最近整理了一下自己所掌握的技术栈，发现自己所掌握的技术实在是太少了，无非就是使用几个简单的技术，搭建一个项目环境，然后进行编写代码。虽然现在自己创建的项目技术选型都是固定的那几种技术，如springboot、shiro、vue等，但是自己所能够做的东西都是比较简单，基本上都是增删改查的操作，对这些技术都没有太过深入的研究，只会比较基础的使用。在各大网站上搜了下比较出名的企业面试题，发现那些问题都好难啊，好大一部分都回答不出来，有的是考基础，有的则是考对流行技术的使用。

所以为了增加点自己的知识储备，打算在网络上找些视频教程继续学习一下内容。这次我找的是spring cloud的视频教程。现在互联网发展迅猛，一个网站系统每日的工作量是巨大的，将项目分开部署、搭建集群是现在最通用也是最有效降低服务器负担的方法。spring cloud正是为了微服务所诞生的技术，之前在github搜索项目时，也发现了好多项目都是选用了这项技术，所以这也是我选这门技术学习的原因之一。

软件工程专业，想要技术有所提升，多敲代码是难免的一项步骤。视频的第一节课，老师对这次视频教程进行简单概述，要求我们在看同时，也要跟着视频敲代码，巩固学习。并且老师在视频的最后说了一句话“不在能知，乃在能行。知行合一，方能进步”。

学习一项技能之前，必须要先理解其概念。理论掌握好了，学习与使用便会更加顺手。老师在开头先是介绍了微服务架构的理念。分久必合合久必分，微服务架构提倡将单一应用程序划分成一组小的服务，并且服务与服务之间互相协调、互相配合。

之前认为spring cloud像是阿里巴巴的dubbo框架一样，是一项RPC的技术框架。但是经过老师的介绍，发现原来之前对spring cloud的了解大相径庭。Spring cloud是一系列框架的有序集合，它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。

Spring cloud它将各大产商较为成熟的框架结合起来，其中包含了大概二十种左右的技术。虽然里面已经大部分技术已经过时并且停止更新，但是还是需要对其有的技术进行了解，如果在之后如果接触到，能够对其有一个大致的印象。虽然一些技术已经被抛弃，但是他们的设计理念大部分得到保留，之后新出现的技术，可能遵循的也是一样的架构。比如RPC的设计，遵守的都是消费者、服务提供者和注册中心之间的相互通信。像是冯诺依曼的计算机体系，他是早在一九四五年提出，但是到现在计算机依然遵守的是如此的设计架构。所以学习知识是不会无用，总是会有其发挥作用的地方。

今天学习的主要是服务的注册与发现，老师在这个系列中会介绍eureka、zookepper、consul和nacos的使用。虽然需要学习的技术有好几种，并且在一个系统的开发中只会选取一种进行使用，但是对于这些技术，必须都要有所掌握与了解，权衡各项技术中的优点与缺点，选出最适合当前项目的技术。并且掌握的技术越多，在今后的工作中，对公司所选择的技术能够有所了解与使用，有利于今后的开发工作。

老师先是不使用框架进行微服务的调用以此进行作对比，所使用的是RestTemplate类中的方法，直接填写微服务的地址进行调用。使用此方法进行调用，开发时必须知道所调用服务的地址，开发较为复杂。

约定大于配置，配置大于编码。在对项目导入一项技术时，对于环境的搭建是至关重要的。环境搭建错误，后面的编码都将不起作用。老师在导入eureka配置时，打开了思维导图文件，其中记录了导入框架时的每一步操作。并且按步骤顺序罗列好，对于该环境的搭建有了十分直观的感受。

经过技术的迭代更新，eureka从之前导入的同一个jar包，变成服务端与客户端的jar包分开导入，对于两端有了更明显的划分。将eureka搭建完成后，创建了多个同类的服务提供者注册进eureka的服务端中，体验到消费者对于服务的调用更加的方便，但是对于单点故障的原因还没有得到很好的解决。

今天继续学习服务的注册与发现中的zookeeper和consul框架的使用。类似功能的框架感觉大部分都是换汤不换药。底层原理都是相似的。Eureka、zookeeper和consul他们都是充当服务注册中心的角色，使用步骤基本一致，服务提供者注册，消费者获取使用。

