 **Nacos**

# Nacos Config

**META-INF/spring.factories**文件

**org.springframework.cloud.bootstrap.BootstrapConfiguration**=**\  
org.springframework.cloud.alibaba.nacos.NacosConfigBootstrapConfiguration  
org.springframework.boot.autoconfigure.EnableAutoConfiguration**=**\  
org.springframework.cloud.alibaba.nacos.NacosConfigAutoConfiguration,\  
org.springframework.cloud.alibaba.nacos.endpoint.NacosConfigEndpointAutoConfiguration  
org.springframework.boot.diagnostics.FailureAnalyzer**=**\  
org.springframework.cloud.alibaba.nacos.diagnostics.analyzer.NacosConnectionFailureAnalyzer**

## BootstrapConfiguration

### NacosConfigBootstrapConfiguration

Springcloud配置，最大配置

配置以下Bean

#### NacosConfigProperties

初始化参数（**spring.cloud.nacos.config**），注入到该Bean中

configServiceInstance：获取配置服务。

静态类获取配置服务：NacosFactory.createConfigService

静态类获取配置服务：ConfigFactory.createConfigService，根据String serverAddr或者Properties获取配置服务,自动配置使用Properties参数函数。

1. serverAddr
2. properties

实例化**com.alibaba.nacos.client.config.NacosConfigService**服务。

构造函数中实例化：

1. ServerHttpAgent

初始化ServerListManager，当配置了serverAddr的时候，根据配置获取nacos地址。当使用spring.cloud.nacos.config.cluster配置的时候，将去远方获取配置信息。

多个serverAddr使用逗号分隔。

1. ClientWorker

构造函数中启动三个线程：名为（**com.alibaba.nacos.client.Worker.fix—ip, com.alibaba.nacos.client.Worker.longPulling**）其中ip为当前主机ip，守护进程。检查线程：

启动Http获取nacos服务地址，当系统启动后或者当前系统系统配置了serverAddr，跳过。

#### NacosPropertySourceLocator

实现PropertySourceLocator接口，spring在使用该接口实现获取spring配置文件。

Locate函数获取PropertySource

实例化NacosPropertySourceBuilder

实例化CompositePropertySource

Locate🡪loadApplicationConfiguration

🡪loadNacosDataIfPresent（添加默认配置文件，不使用profile文件的）

🡪nacosPropertySourceBuilder.build

🡪loadNacosData（获取配置文件函数）

🡪configService.getConfig（参数，配置文件名称，群组，超时）

🡪LocalConfigInfoProcessor.getFailover（当本地存在配置文件时，优先使用本地配置文件）

🡪worker.getServerConfig（从远端nacos服务器，获取配置文件）

🡪根据profile获取配置文件

回到springcloud配置文件初始化器中。

## EnableAutoConfiguration

### NacosConfigAutoConfiguration

Springboot自动配置，配置以下Bean

#### NacosPropertySourceRepository

🡪getAll（获取所有NacosPropertySource配置）

#### NacosRefreshProperties

存储配置（**spring.cloud.nacos.config.refresh.enabled:true**）是否允许动态刷新系统

#### NacosRefreshHistory

系统刷新历史Bean

#### NacosContextRefresher

系统事件监听器（监听系统启动事件ApplicationListener<ApplicationReadyEvent>）,获取所有配置文件，根据配置文件ID（dataId），添加监听器。

🡪registerNacosListener（注册配置文件监听器）

🡪configService.addListener（添加监听器）

🡪worker.addTenantListeners（添加监听器）

🡪agent.getTenant（ServerHttpAgent）这个是复杂的东西，从服务端获取nacos地址时用的，暂时用不上。

🡪addCacheDataIfAbsent添加当前配置文件到cache中。这个cache是系统内存级，

使用volatile保证单线程写，使用AtomicReference（不保证读取的准确性，只保证操作的准确性，和AtomicInteger一样，不能当synchronized使用），读取的数值可能是过期的，写入时会再次校验还是获取忘了。

