

NETFLIX
EUREKA

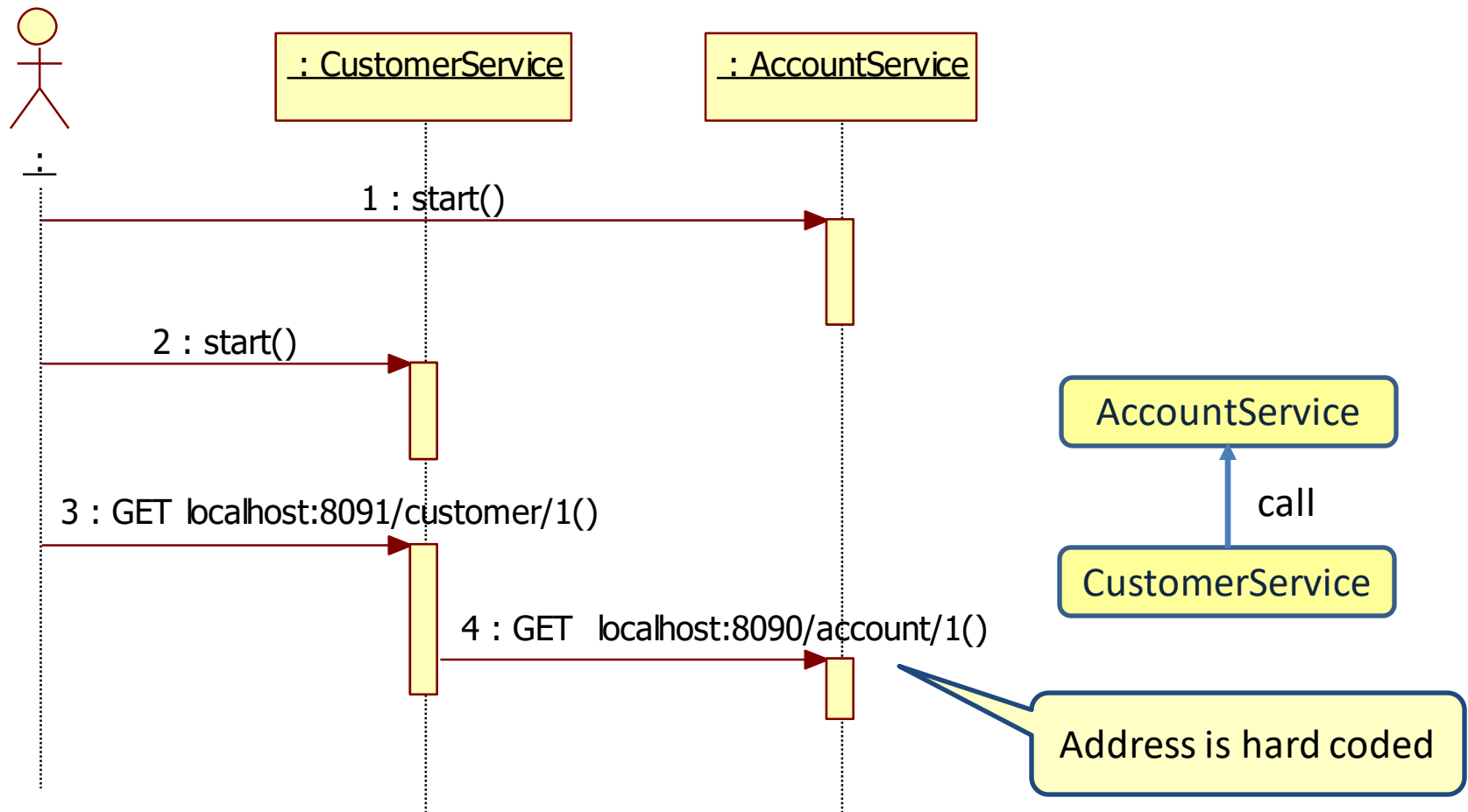
SERVICE REGISTRY: EUREKA



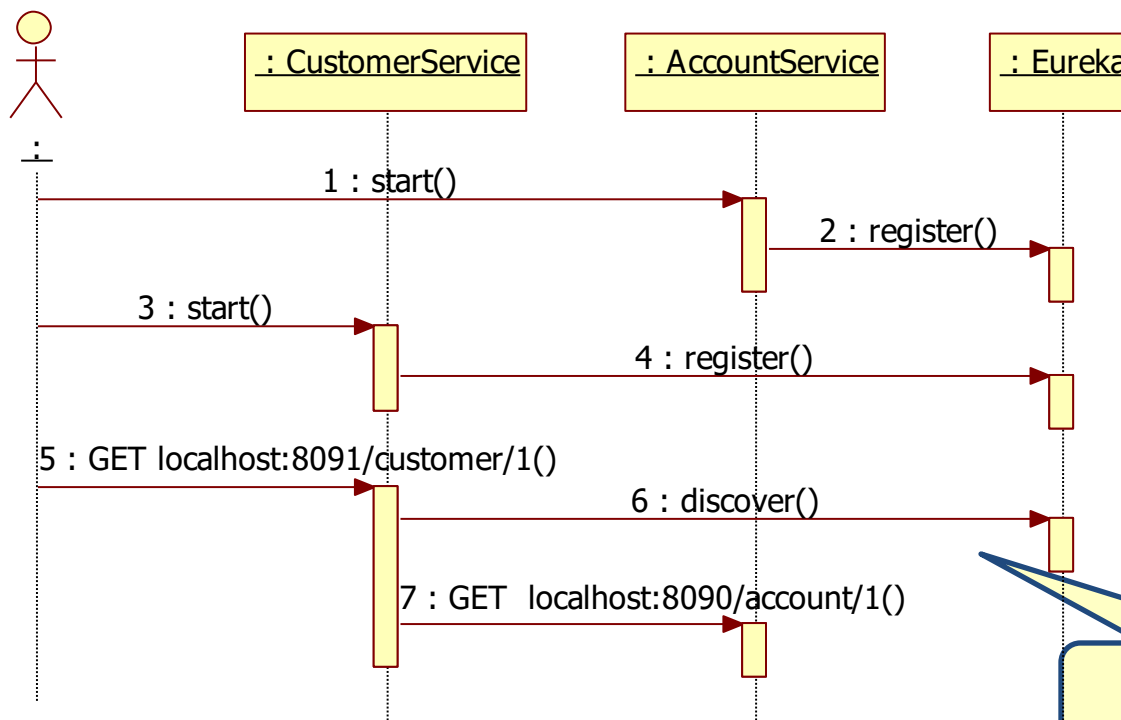
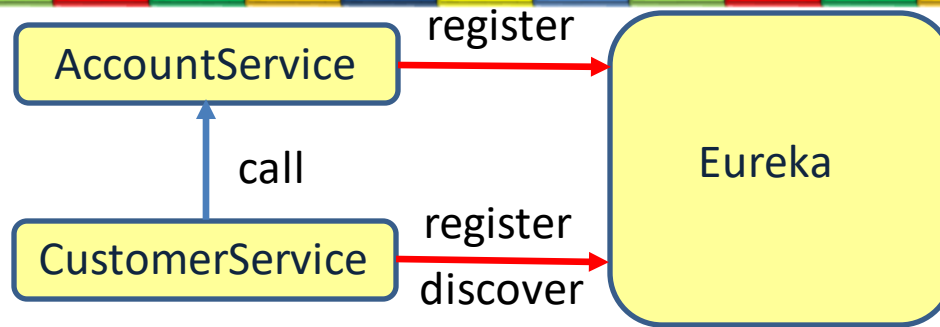
Service Registry

- Like the phone book for microservices
 - Services register themselves with their location and other meta-data
 - Clients can lookup other services
- Netflix Eureka

Without Eureka



Using Eureka



Address is fetched from the registry

Why service registry/discovery?

1. Loosely coupled services

- Service consumers should not know the physical location of service instances.
