目　录

致谢

当前文档 《PHPer 面试指南》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-05-09。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常生活、工作和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址：http://www.bookstack.cn/books/PHPerInterviewGuide

书栈官网：http://www.bookstack.cn

书栈开源：https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

前言

作为一位程序员，面试过多次，也面试过很多人，最近又在找工作，总结一下面试经验和面试题，希望可以帮到正在找工作的小伙伴们。

先说一下面试时的心态，刚入门的程序员，技术实力不高，又大多不善言谈，面试一旦遇到难题，很容易心态失衡、惊慌失措、语无伦次，最终丢掉了 Offer。

其实大可不必，心态坦然，是面试必备的一点。

面试新手，面试官心中很清楚，你的实力有几分几两，一般不会过意的为难人，就算是面试真的出了比较难的题，最多也就是要压一压，你的心理预期的薪水，或者就是考验一下你随机应变的能力。

而对于那种内心十分渴望，但是又对技术水平要求比较高的工作，你要明白，只挣自己能力范围内的薪水，对于梦寐以求的 Offer，你应该不断的提升自己的技术水平，来达到这样 Offer 的标准，而不是只是渴望、碰运气。

想要得到某样东西，最好的办法是让自己配得上它。

PHPer 的开发的技术栈大多是 LAMP 或者是 LNMP，其中 Linux、Apache（Nginx）都比较偏运维，但是 PHP、MySQL 是每一位合格的 PHPer 都必须精通的技术栈。

而 Web 开发又不单单只靠 PHP、MySQL，更多的还需要了前端、Web 安全、高并发、性能优化，甚至还需要学习网络协议、算法等等相关的编程知识。

这次整理了不少面试相关的知识和技巧，给大家简单介绍一下纲要：

- 扩展阅读资源整理

不论你是学习也好，面试也好，都要明白一点，对于技术知识，不应该向背课文一样，死记硬背，理解大于记忆。

希望大家看完之后，都能有所收获，早日找到梦寐以求的 Offer。

GitBook链接

为方便大家阅读，将本 repo 同步至 GitBook: https://todayqq.gitbooks.io/phper/content/

关于作者

赵金超，一位知趣、有趣、有自我觉悟的程序员

个人博客：简书主页

公众号：今朝浅谈，不定期更新区块链相关知识

贡献

本文使用 markdown 编写，提交 PR 时，文章排版请遵循中文文案排版指北。

支持

如果我写的文章，可以帮到你，不妨支持一下:)

转载合作

转载本指南，请注明作者以及 GitHub 链接，谢谢！

协议/License

来源(书栈小编注)

Web 篇

对于大公司，很少会有全栈工程师这个岗位，全栈是个花哨的词，对于现在比较热门的技术，不论是 Vue 还是 Laravel，只要智商不差，看着文档，都能写出一个 CURD 来，但是这就叫全栈了吗？

比如 Vue 中的 MVVM，其中 VM 视图的原理是什么？Laravel 为什么要这么设计？

会用这种技术栈，其实只是学到的只是皮毛，可以会用，但是必须要有自己精通擅长的一面，一定要做到人无我有，人有我优。

面试题

- 谈谈对 Web 语义化的理解

> 语义化的含义就是用正确的标签做正确的事情，语义化让页面的内容结构化，结构更清晰，便于对浏览器、搜索引擎解析，利于 *SEO*，也有利于代码阅读、便于维护。

- 写出一个使用 flex 布局，在 div 垂直居中的 css 代码

```
div {
    display: flex;
    justify-content: center;
    align-items: center;
}
```

- 为什么把 JavaScript 文件放在 Html 底部

> 1. 因为浏览器渲染 *HTML* 文件是从上往下渲染的，*JavaScript* 放在 *Html* 头部，会阻碍浏览器的渲染速度，增加用户的等待时间
> 2. 浏览器加载 *JavaScript* 脚本之后会自动执行，如果放在头部，此时的 *Dom* 树还没有加载完，很容易出 *Bug*

- 谈谈对 JavaScript 闭包的理解

> 闭包是 *JavaScript* 函数的一种，声明即运行，可以在函数内部调用外部变量。

- 如何处理 Ajax 跨域问题

> 1. 代理
> 2. *JsonP*
> 3. *iframe* 等等……

- 一句话解释 JsonP 的原理

> *jsonp*，即 *json padding*，原理就是利用 `<script>` 标签没有跨域限制来达到与第三方通讯的目的。

> 前端的知识比较多，一些比较基础的问题，就不再整理了，比如给 *Http* 常见状态码，*Html5* 多了那些标签，*CSS* 如何清除浮动等等。

如果大家有兴趣，可以阅读这些前端的常见面试题和资料。

扩展阅读

- 收集的前端面试题和答案
- 前端开发面试题
- 史上最全的web前端面试题汇总及答案1
- 前端工程师手册
- HTTP协议：工作原理
- SSL/TLS协议运行机制的概述

协议篇

- 扩展阅读

每次面试多多少少都会被问到 HTTP、HTTPS、TCP、Socket、 OAuth 等等之类协议，协议相关的问题也可以说是面试必备，所以我把这些知识单独收集成了一篇文章。

