什么是微服务

微服务是一种软件开发架构，网上介绍微服务的文章非常多，这里推荐两篇知乎的贴子（通过搜索引擎，你可以很容易找到更多资料）：

- 微服务架构是什么？
- 微服务（Microservices）架构？

既然网上有那么多资料，为啥还要谈微服务？还不是因为吃得太饱，闲的。**有事儿**谁还写这个玩意儿？

其实也不算很闲，因此我尽量把文章写得浅显易懂。下面我们以**选课系统**为例，说明微服务是一种什么架构。

史前时代的软件开发

所谓史前时代是指上个世纪90年代末（互联网浪潮开始）到本世纪前5年（2005年）左右。

早期（90年度末），他们会使用PHP（世界上最好的语言）、ASP、Perl或者CGI，极少数会使用Java，根本不会有人使用Django、Flask、Ruby on rails或者Node.js（根本就没有这些玩意好吗）。那真是一个痛苦的时代，你根本无法想象开发淘宝一样的大型网站是什么概念（马爸爸的第一版阿里巴巴是PHP开发的，好在很快马爸爸就有了第一桶金，程序也用Java重新开发）。那个时代的网站开发，是真的在做网站，因为高级的东西没法做，懂HTML就已经很牛了。这个时代就是传说中的**动态网页架构**时代（名字我取的），做网站等同于做网页，稍微做复杂一点的逻辑就非常痛苦。

2000年左右J2EE（现在叫Java EE）的发布是一个分水岭。Java EE开始提供一系列的开发技术比如Servlet、JSP、EJB。再后来，大家意识到只做网页搞不定复杂应用，于是开始提出了**MVC Web框架**、**ORM映射**、**IOC容器**，之类的玩意。2003年左右，统治现在Web开发领域的SSH（Struts Spring Hibernate）和SSM（Struts Spring Mybatis）就已经出现，经过后面10多年的完善，最终形成了SpringBoot+Mybatis的技术组合。当然还有其他技术，比如云计算、容器化部署等等，在此不再一一叙述。总之，现在的程序员真是非常幸福，不需要太多的技能，只需要掌握几种软件框架的使用，就能轻松领取福报（996是小福报年薪20万起，猝死是大福报赔偿100万起）。

以上是技术方面的发展，从软件架构上说，史前时代的软件都有一个共同的特点：**单体应用**。如果这个时代的程序员要开发**选课系统**会怎么做？

他们会建一个源代码仓库，把html、css、Java（或者PHP、ASP）代码放在一起。程序开发好之后，源代码编译打包成一个应用（一个压缩包），然后部署到用户的服务器上（U盘拷贝过去，解压）。

这种开发模式有什么问题吗？没啥问题。真的没啥问题——只要客服不要求开发手机App和微信小程序。

单体应用的缺点

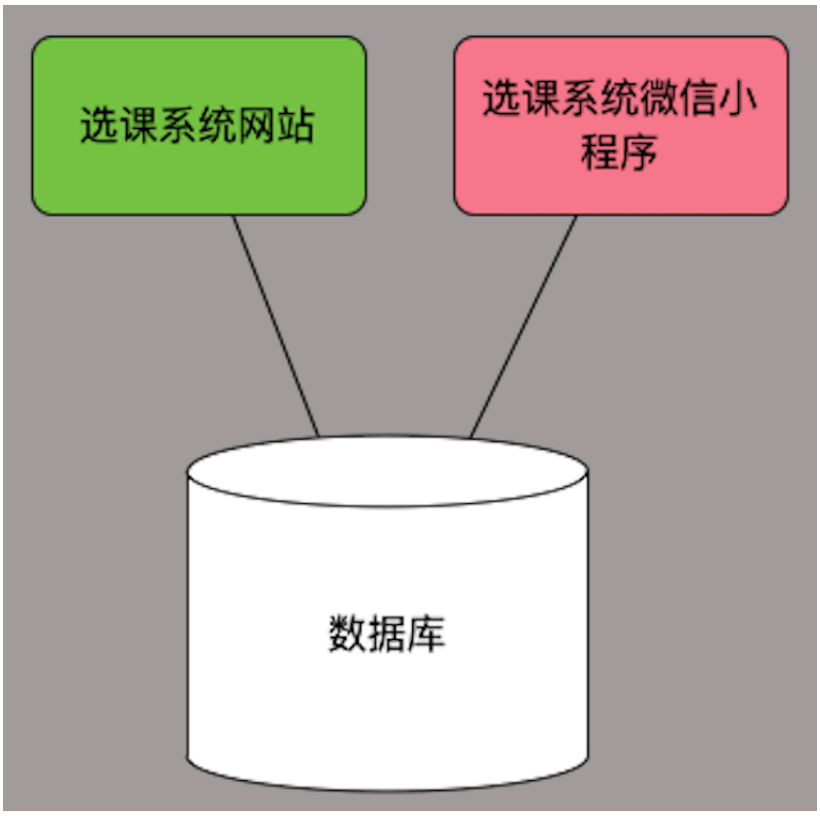
5G手机时代已经到来，选课系统不搞个手机App或者微信小程序，怎么说得过去？至少要开发一个短信通知吧！一旦要做这些应用，单体应用的的缺点马上暴露出来了。请问你会怎么做?

