布统一化
  + 配置发布、更新统一化（云端存储、发布）:配置存储在云端系统，用户统一在平台上进行发布、更新配置。
  + 配置更新自动化：用户在平台更新配置，使用该配置的系统会自动发现该情况，并应用新配置。特殊地，如果用户为此配置定义了回调函数类，则此函数类会被自动调用。
* 配置异构系统管理
  + 异构包部署统一化：这里的异构系统是指一个系统部署多个实例时，由于配置不同，从而需要多个部署包（jar或war）的情况（下同）。使用Disconf后，异构系统的部署只需要一个部署包，不同实例的配置会自动分配。特别地，在业界大量使用部署虚拟化（如JPAAS系统，SAE，BAE）的情况下，同一个系统使用同一个部署包的情景会越来越多，Disconf可以很自然地与他天然契合。 异构主备自动切换：如果一个异构系统存在主备机，主机发生挂机时，备机可以自动获取主机配置从而变成主机。
  + 异构主备机Context共享工具：异构系统下，主备机切换时可能需要共享Context。可以使用Context共享工具来共享主备的Context。
* 注解式编程，极简的使用方式：我们追求的是极简的、用户编程体验良好的编程方式。通过简单的标注+极简单的代码撰写，即可完成复杂的配置分布式化。
* 需要Spring编程环境

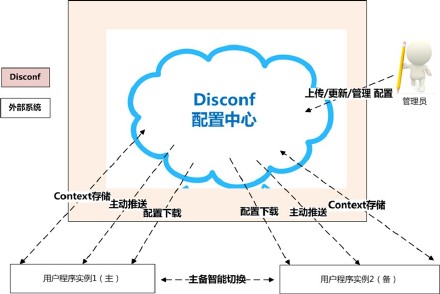
**它的设计理念是：**

* 简单，用户体验良好：
  + 摒弃了打散化配置的管理方式[2,3]，仍旧采用基于配置文件的编程方式，这和程序员以前的编程习惯（配置都是放在配置文件里）一致。特别的，为了支持较为小众的打散化配置功能，还特别支持了配置项。
  + 采用了基于XML无代码侵入编程方式：只需要几行XML配置，即可实现配置文件发布更新统一化、自动化。
  + 采用了基于注解式的弱代码侵入编程方式：通过编程规范，一个配置文件一个配置类，代码结构简单易懂。XML几乎没有任何更改，与原springXML配置一样。真正编程时，几乎感觉不到配置已经分布式化
* 可以托管任何类型的配置文件，这与[2,3]只能支持KV结构的功能有较大的改进。
* 配置更新实时推送
* 提供界面良好Web管理功能，可以非常方便的查看配置被哪些实例使用了。

## 详细设计

### 架构设计

#### disconf服务集群模式：



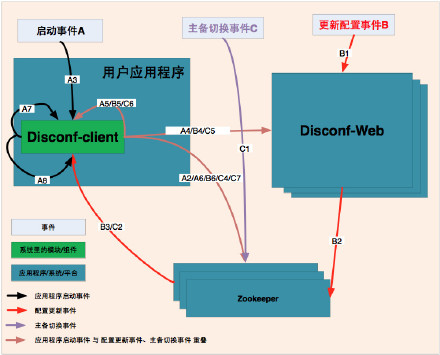
#### disconf的模块架构图：



每个模块的简单介绍如下：

* Disconf-core
  + 分布式通知模块：支持配置更新的实时化通知
  + 路径管理模块：统一管理内部配置路径URL
* Disconf-client
  + 配置仓库容器模块：统一管理用户实例中本地配置文件和配置项的内存数据存储
  + 配置reload模块：监控本地配置文件的变动，并自动reload到指定bean
  + 扫描模块：支持扫描所有disconf注解的类和域
  + 下载模块：restful风格的下载配置文件和配置项
  + watch模块：监控远程配置文件和配置项的变化
  + 主备分配模块：主备竞争结束后，统一管理主备分配与主备监控控制
  + 主备竞争模块：支持分布式环境下的主备竞争
* Disconf-web
  + 配置存储模块：管理所有配置的存储和读取
  + 配置管理模块：支持配置的上传、下载、更新
  + 通知模块：当配置更新后，实时通知使用这些配置的所有实例
  + 配置自检监控模块：自动定时校验实例本地配置与中心配置是否一致
  + 权限控制：web的简单权限控制
* Disconf-tools
  + context共享模块：提供多实例间context的共享