- 网络协议有哪些？

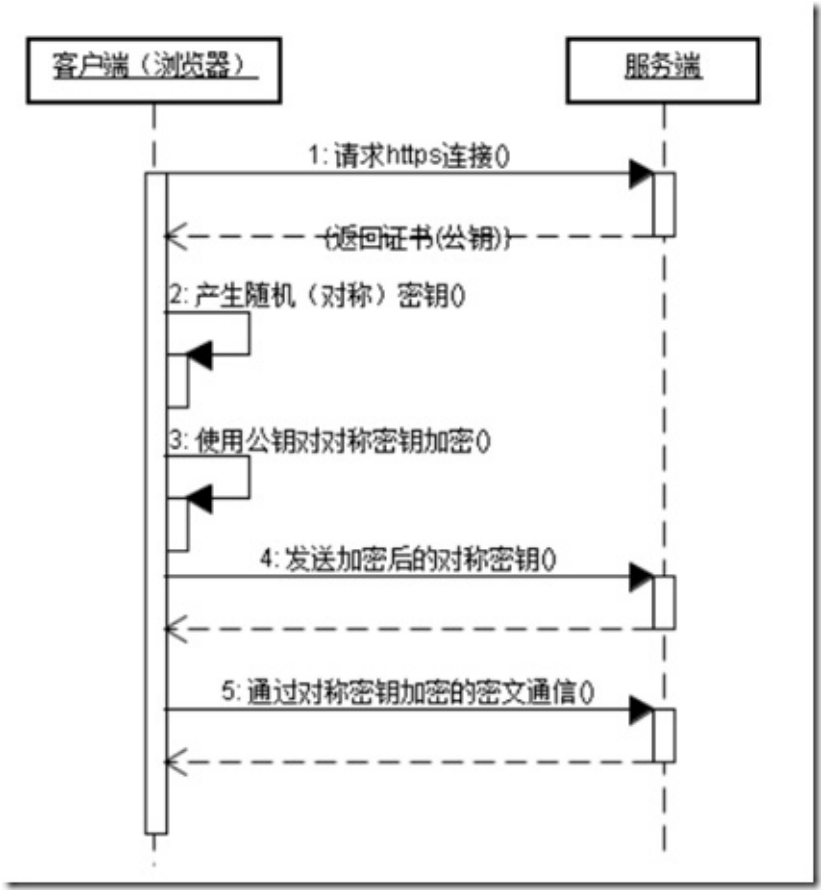
> - 应用层：*HTTP*、*FTP*、*SSH*、*SMTP*
> - 表示层
> - 会话层
> - 传输层：*TCP*、*UDP*
> - 网络层：*IP*
> - 数据链路层
> - 物理层

- 简述 HTTP 协议的工作流程

> 1. 地址解析*;*
>   在浏览器中输入 *URL*，浏览器会从中分解出协议名、主机名、端口、对象路径等部分
> 2. 封装 *HTTP* 请求数据包
> 3. 浏览器获取主机 *IP* 地址，建立 *TCP* 链接（*TCP* 的三次握手）
> 4. *TCP* 链接建立后发送 *HTTP* 请求
>   请求方式的格式为：统一资源标识符（*URL*）、协议版本号，后边是 *MIME* 信息包括请求修饰符、客户机信息和可内容。
> 5. 服务器接到请求后，给予相应的响应信息
>   其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是 *MIME* 信息包括服务器信息、实体信息和可能的内容
> 6. 服务器断开 *TCP* 连接

- 什么是 HTTPS？实现过程是什么？

> *HTTPS*（超文本传输安全协议）是一种通过计算机网络进行安全通信的传输协议，提供对网站服务器的身份认证，保护数据传输的完整性、安全性。
>
> 实现过程：
>
> 1. 客户端发起一个 *https* 的请求
> 2. 服务端接收客户端请求，返回数字证书相关信息
> 3. 客户端收到服务端响应
>   i. 验证证书的合法性
>   ii. 如果证书受信任，生成随机数的密码
>   iii. 使用约定好的 *HASH* 算法计算握手消息，并使用生成的随机数对消息进行加密，然后发送给服务端
> 4. 网站接收浏览器发来的密文后
>   i. 使用私钥来解密握手消息取出随机数密码，再用随机数密码解密，握手消息与 *HASH* 值，并与传过来的*HASH*值做对比确认是否一致
>   ii. 使用密码加密一段握手消息，发送给浏览器
> 5. 浏览器解密并计算握手消息的 *HASH*，如果与服务端发来的 *HASH* 一致，此时握手过程结束，之后所有的通信数据，将由之前浏览器生成的随机密码，并利用对称加密算法进行加密。



- 数字证书都包含那些信息？

> - 证书的版本信息；
> - 证书的序列号，每个证书都有一个唯一的证书序列号；
> - 证书所使用的签名算法；
> - 证书的发行机构名称；
> - 证书的有效期；
> - 证书所有人的名称、公开密钥；
> - 证书发行者对证书的签名

