知识回顾：

1. SpringCloud
   1. SpringBoot和SpringCloud，SpringBoot构建项目更加简单（基于Maven，jar统一管理，自己诞生一个父工程；主要三大框架配置问题，只要支持技术（必然整合包spring-boot-starter-xxx，）都可以大量减少配置文件；启动类方式）
   2. SpringBoot在之前开发方式上更加精简，注解+约定大于配置
   3. SpringCloud微服务霸主，SpringCloud整个蛋糕，Dubbo其中一项，官网专注于基于RPC调用方式。
2. SpringCloud全家桶
   1. Eureka注册中心

CAP定理：构建分布式系统，必须实现P，有两种选择CP，AP

C一致性，mysql、oracle、sqlserver DRMS。数据要有唯一出处。外键。

A可用性，哪怕信息已经旧的。

P 分区容错性。

Eureka适合注册中心，还是ZooKeeper？

针对注册中心特点，注册信息会频繁发生变化，不会。

Eureka设计目标AP，ZooKeeper设计目标CP。这时针对注册中心业务，Eureka更加合适。

* 1. Ribbon前端负载均衡器

Ribbon和Nginx比较？

Nginx反向代理，Ribbon可以在访问消费者本地增加缓存，客户端发起请求前，它的路径已经决定。

Nginx没有故障处理，也就是说，它代理某个服务宕机，或者无法访问。它不能告诉程序，程序调用才出错，等一会（超时时间），抛出异常。Ribbon依赖Eureka动态列表。Eureka客户端有向服务端定时发心跳，如果客户端阀值（几次机会）之后，宣布服务死亡，Eureka会把其服务从列表中移除。Eureka客户端它的心跳处理时，顺便判断列表有无更新。如果判断有更新，从新获取一个新的数据。

* 1. Feign声明式 RESTFul形式支持，RestTemplate，接口更加方便。Jdk动态代理（$ProxyN） cglib动态代理（CGLIB）。

以接口形式体现，接口和业务挂钩，业务程序员必须自己做。做接口同时对业务更加了解。屏蔽复杂API学习成本。

职业发展规划：

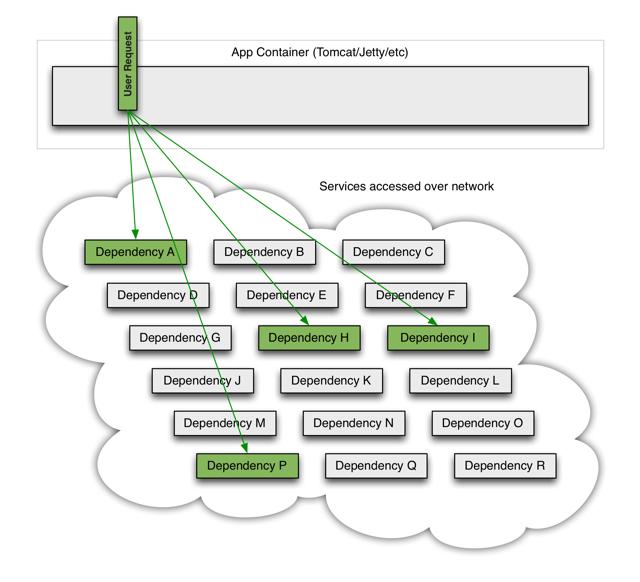
1. 测试和开发
   1. 开发的生命周期更长。谁对业务更加了解？测试还不如开发了解业务。
   2. 测试沦为形式化，薪资
   3. 软件团队中永远开发最重要。

