TAIJI 太极

Spring Cloud交流

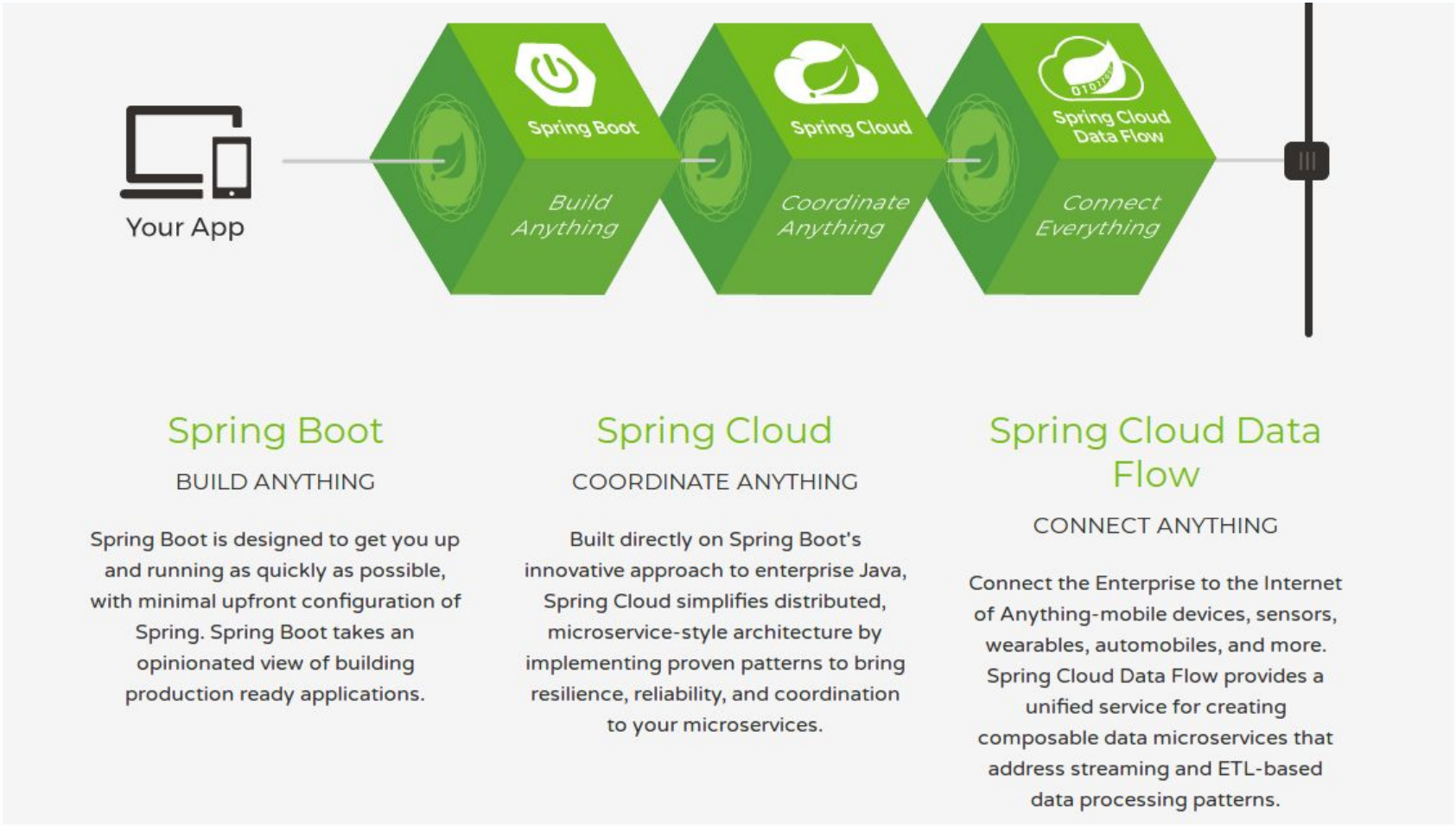
北京市海淀区北四环中路211号 100083 | T:+86-10-89056888 F:+86-10-89056889 | www.taiji.com.cn

一、Spring 生态圈

二、快速开始

三、DEMO

Spring 生态圈


Your App
Spring Boot
Build Anything
Spring Cloud
Coordinate Anything
Spring Cloud Data Flow
Connect Everything
Spring Boot
BUILD ANYTHING
Spring Boot is designed to get you up and running as quickly as possible, with minimal upfront configuration of Spring. Spring Boot takes an opinionated view of building production ready applications.
Spring Cloud
COORDINATE ANYTHING
Built directly on Spring Boot's innovative approach to enterprise Java, Spring Cloud simplifies distributed, microservice-style architecture by implementing proven patterns to bring resilience, reliability, and coordination to your microservices.
Spring Cloud Data Flow
CONNECT ANYTHING
Connect the Enterprise to the Internet of Anything-mobile devices, sensors, wearables, automobiles, and more. Spring Cloud Data Flow provides a unified service for creating composable data microservices that address streaming and ETL-based data processing patterns.