- TCP 三次握手的流程

1. 客户端发送一个 SYN 标志位置 1 的包，指明客户端要连接服务器端的接口，发送完毕后，客户端进入 SYN_SEND 状态
2. 服务器发回确认包 (ACK) 应答。即 SYN 标志位和 ACK 标志位均为1。服务器端选择自己 ISN 序列号，放到 Seq 域里，同时将确认序号(Acknowledgement Number)设置为客户的 ISN 加1，即X+1。 发送完毕后，服务器端进入 SYN_RCVD 状态。
3. 客户端再次发送确认包(ACK)，SYN 标志位为0，ACK 标志位为1，并且把服务器发来 ACK 的序号字段+1，放在确定字段中发送给对方，并且在数据段放写ISN的+1

发送完毕后，客户端进入 ESTABLISHED 状态，当服务器端接收到这个包时，也进入 ESTABLISHED 状态，TCP 握手结束。

- 什么是 Socket？工作流程是怎样的？

> *Socket* 又称网络套接字，是一种操作系统提供的进程间通信机制。
>
> 工作流程：
>
> 1. 服务端先用 *socket* 函数来建立一个套接字，并调用 *listen* 函数，使服务端的这个端口和 *IP* 处于监听状态，等待客户端的连接
> 2. 客户端用 *socket* 函数建立一个套接字，设定远程 *IP* 和端口，并调用 *connect* 函数
> 3. 服务端用 *accept* 函数来接受远程计算机的连接，建立起与客户端之间的通信
> 4. 完成通信以后，最后使用 *close* 函数关闭 *socket* 连接。

- HTTP1.1 与 WebSocket 的区别？

> - *HTTP* 是一个单链接，只能做单向通讯，而 *WebSocket* 是一个持久链接，可用作双向通讯。
> - *WebSocket* 是基于 *HTTP* 来建立连接的，但在建立连接之后，真正的数据传输阶段是不需要 *HTTP* 协议参与的
> - *WebSocket* 的请求的头部和 *HTTP* 请求头部不同
> - *WebSocket* 传输的数据是二进制流，是以帧为单位，*HTTP* 是明文字符串传输

- 什么是 OAuth2.0 协议？运行流程是怎样的？

OAuth(Open Authorization) 协议为用户资源的授权提供了一个安全的、开放而又简易的标准，第三方无需使用用户的用户名与密码，就可以申请获得该用户资源的授权。

运行流程：

1. 用户打开客户端以后，客户端要求用户给予授权。
2. 用户同意给予客户端授权
3. 客户端使用上一步获得的授权，向认证服务器申请令牌。
4. 认证服务器对客户端进行认证以后，确认无误，同意发放令牌。
5. 客户端使用令牌，向资源服务器申请获取资源。
6. 资源服务器确认令牌无误，同意向客户端开放资源

OAuth 2.0 定义了四种授权方式，授权码模式、简化模式、密码模式、客户端模式，具体的授权流程，请看阮一峰老师的文章理解OAuth 2.0。

扩展阅读

- https 原理
- HTTPS 原理解析
- HTTPS 的工作原理
- socket
- HTTP与WebSocket的区别
- 理解OAuth 2.0

PHP 篇

PHP 篇收集了一些常见的基础、进阶面试题。

基础篇

- Get 和 POST 的区别
- 单引号和双引号的区别
- isset 和 empty 的区别
- echo、print_r、print、var_dump 之间的区别
- 什么是 MVC？
- 传值和传引用的区别？
- Cookie 和 Session 的区别和关系

> 1. *Cookie* 在客户端（浏览器），*Session* 在服务器端
> 2. *Session* 比 *Cookie* 安全性更高
> 3. 单个 *Cookie* 保存的数据不能超过 *4K*
> 4. *Session* 是基于 *Cookie*，如果浏览器禁用了 *Cookie*，*Session* 也会失效（但是可以通过其它方式实现，比如在 *url* 中传递 *Session ID*）

进阶篇

- 简述 S.O.L.I.D 设计原则

-	-	-
SRP	单一职责原则	一个类有且只有一个更改的原因
OCP	开闭原则	能够不更改类而扩展类的行为
LSP	里氏替换原则	派生类可以替换基类使用
ISP	接口隔离原则	使用客户端特定的细粒度接口
DIP	依赖反转原则	依赖抽象而不是具体实现

- 列举一些 PHP 中的设计模式？

> - 单例模式：保证在整个应用程序的生命周期中，任何一个时刻，单例类的实例都只存在一个，同时这个类还必须提供一个访问该类的全局访问点。
> - 工厂模式：定义一个创建对象的接口，但是让子类去实例化具体类。工厂方法模式让类的实例化延迟到子类中。
> - 观察者模式：观察者模式有时也被称作发布/订阅模式，该模式用于为对象实现发布/订阅功能：一旦主体对象状态发生改变，与之关联的观察者对象会收到通知，并进行相应操作。

> - 适配器模式：适配器模式将一个类的接口转换成客户希望的另外一个接口，使得原本由于接口不兼容而不能一起工作的那些类可以在一起工作。
> - 依赖注入模式：依赖注入（*Dependency Injection*）是控制反转（*Inversion of Control*）的一种实现方式。要实现控制反转，通常的解决方案是将创建被调用者实例的工作交由 *IoC* 容器来完成，然后在调用者中注入被调用者（通过构造器/方法注入实现），这样我们就实现了调用者与被调用者的解耦，该过程被称为依赖注入。
> - 门面模式：门面模式（*Facade*）又称外观模式，用于为子系统中的一组接口提供一个一致的界面。

