单体引用存在的问题

- 随着业务的发展，开发变得越来越复杂。
- 修改、新增某个功能，需要对整个系统进行测试、重新部署。
- 一个模块出现问题，很可能导致整个系统崩溃。
- 多个开发团队同时对数据进行管理，容易产生安全漏铜。
- 各个模块使用同一种技术进行开发，各个模块很难根据实际情况选择更合适的技术框架，局限性很大。
- 模块内容过于复杂，如果员工离职，可能需要很长时间才能完成工作交接。

分布式、集群

集群：一台服务器无法负荷高并发的数据访问量，那么就设置十台服务器一起分担压力，十台不行就设置一百台。很多人干同一件事，来分摊压力。

分布式：将一个复杂问题拆分成若干个简单的小问题，将一个大型的项目架构拆分成若干个微服务来协同完成。将一个庞大的工作分成若干个小步骤，分别由不同的人完成这些小步骤，最终将所有的结果进行整合实现大的需求。

服务治理的核心由三部分组成：服务提供者、服务消费者、注册中心。

在分布式系统架构中，每个微服务在启动时，将自己的信息存储在注册中心，叫做微服务注册。

服务消费者从注册中心获取服务提供者的网络信息，通过该信息调用服务，叫做服务发现。

Spring Cloud 的服务治理使用 Eureka 来实现，Eureka 是 Netflix 开源的基于 REST 的服务治理解决方案，Spring Cloud 集成了 Eureka，提供服务注册和服务发现的功能，可以和基于 Spring Boot 搭建的微服务应用轻松完成整合，开箱即用。

Spring Cloud Eureka

- Eureka Server，注册中心。
- Eureka Client，所有要进行注册的微服务通过 Eureka Client 连接到 Eureka Server，完成注册。

代码实现

- 创建父工程，pom.xml

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.7.RELEASE</version>
</parent>

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

<dependencyManagement>
    <dependencies>
        <!-- 添加 Spring Cloud 依赖 -->
        <dependency>
```

```
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Finchley.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

创建 Eureka Server

- 在父工程下创建 Eureka Server 模块，pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        <version>2.0.2.RELEASE</version>
    </dependency>
</dependencies>
```

- 创建配置文件，application.yml，添加 Eureka Server 相关配置

```
server:
  port: 8761
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://localhost:8761/eureka/
```

> 属性说明

`server.port`：当前 Eureka Server 服务端口。

`eureka.client.register-with-eureka`：是否将当前的 Eureka Server 服务作为客户端进行注册。

`eureka.client.fetch-registry`：是否获取其他 Eureka Server 服务的数据。

`eureka.client.service-url.defaultZone`：注册中心的访问地址。

- 创建启动类

```
package com.naruto;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

```
}
```