一、Spring 生态圈

二、快速开始

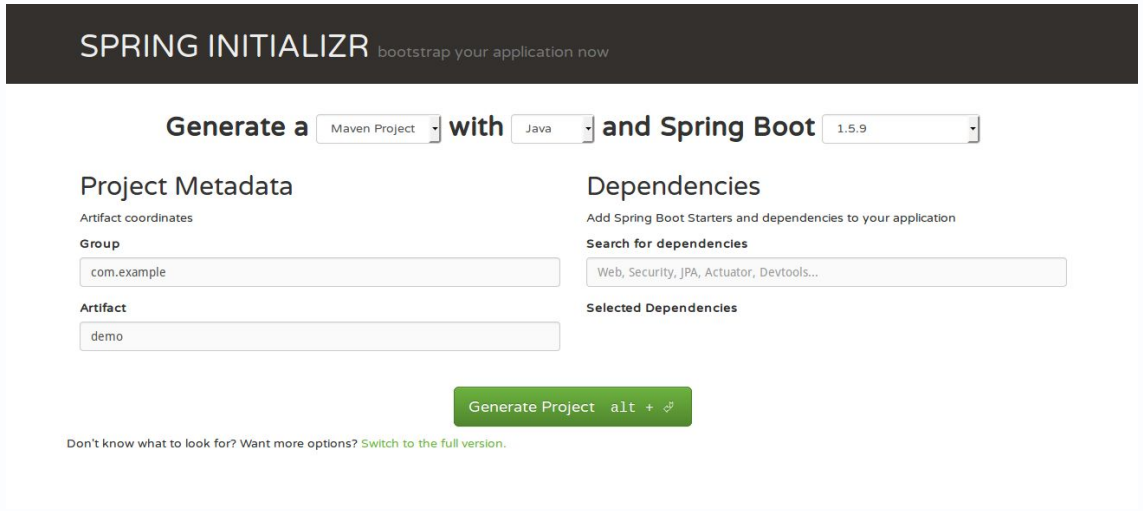
三、DEMO

Build Anything with Spring Boot

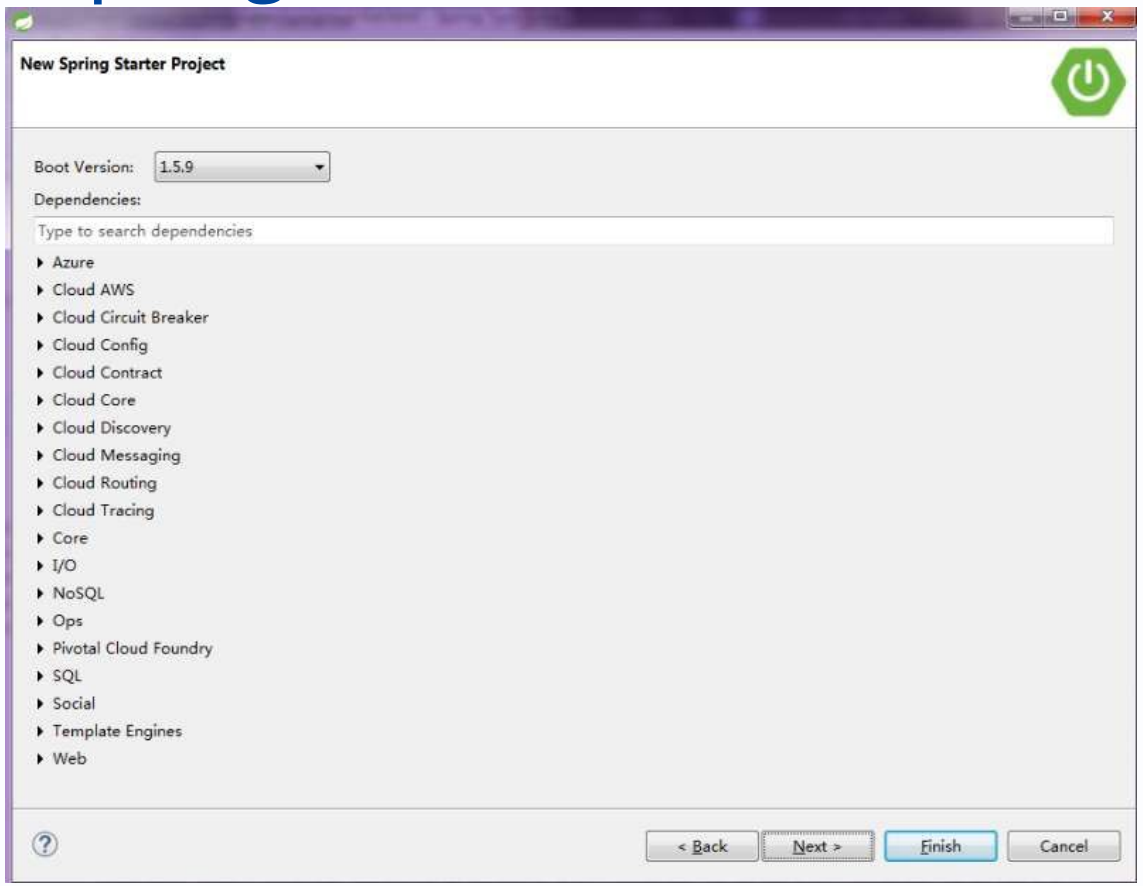
- SpringBoot是构建所有基于Spring的应用程序的起点。SpringBoot的设计是为了让您尽可能快地启动和运行，并且在最小的初始配置中使用Spring。
  - 秒级别的 Spring Initializr
  - 构建一切
    - REST API，WebSocket，网页，流媒体，任务
  - 简化安全
  - 对SQL和NoSQL的丰富支持
  - 嵌入式运行时支持——Tomcat、Jetty和Undertow
  - reload and auto restart
  - 开发人员生产力工具，如重新加载和自动重启
  - 管理的依赖关系
  - 可生产的特性，例如跟踪、指标和健康状态
  - 您最喜欢的IDE——Spring Tool Suite、IntelliJ IDEA和NetBeans

入门指南

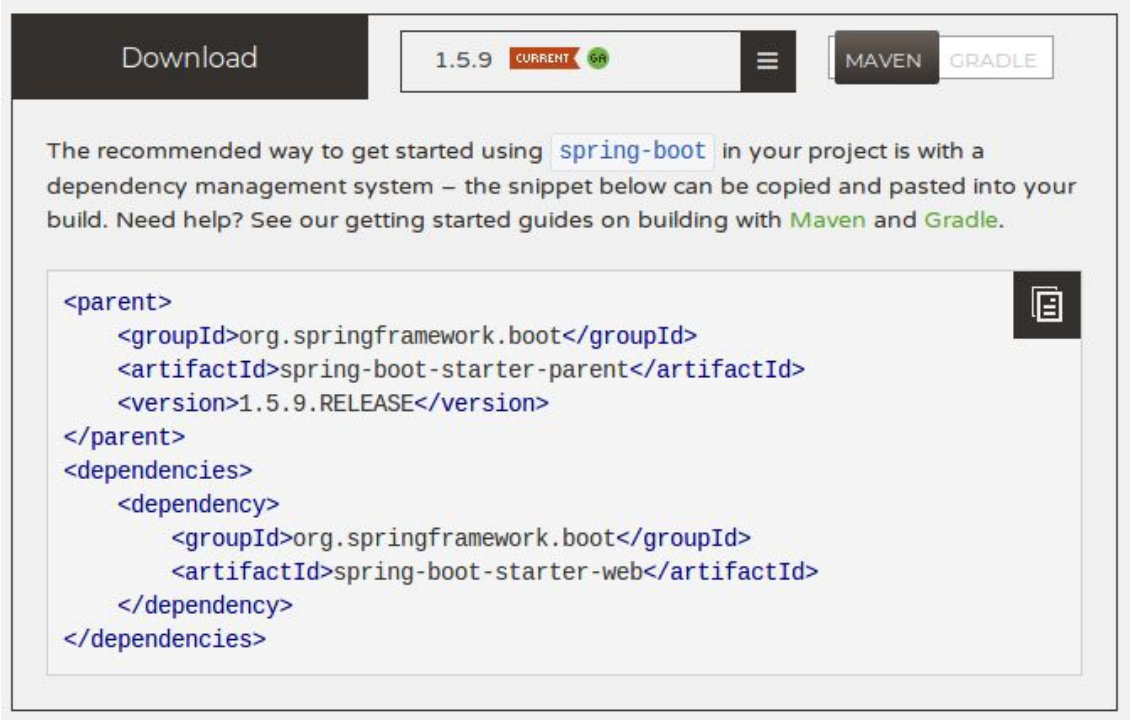
- https://start.spring.io



- Spring Tool Suite™



入门指南




```java
package hello;

import java.util.Arrays;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {

            System.out.println("Let's inspect the beans provided by Spring Boot:");

            String[] beanNames = ctx.getBeanDefinitionNames();
            Arrays.sort(beanNames);
            for (String beanName : beanNames) {
                System.out.println(beanName);
            }

        };
    }

}
```

入门指南

Run the application

To run the application, execute:

```
./gradlew build && java -jar build/libs/gs-spring-boot-0.1.0.jar
```

If you are using Maven, execute:

```
mvn package && java -jar target/gs-spring-boot-0.1.0.jar
```