了解更多，请看PHP 设计模式系列。

- PHP7 和 PHP5 的区别，具体多了哪些新特性？

> 1. 性能提升了两倍
> 2. 增加了结合比较运算符 *(<=>)*
> 3. 增加了标量类型声明、返回类型声明
> 4. `try...catch` 增加多条件判断，更多 *Error* 错误可以进行异常处理
> 5. 增加了匿名类，现在支持通过*new class* 来实例化一个匿名类，这可以用来替代一些“用后即焚”的完整类定义

- 为什么 PHP7 比 PHP5 性能提升了？

> 1. 变量存储字节减小，减少内存占用，提升变量操作速度
> 2. 改善数组结构，数组元素和 *hash* 映射表被分配在同一块内存里，降低了内存占用、提升了 *cpu* 缓存命中率
> 3. 改进了函数的调用机制，通过优化参数传递的环节，减少了一些指令，提高执行效率

- 简述一下 PHP 垃圾回收机制（GC）

> *PHP 5.3* 版本之前都是采用引用计数的方式管理内存，*PHP* 所有的变量存在一个叫 `zval` 的变量容器中，当变量被引用的时候，引用计数会*+1*，变量引用计数变为*0*时，*PHP* 将在内存中销毁这个变量。
>
> 但是引用计数中的循环引用，引用计数不会消减为 *0*，就会导致内存泄露。
>
> 在 *5.3* 版本之后，做了这些优化：
>
> 1. 并不是每次引用计数减少时都进入回收周期，只有根缓冲区满额后在开始垃圾回收；
> 2. 可以解决循环引用问题；
> 3. 可以总将内存泄露保持在一个阈值以下。

了解更多可以查看 PHP 手册，垃圾回收机制。

- 如何解决 PHP 内存溢出问题

> 1. 增大 *PHP* 脚本的内存分配
> 2. 变量引用之后及时销毁
> 3. 将数据分批处理

- Redis、Memecached 这两者有什么区别？

> 1. *Redis* 支持更加丰富的数据存储类型，*String*、*Hash*、*List*、*Set* 和 *Sorted Set*。*Memcached* 仅支持简单的 *key-value* 结构。
> 2. *Memcached key-value*存储比 *Redis* 采用 *hash* 结构来做 *key-value* 存储的内存利用率更高。
> 3. *Redis* 提供了事务的功能，可以保证一系列命令的原子性
> 4. *Redis* 支持数据的持久化，可以将内存中的数据保持在磁盘中
> 5. *Redis* 只使用单核，而 *Memcached* 可以使用多核，所以平均每一个核上 *Redis* 在存储小数据时比 *Memcached* 性能更高。

- Redis 如何实现持久化？

> 1. *RDB* 持久化，将 *Redis* 在内存中的的状态保存到硬盘中，相当于备份数据库状态。
> 2. *AOF* 持久化（*Append-Only-File*），*AOF* 持久化是通过保存 *Redis* 服务器锁执行的写状态来记录数据库的。相当于备份数据库接收到的命令，所有被写入 *AOF* 的命令都是以 *Redis* 的协议格式来保存的。

Web 安全防范

- CSRF 是什么？如何防范？

> *CSRF*（*Cross-site request forgery*）通常被叫做「跨站请求伪造」，可以这么理解：攻击者盗用用户身份，从而欺骗服务器，来完成攻击请求。

防范措施：

1. 使用验证码
2. 给每一个请求添加令牌 token 并验证

- XSS 是什么？如何防范？

> *XSS(Cross Site Scripting)*，跨站脚本攻击，攻击者往 *Web* 页面里插入恶意 *Script* 代码，当用户浏览该页之时，嵌入其中 *Web* 里面的 *Script* 代码会被执行，从而达到恶意攻击用户的目的。

防止 XSS 攻击的方式有很多，其核心的本质是：永远不要相信用户的输入数据，始终保持对用户数据的过滤。

- 什么是 SQL 注入？如何防范？

> *SQL* 注入就是攻击者通过一些方式欺骗服务器，结果执行了一些不该被执行的 *SQL*。

SQL 注入的常见场景

1. 数据库里被注入了大量的垃圾数据，导致服务器运行缓慢、崩溃。
2. 利用 SQL 注入暴露了应用程序的隐私数据

防范措施：

1. 保持对用户数据的过滤
2. 不要使用动态拼装 SQL
3. 增加输入验证，比如验证码
4. 对隐私数据加密，禁止明文存储

扩展阅读

- 3年PHPer的面试总结
- 垃圾回收机制
- S.O.L.I.D 面向对象设计
- 浅谈IOC—说清楚IOC是什么
- Redis和Memcached的区别

