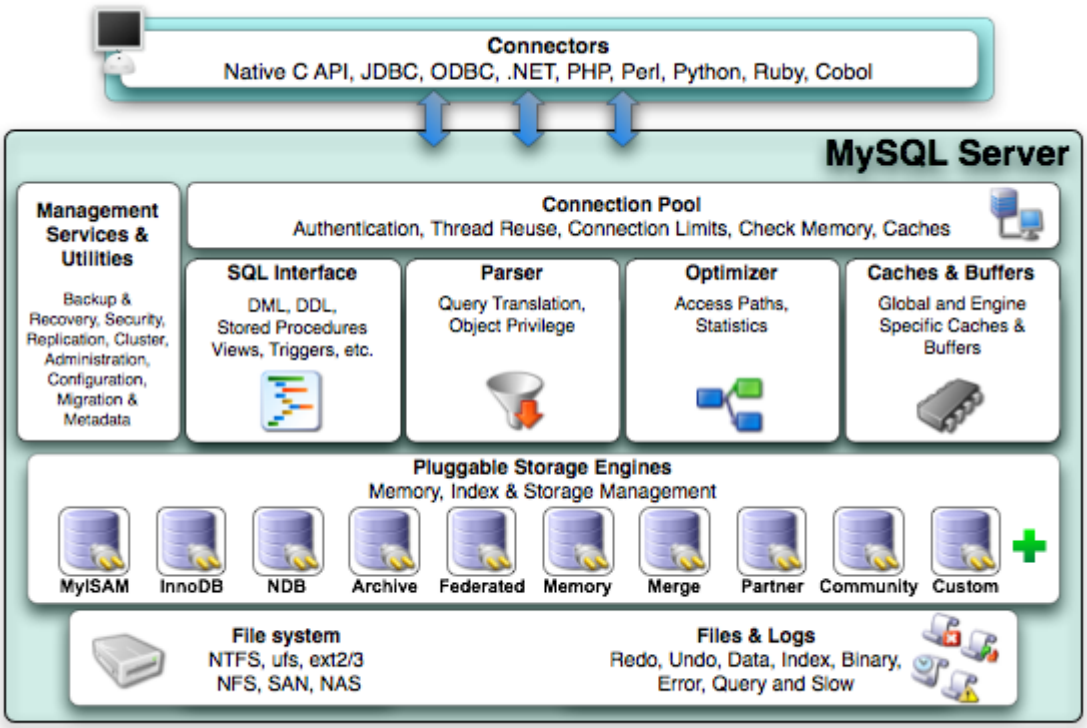
MySQL

1.MySQL体系结构

- 结构体系组成

> SQL接口，解析器，优化器，缓存，存储引擎

- 体系结构图



> Connectors：不同语言中与SQL的交互

> Management Serveices & Utilities： 系统管理和控制工具

> Connection Pool: 连接池

> SQL Interface: SQL接口

> Parser: 解析器

> Optimizer: 查询优化器

> Cache和Buffer：查询缓存

> Engine：存储引擎

2.字段类型

- 数字类型

类型	长度	备注
tinyint	1bit	
smallint	2bit	
mediumint	3bit	
int	4bit	
bigint	8bit	
float	4bit	8位精度
double	8bit	16位精度

- 字符类型

类型	长度	备注
char	2^8长度	固定长度
varchar	2^16长度	可变长度
tinytext	2^8长度	可变长度
text	2^16长度	可变长度
mediumtext	2^24长度	可变长度
longtext	2^32长度	可变长度
blob	二进制数据	

- 日期类型

类型	示例
date	'2018-7-30' 日期
datetime	'2018-7-30 11:20:01' 日期时间
timestamp	自动存储记录修改时间
time	'11:19:47' 时间
year	'2018' 年份

- 数据类型属性

关键字	含义
NULL	数据列可包含NULL值
NOT NULL	数据列不可NULL值
DEFAULT	默认值
PRIMARY KEY	主键
AUTO_INCREMENT	自动递增
UNSIGNED	无符号
CHARACTER SET name	指定字符集

