

后端技术知识一览

整理了一下自己在学习后端时的心得体会，争取可以让各位立志学习后端的朋友快速了解一下后端的技术 📚

如果觉得项目不错，或者给你带来了一些帮助，不妨点个 Star ❤️

本篇文章属于本人的原创作品，如果需要转载，请保留出处，谢谢咯 😊

由于本人现在的技术可能不太成熟，对于一些概念可能理解的会有问题，所以如果你在文章中发现了问题不妨指出问题，提一个issue哦 🙏

为什么会创建这样一个仓库？

很多教程对于后端的技术栈说明的并不全面，或者是太全面了让人不知道从哪里开始学起，这导致初入编程的我们学习的过程中没有较强的连续性，很容易让人产生不知道学了有什么用，有那么多技术框架，我该先学哪个的疑问。

所以作为一名普通的在校大学生，我想在此分享一下自己学习后端的整个心路历程和经验，将整个后端要学习的技术尽量以一种启发式的方式介绍给大家，希望能为所有想学习后端的同学们提供一些帮助。

这个仓库包含了哪些知识的介绍？

- 后端语言 --- Java
- 关系型数据库 --- MySQL
- ORM框架 --- Mybatis / Mybatis-plus
- Web开发框架 --- Spring, SpringMVC, SpringBoot
- 接口管理工具 --- Postman / Apifox / Swagger / YAp
- 安全管理框架 --- Spring Security / Shiro
- 依赖管理工具 --- Maven / Gradle
- 版本控制工具 --- Git
- 服务器 --- Linux
- 前端基础 --- 前端三剑客
- 设计模式 --- 优化代码
- 缓存 --- Redis
- 性能压测 --- JMeter
- 消息队列 --- RabbitMQ
- 反向代理服务器 --- Nginx
- 网络编程 --- Netty
- 微服务框架 --- SpringCloud
- 容器 --- Docker
- 容器编排 --- Kubernetes
- CI / CD --- Jenkins
- 并发编程 --- JUC

- 虚拟机 --- JVM
- 任务调度 --- XXL-JOB
- 搜索引擎 --- Elasticsearch
- 链路追踪 --- SkyWalking
- 读写分离 --- ShardingSphere / MyCat
- 同步数据 --- Canal
- 存储图片 --- OSS
- 好用的工具包 --- Hutool

个人认为除了四大件和其他CS专业知识的介绍，关于纯后端知识的介绍已经非常全面了。

你可能会觉得还不是很全，比如这里有一些概念笔者并没有提及：CDN内容分发网络，DDD架构，分布式的Raft，拥抱云的GraalVM，新时代GC的ZGC，服务网格Istio，提高代码质量的SonarQube等等。

其实当然，后端知识浩如烟海，怎么可能仅仅只用一篇文章介绍完。倒不如说，这些后端知识应该是你在有了一定基础之后自己去看前沿文章去了解到的。

写这篇文章主要是觉得现在市面上的学习路线主要有以下两个极端：

- 太全。让真正的初学者难以下手，不知道学什么，只会盲目的去学习一些新框架
- 太简单。让初学者不易于建立完整的后端知识体系，对于后端架构没有一个完整的认识

所以这里就我在大学以来和在团队中学习的经验写一些我对于后端的看法，希望可以让朋友们拨开云雾见青天，知道自己为什么学这些技术，以及要学什么技术。

我该怎么学习后端的技术

我目前学习后端的主要语言是Java，所以在这一章主要介绍一下Java后端的学习路线，当然如果你是其他语言的选手，这并不影响你阅读本章内容，因为业界要解决的问题是不会变的，变的只是框架。

比如你在Java中使用Mybatis-plus操作数据，在Go中使用GORM操作数据，在Java中使用SpringBoot作为基本的Web开发框架，在Go中使用Gin或Echo作为基本的Web开发框架。

下面，各位朋友可以根据自己的需求来学习。

这里直接总结一下最基础的后端开发需求，适合真正的初学者：

- 会一门后端语言的基础语法 --- 比如Java, Python, Go, Rust等
- 会基础的关系型数据库操作 --- 一般会MySQL的基础操作就行了
- 会使用ORM框架来操作数据库 --- 这个ORM框架你可以根据你选的语言去搜
- 会使用Web开发框架做基本的项目开发 --- 这个Web开发框架你可以根据你选的语言去搜
- 会基本的接口管理工具给我们的程序发起请求 --- 会使用Postman / Apifox其中一个就行了
- 会一个依赖管理工具来管理我们的依赖 --- 这个依赖管理工具你可以根据你选的语言去搜
- 会版本控制工具 --- Git