🡪cache.addListener(listener) 添加监听器到cache中，以便于修改后，会刷新。

### NacosConfigEndpointAutoConfiguration

监听

#### NacosConfigEndpoint

配置管理端点，配合actuator

#### NacosConfigHealthIndicator

系统健康检查，主要是检查配置文件的有效性，主要是远端检查，本地的情况下，只要不删，永远都是健康的。

## FailureAnalyzer

### NacosConnectionFailureAnalyzer

FailureAnalyzer，后续补充

# Nacos Discovery

**org.springframework.boot.autoconfigure.EnableAutoConfiguration**=**\  
 org.springframework.cloud.alibaba.nacos.NacosDiscoveryAutoConfiguration,\  
 org.springframework.cloud.alibaba.nacos.ribbon.RibbonNacosAutoConfiguration,\  
 org.springframework.cloud.alibaba.nacos.endpoint.NacosDiscoveryEndpointAutoConfiguration  
org.springframework.cloud.client.discovery.EnableDiscoveryClient**=**\  
org.springframework.cloud.alibaba.nacos.NacosDiscoveryClientAutoConfiguration**

## EnableAutoConfiguration

### NacosDiscoveryAutoConfiguration

@ConditionalOnNacosDiscoveryEnabled

判定nacos客户端是否启用（**spring.cloud.nacos.discovery.enabled**）

🡪 @ConditionalOnProperty(value = **"spring.cloud.nacos.discovery.enabled"**, matchIfMissing = **true**)

@ConditionalOnClass(name = **"org.springframework.boot.web.servlet.context.ServletWebServerInitializedEvent"**)  
@ConditionalOnProperty(value = **"spring.cloud.service-registry.auto-registration.enabled"**, matchIfMissing = **true**)

@AutoConfigureBefore({ AutoServiceRegistrationAutoConfiguration.**class**,  
 NacosDiscoveryClientAutoConfiguration.**class** })

@ConditionalOnBean(AutoServiceRegistrationProperties.**class**)

注册信息@ConfigurationProperties(**"spring.cloud.service-registry.auto-registration"**)

#### NacosRegistration

🡪init（PostContruct，实例化之后第一时间调用）

🡪 ManagementServerPortUtils.getPort（获取管理端口，ManagementServerProperties.class--@ConfigurationProperties(prefix = **"management.server"**, ignoreUnknownFields = **true**)）

注册管理端口等

**private static final** String ***MANAGEMENT\_PORT*** = **"management.port"**;  
**private static final** String ***MANAGEMENT\_CONTEXT\_PATH*** = **"management.context-path"**;  
**private static final** String ***MANAGEMENT\_ADDRESS*** = **"management.address"**;

#### NacosAutoServiceRegistration

服务注册函数

详细的在springcloud注册中心中

@EventListener(WebServerInitializedEvent.**class**)  
**public void** bind(WebServerInitializedEvent event) {  
 ApplicationContext context = event.getApplicationContext();  
 **if** (context **instanceof** ConfigurableWebServerApplicationContext) {  
 **if** (**"management"**.equals(  
 ((ConfigurableWebServerApplicationContext) context).getServerNamespace())) {  
 **return**;  
 }  
 }  
 **this**.**port**.compareAndSet(0, event.getWebServer().getPort());  
 **this**.start();  
}

### RibbonNacosAutoConfiguration

@Configuration  
@EnableConfigurationProperties  
@ConditionalOnBean(SpringClientFactory.**class**)  
@ConditionalOnRibbonNacos  
@AutoConfigureAfter(RibbonAutoConfiguration.**class**)  
@RibbonClients(defaultConfiguration = NacosRibbonClientConfiguration.**class**)  
**public class** RibbonNacosAutoConfiguration {  
}  
自定义实现ribbon相关。

### NacosDiscoveryEndpointAutoConfiguration

**待续**

## EnableDiscoveryClient

### NacosDiscoveryClientAutoConfiguration

同上，判定是否需要配置客户端

#### NacosDiscoveryProperties

配置文件（@ConfigurationProperties(**"spring.cloud.nacos.discovery"**)）

1. **serverAddr** Nacos服务端地址
2. **endpoint** the domain name of a service, through which the server address can be dynamically obtained，应该是DNS服务的地址。
3. **namespace**不同环境，不同的ns
4. **service**服务名（@Value(**"${spring.cloud.nacos.discovery.service:${spring.application.name:}}"**) ）
5. **weight**负载分压的比重
6. **clusterName** Nacos服务端集群名称
7. **registerEnabled**是否注册当前服务到服务端，和springcloud的那个一样
8. **ip port**
9. **networkInterface**你要发布的服务（不同于springcloud，可以单独发布一部分服务，默认发布所有服务）

🡪init（PostConstruct）获取当前服务的IP，有时间看下**NetworkInterface**

#### DiscoveryClient

获取客户端信息的，将来看