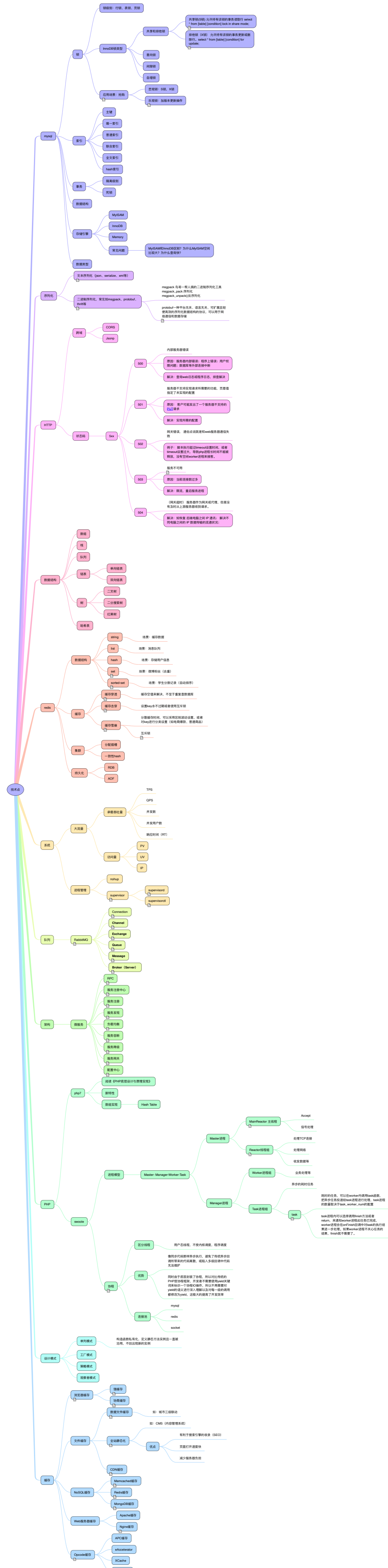

技术点
mysql
锁
锁级别：行锁、表锁、页锁
InnoDB锁类型
共享和排他锁
共享锁(S锁) 允许持有该锁的事务读取行 select * from [table] [condition] lock in share mode;
排他锁（X锁） 允许持有该锁的事务更新或删除行。select * from [table] [condition] for update;
意向锁
间隙锁
自增锁
应用场景：抢购
悲观锁：S锁、X锁
乐观锁：加版本更新操作
索引
主键
唯一索引
普通索引
联合索引
全文索引
hash索引
事务
隔离级别
死锁
数据结构
存储引擎
MyISAM
InnoDB
Memory
常见问题
MyISAM和InnoDB区别？为什么MyISAM空间比较大？为什么查询快？
数据类型
序列化
文本序列化（json、serialize、xml等）
二进制序列化，常见如msgpack、protobuf、thrift等
msgpack 鸟哥一帮人搞的二进制序列化工具
msgpack_pack 序列化
msgpack_unpack()反序列化
protobuf一种平台无关、语言无关、可扩展且轻便高效的序列化数据结构的协议，可以用于网络通信和数据存储
HTTP
跨域
CORS
Jsonp
状态码
5xx
500
内部服务器错误
原因：服务器内部错误；程序上错误；用户权限问题；数据库等外部连接中断
解决：查询web日志或程序日志，排查解决
501
服务器不支持实现请求所需要的功能，页眉值指定了未实现的配置
原因： 客户可能发出了一个服务器不支持的PUT请求
解决：实现所需的配置
502
网关错误， 通俗点说就是和web服务器通信失败
例子： 脚本执行超过timeout设置时间，或者timeout设置过大，导致php进程长时间不能被释放，没有空闲worker进程来接客。
503
服务不可用
原因：当前连接数过多
解决：限流，重启服务进程
504
（网关超时） 服务器作为网关或代理，但是没有及时从上游服务器收到请求。
解决：如恢复 后端电脑之间 IP 通讯； 解决不同电脑之间的 IP 数据传输的流通状况；
数据结构
数组
栈
队列
链表
单向链表
双向链表
树
二叉树
二分搜索树
红黑树
哈希表
redis
数据结构
string
场景：缓存数据
list
场景：消息队列
hash
场景：存储用户信息
set
场景：微博粉丝（去重）
sorted-set
场景：学生分数记录（自动排序）
缓存
缓存穿透
缓存空值来解决，不至于重复查数据库
缓存击穿
设置key永不过期或者使用互斥锁
缓存雪崩
分散缓存时间，可以采用区别波动设置，或者对key进行分类设置（如电商爆款、普通商品）
互斥锁
集群
分配插槽
一致性hash
持久化
RDB
AOF
系统
大流量
承载吞吐量
TPS
QPS
并发数
并发用户数
响应时间（RT）
访问量
PV
UV
IP
进程管理
nohup
supervisor
supervisord
supervisorctl
队列
RabbitMQ
Connection
Channel
Exchange
Queue
Message