比如，我这里以Java为例：

- 会Java的基础语法：掌握数据类型、控制结构、面向对象编程等基本概念
- 会关系型数据库MySQL的基础操作：能够执行基本的SQL查询、插入、更新和删除操作
- 会Mybatis来操作数据库：了解如何使用MyBatis进行数据持久化
- 会SpringBoot来做基本的项目开发：能够创建和配置SpringBoot应用，理解基本的注解和配置方式
- 会使用Apifox给我们的程序发起请求：能够使用Apifox测试和管理API接口
- 会使用Maven来管理我们的依赖：熟悉Maven的基本命令和POM文件配置
- 会使用Git提交推送代码到GitHub上：能够使用 Git 进行版本控制，掌握基本的提交、推送和分支操作

又或者，以Go为例：

- 会Go的基础语法：理解数据类型、控制结构、函数和并发编程等基本概念
- 会关系型数据库MySQL的基础操作：能够执行基本的 SQL 查询、插入、更新和删除操作
- 会GORM来操作数据库：了解如何使用GORM进行数据持久化
- 会Echo / Gin来做基本的项目开发：能够使用 Echo 或 Gin 框架搭建基本的 Web 应用
- 会使用Apifox给我们的程序发起请求：能够使用 Apifox 测试和管理 API 接口
- 会使用Go Modules来管理我们的依赖：熟悉 Go Modules 的基本用法
- 会使用Git提交推送代码到GitHub上：能够使用 Git 进行版本控制，掌握基本的提交、推送和分支操作

除了这些，建议你了解学习以下内容：

- 如何使用Markdown规范自己的文档
- 如何规范自己的commit信息
- 如何遵守接口设计规范 RESTful API

当你有了一定的基础后，你可以根据后面写的这些来查缺补漏，丰富自己的技术栈。

语言---Java(必学)

首先我们要编程，写项目，一定要会一门语言，正所谓万丈高楼平地起，语言就是我们搭建高楼的砖块。

不过初学一门编程语言的时候可能你会遇到很多问题，比如思考为什么要写OOP，为什么要有多线程，很多东西你可能暂时不太了解，不过这没有关系，当你学到后面的时候自然会体会到这样做的原因。

这里介绍一下我初学Java语言时的技巧：

- **多敲代码**(代码是手上功夫，光看不练假把式)
- **学习 Debug** (当你遇到问题无法得到你预期的结果时，可以试试断点调试)
- **学习Stream流和Lambda 表达式**(这两个技术可以使我们的程序更加优雅---优化数据处理和匿名函数的编写)
- **看看阿里巴巴的开发规范**(这样更有利于你的程序写出更少的Bug，可读性更好，可扩展性更高)

推荐文档和工具：

- 简介 - Java教程 - 廖雪峰的官方网站 (liaoxuefeng.com)
- Overview (Java Platform SE 8) (oracle.com)
- Java 8 中文版 - 在线API手册 - 码工具 (matoools.com)
- [akullpp/awesome-java: A curated list of awesome frameworks, libraries and software for the Java programming language.](https://github.com/akullpp/awesome-java) ([github.com](https://github.com/akullpp/awesome-java))
- 《阿里巴巴Java开发手册 (终极版)》免费下载在线阅读藏经阁-阿里云开发者社区 (aliyun.com)

关系型数据库---MySQL(必学)

当我们学完一门语言后，一般学校里面都会要求我们做一个课设(这里用贪吃蛇举例)，我们初步了解Java并上手之后，知道了可以使用文件来存储读取数据，那么一旦当数据多起来的时候，我们就不好处理了，这时我们不妨在网上搜搜看有什么东西可以帮助我们存储数据呢？

于是你找到了 MySQL (这里还推荐你了解一下 MariaDB 和 PostgreSQL)。

那么我们学习MySQL到底要到一个什么程度才能进行开发呢？

- SQL语句编写(**主要会CURD就行了**, create、update、read和delete)
- 设计数据库表，字段(这里建议看看阿里巴巴对于数据库设计的规范，帮助你更好的设计数据库)

推荐文档：

- MySQL :: MySQL Documentation
- MySQL 中文网
- [MySQL 中文文档 | MySQL 中文网 \(mysqlzh.com\)](http://MySQL 中文文档 | MySQL 中文网 (mysqlzh.com))
- 《阿里巴巴Java开发手册 (终极版)》免费下载在线阅读藏经阁-阿里云开发者社区 (aliyun.com)

ORM框架---MyBatis / Mybatis-plus(必学)

当你写多了原生的 JDBC 来操作数据库之后，会发现编码效率很低，写的过程中也比较容易出错，为了更好的和数据库进行交互，于是你又在网上看别人的博客，发现了 ORM 这个东西，这个框架可以帮助我们更快的写 SQL，甚至有时候都不用写，而是自动生成。