 - We can easily scale up or scale down service instances

2. Increase application resilience

- If a service instance becomes unhealthy or unavailable, the service discovery engine will remove that instance from the list of available services.



Eureka Server

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

application.yml

```
server:
  port: 8761

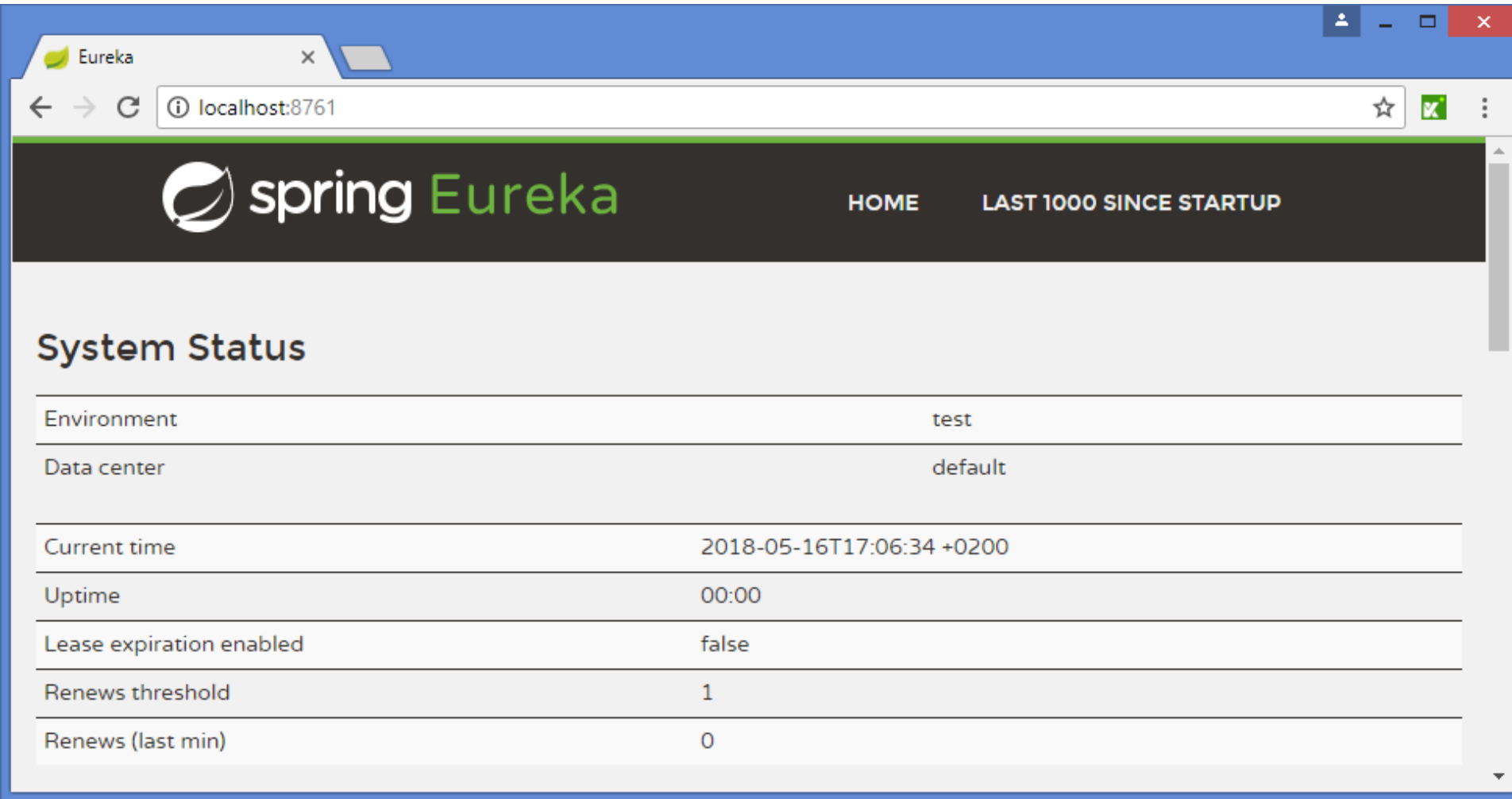
eureka:
  client:
    registerWithEureka: false    #telling the server not to register himself
    fetchRegistry: false
```