把源代码打开，添加一个微信小程序的模块，然后打包，最后拷贝到用户的手机上，解压。。。。等等，不对吧！你能这么做？开玩笑吧！

用户用的是手机！手机！没办法跑你的Java（或者PHP），它只能跑马化腾的微信！你要把程序开发出来，发布到微信的平台。

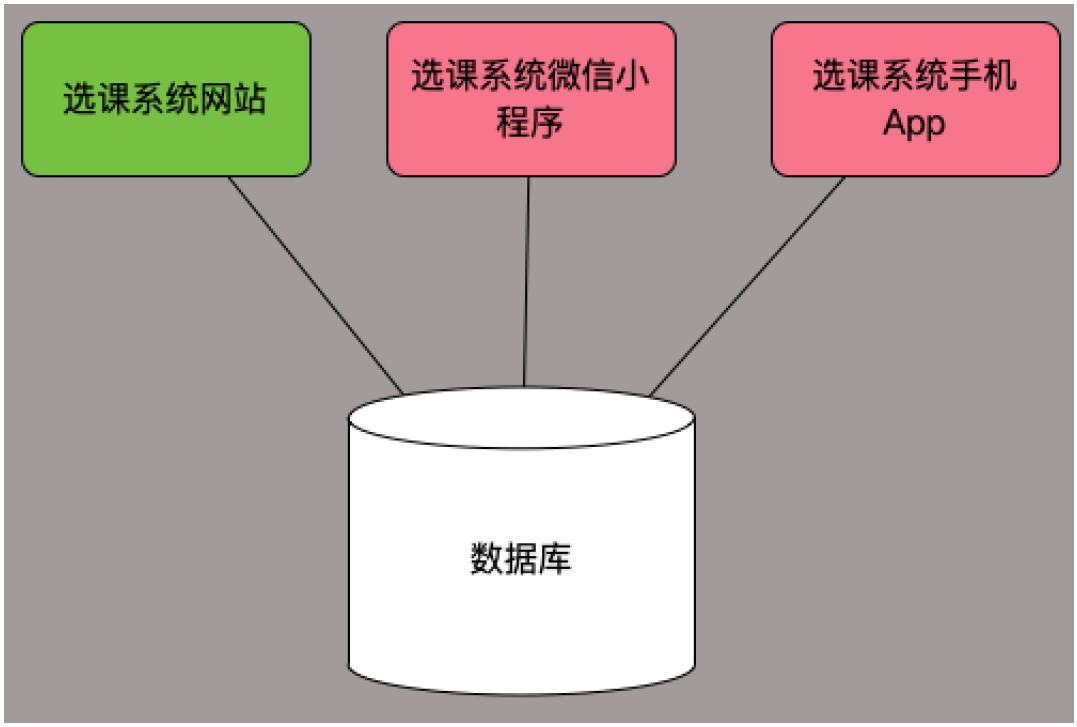
OK！把源代码打开，添加一个微信小程序的模块，然后打包，最后拷贝给微信，解压。。。。搞不定了吧?