这里简单介绍一下我写项目时的搭配(MyBatis / Mybatis-plus)，当然 JPA 也是可以的：

- Mybatis在这种搭配下常常用于写复杂的SQL
- Mybatis-plus天生支持单表查询，不用自己手写单表查询，这里推荐写一些简单的SQL，对于复杂的SQL建议还是使用Mybatis来写

这里简单说明一下，为什么这样搭配，因为在写复杂SQL的时候Mybatis-plus的wrapper会很复杂，可读性很低，我们写的时候犯错的几率也会更大，所以推荐这时使用Mybatis来写。

推荐文档：

- mybatis - MyBatis 3 | 简介
- [MyBatis-Plus 为简化开发而生 \(baomidou.com\)](http://MyBatis-Plus 为简化开发而生 (baomidou.com))

Web开发框架---Spring, SpringMVC , Spring Boot(必学)

这是我们做 Web服务 时常用的框架，如果不从麻烦的原生配置开始做起，建议直接学习 SpringBoot2。

SpringBoot3相对于SpringBoot2就是功能更多，但是API发生了较大的变化，可能你学习的时候大部分时间都用在处理冲突上了，所以建议新手入门直接学习SpringBoot2，等有了一定基础再学Spring家族的其他框架，做一般的项目SpringBoot也够用了。

还有就是不要看Spring, SpringMVC, Spring Boot有三个，其实Spring + SpringMVC ≈ SpringBoot，所以你其实可以直接学习SpringBoot。但是可能对于一些在前者(Spring, SpringMVC)中约定俗称的配置有些迷惑，所以这边还是建议有时间的话先把Spring, SpringMVC学了再学SpringBoot。

推荐文档：

- [Spring | Home](#)
- [spring 中文文档 - spring 中文网 \(springdoc.cn\)](#)
- [Spring 项目 :: Spring 框架 - Spring 中文 \(springframework.org.cn\)](#)

接口管理工具---Postman / Apifox / Swagger / YApi(必学)

什么，你还在使用 HttpClient 发送请求？

那么不妨试一试我上面提到的工具吧，这些工具的界面更加友好，功能更加强大，我们没有理由不去使用它。

最开始我使用的是Postman，后面使用过Apifox, Swagger, YApi等工具，就我个人这么久的使用体验而言，感觉Apifox的功能更多更强大，界面是中文，而且还可以导出接口文档，很方便，不用我们自己去写。

这个就不用什么文档了，建议直接 CSDN 启动。

安全管理框架---Spring Security / Shiro框架(选学，项目有安全需要的学)

当你的系统缺少安全管理部分可以使用到它，你也许还听说过Shiro框架，它也是做安全的，只不过相比于Spring Security更加轻量。

当然，我们说安全管理框架可能你还不太清楚，为什么要这个东西，如果你写过登录系统，就应该明白，没有框架来管理接口访问，我们将无法对用户进行相应的验证和授权，也就无法区分管理员与普通用户的权限。

除了上面那些框架，我还可以尝试什么？

- 当然，如果你觉得上面提到的框架过于笨重，那么这里还建议你尝试一下 [JWT](#)，它可比上面两个轻量多了
- 同时也建议你了解一下 [RBAC理论](#)，这个在Spring Security中也有体现
- 还有就是建议你了解一下 [OAuth2.0协议](#)

推荐文档：

- [Spring | Home](#)
- [spring 中文文档 - spring 中文网 \(springdoc.cn\)](#)
- [Spring 项目 :: Spring 框架 - Spring 中文 \(springframework.org.cn\)](#)

依赖管理工具---Maven / Gradle(必学)

相信你之前使用依赖的方式是直接将对应的 [JAR包](#) 放入项目中，但是随着你项目中使用的依赖越来越多，自己管理依赖逐渐变成了一件麻烦的事情。

于是你发现了Maven，这个工具可以帮助你管理你的依赖，并且你可以通过 [pom.xml](#) 的形式配置jar包的版本。从我们自己在Maven仓库中找到对应依赖的jar包，自己导入项目，自己管理，到Maven的一键配置，大大减少了我们为管理依赖头疼的时间。

相关建议：

- **当成工具用就行**，重要性没有那些框架重要
- 如果缺什么的工具包（比如处理 [JSON](#) 的），**建议在awesome-java中找找看有没有推荐的工具包**，然后在Maven仓库中找到对应的依赖，得到对应的pom文件导入就行了

推荐文档和仓库：

- [Maven – Maven Documentation \(apache.org\)](#)
- [Maven Repository: Search/Browse/Explore \(mvnrepository.com\)](#)
- [Maven Central \(sonatype.com\)](#)
- [Maven 中文网](#)