bootstrap.yml

```
spring:
  application:
    name: Eureka Server
```



Running Eureka



The screenshot displays the Spring Eureka web application interface. The browser window shows the URL `localhost:8761`. The page header includes the Spring Eureka logo and navigation links for `HOME` and `LAST 1000 SINCE STARTUP`. The main content area is titled `System Status` and contains a table with the following data:

Environment	test
Data center	default
Current time	2018-05-16T17:06:34 +0200
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0



AccountService

```
@SpringBootApplication
@EnableDiscoveryClient
public class AccountServiceApplication {

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

application.yml

```
server:
  port: 8090

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

bootstrap.yml

```
spring:
  application:
    name: AccountService
```


AccountService

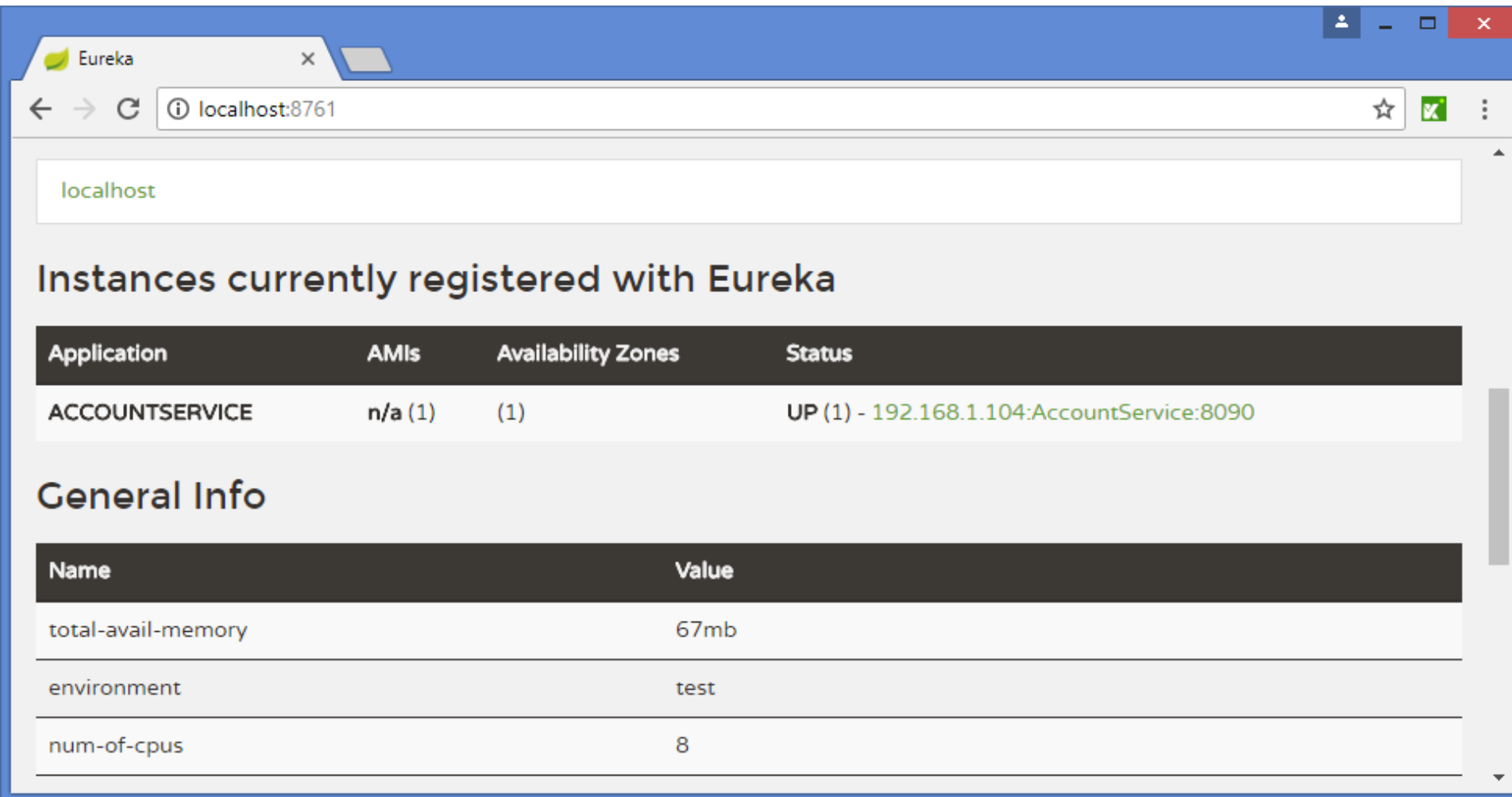


```
@RestController
public class AccountController {
    @RequestMapping("/account/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId) {
        return new Account("1234", "1000.00");
    }
}
```

```
public class Account {
    private String accountNumber;
    private String balance;
    ...
}
```



