Fiegn 声明式接口调用

什么是Feign

Netflix，Feign 是一个提供模版的声明式 Web Service 客户端，使用 Feign 可以简化 Web Service 客户端的编写，开发者可以通过简单的接口和注解来调用 HTTP API。

Spring Cloud Feign：可插拔、基于注解、负载均衡、服务熔断

只需要创建接口，同时在接口上添加相关注解即可。

Ribbon 和 Feign 的区别：

- Ribbon 是一个通用的 HTTP 客户端工具，Feign 是基于 Ribbon ，更加灵动

Feign 的特点：

- Feign 是一个声明式 Web Service 客户端
- 支持 Feign 注解、Spring MVC 注解
- Feign 基于 Ribbon 实现
- Feign 集成了 Hystrix

1、创建 Module，pom.xml

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

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

2、application.yml

```yaml
server:
  port: 8050
spring:
  application:
    name: feign
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
```

3、启动类

```java
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

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

4、实体类

```java
package com.southwind.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {
    private Integer id;
    private String name;
}
```

5、FeignProviderClient 接口

```java
package com.southwind.feign;
```

```
import com.southwind.entity.Student;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@FeignClient(value = "provider")
public interface FeignProviderClient {

    @GetMapping("/provider/findAll")
    public Collection<Student> findAll();

    @GetMapping("/provider/findById/{id}")
    public Student findById(@PathVariable("id") Integer id);

    @PostMapping("/provider/save")
    public void save(@RequestBody Student student);

    @PutMapping("/provider/update")
    public void update(@RequestBody Student student);

    @DeleteMapping("/provider/deleteById/{id}")
    public void deleteById(@PathVariable("id") Integer id);
}
```

6、FeignHandler

```
package com.southwind.controller;

import com.southwind.entity.Student;
import com.southwind.feign.FeignProviderClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@RestController
@RequestMapping("/feign")
public class FeignHandler {

    @Autowired
    private FeignProviderClient feignProviderClient;

    @GetMapping("/findAll")
    public Collection<Student> findAll(){
        return feignProviderClient.findAll();
    }
```

```
    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id") Integer id){
        return feignProviderClient.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student){
        feignProviderClient.save(student);
    }

    @PutMapping("/update")
    public void update(@RequestBody Student student){
        feignProviderClient.update(student);
    }

    @DeleteMapping("/deleteById/{id}")
    public void deleteById(@PathVariable("id") Integer id){
        feignProviderClient.deleteById(id);
    }
}
```

Hystrix 容错监控机制

什么是微服务的容错机制

提前预设解决方案，系统进行自主调节，遇到问题及时处理

什么是 Hystrix

Netflix

设计原则

- 服务隔离机制
- 服务降级机制
- 熔断机制
- 提供实时的监控和报警功能
- 提供实时的配置修改功能

1、创建 Module，pom.xml

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

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

```
    </dependency>

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

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

2、application.yml

```
server:
  port: 8060
spring:
  application:
    name: hystrix
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
feign:
  hystrix:
    enabled: true
management:
  endpoints:
    web:
      exposure:
        include: 'hystrix.stream'
```

3、创建启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
@EnableCircuitBreaker
public class HystrixApplication {
```

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

4、实体类

```
package com.southwind.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {
    private Integer id;
    private String name;
}
```

5、FeignProviderClient 接口

```
package com.southwind.feign;

import com.southwind.entity.Student;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@FeignClient(value = "provider")
public interface FeignProviderClient {

    @GetMapping("/provider/findAll")
    public Collection<Student> findAll();

    @GetMapping("/provider/findById/{id}")
    public Student findById(@PathVariable("id") Integer id);

    @PostMapping("/provider/save")
    public void save(@RequestBody Student student);

    @PutMapping("/provider/update")
    public void update(@RequestBody Student student);

    @DeleteMapping("/provider/deleteById/{id}")
    public void deleteById(@PathVariable("id") Integer id);
```

```
}
```

6、HystrixHandler

```
package com.southwind.controller;

import com.southwind.entity.Student;
import com.southwind.feign.FeignProviderClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@RestController
@RequestMapping("/hystrix")
public class HystrixHandler {
    @Autowired
    private FeignProviderClient feignProviderClient;

    @GetMapping("/findAll")
    public Collection<Student> findAll(){
        return feignProviderClient.findAll();
    }

    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id") Integer id){
        return feignProviderClient.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student){
        feignProviderClient.save(student);
    }

    @PutMapping("/update")
    public void update(@RequestBody Student student){
        feignProviderClient.update(student);
    }