版本控制工具---Git(必学)

你可能想过：有没有一个工具记录自己做过的事情并且可以标准的区分自己各个版本的功能呢？

当然有咯，那就是Git，于此相应的概念还有 [GitHub](#)，不过前者是一个版本控制工具，类似的还有 [SVN](#)，后者是一个代码托管平台，类似的还有 [GitLab](#)。

我们该怎么学习Git？

- 看过 [Git底层设计](#) 的朋友都知道，Git对于底层的抽象做的很好，比如底层基于快照的存储，以及无环图的设计，但是对于上层 [API](#) 暴露的并不友好，所以推荐先看一下**Git的底层设计**之后，再学习一下Git的API，并结合Git底层的数据结构来思考，每次API操作的是数据的那一部分，时间久了，你对于Git就会非常熟悉了

以及，我们主要要学习Git的哪些API呢？

- 主要是学习提交、推送、拉取、回退、重置、克隆，代码合并、解决冲突的命令。后续根据对应项目场景的需求再对应去学即可

推荐文档和网站：

- Git - Book (git-scm.com)
- GitHub Docs
- Simple Git tutorial for beginners | Nulab
- Learn Git Branching
- GitHub Git 备忘单 - GitHub Cheatsheets

服务器---Linux(必学)

你是不是有疑惑，为什么自己写的项目只能被自己的电脑访问，而不能被其他的电脑访问？

其实是因为你没有自己的服务器，如果你要让别人访问你的服务，你可以去买一个属于你的域名和一个云服务器，将自己写的服务部署到服务器上，供别人访问。

那么，使用Linux服务器最基本的技能是什么呢？

由于现在的很多前后台项目大多都是在Linux环境下部署的，所以我们至少要了解一些常用的Linux指令。

当然如果你不买现成的云服务器，你还可以使用虚拟机，这样同样可以模拟在云服务器上Linux的操作。学会了基础的Linux操作之后，你将会在之后学习微服务架构时将会如鱼得水。

那么多命令我该怎么记下来呢？

- 你可以使用man或者tldr，这些命令会解释每一个命令具体的含义或给出详细的示例，你可以照葫芦画瓢的去使用
- 每一个命令一般都是对应英文的缩写，你可以去网上找找看，这样便于你记忆和理解

我该看什么了解Linux命令呢？

- 你可以看看鸟哥的 Linux 私房菜，这本书对于Linux有一个较为全面的介绍

我该怎么用Linux系统呢？

- 我目前主要通过 VMware , VirtualBox 和 WSL2 来使用，推荐使用 Ubuntu，当然使用 CentOS 学习也行
- VMware是我一开始学习Linux的时候用的工具，其实也是挺好用的
- VirtualBox可以结合 Vagrant 来使用，实现一键部署虚拟机，这个倒是挺方便的
- 对了对了，如果你要使用虚拟机来进行学习的话，推荐你使用 Xshell 或者 FinalShell 来作为 SSH 连接的工具，要不然你就只能使用黑黑的终端来连接了（这边建议你实在是要用最好自己配置一下 ohmyzsh，它提供了非常友好的终端界面）
- WSL2可以直接在Windows上运行，不用每次学习都要开虚拟机，更加方便

这个Linux上有很多好玩的东西，下面我来介绍一下：

- 脚本控制项目部署和起起停停 --- Shell
- 最流行的基于命令行的编辑器 --- Vim

- 最常用的构建系统之一 --- `make`

推荐网站：

- [Linux命令搜索引擎 命令](#), [Linux命令详解：最专业的Linux命令大全](#), 内容包含Linux命令手册、详解、学习, 值得收藏的Linux命令速查手册。 - [Linux 命令搜索引擎 \(wangchujiang.com\)](#)
- [Linux命令大全\(手册\) - 真正好用的Linux命令在线查询网站 \(linuxcool.com\)](#)
- 清华大学开源软件镜像站 | [Tsinghua Open Source Mirror](#)

前端基础---前端三剑客HTML , CSS , JavaScript(选学, 这个不做要求, 感兴趣可以了解一下)

作为一位想要从事后端工作的程序员来说，了解一点前端知识是很有用处的。

正如你所了解的一样，后端负责给前端提供数据，前端负责在页面上展示数据。为了使我们和前端的小伙伴们配合的更好，我们可以适当了解一些关于前端的知识，知道数据怎么提交给前端，以便于前端处理，知道前端是怎么获取到后端数据的等等。