- CSRF攻击与防御
- XSS跨站脚本攻击
- PHP 设计模式系列

Laravel 篇

Laravel 作为现在最流行的 PHP 框架，其中的知识较多，所以单独拿出来写一篇。

- 简述 Laravel 的生命周期

> *Laravel* 采用了单一入口模式，应用的所有请求入口都是 *public/index.php* 文件。
>
> 1. 注册类文件自动加载器 *：Laravel*通过 *composer* 进行依赖管理，无需开发者手动导入各种类文件，而由自动加载器自行导入。
> 2. 创建服务容器：从 *bootstrap/app.php* 文件中取得 *Laravel* 应用实例 *$app (*服务容器*)*
> 3. 创建 *HTTP / Console* 内核：传入的请求会被发送给 *HTTP* 内核或者 *console* 内核进行处理
> 4. 载入服务提供者至容器：
>   在内核引导启动的过程中最重要的动作之一就是载入服务提供者到你的应用，服务提供者负责引导启动框架的全部各种组件，例如数据库、队列、验证器以及路由组件。
> 5. 分发请求：一旦应用完成引导和所有服务提供者都注册完成，*Request* 将会移交给路由进行分发。路由将分发请求给一个路由或控制器，同时运行路由指定的中间件

- 服务提供者是什么？

> 服务提供者是所有 *Laravel* 应用程序引导启动的中心*, Laravel* 的核心服务器、注册服务容器绑定、事件监听、中间件、路由注册以及我们的应用程序都是由服务提供者引导启动的。

- IoC 容器是什么？

> *IoC*（*Inversion of Control*）译为 「控制反转」，也被叫做「依赖注入」*(DI)*。什么是「控制反转」？对象 *A* 功能依赖于对象 *B*，但是控制权由对象 *A* 来控制，控制权被颠倒，所以叫做「控制反转」，而「依赖注入」是实现 *IoC* 的方法，就是由 *IoC* 容器在运行期间，动态地将某种依赖关系注入到对象之中。
>
> 其作用简单来讲就是利用依赖关系注入的方式，把复杂的应用程序分解为互相合作的对象，从而降低解决问题的复杂度，实现应用程序代码的低耦合、高扩展。
>
> *Laravel* 中的服务容器是用于管理类的依赖和执行依赖注入的工具。

- Facades 是什么？

> *Facades*（一种设计模式，通常翻译为外观模式）提供了一个*"static"*（静态）接口去访问注册到 *IoC* 容器中的类。提供了简单、易记的语法，而无需记住必须手动注入或配置的长长的类名。此外，由于对 *PHP* 动态方法的独特用法，也使测试起来非常容易。

- Contract 是什么？

> *Contract*（契约）是 *laravel* 定义框架提供的核心服务的接口。*Contract* 和 *Facades* 并没有本质意义上的区别，其作用就是使接口低耦合、更简单。

- 依赖注入的原理？

> 这个不解释，这是理解 *IoC* 容器的前提。

- 谈谈 Laravel 和 YII 框架的区别

> 1. 在 *YII* 框架中的路由是通过书写 *Controller*、*Action* 间接定义路由，而 *Laravel* 中是在 *route* 路由文件中直接定义路由入口
> 2. *Laravel* 提供 *ORM* 对象关系映射，使读写数据库的操作更加简单
> 3. *Laravel* 提供更多的 *Artisan* 命令和脚手架开发
> 4. *Laravel* 的 *Composer* 扩展包比 *Yii* 框架更多，开发更加高效

算法篇

- 扩展阅读

> 本书的 *GitHub* 地址：*https://github.com/todayqq/PHPerInterviewGuide*

算法可以说是大厂的必考题，对于算法，一定要理解其中的精髓、原理。

- 冒泡排序

冒泡排序的原理：一组数据，比较相邻数据的大小，将值小数据在前面，值大的数据放在后面。

```
function bubble_sort($arr)
{
    $count = count($arr);
    if (0 == $count) {
        return false;
    }

    for($i = 0; $i < $count; $i++){
        for($j = 0; $j< $count-1-$i; $j++){
             if($arr[$j] > $arr[$j+1]){
                 $temp        = $arr[$j];
                 $arr[$j]     = $arr[$j+1];
                 $arr[$j+1]   = $temp;
             }
        }
    }
    return $arr;
}
```

这样的一个数组 `array(6, 3, 8, 2, 9, 1)` ，排序过程是怎样的？细节问题不在过多论述，有兴趣可以从扩展阅读中寻找答案。

- 快速排序

快速排序是对冒泡排序的一种改进。

实现过程是：

1. 先从数列中取出一个数作为基准数。
2. 分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。
3. 再对左右区间重复第二步，直到各区间只有一个数。

```
function quick_sort(array $list) {
    $len = count($list);
```

```
    if ($len <= 1) {
        return $list;
    }
    $pivotValue = $list[0];
    $left = array();
    $right = array();
    for ($i = 1; $i < $len; $i++) {
        if ($list[$i] < $pivotValue) {
            $left[] = $list[$i];
        }else{
            $right[] = $list[$i];
        }
    }
    $left = quick_sort($left);
    $right = quick_sort($right);
    return array_merge($left, array($pivotValue), $right);
}
```