3.char和varchar数据类型区别

- char

> 善于存储经常改变的值，或者长度相对固定的值，比如type、ip地址或md5之类的数据，不容易产生碎片

- varchar

> 善于存储值的长短不一的列，也是用的最多的一种类型，节省磁盘空间 保存可变长度字符串，范围0-65535（但受到单行最大64kb的限制）。比如用 varchar(30) 去存放abcd，实际使用5个字节，因为还需要使用额外1个字节来标识字串长度（0-255使用1个字节，超过255需要2个字节） update时varchar列时，如果新数据比原数据大，数据库需要重新开辟空间，这一点会有性能略有损耗，但innodb引擎下查询效率比char高一点。这也是innodb官方推荐的类型

4.存储引擎

- MyISAM

> 不支持事务，SELECT/INSERT速度较快，非聚簇索引

- InnoDB

> 支持事务

> 更新密集型表，并发场景

> 行级锁定

> 自动容灾恢复

> 外键约束

> 聚簇索引

- MyISAM和InnoDB比较

> MyISAM必须依靠操作系统来管理读取与写入的缓存，而InnoDB则是有自己的读写缓存管理机制

- Merge存储引擎(MRG_MyISAM)

> 允许将一组使用MyISAM存储引擎的并且表结构相同（即每张表的字段顺序、字段名称、字段类型、索引定义的顺序及其定义的方式必须相同）的数据表合并为一个表，方便了数据的查询。常用于分表日志查询

5.常见索引

- 索引概念

> 索引好比一本书的目录，用来更快的找到内容，索引不是越多越好，索引也需要占用空间

- 索引分类

- 索引创建

```
普通索引    ALTER TABLE `table_name` ADD INDEX index_name (`column`)
唯一索引    ALTER TABLE `table_name` ADD UNIQUE (`column`)
主键索引    ALTER TABLE `table_name` ADD PRIMARY KEY (`column`)
全文索引    ALTER TABLE `table_name` ADD FULLTEXT (`column`)
组合索引    ALTER TABLE `table_name` ADD INDEX index_name (`column1`, `column2`, `column3`)
```

- 索引区别

> 普通索引：最基本的索引，没有任何限制

> 唯一索引：与"普通索引"类似，不同的就是：索引列的值必须唯一，但允许有空值

> 主键索引：它是一种特殊的唯一索引，不允许有空值

> 全文索引：仅可用于MyISAM表，针对较大的数据，生成全文索引很耗时好空间

> 组合索引：为了更多的提高mysql效率可建立组合索引，遵循"最左前缀"原则

6.聚族索引与非聚族索引的区别

> 按物理存储分类：聚簇索引(clustered index)、非聚簇索引(non-clustered index)

> 聚簇索引的叶子节点就是数据节点，而非聚簇索引的叶子节点仍然是索引节点，只不过有指向对应数据块的指针

7.事务机制

- 数据库事务(Database Transaction)

> 是指作为单个逻辑工作单元执行的一系列操作，要么完全执行，要么完全地不执行

- ACID特性

> 原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)

> 原子性：指事务包含的所有操作要么全部成功，要么全部失败回滚

> 一致性：指事务必须使数据库从一个一致的状态变到另外一个一致的状态，也就是执行事务之前和之后的状态都必须处于一致的状态

> 隔离性：指当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离

> 持久性：指一个事务一旦被提交了，那么对于数据库中的数据改变就是永久性的，即便是在数据库系统遭遇到故障的情况下也不会丢失提交事务的操作

- 事务隔离性

> 当多个线程都开启事务操作数据库中数据时，数据库系统要能进行隔离操作，以保证各个线程获取数据的准确性

- 无隔离产生问题

> 脏读(Dirty Read)：一个事务处理过程里读取了另一个未提交的事务中的数据

> 不可重复读(NonRepeatable Read)：一个事务范围内多次查询却返回了不同的数据值

> 幻读(Phantom Read)：在一个事务中读取到了别的事务插入的数据，导致前后不一致

- 隔离级别

> Serializable、Repeatable read、Read committed、Read uncommitted

隔离级别	脏读	不可重复读	幻读
未提交读(Read uncommitted)	可能	可能	可能
已提交读(Read committed)	不可能	可能	可能
可重复读(Repeatable read)	不可能	不可能	可能
可串行化(Serializable)	不可能	不可能	不可能