那么，我们具体需要掌握到什么层次呢？

- 当然，对于后端程序员来说，**你不需要知道太多的前端知识**，但是基础的你还是要会的，比如说：HTML, CSS, JavaScript。当然你还可以了解一点 Vue 等等前端框架，太多的东西我们不需要掌握，了解一下目前主流前端开发框架就行了

好的，完成上面内容的基本学习之后，你将掌握基本的项目开发能力，但是想要做到一个合格的后端程序员是不容易的，接下来，我们还需要学会这些知识。

设计模式---优化代码(选学, 想要项目有更好设计的学)

也许你已经写出过几千行的代码了，但是你或许会发现自己每改动一个位置的代码，很多位置都会发生变化，你常常因为这件事情焦头烂额，觉得工作量太大。

其实大可不必，很多时候都是我们没有设计好类与类之间的关系，使得我们后期加需求，维护时的难度大大提升。

简而言之，设计模式可以让你写出**高质量的代码(Safe from bugs, Easy to understand, Ready for change)**。

设计模式简单的说有三个模式：创建型模式，结构型模式，行为型模式。当然，你肯定不想让我在这里介绍一些枯燥的知识，所以我这里直接说说我对设计模式的看法：

- 设计模式不应该一开始就强行运用上。也许你会疑惑，刚才不是说了设计模式很好很好云云之类的，为什么这里又不推荐用呢？其实是因为我们项目最开始的时候，需求并不清

楚，代码并不复杂，如果直接使用设计模式会使得代码可读性降低，而且不能有的放矢（不是所有的类都需要通过设计模式来设计），只有当后期需求逐渐完整，类与类之间的关系变得逐渐复杂，你发现可以优化的时候再推荐使用（毕竟软件开发又不是一次就结束了，而是会经历很多轮的迭代优化）

- 最开始的时候不要为了使用设计模式而使用设计模式。学习设计模式可以让我们在读对应优秀框架源码的时候游刃有余（比如Spring就是用了很多的设计模式，如果你不学，很可能看都看不懂这是在干什么）

推荐资料(除了设计模式本身，也包含了其他让你写出高质量代码的资料)：

- [Explore, Learn, and Master Industry-Standard Patterns | Java Design Patterns \(java-design-patterns.com\)](#)
- [Readings | Software Construction | Electrical Engineering and Computer Science | MIT OpenCourseWare](#)

缓存---Redis(必学)

在你使用MySQL的时候不知道你有没有想到过，如果频繁的对于一些经常要访问的数据进行SQL查询是十分浪费性能的（因为数据库是从硬盘里面读取数据到内存中）。

在业务处理中IO往往是导致处理请求慢的罪魁祸首，所以我们不妨使用一下缓存技术吧，将数据缓存到内存中，然后直接省去了从硬盘中读取这一步！

要知道缓存经常访问的不易变化的数据对于提升系统性能是有很大帮助的！

缓存其实和数据库类似，毕竟都是对数据进行存储的，所以一般而言，我们对其的操作也不会超过增删改查。

但是与关系型数据库不同的是，缓存常常和业务是强耦合的，你需要根据实际的情况判断哪些需要缓存，哪些不需要缓存（对于经常变化的数据，不建议缓存，对于不经常变化的数据，建议缓存）。

如果你学完之后意犹未尽，不妨看看 [Caffeine](#)（这是一个很优秀的本地缓存框架）和多级缓存（可以使用 [Lua脚本](#) 在 [Nginx](#) 里构建缓存）。

推荐文档：

- [Commands | Docs \(redis.io\)](#)
- [文档 - Redis 中文](#)
- [redis中文文档](#)

性能压测---JMeter(选学，想要测试一下自己程序性能的可以试试)

对了，看到这里你可能会问，为什么用了缓存之后，你就知道性能提升了啊？

没错，我们接下来介绍一下JMeter。

JMeter就是用来做压测的，你可以多关注一下你加了缓存和没有加缓存的QPS分别是多少，这时你就会发现缓存的厉害之处了。

这个就不用专门去看文档学习了，它只是一个工具，具体的可以去CSDN上看看是怎么实现压测的。

当然，这里提一下，我们这个压测仅仅只是初略的估计，因为你电脑不可能只跑你的服务，可能会打开一些Edge浏览器窗口，可能会挂着QQ，它们都会占用你电脑的系统资源。

同时，JMeter发送请求的时候也会占用你电脑的资源(毕竟是你电脑的一个进程)，所以我们这里只是大概看一下自己写的程序的性能罢了。

要想真正的测试你程序的性能，建议你把你的服务放在一台远程服务器上跑，然后用本地电脑发送JMeter请求，这样测试的相较于上面的就更加精确了。

消息队列---RabbitMQ(必学)