Running the AccountService



The screenshot shows a web browser window with the Eureka application running at localhost:8761. The page displays a search bar with 'localhost' entered. Below the search bar, the title 'Instances currently registered with Eureka' is followed by a table of registered instances. The table has four columns: Application, AMIs, Availability Zones, and Status. One instance, ACCOUNTSERVICE, is listed with status 'UP (1)' and IP '192.168.1.104:AccountService:8090'. Below the table, the 'General Info' section shows a table with three rows: total-avail-memory (67mb), environment (test), and num-of-cpus (8).

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 192.168.1.104:AccountService:8090

Name	Value
total-avail-memory	67mb
environment	test
num-of-cpus	8



CustomerService

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class AccountServiceApplication {

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

Use Feign

application.yml

```
server:
  port: 8091

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

bootstrap.yml

```
spring:
  application:
    name: CustomerService
```

CustomerService: the controller

```
@RestController
public class CustomerController {
    @Autowired
    AccountFeignClient accountClient;

    @RequestMapping("/customer/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId) {
        Account account = accountClient.getName(customerId);
        return account;
    }

    @FeignClient("AccountService")
    interface AccountFeignClient {
        @RequestMapping("/account/{customerid}")
        public Account getName(@PathVariable("customerid") String customerId);
    }
}
```

Name of the service

Use Feign to access
the AccountService

application.yml