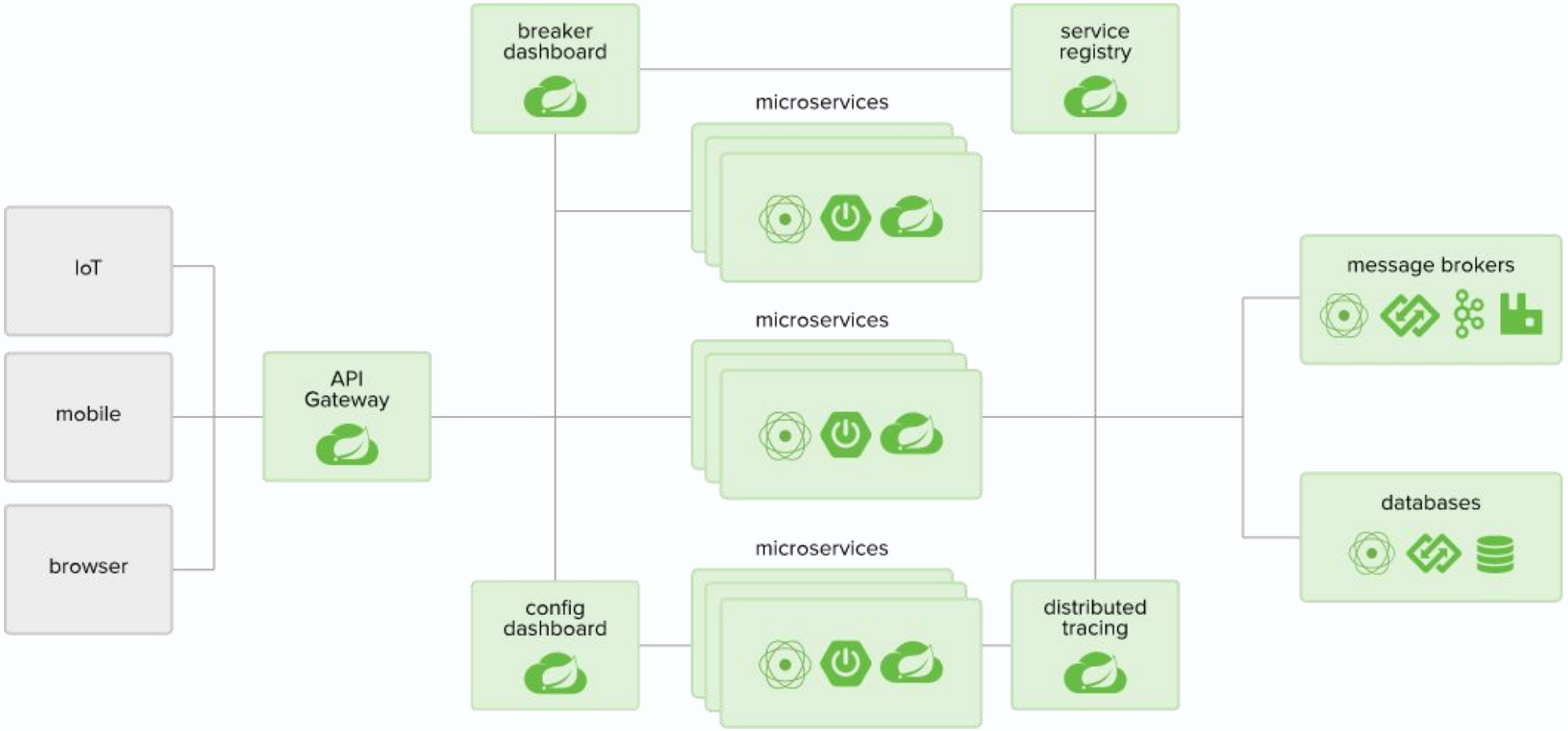
Spring Boot项目

- **Spring IO平台：**版本化应用程序的企业级分发
- **Spring框架：**用于事务管理，依赖注入，数据访问，消息传递和Web应用程序
- **Spring Cloud：**用于分布式系统，用于构建或部署您的微服务器
- **Spring数据：**对于与数据访问相关的微服务，无论是映射减少，关系还是非关系
- **Spring Batch：**用于批处理作业等操作
- **Spring Security：**用于授权和身份验证支持
- **Spring REST文档：**用于记录RESTful服务
- **Spring Social：**用于连接社交媒体API
- **Spring Mobile：**适用于移动网络应用

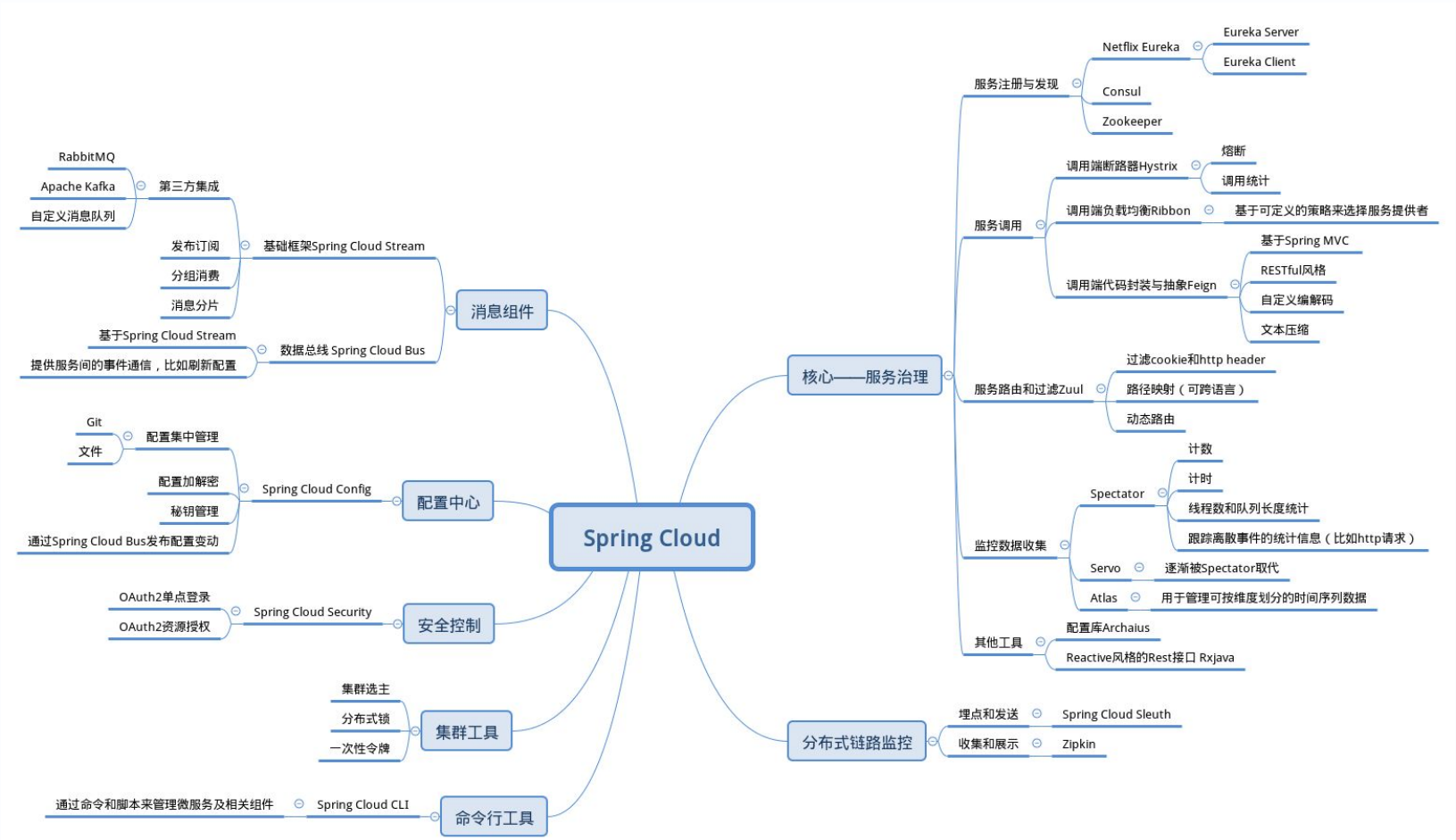
Spring Boot 期望解决程序员的问题

- 1.减轻Spring 开发者开始的难度
- 2.大部分的技术开箱即用
- 3.提供给开发者不用关注的组件（如服务器、安全、度量、外部配置）
- 4.可以免除XML配置文件
- Spring IO platform为开发者管理经过验证的各种组件的版本管理，如数据库连接池、页面框架、安全、大数据等。

Coordinate Anything With Spring Cloud



SpringCloud



快速构建

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.M6</version>
</parent>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Finchley.M5</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
<dependencies>
    <dependency>
        <groupId></groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
    </dependency>
    <dependency>
        <groupId></groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
    </dependency>
</dependencies><repositories>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/libs-milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

```java
@SpringBootApplication
@EnableDiscoveryClient
public class Application {
        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }
}
```

SpringCloud 特性

- 分布式/版本化得配置管理
- 服务注册和发现
- 路由
- 服务调用
- 负载均衡（服务端和客户端）
- 短路
- 全局锁
- 集群选举和状态
- 分布式消息系统

