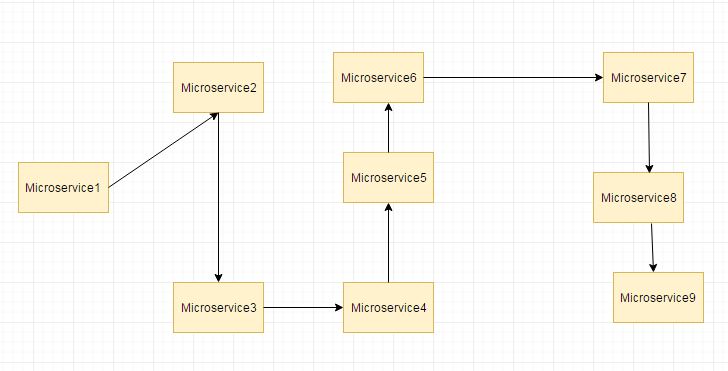
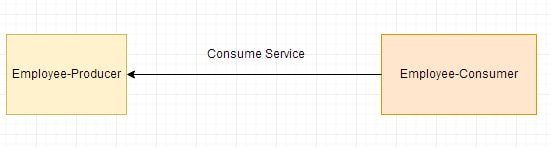
What is Netflix Hystrix? Need for it?

**Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.**  
Usually for systems developed using Microservices architecture, there are many microservices involved. These microservices collaborate with each other.  
Consider the following microservices-



Suppose if the microservice 9 in the above diagram failed, then using the traditional approach we will propagate an exception. But this will still cause the whole system to crash anyways.  
This problem gets more complex as the number of microservices increase. The number of microservices can be as high as 1000. This is where hystrix comes into picture-  
We will be using two features of Hystrix-

* Fallback method
* Circuit Breaker

In this post we will have a look at Hsytrix Fallback method. In next post we will have implement the Hystrix Circuit Breaker.  
  
[In previous posts we had two services-](https://www.javainuse.com/spring/spring_ribbon) employee-consumer consuming the service exposed by the employee-producer.  
The simplified diagram is as below-  
  
  
Now suppose due to some reason the employee-producer exposed service throws an exception. In this case using Hystrix we define a fallback method. This fallback method should have the same return type as the exposed service. In case of exception in the exposed service the fallback method will return some value.

Lets Begin-

Modify the employee-producer pom.xml to include the **spring cloud hystrix starter dependency**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>employee-consumer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.4.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-hystrix</artifactId>**

**</dependency>**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Camden.SR6</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

</project>

Next modify the SpringBootHelloWorldApplication code to add the EnableHystrix syntax

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication

**@EnableCircuitBreaker**

@EnableDiscoveryClient

public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

Next we modify the TestController.java to add the fallback method.

package com.javainuse.controllers;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.javainuse.model.Employee;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController

public class TestController {

@RequestMapping(value = "/employee", method = RequestMethod.GET)

**@HystrixCommand(fallbackMethod = "getDataFallBack")**

public Employee firstPage() {

Employee emp = new Employee();

emp.setName("emp1");

emp.setDesignation("manager");

emp.setEmpId("1");

emp.setSalary(3000);

if(emp.getName().equalsIgnoreCase("emp1"))

throw new RuntimeException();

return emp;