- 锁方案

> 一次性锁、两端锁

- MySQL锁方案

> 表锁:对一整张表加锁，并发能力低下

> 行锁:只锁住特定行的数据，并发能力强，MySQL一般都是用行锁来处理并发事务

8.BTree与BTree-/BTree+索引原理

- BTree

> 二叉树导致树高度非常高，逻辑上很近的节点，物理上非常远，无法利用局部性，IO次数多，查找效率低

- BTree-

> 每个节点都是二元数组[key,data]，所有节点都可以存储数据,key为索引，data为索引外的数据。插入删除数据会破坏BTree性质，插入数据时候，需要对数据进行分裂、合并、转移等操作保持BTree性质，造成IO操作频繁

- BTree+

> 非叶子节点不存储data，只存储索引key，只有叶子节点才存储data

- MySQL中的BTree+

> 在经典BTree+的基础上进行了优化，增加了顺序访问指针。在BTree+的每个叶子节点增加了一个指向相邻叶子节点的指针，形成了带顺序访问指针的BTree+，提高了区间访问性能

参考资料

- MySQL体系结构
- MySQL字符数据类型char与varchar的区别
- MySQL有哪些索引
- MySQL聚簇索引
- MySQL事务处理机制
- MySQL索引背后的数据结构及算法原理

Redis

1.Redis主要特点

> Redis是一个高性能的KV数据库，支持丰富的数据类型，提供多种语言的API，支持数据的持久化，性能极高，常用于Cache

2.Redis数据类型

> STRING,HASH,LIST,SET,SORTEDSET,GEO,PUB/SUB

3.跳跃表与Redis

- 跳跃表

> 跳跃表是一种随机化数据结构，查找、添加、删除操作都可以在对数期望时间下完成

- 跳跃表在Redis的应用

> 跳跃表在Redis的唯一作用，就是实现有序集数据类型

> 跳跃表将指向有序集的score值和member域的指针作为元素，并以score值为索引对有序集元素进行排序

参考资料

- 跳跃表

Web

1.JavaScript事件的三个阶段

> 捕获，目标，冒泡阶段，低版本IE不支持捕获阶段

2.闭包原理及应用

> 闭包就是将函数内部和函数外部连接起来的一座桥梁

> 读取函数内部的变量，让这些变量的值始终保持在内存中

3.跨域

- 什么是跨域

> 简单地理解就是因为JavaScript同源策略的限制

- 什么是同源策略

> 在JavaScript中，同源策略是一个很重要的安全理念，它保证数据的安全性有着重要的意义。同源策略规定跨域之间脚本是隔离的

- 相同域

> 相同协议，相同端口，相同host

- 跨域资源

> 单向数据请求(JSONP)、双向消息通信

4.JSONP原理

> HTML里面所有带src属性的标签都可以跨域，如iframe，img，script等

> 所以可以把需要跨域的请求改成用script脚本加载即可，服务器返回执行字符串，但是这个字符串是在window全局作用域下执行的，你需要把他返回到你的代码的作用域内，这里就需要临时创建一个全局的回调函数，并把到传到后台，最后再整合实际要请求的数组，返回给前端，让浏览器直接调用，用回调的形式回到你的原代码流程中

5.CSS选择器的优先级

- 优先级

> !important > 内联 > id选择器 > 类选择器 > 标签选择器

- CSS选择器的种类：

> 1.id选择器(# myid)

> 2.类选择器(.myclassname)

> 3.标签选择器(div, h1, p)

> 4.相邻选择器(h1 + p)

> 5.子选择器(ul > li)

> 6.后代选择器(li a)

> 7.通配符选择器(*)

> 8.属性选择器(a[rel = "external"])

> 9.伪类选择器(a: hover, li:nth-child)

6.CSS盒子模型

> 属性：element、padding、border、margin

> 显示方式：block、inline、flex

7.CSS清除浮动

8.相对定位relative、浮动float、绝对定位absolute区别

> 相对定位：按一定偏移量依次排列定位

> 浮动定位：浮动框可以向左/右便宜，不影响后续框

> 绝对定位：每个定位框都是一个单独的图层，不会对其他层框的定位产生影响

9.VUE双向绑定原理

> 发布者-订阅模式(backbone.js)、脏值检查(angular.js)、数据劫持(vue.js)