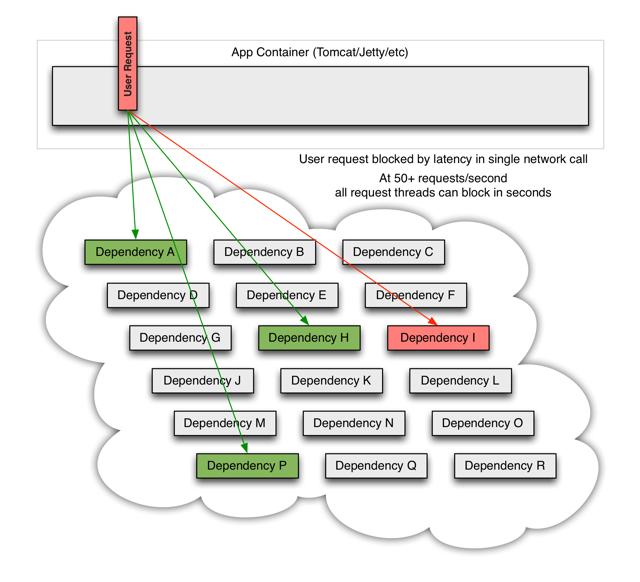
SpringCloud

1. Hystrix断路器（熔断器）

Why？

微服务雪崩





超时机制，30s，30m。

缺点：

1. 超时时间可能设置太长了
2. 手工去解决

Hystrix断路器（熔断器）

电闸（保险丝）

1）类似保险丝，熔断，此时用户请求就不能去访问后面微服务。

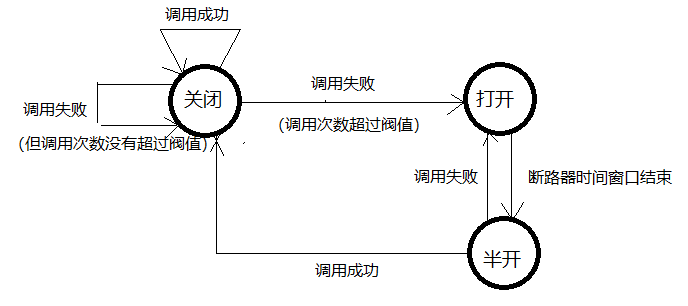
问题：

1. 传统方式是错误页面，用户体验差。

解决：返回默认值（降级）

1. 如果微服务正常可以访问了Hystrix自动恢复到正常访问连接

Hystrix断路器状态表：3种状态：关闭、打开、半开

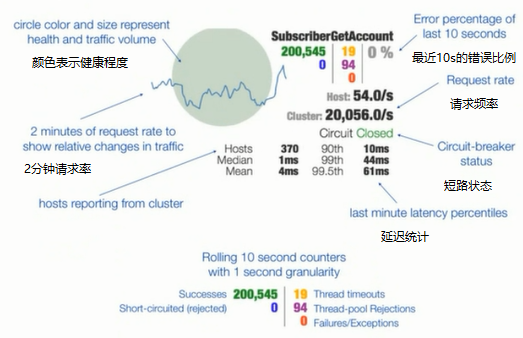


Netflux出品Hystrix，API繁琐，javanica把hystrix又封装一层，但是调用更加简洁。

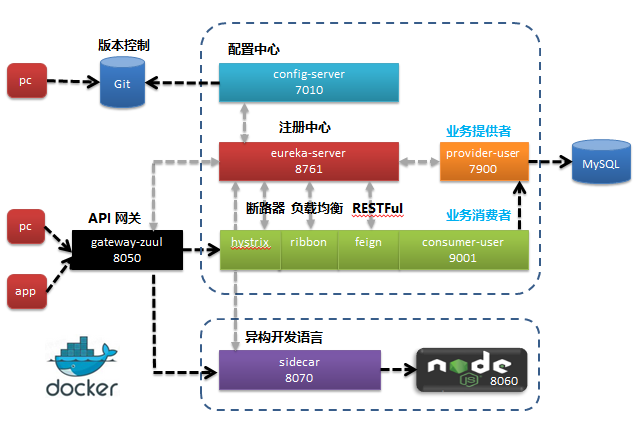
spring-boot-starter-actuator

一旦我们需要监控系统的信息，需要系统提供给我们丰富日志信息，方便监控

必须开启这个依赖



1. Zuul API Gateway网关



Gateway网关开发方式

1. nginx+lua(linux shell脚本) 5w/s
2. springcloud gateway
3. springcloud zuul2.0 100w/s

Zuul怪兽

静态资源怎么处理？

静态资源：js、css、image、html

1. tomcat
2. nginx
3. browser
4. zuul

browser>zuul>nginx>tomcat

底层原理：拦截器、过滤器

aplication.yml

zuul:

routes: 路由

app-a: 随便写，唯一代表

path: /user/\*\*映射了一个路径/user/hello/tony\*\*当前层下所有子路径

serviceId: provider-user 提供者appliaciton.name

方便过滤，/admin，不同的前缀就可以支持不同的权限过滤，前缀就相当于一个分类

3）Sidecar异构开发语言支持，调用NodeJS

异构开发语言，多种语言互相访问

传统怎么处理异构开发语言支持

1. 二进制，截串 tcp 2jfkj389（通讯协议）电信
2. Txt（id,name,age）,edi报文（国际物流）kv
3. xml 标签，灵活，dom4j。sax。WebService ApacheCXF。废弃性能太低。Xml传输标签太多，影响传输性能。Xml纯文本，安全性差。它事先要自动生成java类。符合标准：基于HTTP协议SOAP简单对象方法协议。封装多层后转化多次时间都浪费了。
4. json，京淘httpclient+json，json内容没有xml那么多标签，节约大量空间，传输效率高，但比txt有结构，直接获取其中一个值，方便转化成object。