在搭建成功eureka之后，接下来是搭建zookeeper的环境，在两者切换时，只需要替换一个专属jar包、修改application.yml文件和更改启动类的一个注解便能够生效使用。Consul也是与前两者类似，业务代码不需要变更，只需要更改配置便能够进行替换。

最后学习了这三个框架的不同点。老师先是介绍了CAP的核心理论，一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求。C表示一致性，A表示可用性，P表示分区容错性。Eureka、zookeeper和consul他们根据CAP架构可以分成CP、与AP。Zookeeper和consul划分为CP，使用他们作为注册中心时，一旦服务提供方不可用时，就直接将其剔除，不会对其进行保留。Eureka划分到AP，因为eureka默认开启自我保护机制，如果服务提供方出错时，eureka会对其进行保留一段时间，期待之后能够继续使用。

不能同时满足CAP的原因是，可能机器之间难免会出现不可能避免的故障，如网络异常等情况，系统A的数据不能同步到系统B，导致B的数据是异常的。如果需要保证B系统可用就不能保证数据一致性，剔除B系统便会失去可用性的特点。

学习完服务的注册与发现后，接下来便是学习负载均衡工具的使用。这次所学习的工具是Spring cloud ribbon，ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具。简单地说就是，Ribbon主要功能是提供客户端的软件负载均衡算法和服务调用。Ribbon客户端组件提供了一系列完善的配置如连接超时，重试等。项目在使用他时，只需要在application.yml中编写相应的配置便能够生效。Ribbon会找出服务提供方的所有机器，然后自动的执行某项规则，如随机连接、简单轮询等自带的七种规则去链接这些机器。同样是具有负载均衡功能的工具是nginx，ribbon与nginx的最大区别是，ribbon是在客户端进行负载均衡，而nginx是在服务端所进行的。Nginx会拦截所有相应的请求，然后转发到对应的机器上，客户端只需要访问nginx就行。而ribbon是在所有的服务方地址缓存进自己的jvm虚拟机，然后通过自己的计算，自行访问相应的机器。

工具不仅仅只会是去用，还需要去了解他的原理，了解它的实现。老师打开ribbon的源码，让我们去了解Ribbon负载规则算法的实现。老师让我们先去看轮询算法的实现，基本原理便是，客户端去注册中心获取到所有可用的服务提供方地址，然后存进一个集合中进行保存，之后根据请求，每次请求访问次数就加1，然后集合总数除以访问数求余，最后根据结果去集合中get(index)获取目标主机。其中在RoundRobinRule类中，记录访问次数的变量时，它使用了AtomicInteger原子类，并且使用自旋锁的方式，确保了每次访问该变量的线程安全。

今天学习的是服务的调用OpenFeign的使用。Openfeign是spring cloud在feign的基础上支持了spring mvc的注解，使之使用起来更加的方便。Feign旨在使编写Java Http客户端变得更加容易。在之前的学习中都是使用Ribbon+restTenplate进行使用，利用restTenplate对http请求进行封装处理，形成了一套模板化的调用方法。但是在实际开发环境中，由于对服务依赖的调用可能不止一处，往往一个接口会被多处调用，所以通常都会针对每个微服务自行封装一些客户端类来包装这些依赖服务的调用。所以，在Feign在此基础上做了进一步封装，由它来帮助我们定义和实现依赖服务接口的定义，在feign的实现下，我们只需要创建一个接口并使用注解的方式来配置它，即可完成对服务提供方的接口绑定，简化了使用spring cloud ribbon时，自动封装服务调用客户端的开发量。

Feign在定义调用接口时，与Mybatis定义的Mapper接口十分类似，都是编写接口方法，填入相应的注解，框架便会自己去调用内部的方法。Feign是在服务调用方进行定义使用，在使用时，需要在启动类上使用EnableFeignClients注解，使Feign能够在项目中进行生效。并且在定义Feign的接口时，方法与调用方controller层的接口方法完全一致。使开发时，提高了面向对象编程的理念。Feign还提供了超时控制与日志记录的功能，更加完善了feign在使用时的调用逻辑，避免长时间等待，影响程序的正常运行。