```
server:
  port: 8091
```



Running the CustomerService



Eureka

localhost:8761

localhost

Instances currently registered with Eureka

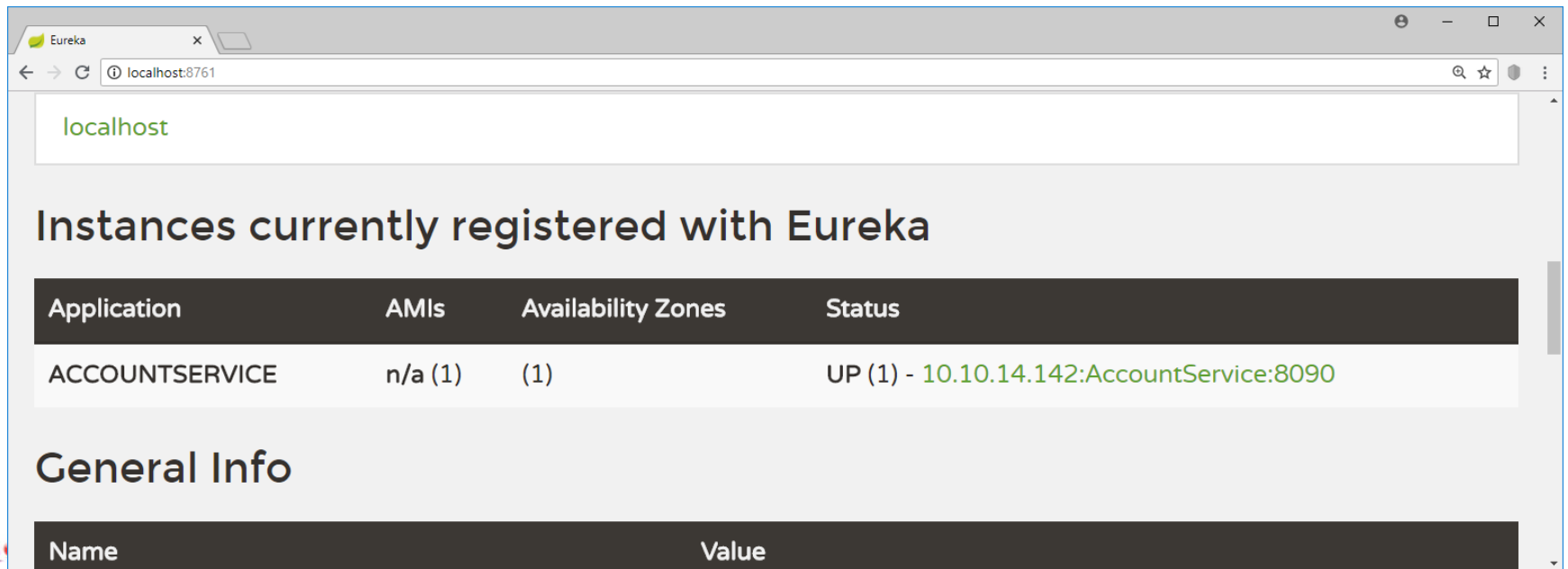
Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:AccountService:8090
CUSTOMERSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:CustomerService:8091

General Info



Stopping the CustomerService

- Eureka monitors the health of registered services.
- If we stop the CustomerService, Eureka will notice that automatically

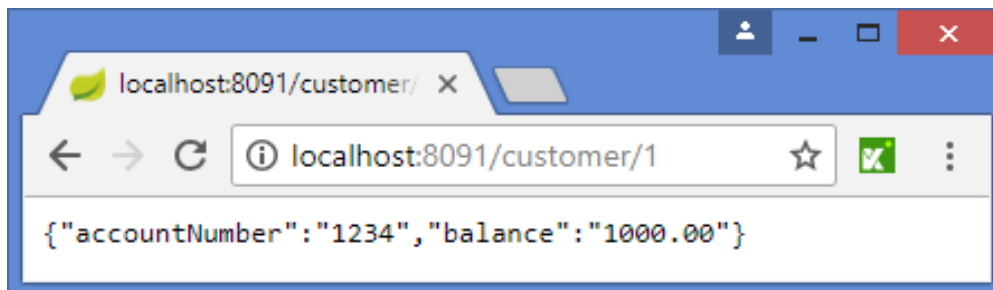
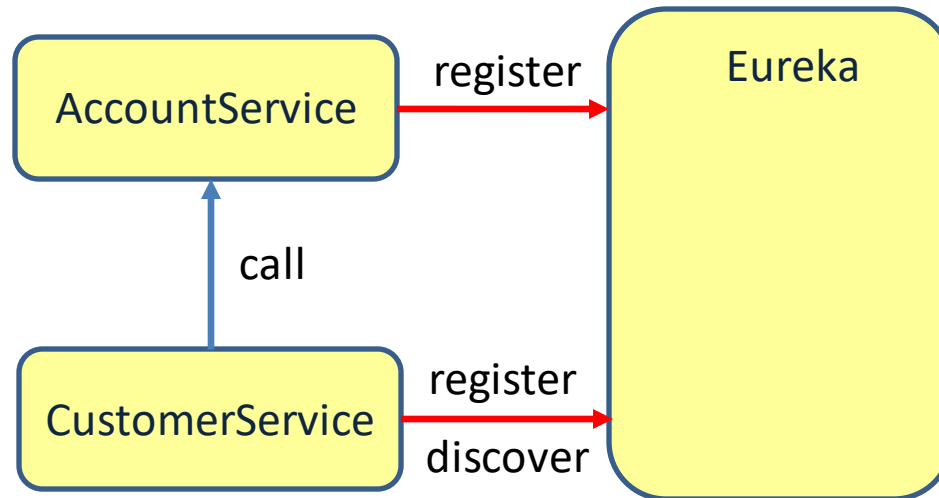
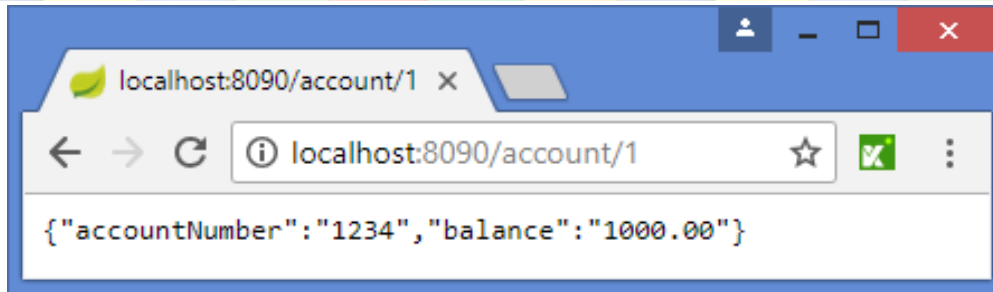


The screenshot shows the Eureka web interface in a browser window. The address bar shows 'localhost:8761'. The page title is 'Eureka'. Below the address bar, there is a search bar with 'localhost' entered. The main content area is titled 'Instances currently registered with Eureka'. It contains a table with the following data:

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:AccountService:8090

Below the table, there is a section titled 'General Info' with a table that has two columns: 'Name' and 'Value'.

Using Eureka



Registering with Eureka

- When a service registers with Eureka, Eureka will wait for 3 successive health checks over the course of 30 seconds before the service becomes available in Eureka



Eureka high availability

- Multiple Eureka servers can be configured as such that they replicate the contents of their registries.

application.yml