> vue.js则是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调

10.性能优化

> js/css合并、使用CDN、图片合并、利用HTTP缓存机制、开启gzip压缩、浏览器加载/解析/渲染机制

参考资料

- 大部分人都会做错的经典JS闭包面试题
- 学习Javascript闭包
- 跨域-知识
- 剖析Vue原理&实现双向绑定MVVM
- 网站性能优化实践总结

安全问题

1.CSRF攻击

> 全称Cross-site request forgery，跨站请求伪造，攻击者盗用了你的身份，以你的名义发送恶意请求

> 登录受信任网站A，并在本地生成Cookie。在不登出A的情况下，访问危险网站B

> 如何防御:在客户端页面增加随机数、提交表单增加CSRF token

- 具体实例

2.XSS攻击

> 全称Cross SiteScript，跨站脚本攻击

> 分类:存储型XSS、反射型XSS、DOM-XSS

> 如何防御:从输入到输出都需要过滤、转义

- 具体实例

3.SQL注入

> SQL注入就是通过操作输入来修改后台SQL语句达到代码执行进行攻击目的的技术

- 如何防御

> 严格限制web应用数据库权限，给用户提供仅仅能满足工作的最低权限

> 检查输入的数据是否具有所期望的数据格式

> 避免打印SQL错误信息，暴露SQL语句

- 具体实例

```
SELECT * FROM table WHERE id=1;DELECT FROM table WHERE id=1;
SELECT * FROM table WHERE id=1 AND (SELECT COUNT(*) FROM try_table)>-1;
```

