**1. Features**

SC专注于为经典案例提供良好的开箱即用体验，以及提供扩展机制用于覆盖其它案例。

* 分布式/版本化配置
* 服务注册和发现
* 路由
* 服务与服务之间的调用
* 负载均衡
* 断路器
* 分布式消息传递

# Part I. Cloud Native Applications

Cloud Native 是一种开发风格，它鼓励在持续交付和价值驱动开发领域采用最佳实践。持续交付最佳实践等相关科目参见[12-factor Apps](http://12factor.net/)（<https://12factor.net/zh_cn/），12-factor-apps> 主要阐述了开发实践应该与交付运营目标相一致，比如通过声明式编程、管理、监控。SC以多种方式促进这种开发风格，起点是一组易用的分布式组件。

许多特性都已被Spring boot覆盖，SC基于Spring boot构建。SC提供了两个库（SC context / SC commons）包含了其它一些特性。SC context提供应用上下文服务（引导上下文，加密，刷新范围和环境端点）。SC commons是一组抽象类和常用类，用于对不同具体实现的抽象（SC Netflix vs SC Consul）。

如果使用SUN JDK过程中遇到“Illegal key size”，请安装“java加密扩展”。

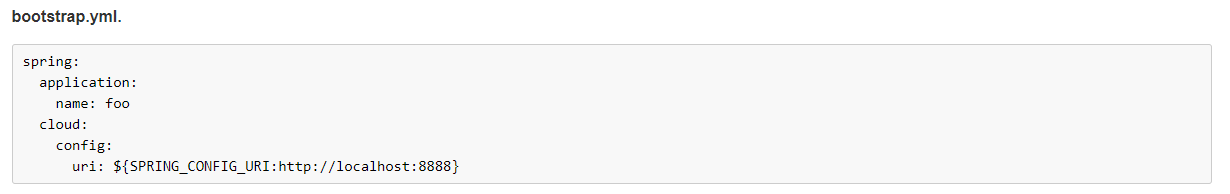
## 2. Spring Cloud Context: Application Context Services

Spring Boot对于如何使用Spring构建应用程序有一些约定看法：例如约定了常规文件位置，以及用于常见管理和监视任务的端点。Spring Cloud建立在此之上，并添加了一些系统中所有组件都要使用或偶尔需要的功能。

## 2.1 The Bootstrap Application Context

SC Application通过创建一个Bootstrap Context来运转，这个上下文是SC Application的父上下文。初始化的时候，Bootstrap Context负责从外部源中加载配置属性，并解密本地配置文件中的属性。这两个上下文共享一个Environment，这个Environment是任何Spring application的外部属性的来源。Bootstrap属性有高优先级，默认情况下，它们不会被本地配置覆盖。

Bootstrap Context与application context在定位外部配置约定不同，因此使用bootstrap.yml来作为bootstrap context的配置文件，而不是用application.yml，保持两个上下午在外部配置上的良好分离。



如果需要从SC config中获取应用特定配置的话，那么推荐设置spring.application.name (in bootstrap.yml or application.yml)。

您可以通过设置spring.cloud.bootstrap.enabled=false（例如在系统属性中）来完全禁用引导过程。

## 2.2 Application Context Hierarchies

如果你通过SpringApplication或者SpringApplicationBuilder创建一个Application Context,那么会创建Bootstrap Context作为 Application Context的父上下文。在Spring里有个特性，子上下文会继承父上下文的属性源和配置文件 ，所以main application context 相对于没有使用Spring Cloud Config的application context，会新增额外的属性源。额外的属性源有：

“bootstrap” : 如果在Bootstrap Context扫描到PropertySourceLocator（属性源定位）并且有属性，那么会添加到CompositePropertySource（混合属性源）。Spirng Cloud Config就是通过这种方式来添加的属性的，有关如何自定义此属性源的内容的[说明](https://springcloud.cc/spring-cloud-dalston.html#customizing-bootstrap-property-sources)，请参阅[下文](https://springcloud.cc/spring-cloud-dalston.html#customizing-bootstrap-property-sources)。

“applicationConfig: [classpath:bootstrap.yml]” (and friends if Spring profiles are active): 如果项目存在bootstrap.yml，里面的属性用来配置Bootstrap Context，这些属性也添加到了子上下文通过子上下文继承父上下文，它比application.yml优先级低，同时也比其它被添加到子上下文中用于创建Spring boot application的属性源优先级低。下文有介绍。