```
server:  
  port: 8091  
  
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8761/eureka/
```

This can be a comma separated list of Eureka instances.
If the first instance does not respond, we try the next instance

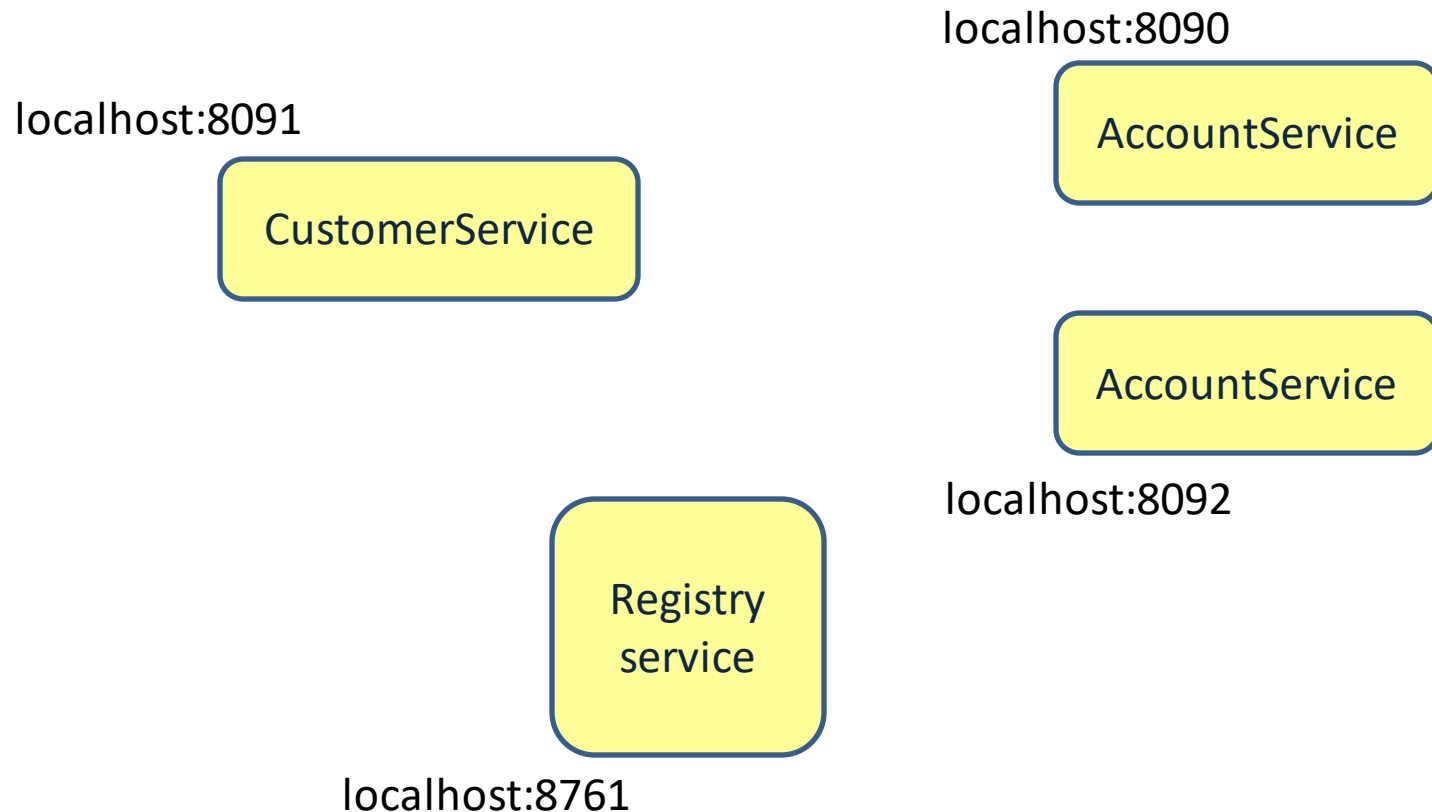


NETFLIX RIBBON

LOAD BALANCING: RIBBON



Running 2 AccountServices using profiles



Spring Profiles

```
@RestController  
@Profile("One")
```

Define a profile

```
public class AccountController1 {  
    @GetMapping("/account/{customerid}")  
    public Account getName(@PathVariable("customerid") String customerId) {  
        System.out.println("getName() on AccountController1 is called");  
        return new Account("1234", "1000.00");  
    }  
}
```