4.IP地址能被伪造吗

5.include请求参数

6.md5逆向原理

7.DOS攻击

参考资料

- 浅谈CSRF攻击方式
- XSS攻击和防御详解
- 避免SQL注入

网络协议

1.UDP的主要特点

> 无连接的、尽最大努力交付、面向报文、没有拥塞控制、支持一对一，一对多和多对多的交互通信、首部开销小

2.TCP握手三次，断开四次，TIME-WAIT

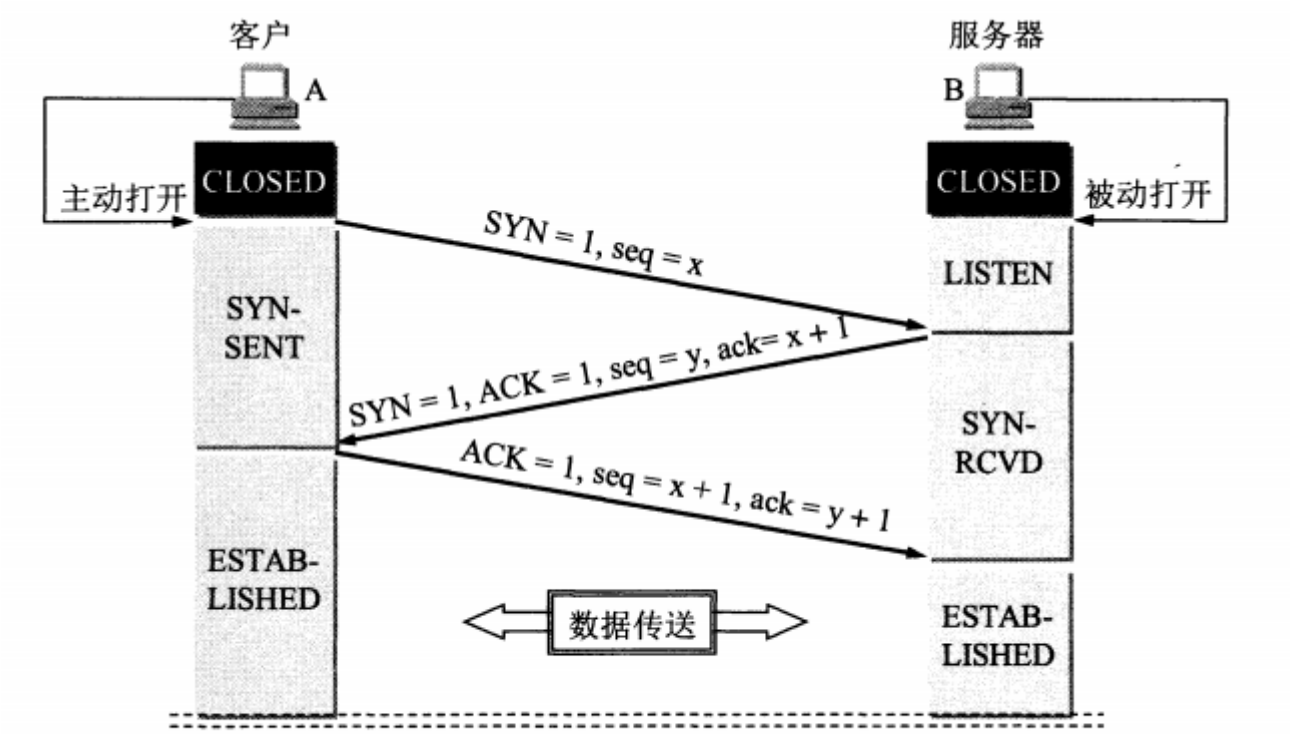


图 5-28 用三报文握手建立 TCP 连接

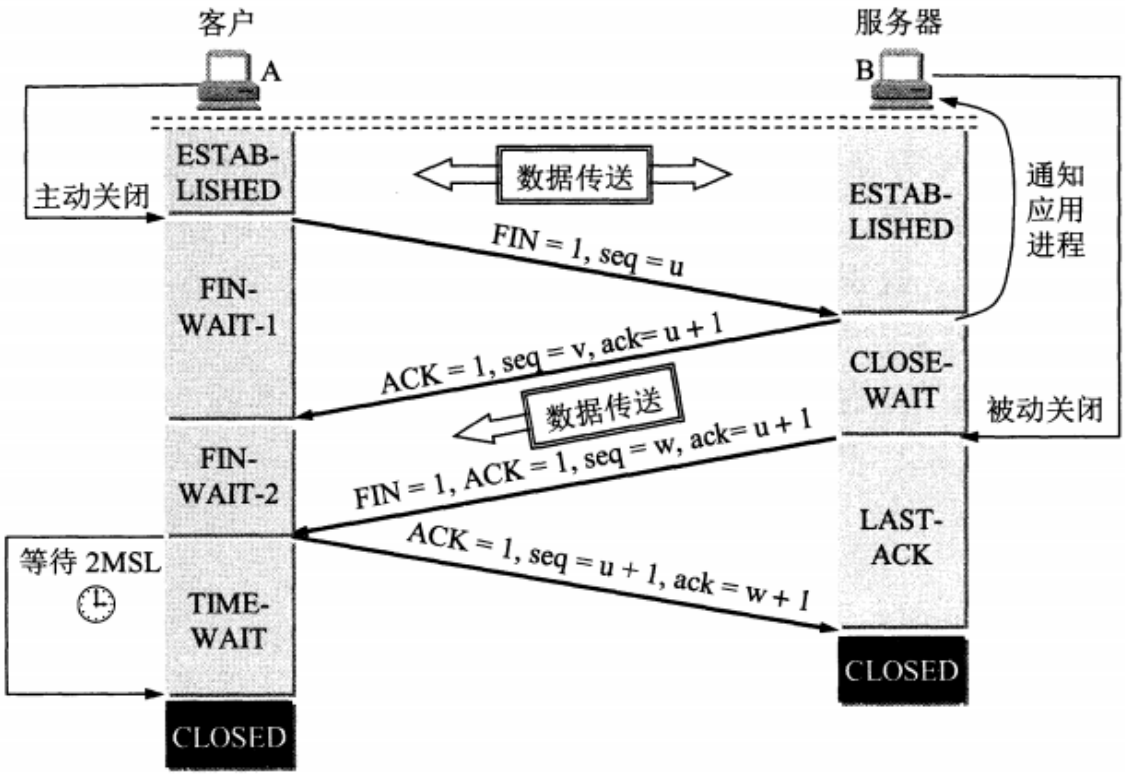


图 5-29　TCP 连接释放的过程

3.socket

> socket：网络中进程通过socket进行通信

> 基本操作：socket()、bind()、listen()/connect()、accept()、read()/write()、close()

4.HTTP协议

> HTTP方法

> HTTP首部字段

> HTTP状态码

> HTTP2特点：二进制协议、多路复用、头压缩、服务器推送

5.websocket协议

> HTTP握手，通过首部字段upgrade标识websocket，切换协议，进行通信

6.GET与POST请求方式区别

> GET请求会将参数跟在URL后进行传递，而POST请求则是作为HTTP消息实体内容发送，在AJAX请求中，这种区别对用户不可见

> GET方式对传输有大小限制，而POST则大的多

> GET请求和数据会被浏览器缓存起来

> GET方式和POST方式传递的数据在服务器的获取也不同

参考资料

HTTP/2协议–特性扫盲篇 Socket通信原理和实践

PHP

1.echo、print、print_r、var_dump的区别

> echo：输出一个或多个字符串

> print：输出字符串

> print_r：打印关于变量的易于理解的信息

> var_dump：打印关于变量的易于理解的信息(带类型)