消息是消息，队列是队列，两个合在一起就是消息队列(在队列里存放的是一个一个的消息对象)。

这项技术主要是用于异步消息，比如说调用一个服务的链路很长，就以订单服务来说，首先创建订单，扣减库存，扣减账户余额，创建完成订单，一下子就调用到了订单服务，库存服务，用户账户服务这三个服务，如果是依次调用的话那么用户可能买完东西后会等好一会才有反应，那么如果链路变的更长呢？

所以我们急需一种技术可以实现传输和保存消息，将各个服务的请求交由消息队列去处理(这里再举一个小例子，快递员是直接上楼将快递给你快，还是统一将快递交由快递柜快，答案显而易见)。

消息队列也是同样的设计思想，与其将消息传递给对接的服务，不如解耦，去找一个代理的去异步处理消息。

消息队列那么多，我该学习哪个呢？

- 推荐学习RabbitMQ，主要是因为RabbitMQ比Kafka更简单，当然建议感兴趣的可以学学Kafka的设计理念。还有其他的消息队列你可以适当了解，比如RocketMQ和ActiveMQ，但它们的社区没有前两者活跃，资料也比较少，所以不是很推荐初学者进行学习

什么时候使用到它呢？

- 当你的调用链路过长的时候，你就可以考虑使用消息队列来处理你项目中遇到的问题了

推荐文档：

- [RabbitMQ Documentation | RabbitMQ](#)
- [RabbitMQ: 一个经纪人来管理所有队列 | RabbitMQ 中文](#)

反向代理服务器---Nginx(选学，一般只要了解一下怎么用即可)

你是否还在为Tomcat的性能而担忧，那么，来试试使用Nginx吧，Nginx一般使用在Tomcat的前面，用于抗拒较高的并发量。

一般用于存储前端的静态资源，实现动静分离，屏蔽后面的服务器，实现反向代理。

如果你要学习Nginx的话，建议你了解一下OpenResty，相较于Nginx，OpenResty的功能更多，而且我们也更容易使用Lua脚本语言来改造，加上一些自己特制的功能，当然Tengine也不错。

推荐文档：

- [nginx documentation](#)
- [nginx 文档 - Nginx 中文 \(nginxserver.cn\)](#)
- [OpenResty® - 开源官方站](#)
- [文档 - The Tengine Web Server \(taobao.org\)](#)

网路编程---Netty(选学，除非你对于网络编程很感兴趣)

如果你对于框架底层的网络编程很感兴趣，那么一定要学学Netty，可以说，很多框架的高性能是多亏了它。

可以说Netty才是集Java并发的大成者，它才是Java真正的高并发，有兴趣的可以了解一下，个人觉得难度还是不小的。

也许你觉得它和你太过遥远，其实不然，Redis客户端用过Netty，Elasticsearch底层使用的是Netty，很多微服务框架底层使用的也是Netty，比如Spring Cloud、Dubbo等，Netty的高性能使得很多框架都喜欢使用它。

微服务框架---SpringCloud(必学)

微服务，什么是微服务，微服务就是将原来巨大的单体服务拆分成一个个职责明确、功能独立的细小模块，然后再将这些模块分别部署到服务器上。

那么，为什么会出现微服务呢？

- 答案很简单，当你的项目不断的增加新的需求变的越来越大，越复杂时，总有一天会迎来被拆分的结局。就类似于大学物理，为了你更好理解，会将对应的章节拆分开，分成第几章第几节，不会说直接一大坨没有目录没有拆分的让你学。对于程序员也是一样，如果一个项目越来越庞大，那么为了使开发效率变高，必然会将一个庞大的项目拆分成一个个功能独立的小模块

推荐学习的路线：

- 先学习Dubbo，因为Dubbo对于底层的封装更少，可以使你快速的了解到分布式，RPC，然后再是Spring Cloud和Spring Cloud Alibaba，因为后两者对于底层的封装太完善了，如果直接学很有可能使你不知道一下子是怎么来的

组件那么多，我该先学哪些？

- 其实我觉得比较重要的组件是Consul / Nacos服务注册中心，OpenFeign远程服务调用，Gateway服务网关，Sentinel熔断限流，Seata分布式事务，Micrometer Tracing服务链路追踪 + Zipkin 链路可视化
- 可能你会了解到一些其他的组件和我这里说的不一样，比如这里的Micrometer Tracing，以前是Sleuth，不过现在有点过时了，因为微服务作为一个新兴领域发展的很快嘛，所以一些组件可能过了一段事件之后就会变化，所以我们没有必要同一种类型的技术学那么多，学好一种，到时候直接带着原有的知识迁移就行了

容器---Docker(必学)