- 二分查找（折半查找）

实现思想：将表中间位置记录的关键字与查找关键字比较，如果两者相等，则查找成功；否则利用中间位置记录将表分成前、后两个子表，如果中间位置记 录的关键字大于查找关键字，则进一步查找前一子表，否则进一步查找后一子表。

```
function binSearch($arr, $target){
    $height = count($arr)-1;
    $low = 0;

    while($low <= $height){
        $mid = floor(($low+$height)/2);//获取中间数

        //两值相等，返回
        if($arr[$mid] == $target){
            return $mid;

        //元素比目标大，查找左部
        } elseif ($arr[$mid] < $target){
            $low = $mid + 1;

        //元素比目标小，查找右部
        } elseif ($arr[$mid] > $target){
            $height = $mid - 1;
        }
    }
    return "查找失败";
}
```

扩展阅读

- PHP 冒泡排序
- php四种基础算法
- PHP实现各种经典算法
- PHP常见算法-面试篇
- php实现二分查找法

MySQL 篇

- 扩展阅读

- MyISAM 和 InnoDB 的区别

> 1. *MyISAM* 查询效率更高，但是不支持事物
> 2. *InnoDB* 插入、更新较高，支持事物处理
> 3. *MyISAM* 支持表锁， *InnoDb* 支持行锁
> 4. *MyISAM* 是默认引擎，*InnoDB* 需要指定
> 5. *InnoDB* 不支持 *FULLTEXT* 类型的索引

- 什么是索引，作用是什么？常见索引类型有那些？Mysql 建立索引的原则？

> 索引是一种特殊的文件，它们包含着对数据表里所有记录的引用指针，相当于书本的目录。其作用就是加快数据的检索效率。常见索引类型有主键、唯一索引、复合索引、全文索引。
>
> - 索引创建的原则
>   - 最左前缀原理
>   - 尽量的去扩展索引，而不是重复的新建新索引

- SQL 语句的优化方式？

> 1. 避免使用 *Like* 模糊查询
> 2. 只列出需要查询的字段，而不是所有
> 3. 不在 *MySQL* 中进行运算，减轻 *MySQL* 的压力
> 4. 经常查询的字段，创建合适的索引，提高查询效率

- 数据库的优化方式？

> 1. 合理的设计表结构、表索引
> 2. 不在数据库中做运算
> 3. 控制单表数据量
> 4. 数据量过大进行读写分离，使用*INNODB*存储引擎
> 5. 不再数据表中存储图片

- 什么是 MySQL 慢查询？又该如何优化？

> *MySQL* 中查询超过指定时间的语句，被称之为「慢查询」。该如何优化呢？优化 *SQL* 语句，创建合适的索引，如以上两个问题。

- MySQL 分库分表怎么设计

> 1. 垂直分表
>
> 垂直分表在日常开发和设计中比较常见，通俗的说法叫做“大表拆小表”，某个表中的字段比较多，可以新建立一张“扩展表”，将不经常使用或者长度较大的字段，拆分出去放到“扩展表”中。
>
> 1. 垂直分库
>
> 基本的思路就是按照业务模块来划分出不同的数据库，而不是像早期一样将所有的数据表都放到同一个数据库中。
>
> 1. 水平分表

> 水平分表也称为横向分表，比较容易理解，就是将表中不同的数据行按照一定规律分布到不同的数据库表中（这些表保存在同一个数据库中），这样来降低单表数据量，优化查询性能。
>
> 1. 水平分库分表
>
> 水平分库分表与上面讲到的水平分表的思想相同，唯一不同的就是将这些拆分出来的表保存在不同的数据库中。

- 什么是 MySQL 死锁？如何有效降低死锁？

> 死锁：死锁一般是事务相互等待对方资源，最后形成环路，而无法继续运行。
>
> 产生死锁的原因：
>
> 1. 系统资源不足；
> 2. 进程运行推进的顺序不合适；
> 3. 资源分配不当等；
>
> 如何有效降低死锁：
>
> 1. 按同一顺序访问资源；
> 2. 避免事务中的用户交互；
> 3. 保持事务简短并在一个批处理中；
> 4. 使用低隔离级别；
> 5. 使用绑定连接；

- 什么是 MySQL 的事物？事物都有那些特性？

> 事务是一个序列操作，其中的操作要么都执行，要么都不执行，它是一个不可分割的工作单位，*ACID* 四大特性是事务的基础。
>
> - 原子性：一个事务（*transaction*）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。
> - 一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。
> - 隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。
> - 持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

说起 MySQL 的事物，就会谈起并行事物而引发的问题，比如脏读、幻读和不可重复读。

- 什么是脏读、幻读和不可重复读？会产生什么问题？该如何解决？

> - 脏读 ：脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。
> - 不可重复读 ：是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务再修改数据。那么第一个事务两次读到的的数据可能是不一样的，因此称为是不可重复读。
> - 幻读：当某事物正在执行插入或删除操作同时，第二个事物也在操作此表的数据，就会显示有一行还未存在的数据，就像发生了幻觉一样。
>
> 解决办法：如果在操作事务完成数据处理之前，任何其他事务都不可以操作此数据，则可避免该问题。

扩展阅读

- MySQL索引原理及慢查询优化
- 分库分表的几种常见形式
- 大众点评订单系统分库分表实践

- MySQL 死锁问题及解决
- MySQL索引背后的数据结构及算法原理
- MySQL存储引擎InnoDB与Myisam的六大区别
- 『浅入深出』MySQL 中事务的实现
- 脏读、幻读和不可重复读