```
@RestController  
@Profile("Two")
```

```
public class AccountController2 {  
    @GetMapping("/account/{customerid}")  
    public Account getName(@PathVariable("customerid") String customerId) {  
        System.out.println("getName() on AccountController2 is called");  
        return new Account("1234", "1000.00");  
    }  
}
```



Start the first instance

The screenshot shows the 'Run Configurations' dialog in Eclipse IDE. The left sidebar lists various projects, with 'AccountServiceProfile' selected. The main panel shows the configuration for 'AccountServiceProfile - AccountServiceApplication'.

Annotations:

- Activate profile "One"**: A yellow callout box pointing to the 'Profile' dropdown menu, which is set to 'One'.
- Set server.port**: A yellow callout box pointing to the 'server.port' property in the 'Override properties' table.

Configuration Details:

- Name:** AccountServiceProfile - AccountServiceApplication
- Spring Boot** tab is active.
- Project:** AccountServiceProfile
- Main type:** accounts.AccountServiceApplication
- Profile:** One
- ☒ Enable debug output
- ☐ Hide from Boot Dash
- ☒ Fast startup
- ☒ Enable JMX Port: 0
- ☒ Enable Life Cycle Management. Termination timeout (ms): 15000
- Override properties:**

property	value
<input checked="" type="checkbox"/> server.port	8090
<input type="checkbox"/>	

Buttons at the bottom: **Run** (highlighted), **Close**, **Revert**, **Apply**.

Start the second instance

Run Configurations

Create, manage, and run configurations

Name: AccountServiceProfile - AccountServiceApplication

Spring Boot (x) Arguments JRE Classpath Source Environment Common

Project: AccountServiceProfile

Main type: accounts.AccountServiceApplication

Profile: Two

☒ Enable debug output ☐ Hide from Boot Dash

☒ Fast startup

☒ Enable JMX Port: 0

☒ Enable Life Cycle Management. Termination timeout (ms): 15000

Override properties:

property	value
<input checked="" type="checkbox"/> server.port	8092
<input type="checkbox"/>	

Revert Apply

Run Close

Filter matched 96 of 272 items

Activate profile "Two"

Set server.port

2 instances of AccountService

localhost:8091

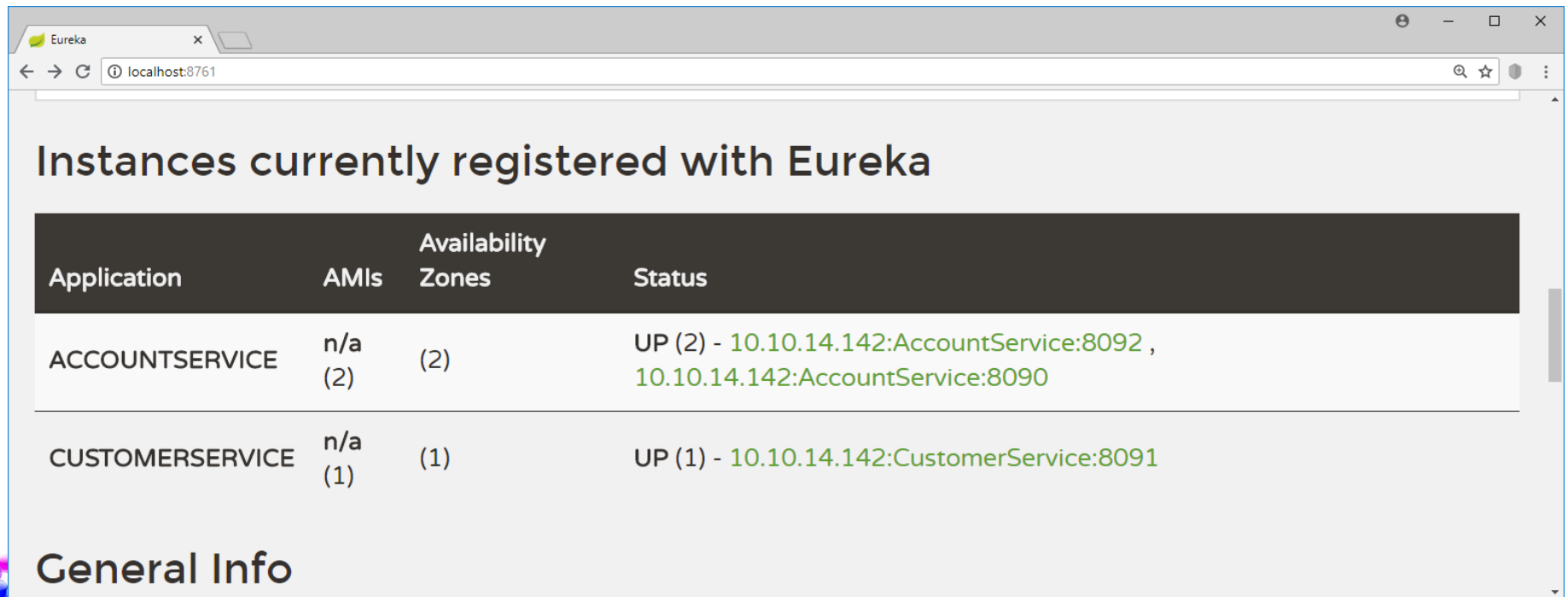
CustomerService

localhost:8090

AccountService

AccountService

localhost:8092

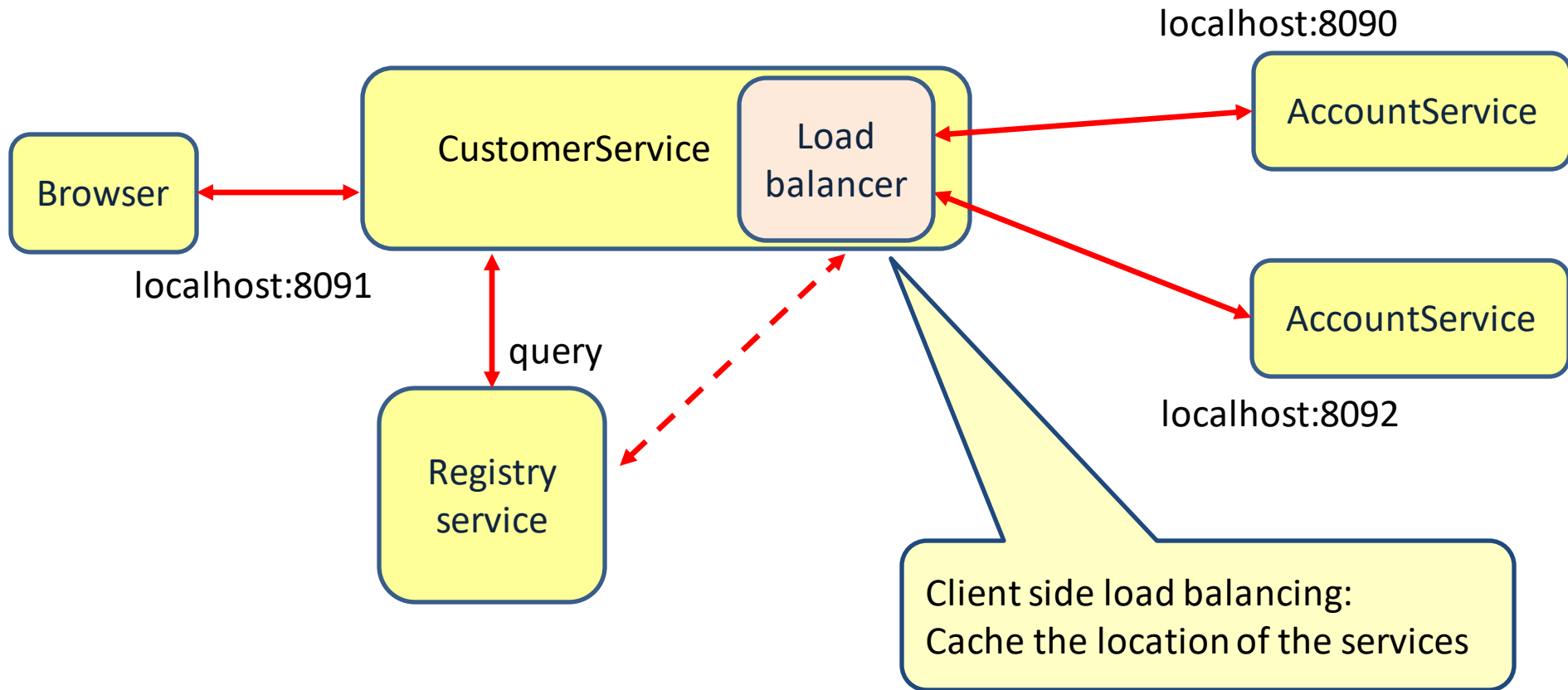


The screenshot shows the Eureka web interface in a browser window. The address bar shows 'localhost:8761'. The main heading is 'Instances currently registered with Eureka'. Below this is a table with the following data:

Application	AMIs	Availability Zones	Status
ACCOUNTSERVICE	n/a (2)	(2)	UP (2) - 10.10.14.142:AccountService:8092 , 10.10.14.142:AccountService:8090
CUSTOMERSERVICE	n/a (1)	(1)	UP (1) - 10.10.14.142:CustomerService:8091

Below the table, there is a section titled 'General Info'.

Load balancer



CustomerService calls AccountService