### 流程设计



运行流程详细介绍：

与2.0版本的主要区别是支持了：主备分配功能/主备切换事件。

启动事件A：以下按顺序发生。

A3：扫描静态注解类数据，并注入到配置仓库里。

A4+A2：根据仓库里的配置文件、配置项，去 disconf-web 平台里下载配置数据。这里会有主备竞争

A5：将下载得到的配置数据值注入到仓库里。

A6：根据仓库里的配置文件、配置项，去ZK上监控结点。

A7+A2：根据XML配置定义，到 disconf-web 平台里下载配置文件，放在仓库里，并监控ZK结点。这里会有主备竞争。

A8：A1-A6均是处理静态类数据。A7是处理动态类数据，包括：实例化配置的回调函数类；将配置的值注入到配置实体里。

更新配置事件B：以下按顺序发生。

B1：管理员在 Disconf-web 平台上更新配置。

B2：Disconf-web 平台发送配置更新消息给ZK指定的结点。

B3：ZK通知 Disconf-cient 模块。

B4：与A4一样。

B5：与A5一样。

B6：基本与A4一样，唯一的区别是，这里还会将配置的新值注入到配置实体里。

主备机切换事件C：以下按顺序发生。

C1：发生主机挂机事件。

C2：ZK通知所有被影响到的备机。

C4：与A2一样。

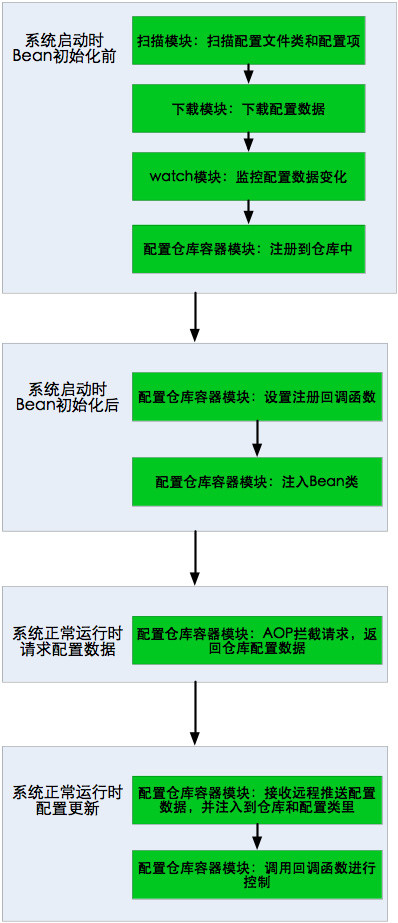
C5：与A4一样。

C6：与A5一样。

C7：与A6一样。

### 注解式disconf实现

本实现会涉及到 配置仓库容器模块、扫描模块、下载模块、watch模块，



使用AOP拦截的一个好处是可以比较轻松的实现配置控制，比如并发环境下的配置统一生效。关于这方面的讨论可以见这里。

特别地，本方式提供的编程模式非常简单，例如使用以下配置类的程序在使用它时，可以直接@Autowired进来进行调用，使用它时就和平常使用普通的JavaBean一样，但其实它已经分布式化了。配置更新时，配置类亦会自动更新。

### 基于XML配置disconf实现

本实现提供了无任何代码侵入方式的分布式配置。

ReloadablePropertiesFactoryBean继承了Spring Properties文件的PropertiesFactoryBean类，管理所有当配置更新时要进行reload的配置文件。对于被管理的每一个配置文件，都会通过 配置仓库容器模块、扫描模块、下载模块、watch模块 进行配置获取至配置仓库里。

ReloadingPropertyPlaceholderConfigurer继承了Spring Bean配置值控制类PropertyPlaceholderConfigurer。在第一次扫描spring bean 时，disconf会记录配置文件的配置与哪些bean有关联。

ReloadConfigurationMonitor是一个定时任务，定时check本地配置文件是否有更新。

当配置中心的配置被更新时，配置文件会被下载至实例本地，ReloadConfigurationMonitor即会监控到此行为，并且通知 ReloadingPropertyPlaceholderConfigurer 对相关的bean类进行值更新。

特别的，此种方式无法解决并发情况下配置统一生效的问题。

### 主备分配实现

在实现中，为每个配置提供主备选择的概念。用户实例在获取配置前需要先进行全局唯一性竞争才能得到配置值。在这里，我们采用基于zookeeper的全局唯一性锁来实现。