当你想要在你的服务器上运行你自己的项目时，你会发现，你要配置对应的环境，而且自己的项目没有一个统一的工具进行管理。

那么为什么我们不能将这个流程自动化呢？毕竟计算机就是解放人类的，如果你想要减轻自己配置环境的负担而且不想因为自己小测试需要的环境污染整个计算机环境，那么Docker将会是你的一个不错的选择。

Docker主要做的是将你的项目和你项目所需的环境进行封装，可以使我们的程序直接在Linux服务器上跑，省去了我们亲自配置环境的麻烦事（直接写好一个 Dockerfile 就行了），而且由于其很好的隔离性，不会使得程序之间相互影响。

主要是多练练相关的命令，可以试着将自己的项目采用Docker的方式部署在虚拟机上试试，或者使用WSL2，这也是一个不错的选择。

推荐文档和仓库：

- [Docker Docs](#)
- [Docker中文网 \(github.net.cn\)](#)
- [Docker Hub Container Image Library | App Containerization](#)

容器编排---Kubernetes(选学，有自动化部署集成需求的学)

当你的容器启动的越来越多时，再采用原始的手动控制容器已经不适合了，毕竟你也不想将时间浪费在容器的起起停停上吧，于是我们有了更加自动化的工具Kubernetes，帮助我们对这些容器进行管理。

这里说说我一开始使用Kubernetes遇到的问题：

- 不要直接在笔记本上部署完整的Kubernetes集群，要不然你的电脑可能会承受不住直接死机，对于性能较差的笔记本电脑来说，部署 Minikube 或 K3s 是一个更加明智的选择

择(比起Kubkernetes来说更加轻量)，毕竟我们只是想学习它，不是非要部署一个完整的才行

推荐文档：

- [Kubernetes 文档 | Kubernetes](#)
- [Kubernetes \(K8s\) 中文](#)

CI / CD---Jenkins(选学，有自动化部署集成需求的学)

不知道各位有没有使用过 [Github Pages](#) 这个功能，每次你将最新的静态网站代码交上 Github之后，再次访问自己的网站就会发现内容更新了，那么到底是谁偷偷的为我们做了哪些本该由我们做的部署呢？

答案是 [Github Actions](#)，GitHub有属于自己的一套 CI /CD (持续集成，持续交付，简单的理解就是持续交代码，持续帮你部署发布)。

这里讲讲我在学习 Jenkins 时遇到的一些问题：

- 其实也没有多少，但是一定要注意 Jenkins 插件对于你下载 Jenkins 的版本是有要求的，为了较为快乐的学习这个工具，建议使用最新版的 Jenkins 来学习，至少你不会遇见一些因为插件版本不匹配导致的奇怪 Bug

推荐文档：

- [Jenkins 用户手册](#)

并发编程---JUC(必学)

都说 Java 百万并发，千万并发的，其实说的就是 Java 的 JUC，JUC 旨在教导你使用计算机有限的资源去应对更多的请求压力，看到这里相信你也明白了，这是属于程序优化的内容。

建议就是菜就多练，这部分是 Java 最难的部分之一，没有长时间的实践积累，学习原理，你是拿不下这块内容的。

推荐网站(建议自己找书看)：

- [Z-Library - 世界上最大的电子图书馆。自由访问知识和文化 \(singlelogin.re\)](#)

虚拟机---JVM(必学)

和上面的 JUC 一样，都是 Java 中的大头，对于 JVM 而言，相信你可能听过 JVM 调优，没错，又是对 Java 程序优化，在这里你将学习到如何从 JVM 的层面了解 Java 代码，会使你看待 Java 程序的观点产生质的变化。

这边建议结合视频和书(《深入理解 Java 虚拟机 (第3版)》)一起看，毕竟这个东西比较枯燥，可能你会看不下去书，还有就是推荐使用 [Arthas](#) 这个工具来帮助我们学习。

推荐工具和文档(建议自己找书看)：

- 命令列表 | [arthas \(aliyun.com\)](#)
- [java \(oracle.com\)](#)
- Z-Library - 世界上最大的电子图书馆。自由访问知识和文化 ([singlelogin.re](#))

任务调度---XXL-JOB(选学，一般了解即可)

任务交由谁来执行呢，定时任务该怎么做，如果你有这样的需求，那么XXL-JOB一定对你的胃口。

个人感觉和RabbitMQ的功能比较类似，但是比起RabbitMQ，XXL-JOB在实际的项目使用中更加轻量化。

推荐文档：

- 分布式任务调度平台XXL-JOB ([xuxueli.com](#))

搜索引擎---Elasticsearch(选学，等你项目需要搜索功能的时候再学)