正确的做法是必须单独搞一个微信小程序的源代码，形成如下的程序结构：



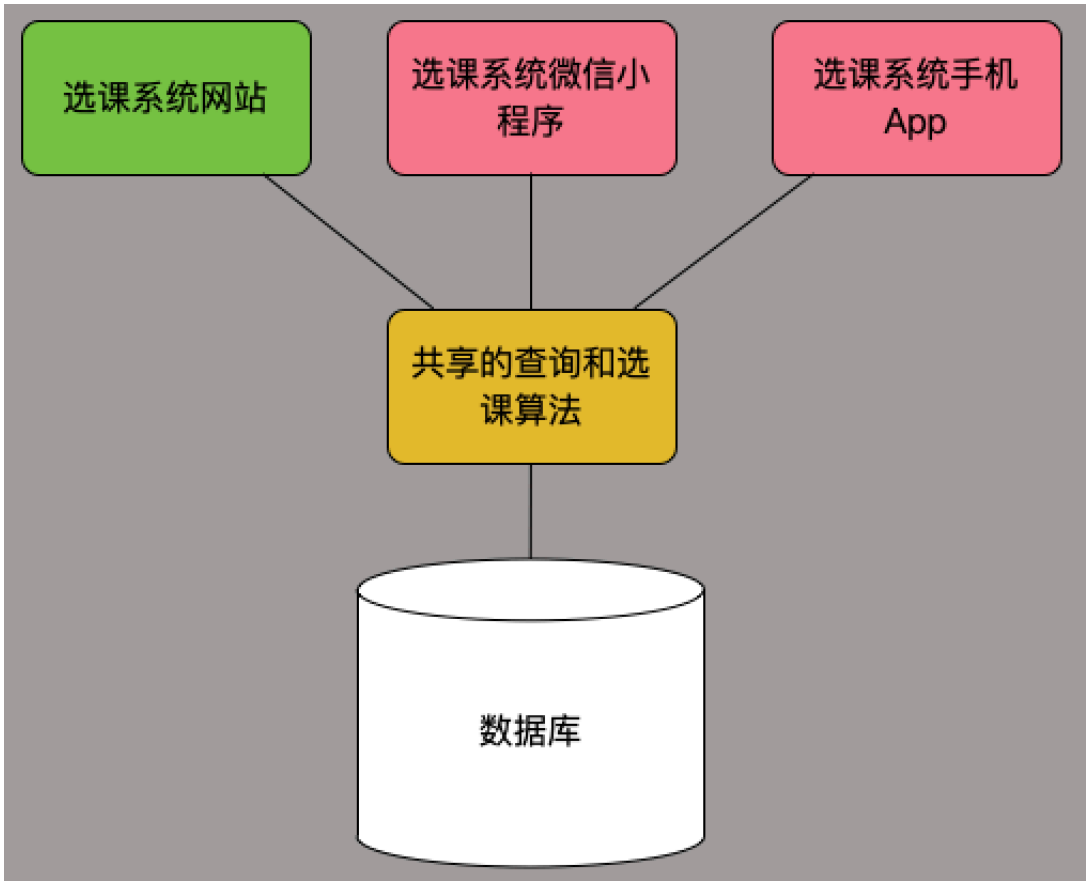
完美吧？**问题更大了！**小程序怎么查询数据库呢？需要写一段代码！小程序怎么选课呢？需要写一段代码！这意味着原来的程序逻辑重写一遍！

教务处说，我们还需要一个手机App！



赶快再把程序逻辑重写一遍！如果教务处还想在另外一个平台开发程序，我感觉程序员的大福报快要来了！

解决问题的方法其实也很简单，如果业务逻辑能够在多个应用中共享，那就不存在这个问题！像下面这样：



问题解决了吗？没有！

3种不同的系统怎么可能共享相同的算法逻辑呢？选课网站使用Java开发，微信小程序使用微信小程序的语言，App可能是Android也可能是IOS，怎么可能调用相同的算法呢？

过去不行现在没有问题。

微服务

解决上面问题的方法是将共享的算法变成服务（service）。这是一个抽象的说法，具体来说，就是变成一种可以跨平台、跨语言、跨网络调用玩意。实现服务的技术有很多很多种，常见的有Web service、RPC和Restful。目前广泛使用的是Restful，如果只学一种服务实现方式，那就是Restful。