    @DeleteMapping("/deleteById/{id}")
    public void deleteById(@PathVariable("id") Integer id){
        feignProviderClient.deleteById(id);
    }
}
```

数据监控 URL：http://localhost:8060/actuator/hystrix.stream

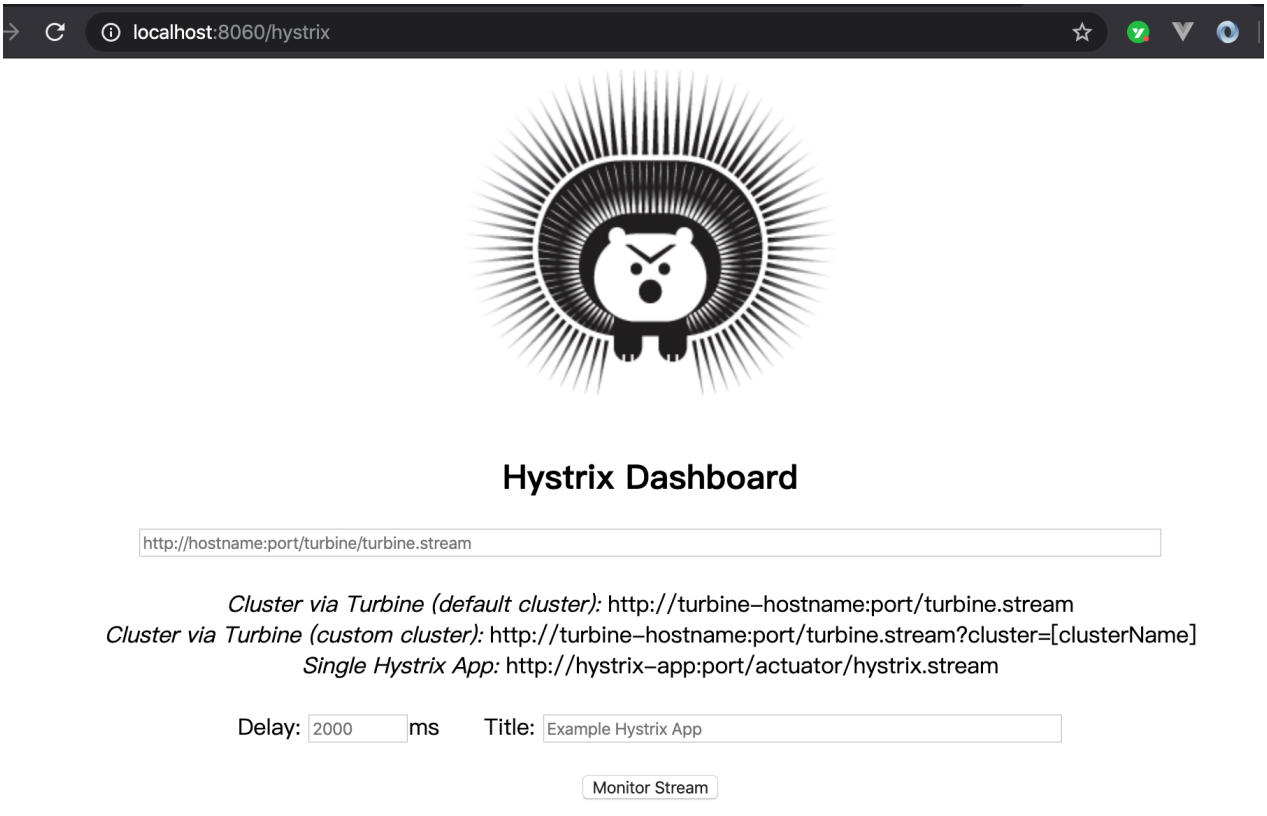
7、添加可视化界面组件

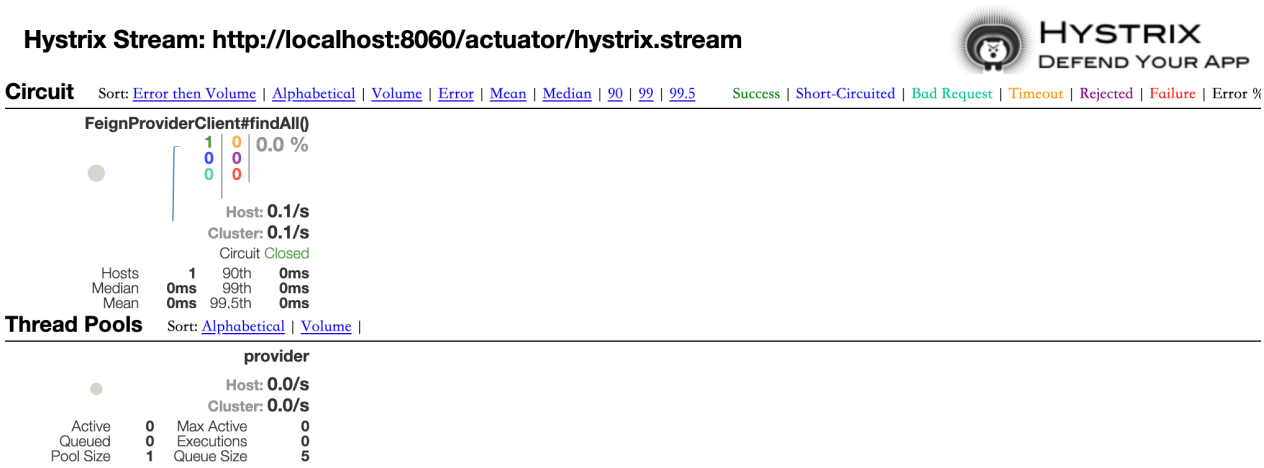
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
```