2.超全局变量

> `$GLOBALS`、`$_SERVER`、`$_GET`、`$_POST`、`$_FILES`、`$_COOKIE`、`$_SESSION`、`$_REQUEST`、`$_ENV`

3.PHP支持回调的函数，实现一个

> array_map、array_filter、array_walk、usort

> is_callable + callbacks + 匿名函数实现

4.发起HTTP请求有哪几种方式，它们有何区别

> cURL、file_get_contents、fopen、fsockopen

5.对象关系映射/ORM(Object Relational Mapping)

- 优点

> 缩短编码时间、减少甚至免除对model的编码，降低数据库学习成本

> 动态的数据表映射，在表结构发生改变时，减少代码修改

> 可以很方便的引入附加功能(cache层)

- 缺点

> 映射消耗性能、ORM对象消耗内存

> SQL语句较为复杂时，ORM语法可读性不高(使用原生SQL)

6.MVC的理解

> MVC架构中M是指数据模型，V是指用户界面，C则是控制器；MVC的思想是模块化分离，为了代码的重用和增强代码的维护性和扩展性出发的，其中MVC的实现有一定的思想和原则

7.类的静态调用和实例化调用

> 调用前初始化，调用时初始化

8.常见PHP框架特点

- ThinkPHP

> URL模式：系统支持普通模式、PATHINFO模式、REWRITE模式和兼容模式的URL方式，支持不同的服务器和运行模式的部署，配合URL路由功能，可以随心所欲地构建需要的URL地址和进行SEO优化工作

> 查询语句：内建丰富的查询机制，包括组合查询、符合查询、区域查询、统计查询、定位查询、动态查询、和原生查询、让数据查询简洁高效

> 分组模块：不用担心大项目的分工协调和部署问题，分组模块解决跨项目的难题

- Laravel

> 包含Web开发、包管理、代码生成、ORM、常见组件(cache/log)、路由管理、中间件、依赖注入

- Biny

> 支持跨库连表，条件复合筛选，查询PK缓存等

> 同步异步请求分离，类的自动化加载管理

> 支持Form表单验证，支持事件触发机制

> 支持浏览器端调试，快速定位程序问题和性能瓶颈

> 具有sql防注入，html自动防xss等特性

9.设计模式(design pattern)

> 设计模式是对软件设计中普遍存在(反复出现)的各种问题，所提出的解决方案

- 常见设计模式:

> 单例模式
>
> > 定义：确保一个类只有一个实例，并提供一个全局访问点 使用场景：单入口模式

> 简单工厂模式
>
> > 定义：简单工厂模式的工厂类一般是使用静态方法，通过接收的参数的不同来返回不同的对象实例 使用场景：对象管理(初始化)

> 工厂方法模式
>
> > 定义：定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类 设计原则：依赖抽象，不要依赖具体类。(依赖倒置) 使用场景：

> 抽象工厂模式

> 定义：提供了一接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类 设计原则：依赖抽象，不要依赖具体类。(依赖倒置) 使用场景：

> 策略模式

> > 定义：定义算法族，分别封装起来，让他们之间可以互相替换，此模式让算法的变化独立于使用算法的客户 设计原则：1.封装变化。2.多用组合，少用继承。3.针对接口编程，不针对实现编程。 使用场景：SDK封装

> 观察者模式

> > 定义：在对象之间定义一对多的依赖，这样一来当一个对象改变状态，依赖它的对象都会收到通知，并自动更新 设计原则：为了交互对象之间的松耦合设计而努力 使用场景：回调机制

> 适配器模式

> > 定义：将一个类的接口，转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间 使用场景：代理服务器协议转换