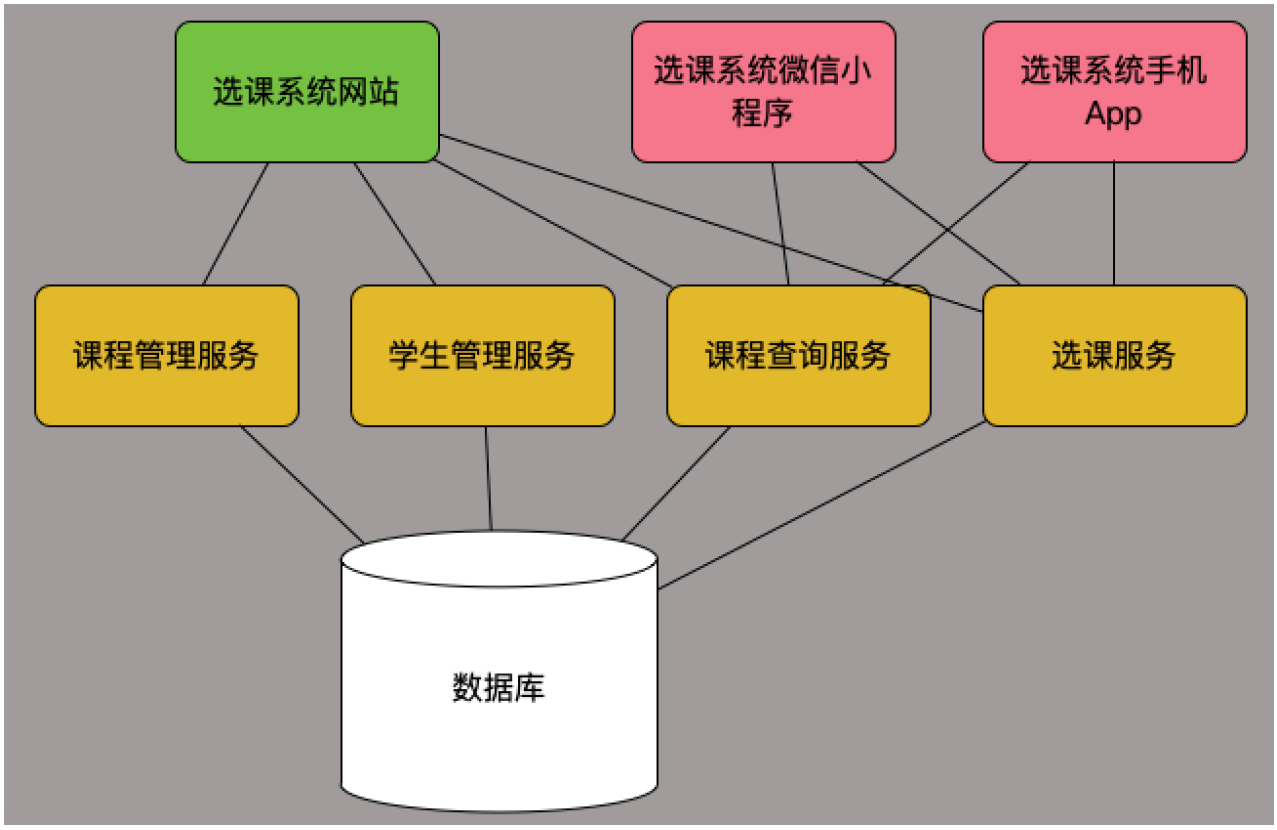
Restful可以用各种语言实现。各种框架（Node.js、Flask、Django、Spring）实现Restful的方法也非常简单。例如Node.js的如下代码

```
const express = require('express')
const app = express()
const port = 8080

app.get('/helloworld', (req, res) => {

    res.send({message:'Hello World!'})
})
app.get('/add', (req, res) => {
    var a=new Number(req.query.a)
    var b=new Number( req.query.b)
    res.send({result:a+b})

})

app.listen(port, () => console.log(`Example app listening on port ${port}!`))
```