}

**public Employee getDataFallBack() {**

**Employee emp = new Employee();**

**emp.setName("fallback-emp1");**

**emp.setDesignation("fallback-manager");**

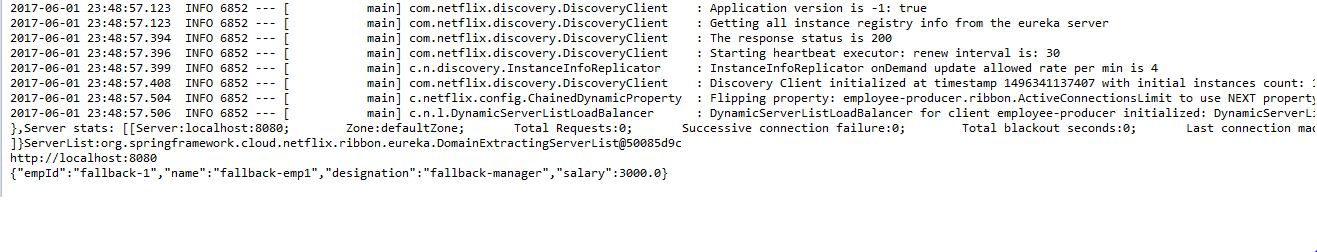
**emp.setEmpId("fallback-1");**

**emp.setSalary(3000);**

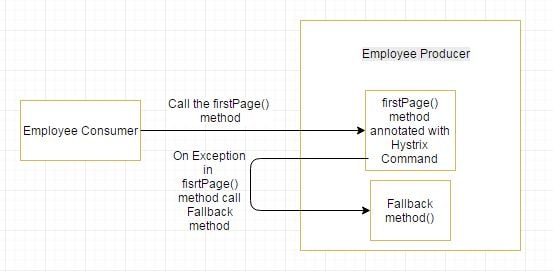
**return emp;**

**}**

}

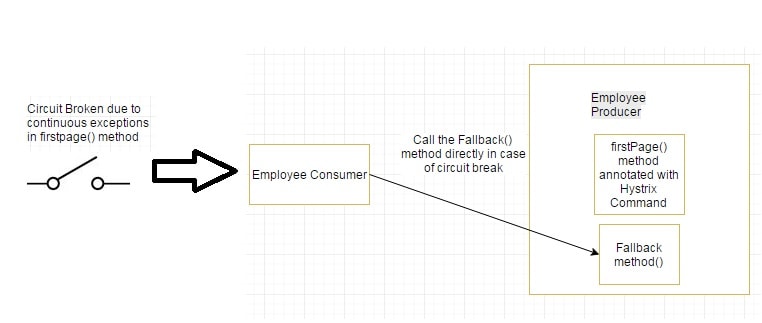
Next start the Eureka server, employee-producer, employee-consumer modules. When we start the employee-consumer, the service exposed by the employee-producer gets called.  
  
**In the employee-consumer console we will see that the Employee object created in the fallback method of TestController class of employee-producer.**

What is the Netflix Hystrix Circuit Breaker Feature? Need for it?

[In previous posts we had two services-](https://www.javainuse.com/spring/spring_hystrix) employee-consumer consuming the service exposed by the employee-producer.  
Due to some reason the employee-producer exposed service throws an exception. In this case using Hystrix we defined a fallback method. In case of exception in the exposed service the fallback method returned some default value.  


If the exceptions keep on occuring in the firstPage method() then the Hystrix circuit will break and the employee consumer will skip the firtsPage method all together and directly call the fallback method.

The purpose of circuit breaker is to give time to the first page method or other methods that the firstpage method might be calling and is causing the exception to recover. It might happen that on less load the issue causing the exceptions have better chance of recovering



Lets Begin-

The employee-producer had the following firstpage method annotated with the hystrix annotation.

package com.javainuse.controllers;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import com.javainuse.model.Employee;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController

public class TestController {

**@RequestMapping(value = "/employee", method = RequestMethod.GET)**

**@HystrixCommand(fallbackMethod = "getDataFallBack")**

public Employee firstPage() {

**System.out.println("Inside firstPage");**

Employee emp = new Employee();

emp.setName("emp1");

emp.setDesignation("manager");

emp.setEmpId("1");

emp.setSalary(3000);

if(emp.getName().equalsIgnoreCase("emp1"))

throw new RuntimeException();

return emp;

}

public Employee getDataFallBack() {

**System.out.println("Inside fallback");**

Employee emp = new Employee();

emp.setName("fallback-emp1");

emp.setDesignation("fallback-manager");

emp.setEmpId("fallback-1");

emp.setSalary(3000);

return emp;

}

}

Previously using employee-consumer we were calling the employee producer only once. Now using for loop we will call it multiple times and check if the circuit trips and the fallback method gets called directly.

package com.javainuse;

import java.io.IOException;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestClientException;

import com.javainuse.controllers.ConsumerControllerClient;

@SpringBootApplication

public class SpringBootHelloWorldApplication {

public static void main(String[] args) throws RestClientException, IOException {

ApplicationContext ctx = SpringApplication.run(

SpringBootHelloWorldApplication.class, args);

ConsumerControllerClient consumerControllerClient=ctx.getBean(ConsumerControllerClient.class);

System.out.println(consumerControllerClient);

**for(int i=0;i<100;i++)**

**consumerControllerClient.getEmployee();**

}

@Bean

public ConsumerControllerClient consumerControllerClient()

{

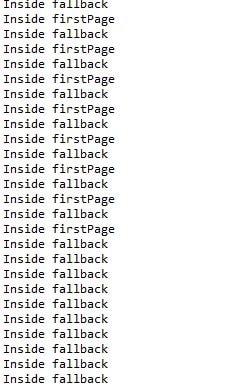
return new ConsumerControllerClient();

}

}

As we had done in previous posts- Start the following Spring Boot Applications-

* eureka-server
* employee-producer
* employee-consumer

On running the employee consumer we get the output in employee-producer module as follows-  
  
We can see that after some exceptions the **fallback method getting called directly and the hystrix annotated method skipped. So the hystrix circuit is open.**