微服务

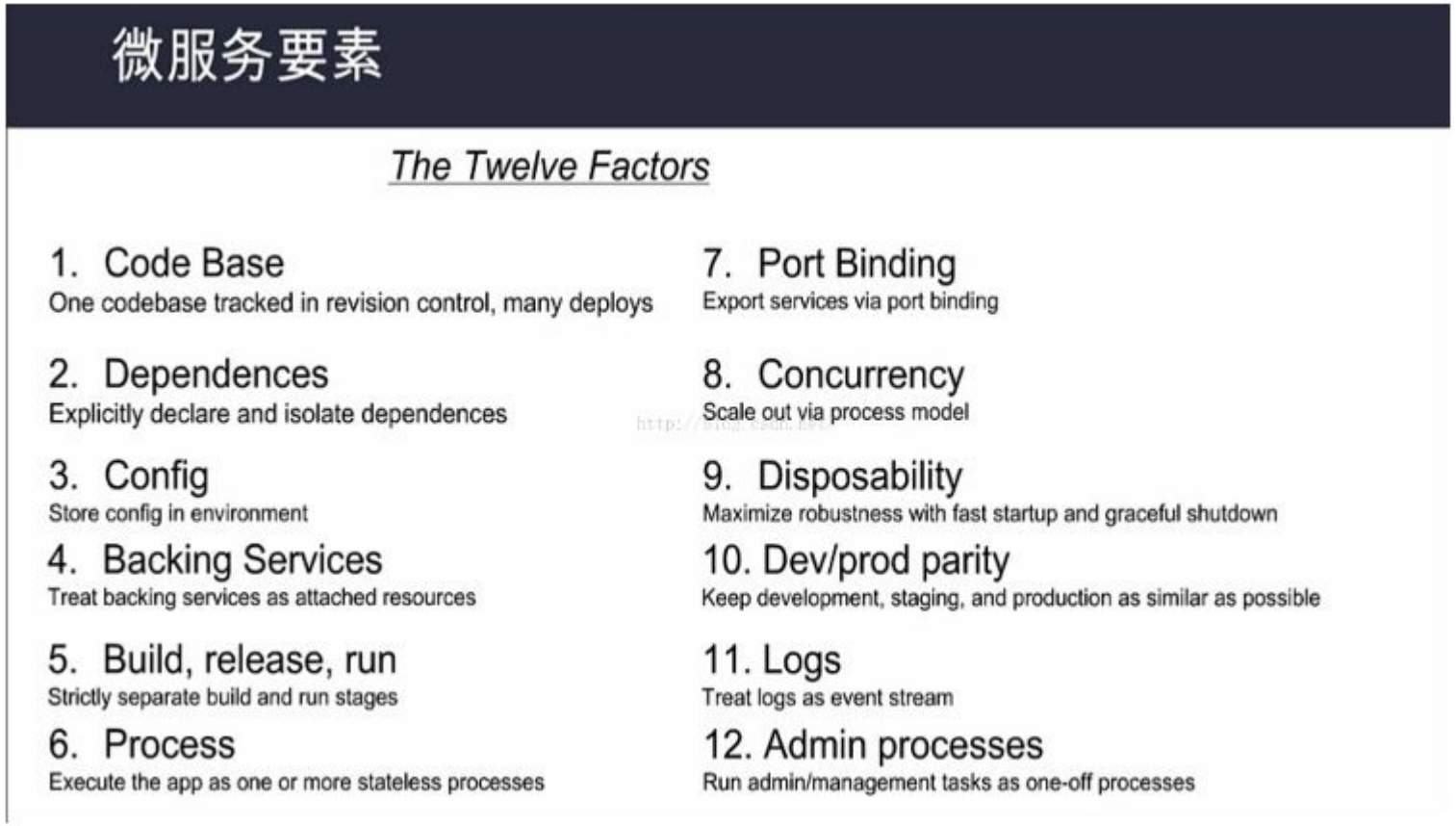
- http://www.martinfowler.com/articles/microservices.html

- 简而言之，微服务架构风格这种开发方法，是以开发一组小型服务的方式来开发一个独立的应用系统的。其中每个小型服务都运行在自己的进程中，并经常采用HTTP资源API这样轻量的机制来相互通信。这些服务围绕业务功能进行构建，并能通过全自动的部署机制来进行独立部署。这些微服务可以使用不同的语言来编写，并且可以使用不同的数据存储技术。对这些微服务我们仅做最低限度的集中管理。

单体vs. SOA vs.微服务架构

	单体	SOA	微服务
优势	l **人所众知：**传统工具、应用和脚步都是这种结构 l **IDE：**Eclipse、Intellij等开发工具多 l **易于测试：**单体应用部署后，服务或特性即可展现，没有额外的依赖，测试可以立刻开始 l **容易部署：**只需将单个WAR文件部署到单个目录即可	l **服务重用性：**通过编排基本服务以用于不同的场景 l **易维护：**单个服务的规模变小，维护相对容易 l **高可靠性：**使用消息机制以及异步机制，提高可靠性	l **简单：**单个服务简单，只关注一个业务功能 l **团队独立性：**每个微服务可以由不同的团队独立开发 l **松耦合：**微服务是松散耦合 l **平台无关性：**微服务可以用不同的语言/工具开发 l **通信协议轻量级：**使用Rest或者RPC进行服务间通信
劣势	l **不够灵活：**任何细节修改需要将整个应用重新构建、部署，这降低了团队的灵活性和功能交付频率 l **妨碍持续交付：**单体应用较大时，构建时间长，不利于频繁部署，阻碍持续交付 l **受技术限制：**必须同一语言、工具、存储等，无法根据具体的场景做出其他选择 l **技术债务：**‘不坏不修’，系统设计、代码难以修改，耦合性高	l **过分使用ESB：**使得系统集成过于复杂 l **使用基于SOAP协议的WS：**使得通信的额外开销很大 l **使用形式化的方式管理：**增加了服务管理的复杂度 l **需要使用可靠的ESB：**初始投资比较高	l 运维成本高。 l 分布式系统的复杂性。 l 异步，消息与并行的方式使得系统的开发门槛高。 l 分布式系统的复杂性也会让测试变得复杂