> 注解说明

`@SpringBootApplication`：声明该类是 Spring Boot 服务的入口

`@EnableEurekaServer`：声明该类是一个 Eureka Server 服务，提供服务注册和服务发现功能，即注册中心

- 如果是 JDK 9 的话，需要在父工程 pom.xml 添加相应的 jar 包

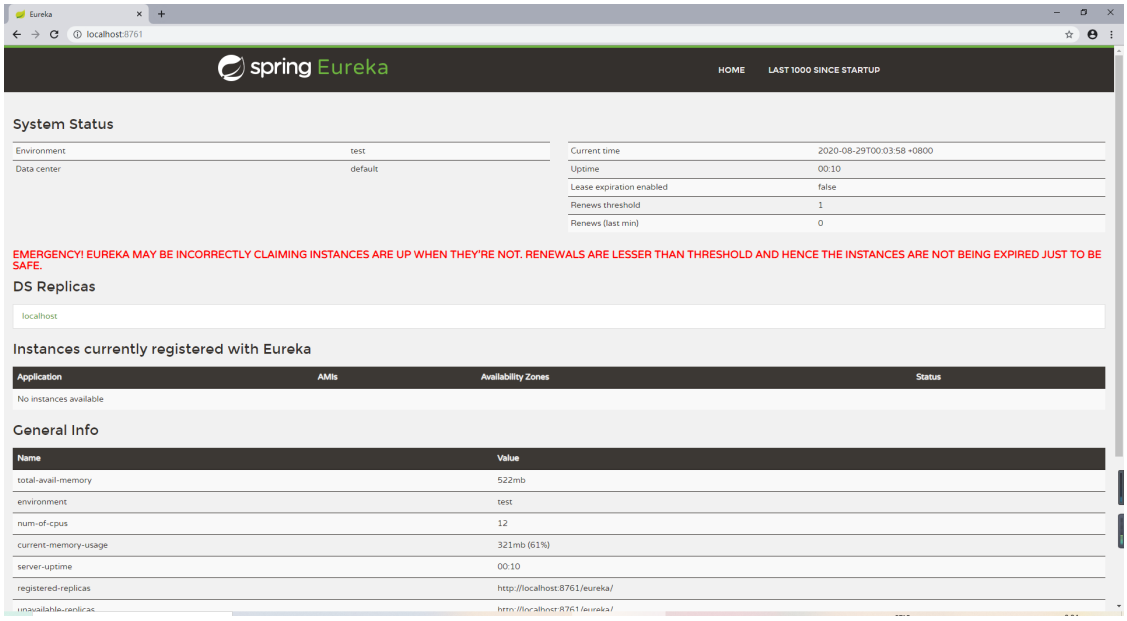
```
<!-- 解决 JDK 9 以上没有 JAXB API 的问题 -->
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>

<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>2.3.0</version>
</dependency>

<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-core</artifactId>
    <version>2.3.0</version>
</dependency>

<dependency>
    <groupId>javax.activation</groupId>
    <artifactId>activation</artifactId>
    <version>1.1.1</version>
</dependency>
```

- 启动 Eureka Server 程序，访问 localhost:8761



创建 Eureka Client

- 在父工程下创建 Eureka Client 模块，pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        <version>2.0.2.RELEASE</version>
    </dependency>
</dependencies>
```

- 创建配置文件，application.yml，添加 Eureka Client 相关配置

```yaml
server:
  port: 8010
spring:
  application:
    name: provider
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka//
  instance:
    prefer-ip-address: true
```

> 属性说明

`spring.application.name`：当前服务注册在 Eureka Server 上的名称

`eureka.client.service-url.defaultZone`：注册中心的访问地址

`eureka.instance.prefer-ip-address`：是否将当前服务的 IP 注册到 Eureka Server

- 创建启动类

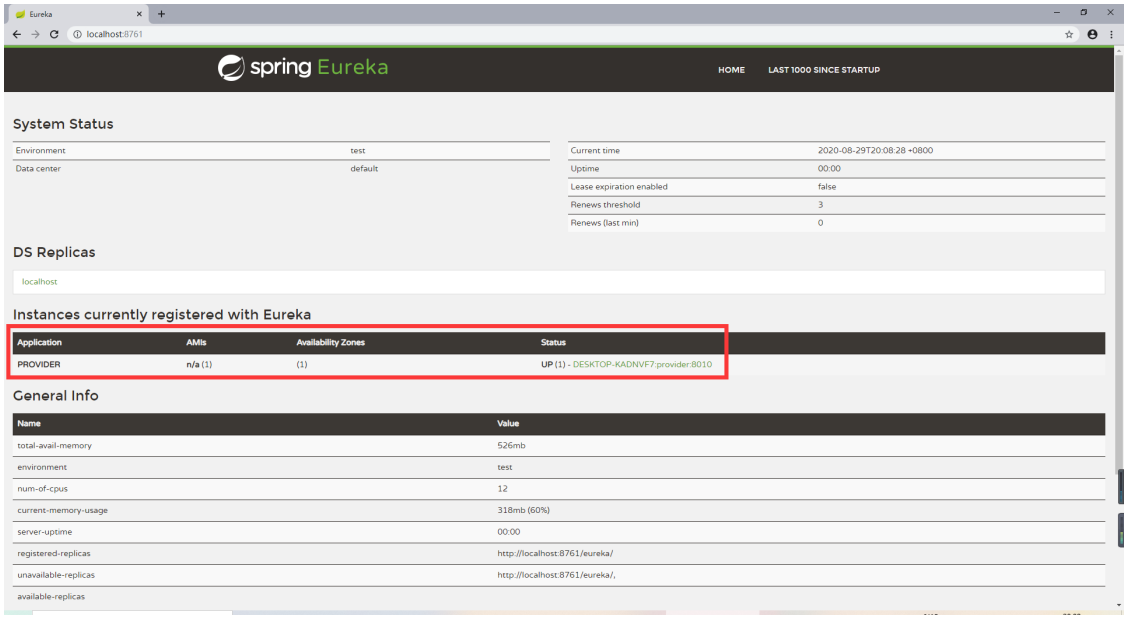
```java
package com.naruto;