> 装饰者模式

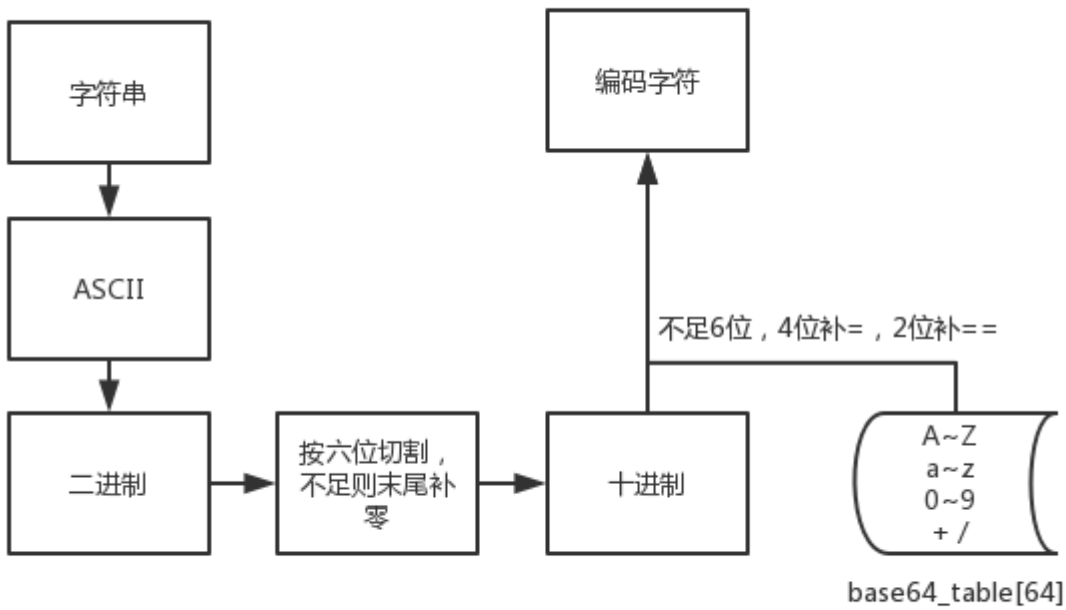
> > 定义：动态的将责任附加到对象上。若要扩展功能，装饰者提供了比继承更有弹性的替代方案 设计原则：对扩展开放，对修改关闭 使用场景：路由功能

10.工厂方法模式与抽象工厂模式区别

> 工厂方法模式只有一个抽象产品类，而抽象工厂模式有多个

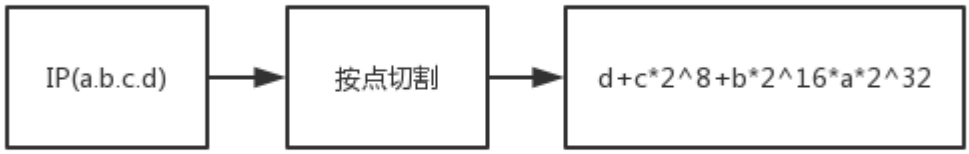
> 工厂方法模式的具体工厂类只能创建一个具体产品类的实例，而抽象工厂模式可以创建多个

11.base64编码原理



base64编码原理

12.ip2long实现



ip2long实现

13.代码执行过程

> PHP代码 => 启动php及zend引擎，加载注册拓展模块 => 对代码进行词法/语法分析 => 编译成opcode(opcache) => 执行opcode

> 当前作用域分配内存，充当运行栈，局部变量分配在当前栈，函数调用时压栈，返回时出栈

14.弱类型变量如何实现

> PHP中声明的变量，在zend引擎中都是用结构体zval来保存，通过共同体实现弱类型变量声明

15.垃圾回收机制

> 引用计数器

16.进程间通信方式

> 消息队列、socket、信号量、共享内存、信号、管道

17.链式调用实现

> 类定义一个内置变量，让类中其他定义方法可访问到

18.多进程同时写一个文件

> 加锁、队列

19.PHP拓展

20.PHP7新特性

> 标量类型声明、返回值类型声明、通过define()定义常量数组、匿名类、相同命名空间类一次性导入

21.PHP7底层优化

> ZVAL结构体优化，占用由24字节降低为16字节

> 内部类型zend_string，结构体成员变量采用char数组，不是用char*

> PHP数组实现由hashtable变为zend array

> 函数调用机制，改进函数调用机制，通过优化参数传递环节，减少了一些指令

22.构造函数和析构函数

23.PHP不实例化调用方法

> CLASS::METHOD() 静态方法

参考资料

- 深入理解PHP内核
- PHP中的回调、匿名函数与闭包
- 从PHP 5.6.x 移植到 PHP 7.0.x
- PHP7革新与性能优化
- 常用设计模式汇总
- 腾讯开源Biny框架
- 类与对象