微服务12要素

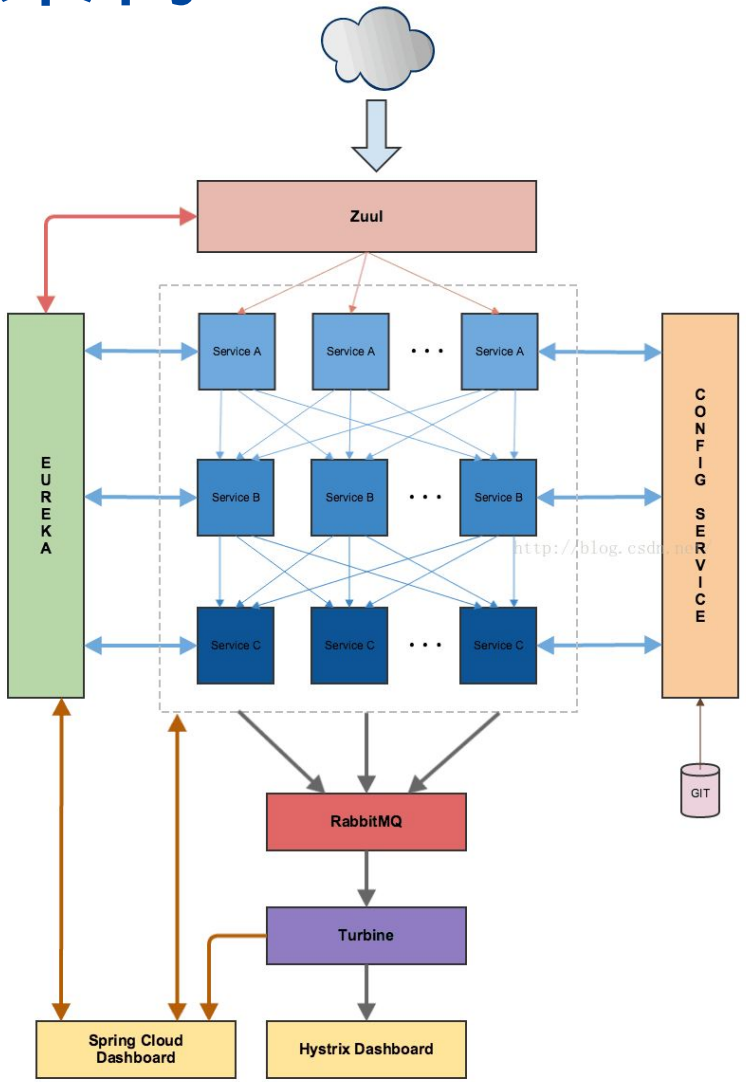

微服务要素
The Twelve Factors
1. Code Base
One codebase tracked in revision control, many deploys
2. Dependences
Explicitly declare and isolate dependences
3. Config
Store config in environment
4. Backing Services
Treat backing services as attached resources
5. Build, release, run
Strictly separate build and run stages
6. Process
Execute the app as one or more stateless processes
7. Port Binding
Export services via port binding
8. Concurrency
Scale out via process model
9. Disposability
Maximize robustness with fast startup and graceful shutdown
10. Dev/prod parity
Keep development, staging, and production as similar as possible
11. Logs
Treat logs as event stream
12. Admin processes
Run admin/management tasks as one-off processes

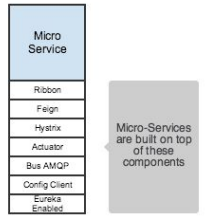
一、介绍

二、快速开始

三、详细介绍

Spring Cloud 微服务组件架构





Ribbon	Ribbon is a client side load balancer
Feign	Declarative web service client
Hystrix	Implementation of circuit breaker pattern
Actuator	Adds features to monitor and manage the application
Bus AMQP	Broadcasts messages through Queue
Config Client	Support for Config Server
Eureka	Discovery Service
Eureka Enabled	Discovery Service client
Config Service	Server for centralised configuration
RabbitMQ	Message broker software
Turbine	Hystrix streams aggregator
Hystrix Dashboard	Displays the health of each circuit breaker
Spring Cloud Dashboard	Admin interface for Spring Cloud Applications

举例说明

- 需求
  - 幼儿园要根据不同年级每天练习10以内加法，随机出题。比如3岁每天5道题，4岁每天10道题
- 使用技术
  - SpringBoot 快速构建
  - 服务发现与消费
  - 服务注册
  - 服务配置
  - 负载均衡
  - 熔断器
  - 监控指标

TAIJI 太极

示例代码清单

启动顺序	工程名称	说明	端口号	访问地址
1	open-eureka-server	服务注册中心	1111	http://127.0.0.1:1111
2	open-config-server	配置中心	8888	
2	open-cloud-admin	应用监控管理	8090	http://127.0.0.1:8090
3	open-cloud-idiom	题库服务，一份代码可运行多个示例	2222/可设置随机端口	http://127.0.0.1:2222/add
4	open-cloud-idiom-ui	答题系统，包含了服务调用，两种方式的客户端负载，以及熔断等操作	3333	http://127.0.0.1:3333/add
5	open-cloud-idiom-hystrix-dashboard	可视化的实时的监控，以仪表盘的形式展现，数据可导出	4444	http://127.0.0.1:4444/hystrix.stream
6	open-cloud-idiom-zuul	路由,服务端的负载	5555	http://127.0.0.1:5555/add

外部配置---服务端(open-config-server)

- 在Git上创建repo仓库，用来存储配置信息
  - 略
- 构建配置中心open-config-server,连接到Git仓库
  - 步骤1.POM.XML中增加spring-cloud-config-server的依赖

```
<dependencies>
        <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-config-server</artifactId>
        </dependency>
</dependencies>
```

外部配置---服务端(open-config-server)

- 步骤2.开启外部配置注解

```
package org.open.config;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.cloud.config.server.EnableConfigServer;

@EnableConfigServer   开启外部配置的注解
@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                new SpringApplicationBuilder(Application.class).web(true).run(args);
        }

}
```

外部配置---服务端(open-config-server)

- 步骤3.在application.properties 或者 bootstrap.properties中配置相关信息

```
# git\u7BA1\u7406\u914D\u7F6E
spring.cloud.config.server.git.uri=https://github.com/startSnow/open-cloud
spring.cloud.config.server.git.searchPaths=repo
spring.cloud.config.server.git.username=startSnow
spring.cloud.config.server.git.password=
spring.profiles.active=prod
```

外部仓库地址

搜索仓库文件路径

外部配置---客户端(open-cloud-idiom)

步骤1.pom.xml增加依赖

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
```

步骤2.配置文件配置信息

```
spring.cloud.config.profile=evn
spring.cloud.config.label=master
spring.cloud.config.uri=http://localhost:8888/

spring.cloud.config.discovery.enabled=true
spring.cloud.config.discovery.serviceId=config-server
```

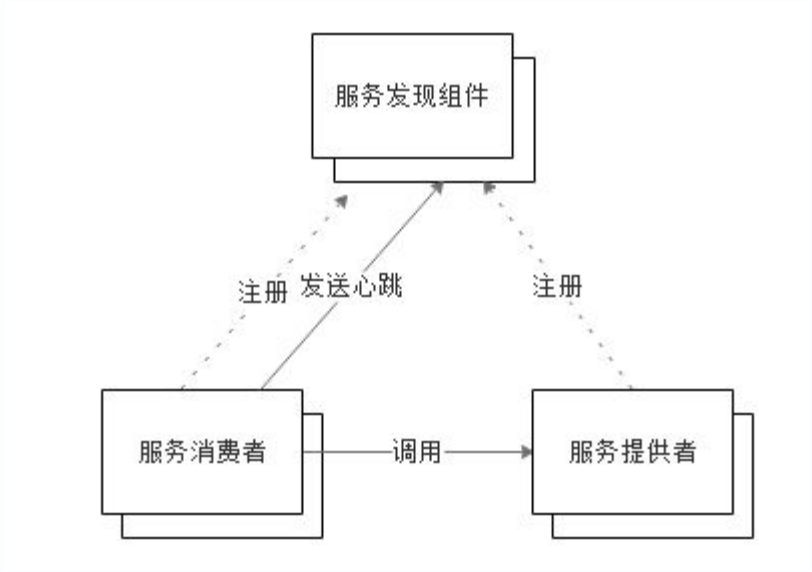
外部配置---客户端 (open-cloud-idiom)

- 步骤3.开启允许刷新注解

```
@RefreshScope
@RestController
public class IdiomController {
```

  - 注意:
    - 当想让配置信息刷新，请以POST 方式请求：http://localhost:2222/refresh

使用Eureka实现服务注册与发现（open-eureka-server）



spring Eureka　HOME　LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2018-01-25T12:57:03 +0800
Data center	default	Uptime	01:34
		Lease expiration enabled	true
		Renews threshold	10
		Renews (last min)	12

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
HYSTRIXDASHBOARD	n/a (1)	(1)	UP (1) - 10.0.32.15:HystrixDashboard:4444
IDIOM-UI	n/a (1)	(1)	UP (1) - 10.0.32.15:idiom-ui:3333
IDIOMS	n/a (1)	(1)	UP (1) - 10.0.32.15:idioms:2222

General Info

Name	Value
total-avail-memory	305mb
environment	test
num-of-cpus	8
current-memory-usage	186mb (60%)
server-uptime	01:34
registered-replicas	http://localhost:1111/eureka/
unavailable-replicas	http://localhost:1111/eureka/,
available-replicas	

Instance Info

Name	Value
ipAddr	10.0.32.15
status	UP

Eureka服务端（open-eureka-server）

- 步骤1.pom.xml增加依赖

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka-server</artifactId>
</dependency>
```