这个其实也和数据库比较像，想想你平时搜索，如果匹配到了是会高亮的，或者说，不是完全匹配也可以找到数据，其实这都是它的功劳(比如你搜索笔记本电脑，但是很明显，没有这个东西，可能是你打掉了一个字，但是搜索的时候依然可以搜索出来笔记本电脑)。

为要搜索的数据建立索引，主要用于快速搜索。

这个Elasticsearch是 [ELK](#) 的一部分，其中 [Kibana](#) 可以可视化你存在Elasticsearch中的数据，而且还提供了一个写 [DSL](#) 语句的页面，有点类似[HttpClient](#)。

推荐文档：

- Documentation ([elastic.co](#))

链路追踪---SkyWalking(选学，除非你的项目很大，链路层级很深)

当你链路长了的时候推荐使用，主要是帮助你分析调用链路的，比如是在哪里调用出现了问题，哪里的请求处理耗时太长等等。

这个你也可以不用学，因为我们之前已经学习了Micrometer Tracing服务链路追踪，已经学习了一个链路追踪的组件了，如果你学了一个的话，其实也没有必要学习太多，学好一个就行了。

推荐网站：

- [Apache SkyWalking](#)
- [SkyWalking 极简入门 | Apache SkyWalking](#)

读写分离---ShardingSphere / MyCat(选学，除非你的项目一个数据库不够用)

可以帮助你实现读写分离，分库分表，一般是在你一个数据库不够应对现有并发量的情况下使用，否则不建议学习，建议有需要了再学习。

这个建议直接看官方文档，主要是官方文档写的比较精炼，而且基本上照着文档上面说的配置一般都没有什么问题，不过要注意使用的数据库连接客户端的版本不能太低，要不然会产生不兼容的问题。

至于这两个我们到底要选择哪一个，我推荐学习使用ShardingSphere，因为它的社区更活跃，功能更多，比MyCat更加完善。

推荐文档：

- 概览 :: ShardingSphere (apache.org)

同步数据---Canal(选学，建议了解即可，毕竟我们也不是要做运维人员)

主要是为了解决主从数据库数据的同步问题，一般要用的话会配合MySQL使用。

存图片---OSS(选学，建议当你遇到存图片的需求的时候再去有针对性的学)

嗯，想想看，当你有了一个用户系统，然后每一个用户都会上传自己的头像，随着用户越来越多，你会发现这些图片好像浪费了你服务器的很多空间，那么你有没有考虑过将图片存储在其他人的服务器上以避免这种情况呢？

如果你想优化一下，那么不妨试一试阿里云的OSS云存储吧，免费又实用。

推荐网站：

- 阿里云-计算，为了无法计算的价值 (aliyun.com)

超好用的工具包---Hutool(必学)

你是不是在为有时候没有好用的工具而烦恼，又或者是自己写的小工具总是有Bug，那么不妨尝试一下这款工具包吧，里面包含了对文件、流、加密解密、转码、正则、线程、XML等JDK方法的封装，用起来既方便又安全。

推荐文档：

- Hutool 一个功能丰富且易用的Java工具库，涵盖了字符串、数字、集合、编码、日期、文件、IO、加密、数据库JDBC、JSON、HTTP客户端等功能

笔者最后想说的话

什么？你竟然看到这里来了，那么不管怎么说，你一定是热爱技术的，不过我想在此提提我学了这么久的一些心得体会：

- 框架是为了解决问题而出现的，我们不应该痴迷于学框架，而是应该把自己的基础搞扎实，比如多看看类似 [SICP](#) 这样的书籍。所以一般推荐你遇到了相应的问题再去找框架解决，而不是盲目的去找框架学习。还有就是记住一点，框架会不停的变化，但是它们底层设计的思想一般是不会变化的(所以我们没有必要每一个都学，学一个经典的其他的都差不多了，感兴趣再多了解一下就行)
- 我们要锻炼自己看文档的能力。就以我为例吧，我之前学习的时候总是去在B站上找视频看，但是后来我发现这样学习的效率实在是太低了，于是开始慢慢尝试去看一些中文文档去学习（上面的ShardingSphere我就是照着文档学习的），你会发现，当你会通过文档学习之后，学习的速度会变得非常快
- 善于使用搜索平台。[Stack Overflow](#)和[Github](#)是你的好帮手，前者可以搜问题，后者可以找代码
- 不要过于依赖别人的回答。有时候，别人的回答效率还不如你自己去解决这个问题的效率高，建议是在自己的程序出现Bug之后，先自己在网上搜，用GPT问，各种Debug的方法尝试了，自己实在是解决不了再去问，关于问问题的规范可以看看这个 [How-To-Ask-Questions-The-Smart-Way](#)