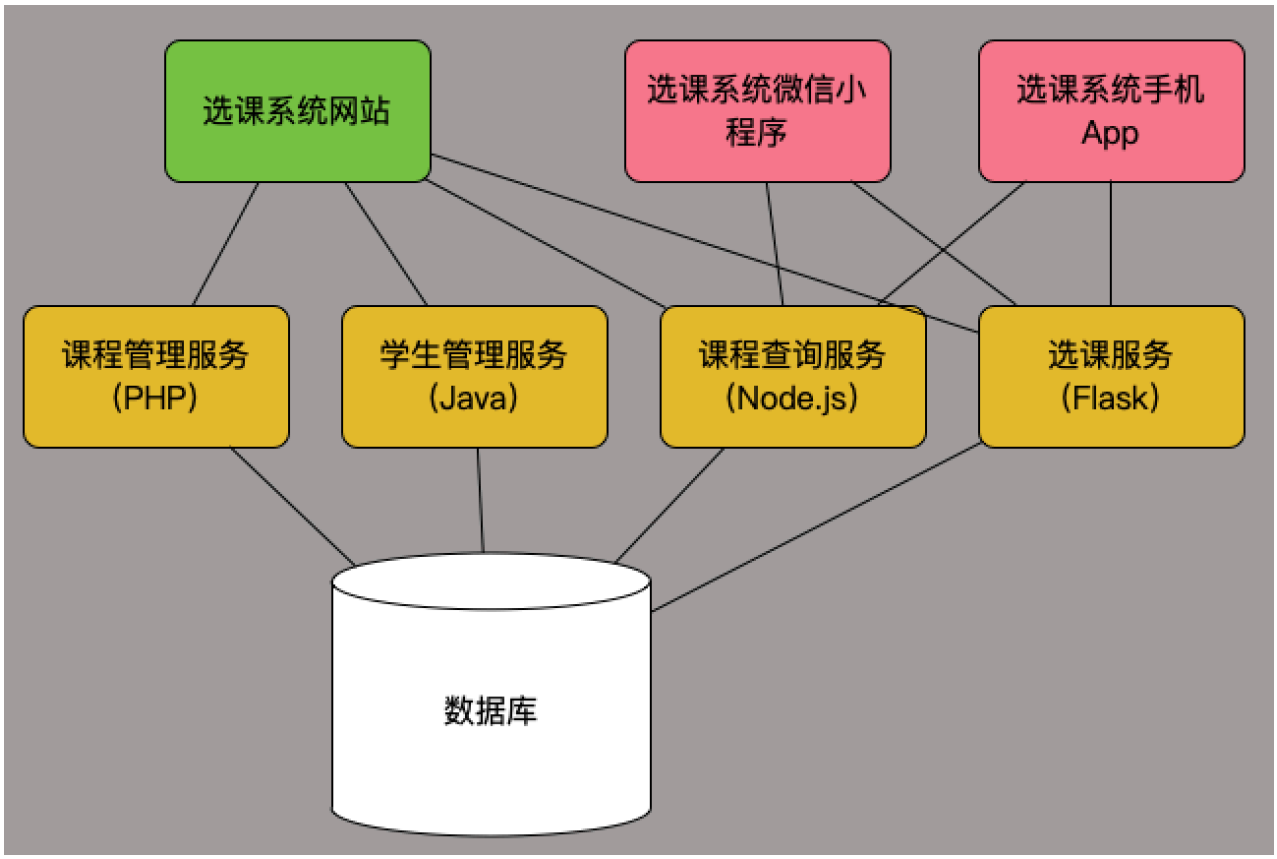
实现了两个Restful接口：

1. /helloworld，输出JSON格式的 `{message:'Hello World!'}`
2. /add, 输入a和b两个参数，输出JSON格式的`{result:a+b}`

微服务和服务的最大区别在于**微**，**微**就是小。有了Restful这样的技术，我们就可以把选课系统重构为微服务架构，也就是把查询和算法拆分为一个个的服务，服务体积要尽量的小巧（1个、2个、3个接口构成一个服务，10个接口的服务就有点多了）。如下图：



在选课系统中，微信小程序和手机app只需要使用系统提供的部分服务，并且每个服务和可能只有简单的CRUD操作，3-4个服务接口。微服务架构不仅解决了系统扩展问题，同时还使得每个模块的复杂度降低。不仅如此，如果我们考虑到每个服务还可以用不同技术予以实现，例如：