Linux、Git 篇

Linux

- 说一些常用的 Linux shell 命令

> 这个问题就不回答了，自由发挥

- Linux 硬链接和软链接有什么区别？

> 1. 硬链接不可以跨分区，软件链可以跨分区
> 2. 硬链接指向一个 *i* 节点，而软链接则是创建一个新的 *i* 节点
> 3. 删除硬链接、软链接文件，对原文件都没有什么影响，但是如果删除原文件，会导致软连接失效，硬链接无影响。

- 建立软链接(快捷方式)，以及硬链接的命令。

> 软链接： *ln -s slink source*
> 硬链接： *ln link source*

- 怎么利用 ps 查看指定进程的信息

> *ps -ef | grep pid*

- Linux 下命令有哪几种可使用的通配符？分别代表什么含义?

> *“?”* 可替代单个字符。
> *“*”* 可替代任意多个字符。
> 中括号*“[charset]”*可替代 *charset* 集中的任何单个字符，如*[a-z]*，*[abABC]*

Git

- Push 代码时发生突破如何处理？

> *1*、使用 *git stash* 将本地文件暂存
>
> *2*、更新代码 *git pull*
>
> *3*、还原暂存的内容 *git stash pop*

- 线上服务器代码出了问题如何回滚？

> *git reset —hard HEAD^*

- GitFlow 中都有那些分支？

> 两个长期维护分支
>
> - 主分支（*master*）
> - 开发分支 （*develop*）
>
> 三种短期分支
>
> - 功能分支（*feature branch*）
> - 补丁分支（*hotfix branch*）
> - 预发分支（*release branch*）

扩展阅读

- linux面试常问命令
- Linux常见面试题
- Git教程
- Gitflow 工作流

程序员如何写好一份简历？

> 程序员的简历在求职的时候，尤为重要，简历就是销售自己的明信片，一份优秀的简历，能为你带来更多的面试机会。

我自己写了不少了简历，也指导过很多朋友写过简历，同时也看过不少程序员的简历，一份好的简历应该这么写：

1. 简历的格式推荐使用 PDF，兼容性强且不易乱序
2. 简历的排版要简单明了、一目了然、结构清晰
3. 清晰罗列出个人基本信息
    - 姓名、性别
    - 学历、毕业院校
    - 电话、邮箱
    - 居住地、期望地
    - 求职岗位、期望薪水
    - 最好有个人博客、GitHub 地址
4. 技术栈
5. 工作经历
    - 哪家公司、担任职位
    - 起止时间
    - 工作职责
6. 项目经历
    - 这个项目是什么
    - 你负责处理了什么
    - 结果是什么
7. 教育经历
8. 个人评价

工作经历、项目经历中尽量避免主观表述，少一点语义词、模糊的形容词，要用语气肯定词，一句话概述成果，一定要将自己的优势和期望明晰地表达出来。

写简历的错误姿势：

1. 简历打开乱码，排版不整齐
2. 个人信息不完善（还真有不写自己年龄、毕业院校的）
3. 语气描述词太多，不够精简，完全不知所云
4. 软件开发一两年，简历中不要有精通一词，面试时分分钟被打脸
5. 最好不要写培训经历，近几年的培训机构的课程质量都不高
6. 简历页数太多，最好不好超过 3 页

简历千万不要造假，不要造假，不要造假！

- https://github.com/todayqq/resume

这是我的简历，大家可以参考一下。

写简历的神器

1. 模板

2. https://github.com/geekcompany/ResumeSample

这份模板中包含了 PHP、iOS、Android、Java、前端等等之类的工程师简历模板，写起简历来太轻松，有木有！

1. 在线工具

2. http://cv.ftqq.com/?fr=github#

在线直接编写，还可以下载、转换 PDF 文件等等。

如果你不会使用 Markdown 的话，我写了一篇 Markdown 的教程，Markdown 写作的神兵利器，你可以学习一下，Markdown 是程序员必备技能。

扩展阅读

- 程序员简历应该怎么写？
- 面试时，如何向公司提问？

结语：写给程序员的一些建议

随着 IT 培训行业雨后春笋般崛起，码农的入门门槛越来越低，每年都会输出数十万码农，对于互联网整个行业来说，这是一件好事，能促进行业的整体发展，但是对于相关开发的从业者来说，程序员的竞争也越来越大。

并不是每一个人都适合软件开发，首先你要确认是否发自内心的热爱这个领域，而不是纯粹为了「高薪职业」。

即使互联网发展的再快，对于码农这个职业，一时半会也很难被取缔，这就像建筑工地的农民工一样，存在了几千年。但是随着程序员的竞争越来越大、互联网的快速发展，码农将不再有竞争优势，或许将来的有一天，码农将和现在的农民工一样，生活在社会的底层。

或许你并怎么热爱这个行业，或许你只是为了生计，但是进入了这个行业，至少可以为自己打上软件工程师的标签，而不是只会 Hello World、Control+C、Control+V 的 Coder。