由于属性源的排序规则，“引导”条目优先，但请注意，这些条目不包含来自bootstrap.yml的任何数据，它具有非常低的优先级，但可用于设置默认值。

您可以通过简单地设置您创建的ApplicationContext的父上下文来扩展上下文层次结构，例如使用spirng自己的接口，或使用SpringApplicationBuilder方便方法（parent()，child()和sibling(兄弟姐妹））。Bootstrap Context将是您创建自己的最高级上下文的父级。层次结构中的每个上下文都将有自己的“引导”属性源（可能为空），以避免无意中将父上下文的属性带到子上下文。层次结构中的每个上下文（原则上）也可以具有不同的spring.application.name，因此如果存在配置服务器，那么这些上下文可以拥有不同的远程属性源。普通的Spring应用程序上下文行为规则适用于属性解析：子上下文的属性会覆盖父上下文的属性，如果属性名或属性源名相同（如果属性源名相同，父上下文属性源不会添加到子中）。

请注意，SpringApplicationBuilder允许您在整个层次结构中共享Environment，但这不是默认值。因此，兄弟上下文不需要具有相同profiles或属性源，尽管它们与父母共享共同点。

## 2.3 Changing the Location of Bootstrap Properties

Bootstrap.xml位置可以使用spring.cloud.bootstrap.name (default "bootstrap") 或者spring.cloud.bootstrap.location (default empty)来自定义，例如，在系统属性中。这些属性的行为与spring.config.\*类似，实际上他们用来通过在其environment中设置这些属性来构建bootstrap appliacationcontext。如果有一个激活的Profiles(spring.profiles.active 或者 通过environment api)，那么对应的bootstrap-env也会被加载，就想常规的spring boot 应用程序。

## 2.4 Overriding the Values of Remote Properties

通过bootstrap contex添加到应用程序的属性源通常是“远程”（例如从配置服务器），并且默认情况下，不能被本地属性覆盖，除了命令行属性。如果你要允许application使用自己的系统属性或配置文件覆盖远程属性，则远程属性源必须通过设置spring.cloud.config.allowOverride=true（在本地设置本身不起作用）授予权限。一旦设置了该标志，就会有一些更精细的设置来控制远程属性与系统属性、应用程序本地配置的优先级：spring.cloud.config.overrideNone=true覆盖任何本地属性源，spring.cloud.config.overrideSystemProperties=false那么只有系统属性和env var能覆盖远程设置，本地配置文件无法覆盖远程。

## 2.5 Customizing the Bootstrap Configuration

Bootstrap context可以被训练做任何你喜欢的事通过在/META-INF/spring.factories（spring cloud context包中）中org.springframework.cloud.bootstrap.BootstrapConfiguration key后面添加条目。Key对应的值为一个用逗号分隔用来创建context的Spring @Configuration类列表。你可以在此处创建要用于自动装配的主应用程序上下文的任何bean，自动装配与标注了@Beans的ApplicationContextInitializer子类有特殊约定。你可以通过@Order来控制启动顺序(the default order is "last") 。

警告：当你添加自定义BootstrapConfiguration时，请不要被错误扫描的到main application context中，main application context不需要它们。要给boot configuration类使用单独的包，这个包不能被@ComponentScan or @SpringBootApplication注解包含。

bootstrap进程通过将初始化器注入SpringApplication实例后结束(即正常的SpringBoot启动顺序,无论是作为独立程序运行还是部署在应用程序服务器中).首先用spring.factories配置的类创建一个bootstrap contet，然后创建@Beans注解的ApplicationContextInitializer子类，这些类都会在main SpringApplication启动前添加到其中。

## 2.6 Customizing the Bootstrap Property Sources

在bootstrap 进程中默认添加的额外的配置源是Config Server,但是你通过添加PropertySourceLocator的beans到bootstrap context中(通过spring.factories),添加额外的资源.你可以使用这个从不同的服务器插入额外的配置,或者从数据库   
example

*@Configuration*

**public** **class** CustomPropertySourceLocator **implements** PropertySourceLocator {

*@Override*

**public** PropertySource<?> locate(Environment environment) {

**return** **new** MapPropertySource("customProperty",

Collections.<String, Object>singletonMap("property.from.sample.custom.source", "worked as intended"));

}

}

传入的Environment是要创建的ApplicationContext的Environment，即为我们提供额外的属性来源的。它将已经具有正常的Spring Boot提供的资源来源，因此您可以使用它们来定位特定于此Environment的属性源（例如通过将其绑定在spring.application.name上，如在默认情况下所做的那样Config Server属性源定位器）。

如果你创建的jar里面有这个类，然后新增META-INF/spring.factories，包含以下配置：

org.springframework.cloud.bootstrap.BootstrapConfiguration=sample.custom.CustomPropertySourceLocator，那么“customProperty”PropertySource将显示在包含该jar的任何应用程序中。

## 2.7 Environment Changes