8、启动类添加相应注解

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import
org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
@EnableCircuitBreaker
@EnableHystrixDashboard
public class HystrixApplication {
    public static void main(String[] args) {
        SpringApplication.run(HystrixApplication.class,args);
    }
}
```

可视化界面 URL：http://localhost:8060/hystrix





Spring Cloud Config 本地配置

本地文件系统

我们可以将微服务的相关配置文件存储到本地文件中，然后让微服务来读取本地配置文件。

创建本地 Config Server

1、创建 Module，pom.xml

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

2、application.yml

```
server:
  port: 8762
spring:
  application:
    name: nativeconfigserver
  profiles:
    active: native
  cloud:
    config:
      server:
        native:
          search-locations: classpath:/shared
```

3、在 resources 路径下创建 shared 文件夹，并在此路径下创建本地配置文件 configclient-dev.yml

```yaml
server:
  port: 8070
foo: foo version 1
```

4、创建启动类

```java
package com.southwind;

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

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

创建客户端读取配置中心的配置文件。

1、创建 Module，pom.xml

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

2、bootstrap.yml

```yaml
spring:
  application:
    name: configclient
  profiles:
    active: dev
  cloud:
    config:
      uri: http://localhost:8762
      fail-fast: true
```

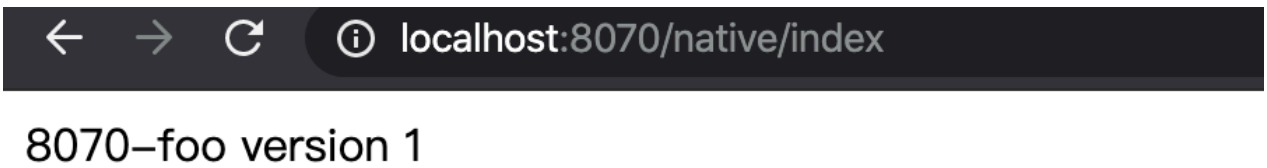
3、创建启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

4、创建 Handler

```
package com.southwind.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/native")
public class NativeConfigHandler {

    @Value("${server.port}")
    private String port;
    @Value("${foo}")
    private String foo;

    @GetMapping("/index")
    public String index(){
        return this.port+"-"+this.foo;
    }
}
```

打开浏览器访问 http://localhost:8070/native/index



Spring Cloud 服务跟踪

Zipkin 实现服务跟踪

什么是 Zipkin

- Zipkin Server
- Zipkin Client

Spring Cloud Sleuth 集成了 Zipkin

创建 Zipkin Server

1、创建 Module，pom.xml

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

    <dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-server</artifactId>
        <version>2.9.4</version>
    </dependency>

    <dependency>
        <groupId>io.zipkin.java</groupId>
        <artifactId>zipkin-autoconfigure-ui</artifactId>
        <version>2.9.4</version>
    </dependency>

</dependencies>
```

2、application.yml

```
server:
  port: 9090
```

3、创建启动类

```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import zipkin.server.internal.EnableZipkinServer;