```
@RestController
public class CustomerController {
    @Autowired
    AccountFeignClient accountClient;

    @RequestMapping("/customer/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId) {
        Account account = accountClient.getName(customerId);
        return account;
    }

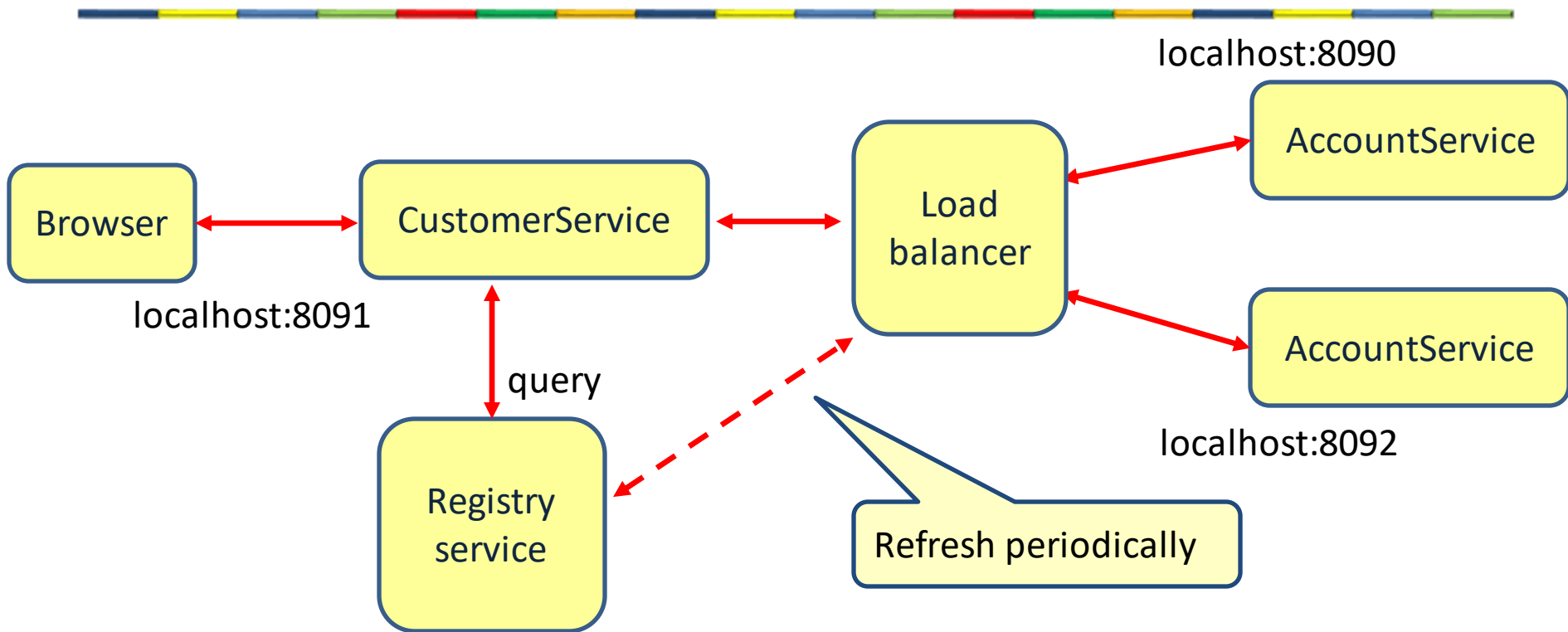
    @FeignClient("AccountService")
    @RibbonClient(name="AccountService")
    interface AccountFeignClient {
        @RequestMapping("/account/{customerid}")
        public Account getName(@PathVariable("customerid") String customerId);
    }
}
```

Use Feign to call another service

Use Ribbon for load balancing



Load balancer



- The load balancer will use Round Robin by default.
- If you stop one instance of AccountService, automatically the other instance will be used.
- If you start the second instance again, it will use Round Robin again.