- 步骤2.启动类中配置启动注解

```
/**
 * EurekaApplication 上面的 EnableEurekaServer 注解就保证 eureka 服务器能正常启动。
 * @author chixue
 */
@EnableEurekaServer
@SpringBootApplication
public class EurekaApplication {
        public static void main(String[] args) {
                new SpringApplicationBuilder(EurekaApplication.class).web(true).run(
                                args);
        }
```

Eureka服务端 (open-eureka-server)

- 其他：
  - application.properties中描述的 eureka 服务器的一些基本配置
    Bootstrap.properties指定所使用的配置服务器。

服务注册到Eureka服务中心

- 步骤1.pom.xml

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

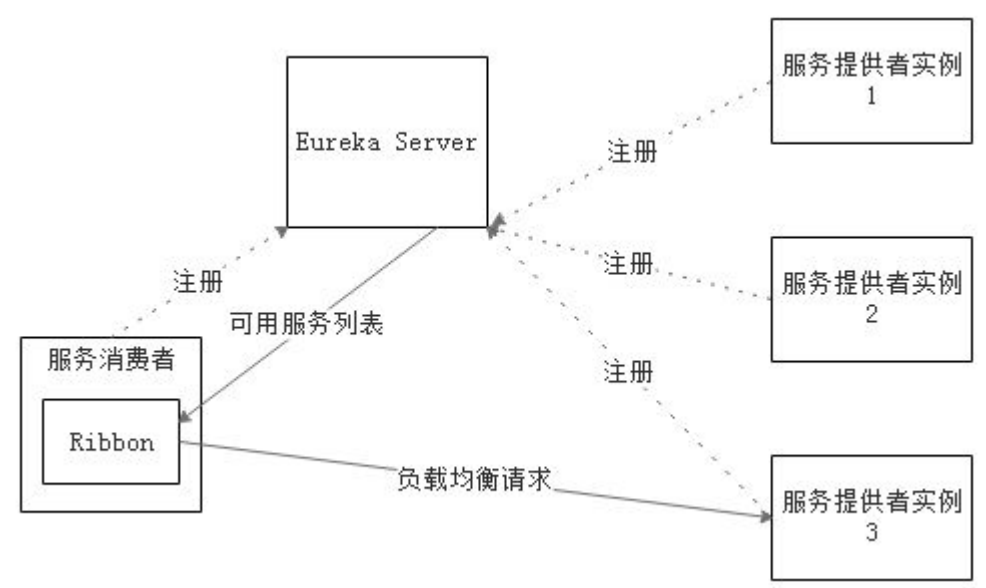
- 步骤2.指定服务中心地址

```
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
```

- 步骤3.开启注解@EnableEurekaClient

```
@SpringBootApplication
@RestController
@EnableDiscoveryClient  //激活eureka中的DiscoveryClient实现
public class ExamApplication {
```

客户端负载均衡Ribbon (open-cloud-idiom-ui)



- 步骤1
  - pom.xml 不用单独依赖，依赖spring-cloud-starter-eureka可继承依赖
- 步骤2.增加@LoadBalanced注解

```
@Bean
@LoadBalanced
RestTemplate restTemplate() {
    return new RestTemplate();
}
```

- 步骤3.restTemplate 调用服务

```
public String addition() {
    return restTemplate.getForObject("http://idioms/add",String.class);
}
```

服务中心注册的服务ID

客户端负载Feign (open-cloud-idiom-ui)

- 步骤1.pom.xml

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
```

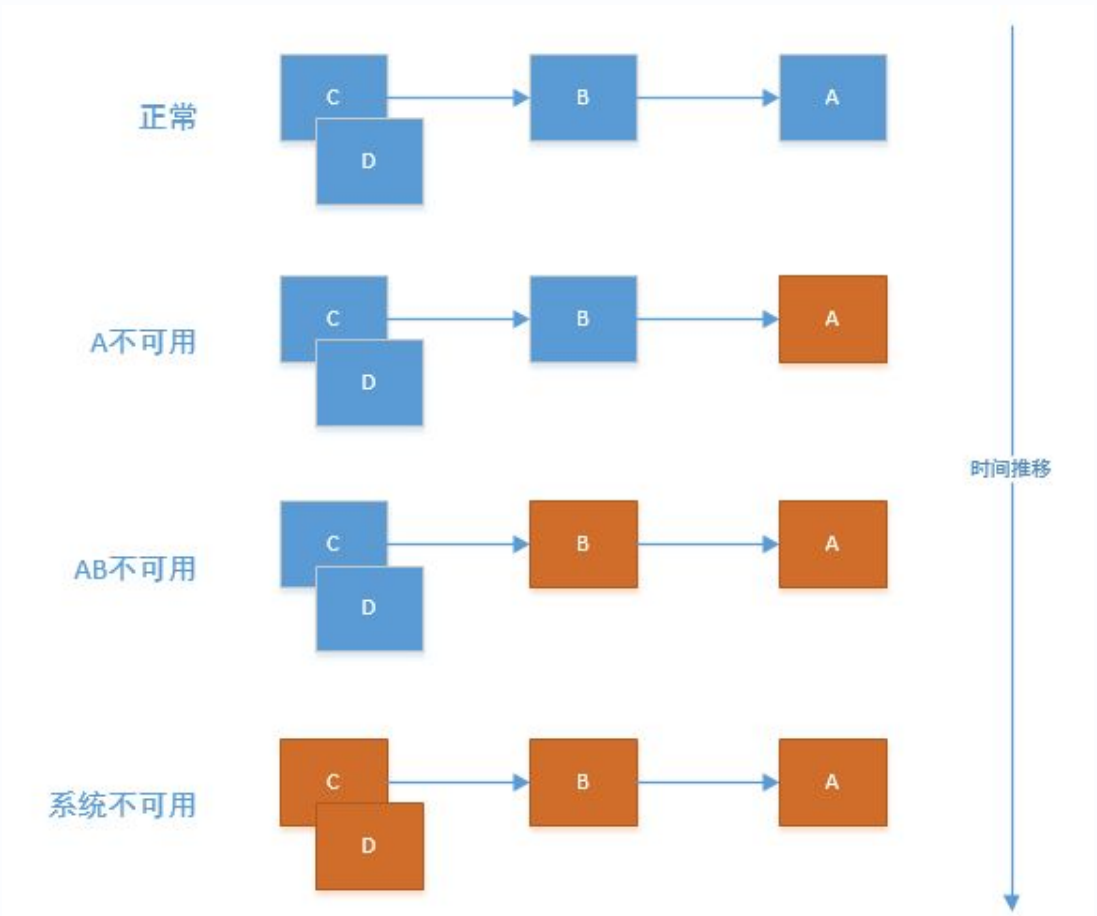
- 步骤2.通过@EnableFeignClients注解开启Feign功能

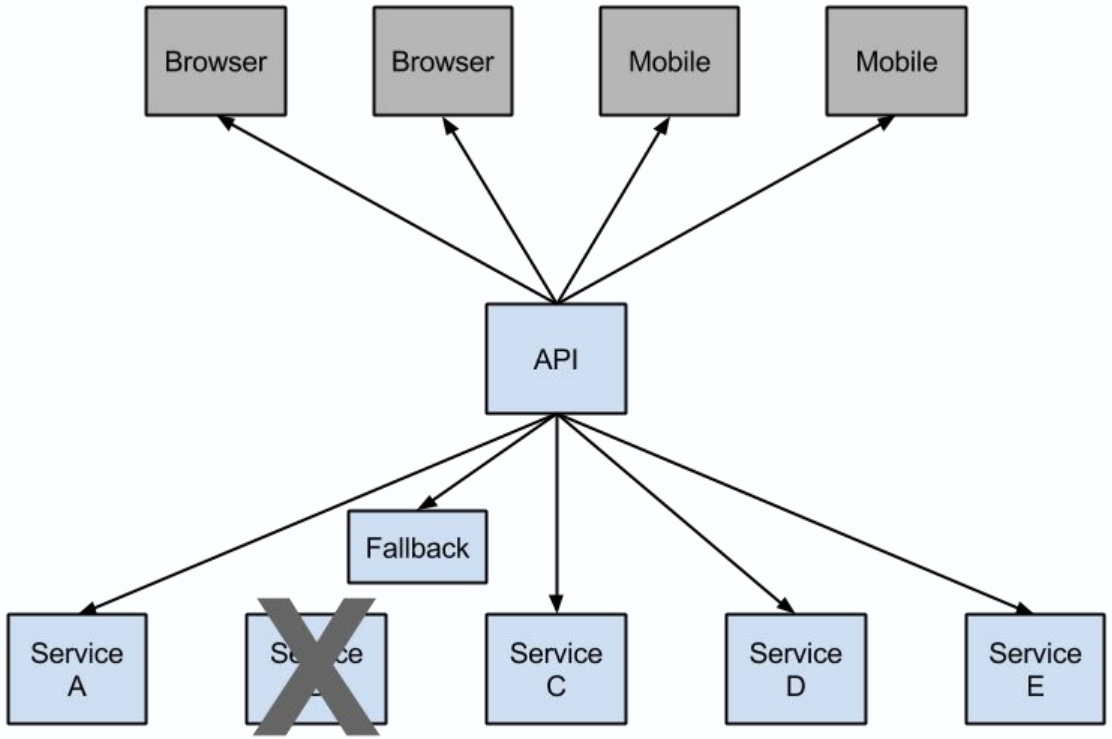
```
@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker
@EnableFeignClients
@RestController
@EnableConfigurationProperties
public class IdiomUiApplication {
```

- 步骤3.定义idioms服务的接口，具体如下：