服务器

1.进程、线程、协程区别

- 进程(process)

> 进程是一个程序在一个数据集中的一次动态执行过程，可以简单理解为“正在执行的程序”，它是CPU资源分配和调度的独立单位

- 线程(thread)

> 线程是在进程之后发展出来的概念。 线程也叫轻量级进程，它是一个基本的CPU执行单元，也是程序执行过程中的最小单元，由线程ID、程序计数器、寄存器集合和堆栈共同组成。一个进程可以包含多个线程

- 协程(coroutine)

> 协程是一种用户态的轻量级线程，又称微线程，英文名Coroutine，协程的调度完全由用户控制

2.Linux进程

> 进程属性：进程号pid、父进程号ppid、进程组号pgid

> 进程状态：就绪、运行、可中断、不可中断、僵死、停止

3.反向代理

- 概述

> 反向代理（Reverse Proxy）方式是指以代理服务器来接受Internet上的连接请求，然后将请求转发给内部网络上的服务器；并将从服务器上得到的结果返回给Internet上请求连接的客户端，此时代理服务器对外就表现为一个服务器

- 工作原理

> 反向代理服务器通常有两种模型：作内容服务器的替身、作为内容服务器的负载均衡器

4.负载均衡

- 概述

> 负载均衡(Load Balance)，意思是将负载(工作任务，访问请求)进行平衡、分摊到多个操作单元(服务器，组件)上进行执行。是解决高性能，单点故障(高可用)，扩展性(水平伸缩)的终极解决方案

- 负载均衡原理

> 系统的拓展可以分为纵向(垂直)拓展和横向(水平)拓展

> 采用横向拓展方式，通过添加机器来满足大型网站服务的处理能力

> 应用集群：将同一应用部署到多台机器上，组成处理集群，接收负载均衡设备分发的请求，进行处理，并返回相应数据。

> 负载均衡设备：将用户访问的请求，根据负载均衡算法，分发到集群中的一台处理服务器

- 负载均衡分类

> DNS负载均衡：DNS服务器配置多个A记录，记录对应的服务器构成集群

> IP负载均衡：在网络层通过修改请求目标地址进行负载均衡

> 链路层负载均衡：通信协议的数据链路层修改mac地址，进行负载均衡，最广泛方式

> 混合型负载均衡

5.nginx中fastcgi_pass监听，unix socket和tcp socket的区别

> nginx和fastcgi通信方式有两种:TCP、unix socket

> TCP和socket方式区别

> socket可以很方便进行进程通信，可以使用字节流和数据队列方式，而管道通信只能通过字节流

> socket比TCP方式消耗资源更少，高并发时tcp方式更稳定

> TCP方式做负载均衡更方便

6.消息队列

> https://cloud.tencent.com/developer/article/1006035 https://www.jianshu.com/p/689ce4205021

参考资料

- 进程、线程与协程的比较
- 反向代理服务器的工作原理

- 大型网站架构系列：负载均衡详解
- nginx和php-fpm 通信使用unix socket还是TCP，及其配置

业务设计

1.网易盖楼

2.秒杀设计

3.消息队列

4.共享SESSION

5.下单后30分钟未支付取消订单

6.IP对应省市效率尽可能高

7.详细描述输入地址到打开网页过程

参考资料

线上故障

1.客户端热更新失败

2.Redis实例used_memory达到80%

3.游戏任务完成了进度未更新

4.测试服HTTP请求未响应

5.游戏账号被盗

参考资料

个人简历

自我介绍

离职原因

1.跳槽频繁

2.这次换工作原因

职业规划

准备问题

- 1.工作挑战大不大?
- 2.项目开发是否写测试用例，项目上线先是否会进行压力测试
- 3.业务前景如何?
- 4.技术氛围如何?
- 5.根据这次面试，对个人进行评价，帮助成长
- 6.融资计划
- 7.是否有加班费/调休，公司福利，社保公积金缴纳基数

结束声明

> 本资料仅供参考，不保证正确性

> 作者：凌枫 Email：colinlets@gmail.com 链接：https://github.com/colinlet/PHP-Interview-QA