就算是面向工资编程，也需要有足够的实力，而程序员该如何精进呢？

1. 每天比别人多做一点、多学一点

想必大家都听过卖油翁中的一句话：“无他，但手熟尔。” 想要在编程领域有所突破，1W 小时定律尤为体现，编程需要不断的实践，才有提高。

编程就像打篮球，不管你看再多 NBA 的视频，只有你自己去拍打篮球的时候，才能真正的去融会贯通，每天比同事多做 1 小时，坚持下去，升职加薪不再是夸夸其谈。

2. 不断学习、持续积累、坚持复盘

IT 行业发展迅速，软件的版本迭代更新也非常快，而每年产生的新技术也越来越多，在这个行业中药持续不断的学习，不断更新自己的操作系统、知识体系。

在公司负责开发完项目之后，就真的完事了吗？从中遇到了那些问题？提出了那些解决方案？掌握了那些新技能？项目复盘、总结不可或缺。

3. 坚持写作

说起写文章，是很多理科生的痛，不善言谈，要写作，简直比登天还难。

但是程序员非常有必要养成写作的习惯，编程和写作有很多类似的地方，最为核心的共同之处在于它们都需要清晰思考的能力。很多优秀的软件工程师也是优秀的作家，文章和代码一样富有逻辑性，行文流畅、优雅。

而且写作还可以打造个人 IP，提高个人影响力，其中最典型的就是你的技术博客，就是你的最好的简历。

4. Follow the master

你现在的圈子，就决定了你将来的生活状态，也基本决定了你的技术水平，想要有所突破，就要不断的学习技术大牛们的学习方式、学习技巧，站在巨人的肩膀上，你将看的更远。

不仅要追随这些牛人的脚步，也要尝试和牛人做朋友，牛人也是普通人，都会有自己的兴趣爱好，比如巴菲特就非常喜欢玩桥牌，如果你桥牌玩的很棒的话，是不是就能有机会，在这个领域和巴菲特成为朋友呢？娱乐的同时，有幸能接收一些指点，是不是就能受益终身呢？

相信我，这些牛人就在我们的生活中，因为互联网的存在，他们离你并不遥远。

5. 坚持独立思考

在软件开发流程里面，程序员只是充当一种工具，用来实现产品经理的软件标准，最终输出可交付的代码。

许多年轻的程序员勤奋工作，从早到晚一刻不停地编码，周末也来加班，努力完成公司的一个个目标，很少会去想为什么要开发这个软件？这个软件有多大的价值？更很少去想"我的人生规划是什么"？

即使软件最终开发完毕，公司获得了利润，但是和程序员又能有多大的关联呢？谁最终能记得这个软件是你开发的呢？

你不应该只是像工具一样工作，坚持独立思考，多规划自己的未来。

PHPer 精进之路

最后在简单聊一下 PHPer 的级别，初级、中级、高级、再往上就是架构师。

简单的区分一下，初级就是指刚入门的程序员，此时的编程水平还比较稚嫩，还不足以独立开发项目的能力；

而中级就是指编程已经有了一定的火候，也积累了较多的业务经验，此时已经具备了单独开发较多项

目的能力；

而高级的工程师，和中级有很大的区别，技术层面而言，最大的不同的是：中级工程师只是知其然，而不知所已然。很多技术只是会用，而且用的很熟练，但是不清楚其中的原理，在处理一些大流量、高并发的情况下，其中的差距尤为明显。

架构师呢，是一个既需要掌控整体又需要洞悉局部瓶颈，并依据具体的业务场景给出解决方案的人。到了这个职位，就不仅仅只是技术层面了，更多的还有沟通和管理。

不知道你属于哪个级别呢？是否有考虑过又该如何进阶呢？

扩展阅读

- 关于程序员生涯的思考，30 岁以后的码农们该何去何从？
- 你的命运不是一头骡子

扩展阅读资源整理

前端篇

- 收集的前端面试题和答案
- 前端开发面试题
- 史上最全的web前端面试题汇总及答案
- 前端工程师手册
- HTTP协议：工作原理
- SSL/TLS协议运行机制的概述

协议篇

- https 原理
- HTTPS 原理解析
- HTTPS 的工作原理
- socket
- HTTP与WebSocket的区别
- 理解OAuth 2.0

后端篇

- 3年PHPer的面试总结
- 垃圾回收机制
- S.O.L.I.D 面向对象设计
- 浅谈IOC—说清楚IOC是什么
- Redis和Memcached的区别
- MySQL索引原理及慢查询优化
- 分库分表的几种常见形式
- 大众点评订单系统分库分表实践
- MySQL 死锁问题及解决
- CSRF攻击与防御
- XSS跨站脚本攻击
- PHP 设计模式系列

- MySQL存储引擎InnoDB与Myisam的六大区别
- 『浅入深出』MySQL 中事务的实现
- 脏读、幻读和不可重复读

算法篇

- PHP 冒泡排序
- php实现快速排序
- PHP实现各种经典算法
- PHP常见算法-面试篇
- php实现二分查找法

Linux、Git 篇

- linux面试常问命令
- Linux常见面试题
- Git教程
- Gitflow 工作流

其他

- 程序员简历应该怎么写？
- 程序员简历模板
- 面试时，如何向公司提问？
- 关于程序员生涯的思考，30 岁以后的码农们该何去何从？