转化工具类：jackson、fastjson阿里

1. SpringCloud出品Sidecar。它把我们的程序封装了一下。对外暴露RESTFul+json

NodeJS

Js天生缺点：同源策略，js跨域。解决跨域问题：jsonp。

同源策略，js称为脚本语言。弱语言。不允许访问本地资源（磁盘、内存、CPU）

NodeJS，它完成一个web服务器（后端），可以存储本地资源。本地隔离。

Js也可以做整个其他语言所能做的事情。

前端工程师什么都能做了？

1. 互联网架构（外延，nginx c语言，rabbitmq erlang高并发语言，docker go语言）
2. 大数据（Hadoop生态链java、Spark生态链scala编译完java，scala下一代java）

Java1.8引入lamda表达式（函数式编程） 面向对象+面向函数

学前端好学java互联网架构？

1. 很快就没得升职，跳槽
2. 薪资

学大数据课，AI课程

1. 起点高，薪资高，然后遇到天花板
2. 算法：数学，概率论、高数、微积分、离散。推荐算法；大数据干的事情太少了（海量数据分析）
3. 大数据产品是否必须

互联网架构（主）+大数据（辅）+AI

分布式、高并发、高可用+海量数据处理+机器模拟人类

node-v8.11.3-x64.msi

一路安装，

执行js文件

node-service.js 格式必须utf-8，中文乱码，不能解析。

问题：

Sidecar有个使用前提，现有项目中必须使用api网关来访问

浏览器>api网关>映射/sidecar>nodejs

SpringCloud

1. Hystrix断路器（熔断器）

Why？定义、概念、用途；和项目来结合；挖陷阱



系统运行时，可靠性。如果永不宕机100%。

一个微服务99.99%，一千次，只有一次失败。

投标时，标书要求：手册。

*99.99%*30*=99.7%*

服务错误存在，随着服务请求次数和服务数量的增加，错误次数也增加。

断路器用途出来，传统只能抛出异常，提示用户不可用。

断路器后快速响应用户，第一种就快速失败。第二种，返回预先定义的值。“假”的吗？术语：降级。

Hystrix怎么和项目结合使用？

1. 创建一个SpringBoot（陷阱）应用程序，手写pom.xml增加依赖starter-hystrix，hystrix基于Ribbon和Feign（陷阱）来开发。
2. 在原来Ribbon和Feign项目基础上，修改它的启动类，增加@EnableCircuitBreaker注解即可
3. 在controller中增加一个注解，@HystrixCommand，有一个参数fallbackMethod，失败返回回调一个方法名。在这个controller下在声明一个回调方法，这个参数和参数类型、返回值、返回值类型必须和配置的这个方法一致。
4. 用户直接请求Hystrix消费者就可以
5. Hystrix提供3种状态，当访问异常时，尝试多次（阀值），如果还无法访问成功，断路器由“关闭”状态切换“打开”状态。业务如果继续有人访问，如果失败，继续处于“半开”状态，一旦有访问成功，状态立即切换到“关闭”。
6. Zuul API网关gateway

SpringCloud是基于RESTFul+json形式，访问链接没有安全性。引入API网关，对于访问者进行授权验证。把非法用户禁止访问，提升整个系统安全。

微服务架构，把服务拆成很多细小的服务，带来一个问题。之前在一个方法内部完成，变成多个微服务远程调用，发起端就需要发起多个微服务请求，每个微服务都有创建和销毁，这样编写代码繁琐，也占用更多资源。Api网关就可以合成一次调用，内部发起多次请求，讲结果拼成一个结果。

屏蔽很多开发结构RPC，json，异构。

拦截器、缓存、授权等等

直接软件公司做微服务来卖！可以监控调用次数和时长。

1. Sidecar异构开发语言调用

实际系统中由于历史原因，.net，php；开发团队技术原因。

SpringCloud选型RESTFul+json，因为完成支持异构。

Sidecar把异构开发语言，对外暴露url地址，做了映射。结合Zuul映射。

启动NodeJS，安装完成node node-service.js直接执行，启动web服务。

Sidecar把nodejs映射来封装

sidecar:

port: 8060 端口

health-uri: <http://localhost:8060/health.json> 访问健康地址

必须，提供平台感知服务死活。如果暂停这个服务。Status:DOWN

Zuul配置中映射：

zuul:

routes:

user-app:

path: /user/\*\*

serviceId: provider-user

sidecar-app: 随便写，不要有重复

path: /car/\*\* 访问多加一级路径，区分不同微服务入口

\*\*代表这个uri，支持多级目录

serviceId: sidecar 映射，到eureka注册application.name