```
@FeignClient("idioms")   调用的服务ID
public interface Idioms {
  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String idiom();
}
```

微服务断熔--雪崩效应


正常
C
D
B
A
A不可用
C
D
B
A
AB不可用
C
D
B
A
系统不可用
C
D
B
A
时间推移


Browser
Browser
Mobile
Mobile
API
Fallback
Service A
Service C
Service D
Service E

Ribbon中引入Hystrix (open-cloud-idiom-ui)

- 步骤1.pom.xml 引用

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>
```

- 步骤2.
  - 在主类IdiomUiApplication中使用@EnableCircuitBreaker注解开启断路器功能

```
@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker
@EnableFeignClients
@RestController
@EnableConfigurationProperties
public class IdiomUiApplication {
```

Ribbon中引入Hystrix (open-cloud-idiom-ui)

- 步骤3
  - 改造原来的服务消费方式，在使用ribbon消费服务的函数上增加@HystrixCommand注解来指定回调方法

```
@HystrixCommand(fallbackMethod = "addError")
public String addition() {
    return restTemplate.getForObject("http://idioms/add",String.class);
}

public String addError() {
    return "no gain";
}
```

Feign使用Hystrix (open-cloud-idiom-ui)

- 注意这里说的是“使用”，没有错，我们不需要在Feigh工程中引入Hystix，Feign中已经依赖了Hystrix。
- 使用@FeignClient注解中的fallback属性指定回调类

```
@HystrixCommand(fallbackMethod = "defaultIdiom")
public String idiom() {
  return idioms.idiom();
}

public String defaultIdiom() {
  return "No pain, no gain";
}
```

实时监控Hystrix的各项指标（open-cloud-idiom-hystrix-dashboard）

- 步骤1.
  - Dashboard服务是个独立的结点，不需要配置eureka信息，只需增加以下依赖：

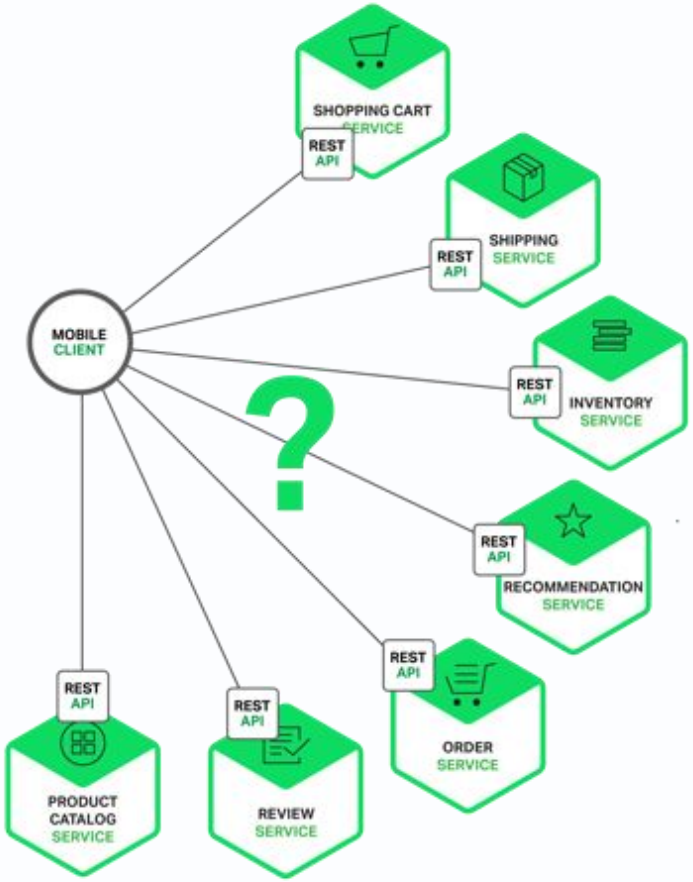
```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
</dependency>
```

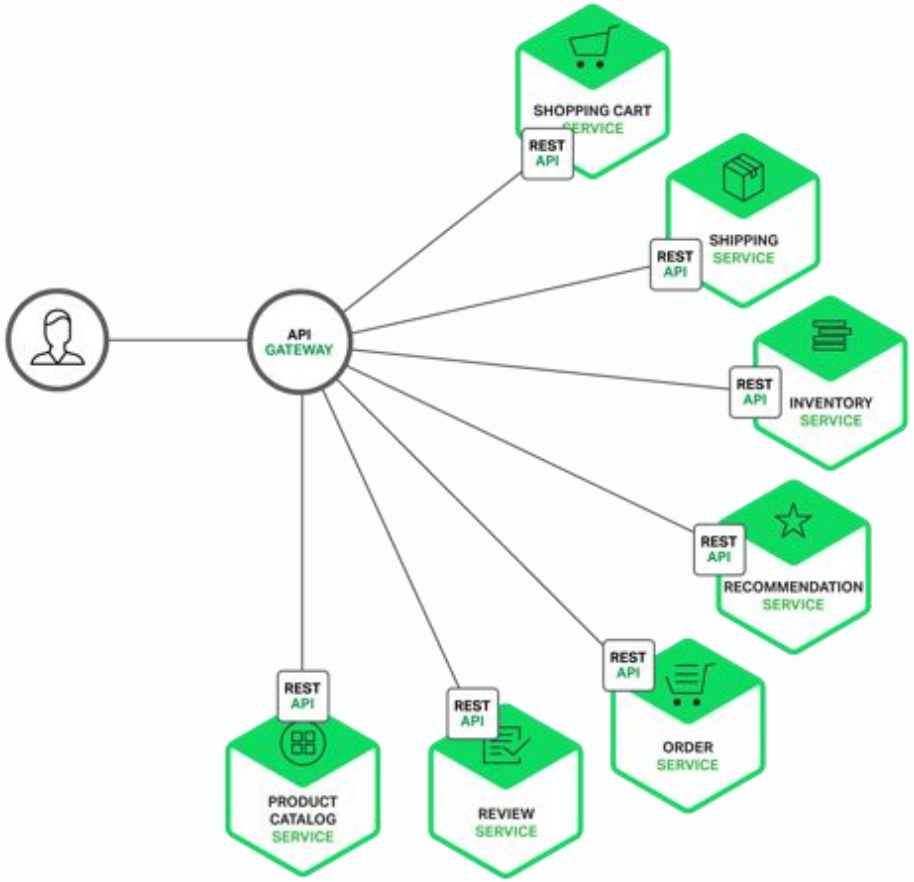
- 步骤2.
  - Dashboard主程序,开启@EnableHystrixDashboard

```
@SpringBootApplication
@EnableHystrixDashboard
public class HystrixDashboardApplication {

    public static void main(String[] args) {
        SpringApplication.run(HystrixDashboardApplication.class, args);
    }
}
```