- 课程管理服务由熟悉PHP的张三去做
- 学生管理服务由熟悉Java的李四去做
- 课程查询服务有熟悉Node.js的王五去做

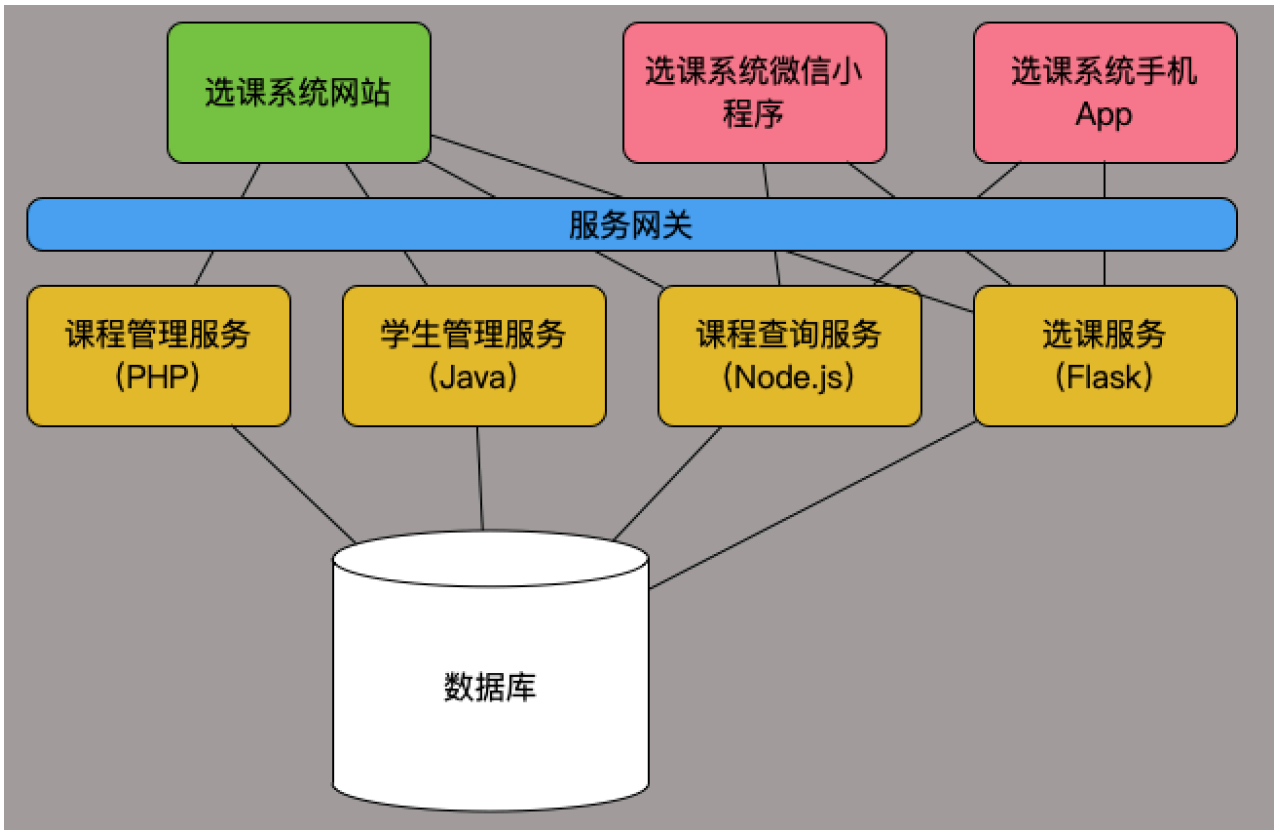
- 选课服务交给熟悉Flask的赵六去做

理论上，可以这么分工，实际上这么做也完全没有问题（如果有问题就是为啥要搞那么多技术）。负责具体服务的团队，可以选择自己熟悉的工具、技术和开发方法。是不是很美好?