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

创建 Zipkin Client

1、创建 Module，pom.xml

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

2、application.yml

```
server:
  port: 8090
spring:
  application:
    name: zipkinclient
  sleuth:
    web:
      client:
        enabled: true
    sampler:
      probability: 1.0
  zipkin:
    base-url: http://localhost:9090/
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

3、创建启动类

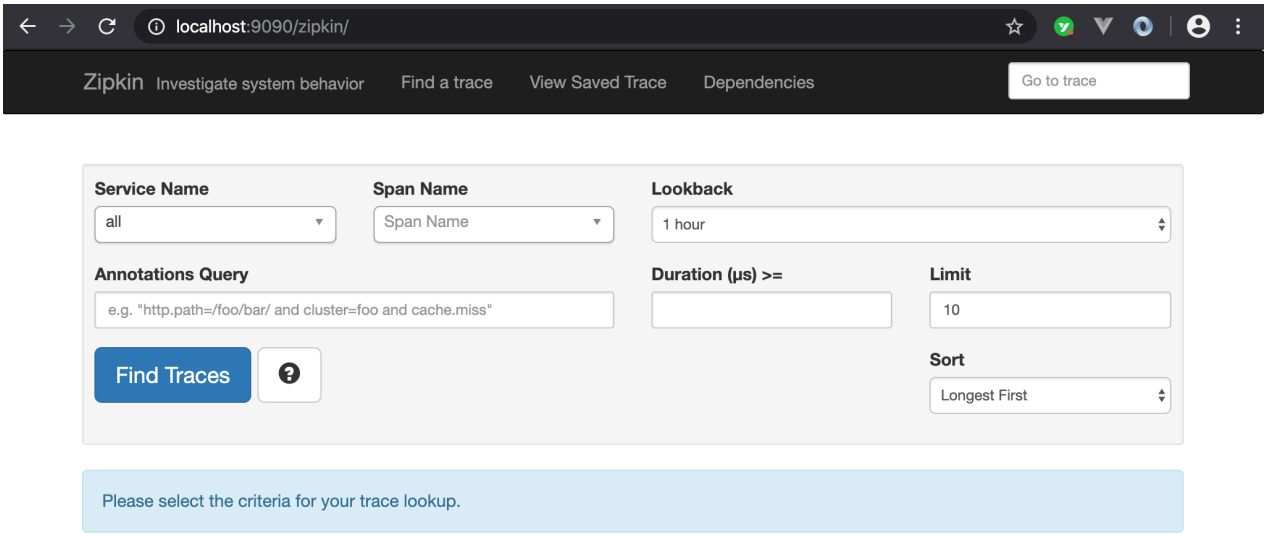
```
package com.southwind;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

4、创建 Handler

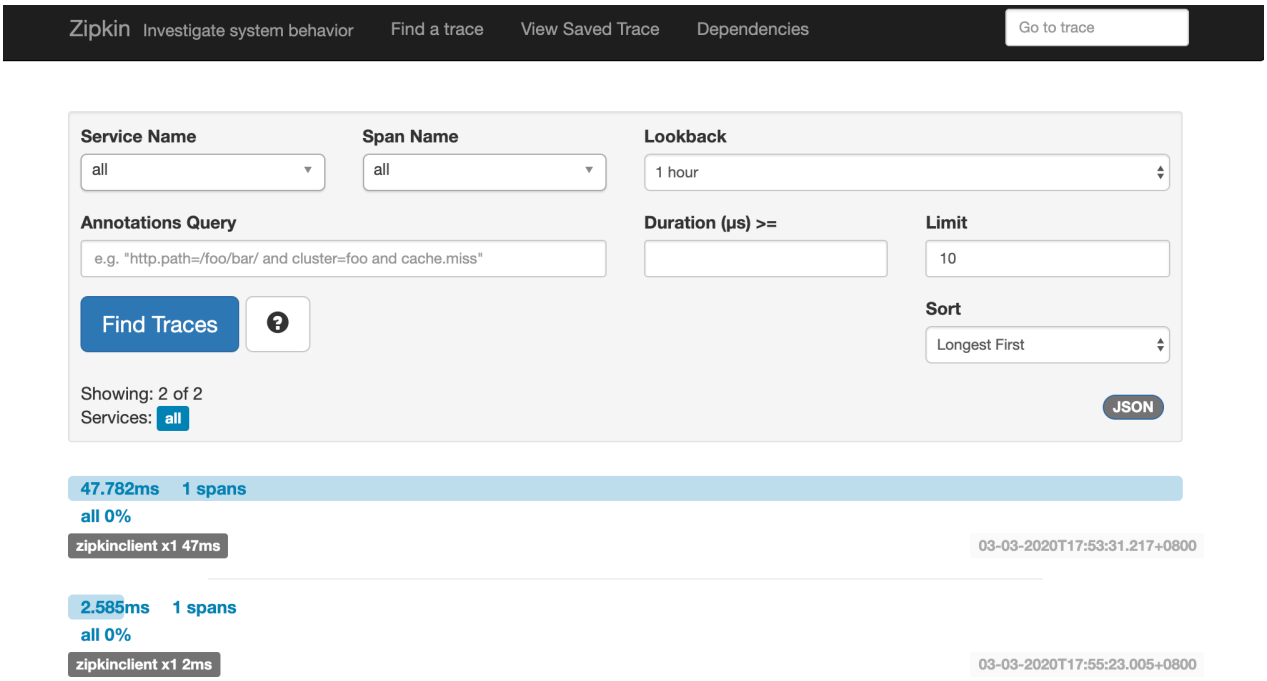
```
package com.southwind.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/zipkin")
public class ZipkinHandler {

    @Value("${server.port}")
    private String port;

    @GetMapping("/index")
    public String index(){
        return this.port;
    }
}
```