- 步骤3.
  - 要确保被监控的服务打开了Actuator（依赖spring-boot-starter-actuator），开启了断路器（@EnableCircuitBreaker注解）。

API 网关


SHOPPING CART
SERVICE
REST
API
SHIPPING
SERVICE
MOBILE
CLIENT
INVENTORY
SERVICE
?
RECOMMENDATION
SERVICE
ORDER
SERVICE
PRODUCT
CATALOG
SERVICE
REVIEW
SERVICE


SHOPPING CART
SERVICE
REST
API
SHIPPING
SERVICE
API
GATEWAY
INVENTORY
SERVICE
RECOMMENDATION
SERVICE
ORDER
SERVICE
PRODUCT
CATALOG
SERVICE
REVIEW
SERVICE

服务网关（open-cloud-idiom-zuul）

- 步骤1.pom.xml增加配置引入spring-cloud-starter-zuul

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-zuul</artifactId>
</dependency>
```

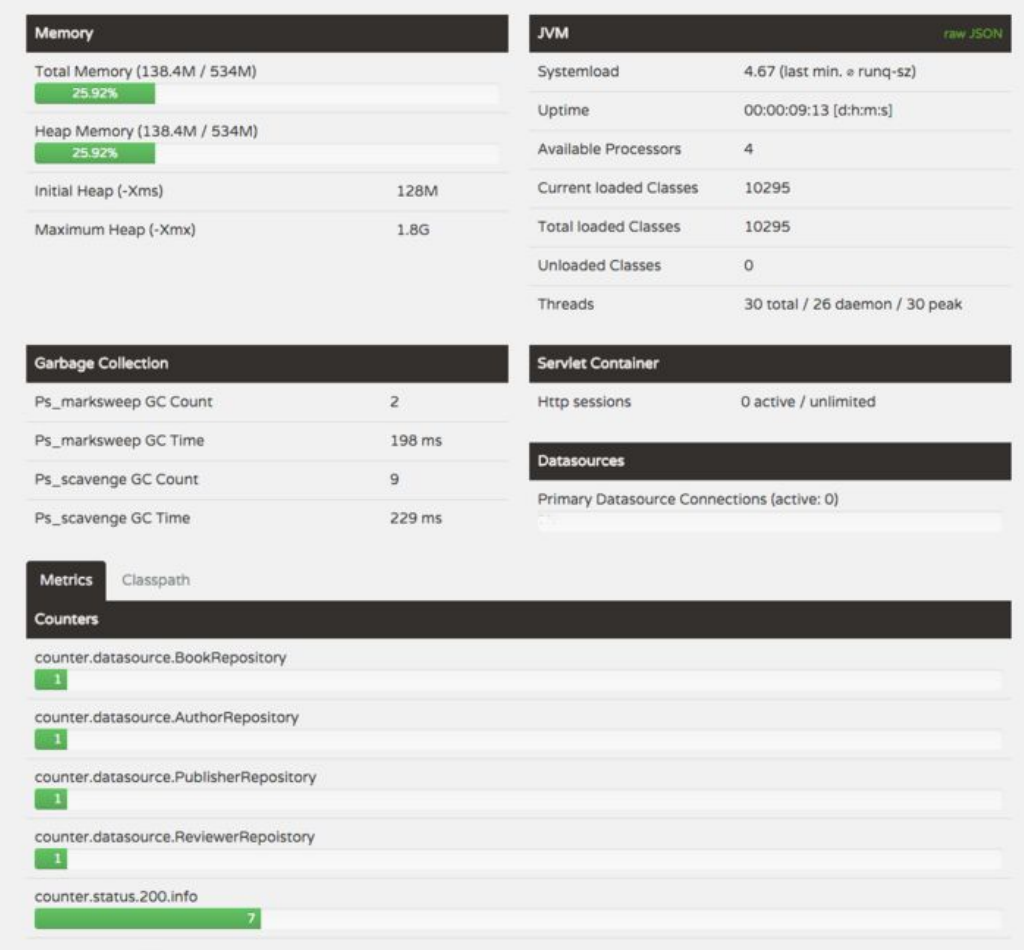
- 步骤2.启动类

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableZuulProxy
public class ZuulApplication {

        public static void main(String[] args) {
                SpringApplication.run(ZuulApplication.class, args);
}
```

- 步骤3.配置文件

```
spring.application.name=idiom-zuul
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/
server.port=5555
zuul.routes.idiom-ui=/**

spring.boot.admin.url=http://localhost:8090
```

健康监控


Memory
Total Memory (138.4M / 534M)
25.92%
Heap Memory (138.4M / 534M)
25.92%
Initial Heap (-Xms) 128M
Maximum Heap (-Xmx) 1.8G
JVM
raw JSON
Systemload 4.67 (last min. ⌀ runq-sz)
Uptime 00:00:09:13 [d:h:m:s]
Available Processors 4
Current loaded Classes 10295
Total loaded Classes 10295
Unloaded Classes 0
Threads 30 total / 26 daemon / 30 peak
Garbage Collection
Ps_marksweep GC Count 2
Ps_marksweep GC Time 198 ms
Ps_scavenge GC Count 9
Ps_scavenge GC Time 229 ms
Servlet Container
Http sessions 0 active / unlimited
Datasources
Primary Datasource Connections (active: 0)
Metrics
Classpath
Counters
counter.datasource.BookRepository
1
counter.datasource.AuthorRepository
1
counter.datasource.PublisherRepository
1
counter.datasource.ReviewerRepoistory
1
counter.status.200.info
7

SpringBootAdmin-server(open-cloud-admin)

- 说明
  - 当Spring Boot工程开启Actuator，提供的监控接口，例如：/health、/info等等，实际上除了之前提到的信息，还有其他信息业需要监控：当前处于活跃状态的会话数量、当前应用的并发数、延迟以及其他度量信息。我们可以利用Spring-boot-admin对应用信息进行可视化。

SpringBootAdmin-server 搭建过程（一）

- 步骤1.pom.xml添加相关依赖

```
<!--admin的服务-->
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-server</artifactId>
  <version>1.5.5</version>
</dependency>
<!--admin的ui依赖-->
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-server-ui</artifactId>
  <version>1.5.5</version>
</dependency>
```

SpringBootAdmin-server 搭建过程（二）

- 步骤2.在主启动类中添加@EnableAdminServer注解

```
@Configuration
@EnableAutoConfiguration
@EnableDiscoveryClient
@EnableAdminServer
public class SpringBootAdminApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootAdminApplication.class, args);
    }
}
```

- 步骤3.在配置文件application.properties中配置

```
spring.boot.admin.url=http://localhost:${server.port}
```

SpringBootAdmin监控---Client

- 步骤1.pom.xml添加依赖

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

- 步骤2.在配置文件中application.properties
  - 触发自动配置、指明 Server 注册地址

```
idiomui.prefix=\u4F20\u7EDF\u6587\u5316:

spring.boot.admin.url=http://localhost:8090
```

相关示例代码

- https://github.com/startSnow/SpringCloudDemo.git

谢谢，大家！

TAIJI 太极
创新融合　让我们的世界更智慧
北京市海淀区北四环中路211号 100083
211 NORTH FOURTH RING ROAD HAIDIAN DISTRICT.BEIJING 100083.CHINA
T:010-51616888 F:010-51616889
www.taiji.com.cn