没有银弹

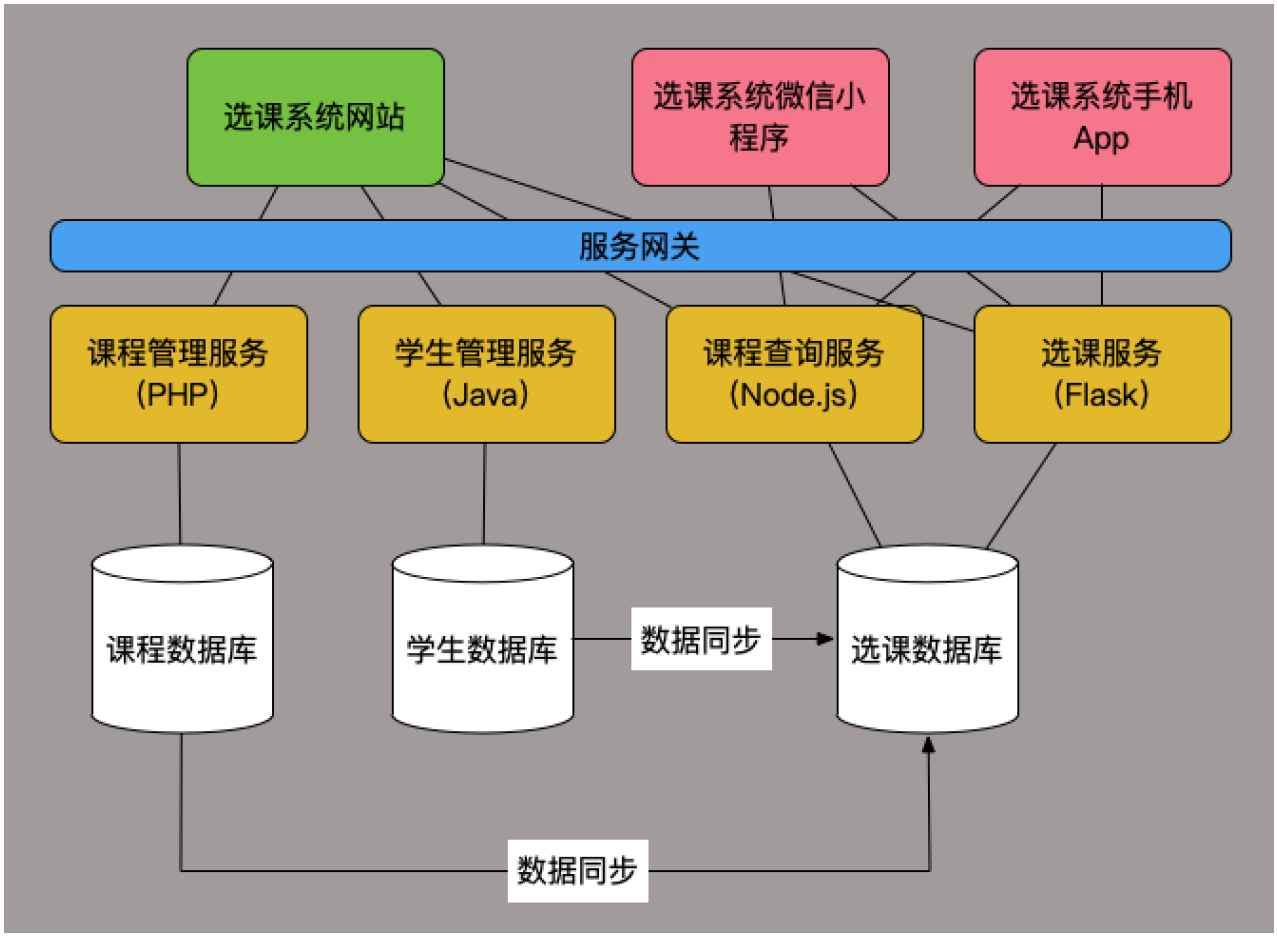
现实很残酷，虽然微服务有那么多好处，它也有很多缺陷。系统集成就是一个大问题！于是程序员又发明了DevOps、容器化部署、持续集成等等来解决这个问题（先给自己挖一个坑，再填坑，填一个，再挖三个）。所以微服务并没有降低程序员领取福报的速度。不过，它的确解决了一些问题，只要合理的使用。

举个例子，上面这个架构没有考虑权限问题，是否应该加一个服务网关，统一控制一下权限呢?



加不加是一个需要考虑的问题（会增加技术复杂度）。如果要加，怎么加也是需要考虑的问题（使用别人的框架，还是自己写一个）。

还比如，是不是分库，也是需要考虑的问题。



分库之后，各个服务实现了数据库独立、逻辑独立，再也不用担心猪队友因为写了错误的代码，把数据清空！但是成本也上去了，福报也增加了！

建议

软件设计是一个复杂的话题（设计都是复杂的话题），没有什么设计是简单的。每个项目都有特殊性，成熟的方案适用于80%的场景，但总有20%是例外。因此整体来说，应该优先使用成熟的方案，遇到例外情况，再发挥想象力。以下是我的几点建议：

1. 对于传统的信息管理系统，单一应用很好用，便于部署，但前后端分离，后端采用Restful仍然是必要的（万一哪天想做app呢）
2. 对于互联网应用，手机App或者微信小程序，应考虑直接使用微服务模式
3. 混合型的系统，单一应用和微服务混合用
4. 依据应用复杂度，确定是否采用微服务框架（spring cloud、nacos、dubbo），框架会导致复杂度提升，有时候得不偿失
5. 数据库不够大就不要分
6. 参加竞赛、做作业，直接用微服务（划水的队友也可以开发一点点，反正他做的不会影响你做的）