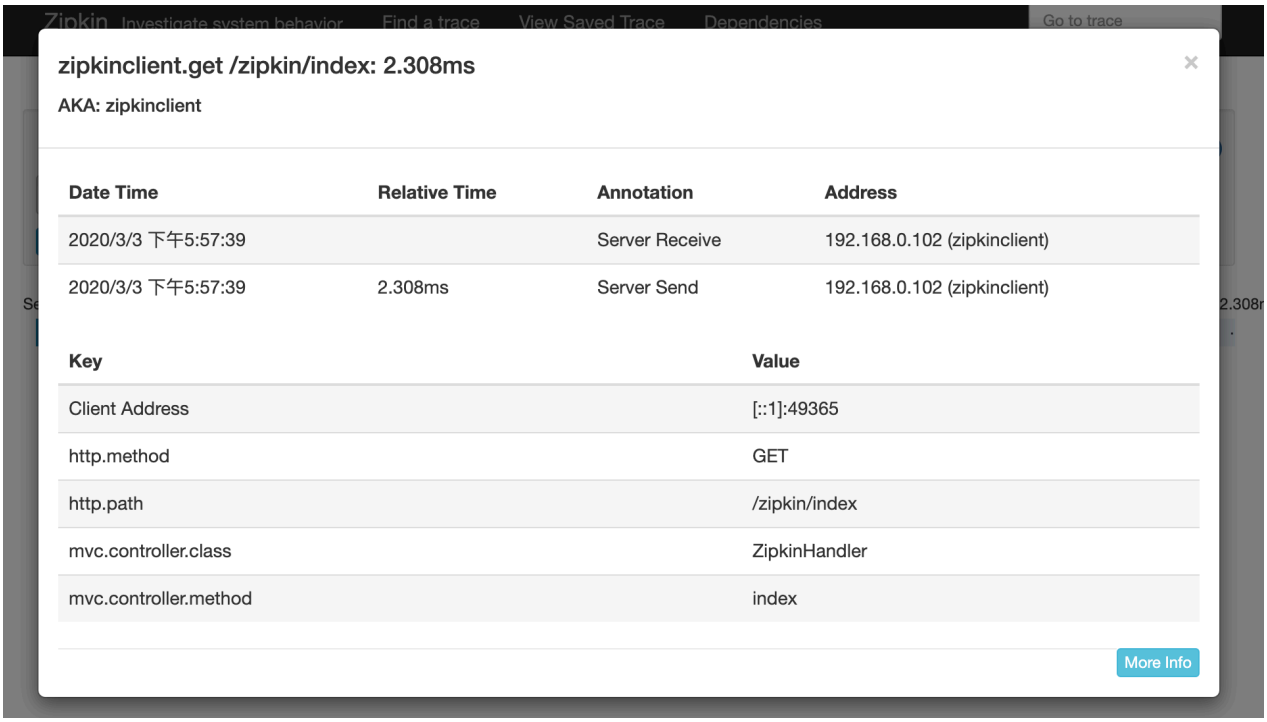
依次启动注册中心、Zipkin Server、Zipkin Client

访问 http://localhost:9090/zipkin/



客户端发起 Zipkin Client 请求，点击 Find Traces，可以看到服务跟踪数据




Zipkin
Investigate system behavior
Find a trace
View Saved Trace
Dependencies
Go to trace
zipkinclient.get /zipkin/index: 2.308ms
AKA: zipkinclient
Date Time
Relative Time
Annotation
Address
2020/3/3 下午5:57:39
Server Receive
192.168.0.102 (zipkinclient)
2020/3/3 下午5:57:39
2.308ms
Server Send
192.168.0.102 (zipkinclient)
Key
Value
Client Address
[::1]:49365
http.method
GET
http.path
/zipkin/index
mvc.controller.class
ZipkinHandler
mvc.controller.method
index
More Info