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

@SpringBootApplication
public class ProviderApplication {

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

- 启动 Eureka Client 程序，访问 localhost:8761



实现一个简单的增删改查

- 在父工程添加 lombok 插件简化实体类的开发

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.6</version>
    <scope>provided</scope>
</dependency>
```

- 在 Eureka Client 工程下创建 entity 实体类

```
package com.naruto.entity;
```

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student implements Serializable {

```
    private Long id;
    private String name;
    private Integer age;
```

}

> 注解说明
>
> `@Data`：生成 Getter Setter 方法
>
> `@AllArgsConstructor`：生成有参构造函数

`@NoArgsConstructor`：生成无参构造函数

- 在 Eureka Client 工程下创建 Repository，即 Dao 层组件

~~~java
package com.naruto.repository;

import com.naruto.entity.Student;

import java.util.Collection;

public interface StudentRepository {

    public Student findById(Long id);

    public Collection<Student> findAll();

    public void saveOrUpdate(Student student);

    public void deleteById(Long id);
}
~~~

- 因没有连接数据库，所以需要创建一个实体类初始化数据

```java
package com.naruto.repository.impl;

import com.naruto.entity.Student;
import com.naruto.repository.StudentRepository;
import org.springframework.stereotype.Repository;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

@Repository
public class StudentRepositoryImpl implements StudentRepository {

    private static Map<Long, Student> studentMap;

    static {
        studentMap = new HashMap<>();
        studentMap.put(1L, new Student(1L, "鸣人", 16));
        studentMap.put(2L, new Student(2L, "佐助", 17));
        studentMap.put(3L, new Student(3L, "小樱", 18));
    }

    @Override
    public Student findById(Long id) {
        return studentMap.get(id);
    }

    @Override
    public Collection<Student> findAll() {
        return studentMap.values();
    }

    @Override
```

```
    public void saveOrUpdate(Student student) {
        return studentMap.put(student.getId(), student);
    }

    @Override
    public void deleteById(Long id) {
        studentMap.remove(id);
    }
}
```

- 在 Eureka Client 工程下创建 Handler，即 Controller 层控制器

```
package com.naruto.handler;

import com.naruto.entity.Student;
import com.naruto.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@RestController
@RequestMapping("/student")
public class StudentHandler {

    @Autowired
    private StudentRepository studentRepository;

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

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

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

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

    @DeleteMapping("/delete/{id}")
    public void deleteById(@PathVariable("id") Long id) {
        studentRepository.deleteById(id);
    }
}
```

- 启动 Eureka Client 程序测试接口

RestTemplate 的使用

- 什么是 RestTemplate

  RestTemplate 是 Spring 框架提供的基于 REST 的服务组件，底层对于 HTTP 请求及响应进行了封装，提供了很多访问 REST 服务的方法，可以简化代码开发。

- 如何使用 RestTemplate
  - 创建 Maven 工程，pom.xml
  - 创建启动类

```
package com.naruto;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class RestTemplateApplication {

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

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

  - 创建 entity 实体类

```
package com.naruto.entity;

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

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student implements Serializable {

    private Long id;
    private String name;
    private Integer age;
}
```

  - 创建 Handler，即 Controller 层控制器

```
package com.naruto.handler;

import com.naruto.entity.Student;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.client.RestTemplate;
```

```
import java.util.Collection;

@RestController
@RequestMapping("/rest")
public class RestHandler {

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id") long id) {
        return
restTemplate.getForEntity("http://localhost:8010/student/findById/{id}"
, Student.class, id).getBody();
    }

    @GetMapping("/findAll")
    public Collection<Student> findAll() {
        return
restTemplate.getForEntity("http://localhost:8010/student/findAll",
Collection.class).getBody();
    }

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

 restTemplate.postForEntity("http://localhost:8010/student/save",
student, null).getBody();
    }

    @PutMapping("/update")
    public void update(@RequestBody Student student) {
        restTemplate.put("http://localhost:8010/student/update",
student);
    }

    @DeleteMapping("delete/{id}")
    public void deleteById(@PathVariable("id") long id) {

 restTemplate.delete("http://localhost:8010/student/delete/{id}", id);
    }
}
```

- 启动 RestTemplate 程序测试接口