Broker（Server）
架构
微服务
RPC
服务注册中心
服务注册
服务发现
负载均衡
服务容断
服务降级
服务网关
配置中心
PHP
php7
阅读《PHP底层设计与原理实现》
新特性
数组实现
Hash Table
swoole
进程模型
Master- Manager-Worker-Task
Master进程
MainReactor 主线程
Accept
信号处理
Reactor线程组
处理TCP连接
处理网络
收发数据等
Manager进程
Worker进程组
业务处理等
Task进程组
异步的耗时任务
task
耗时的任务，可以在worker内调用task函数，把异步任务投递给task进程进行处理，task进程的数量取决于task_worker_num的配置
task进程内可以选择调用finish方法或者return，来通知worker进程此任务已完成，worker进程会在onFinish回调中对task的执行结果进一步处理。如果worker进程不关心任务的结果，finish就不需要了。
协程
区分线程
用户态线程，不受内核调度，程序调度
优势
像同步代码那样异步执行，避免了传统异步回调所带来的代码离散，或陷入多级回调中代码无法维护
同时由于底层封装了协程，所以对比传统的PHP层协程框架，开发者不需要使用yield关键词来标识一个协程IO操作，所以不再需要对yield的语义进行深入理解以及对每一级的调用都修改为yield，这极大的提高了开发效率
连接池
mysql
redis
socket
设计模式
单列模式
构造函数私有化，定义静态方法实例且一直被沿用，不回出现新的实例
工厂模式
策略模式
观察者模式
缓存
浏览器缓存
强缓存
协商缓存
文件缓存
数据文件缓存
如：城市三级联动
全站静态化
如：CMS（内容管理系统）
优点
有利于搜索引擎的收录（SEO）
页面打开速度快
减少服务器负担
CDN缓存
NoSQL缓存
Memcached缓存
Redis缓存
MongoDB缓存
Web服务器缓存
Apache缓存
Nginx缓存
Opcode缓存
APC缓存
eAccelerator
XCache
OPchache

/技术点 / *mysql* / 锁 / *InnoDB*锁类型 / 共享和排他锁 / 共享锁*(S*锁*)* 允许持有该锁的事务读取行 *select * from [table] [condition] lock in share mode;*

符合条件的rows上都加了共享锁，可以共享读， 但无法修改这些记录直到你这个加锁的过程执行完成

/技术点 / *mysql* / 锁 / *InnoDB*锁类型 / 共享和排他锁 / 排他锁（*X*锁） 允许持有该锁的事务更新或删除行。

如果一个事务对对象加了排他锁，其他事务就不能再给它加任何锁

/技术点 / *mysql* / 锁 / 应用场景：抢购

大多数情况是不会使用数据库直接来应对大流量的（mysql最大连接数100大概），这里理解为少数情况或研究

/技术点 / *mysql* / 锁 / 应用场景：抢购 / 乐观锁：加版本更新操作

实践证明，事务中update操作会加行锁，通过版本来锁定更新记录，对于版本不一致的事务就可以做失败处理

/技术点 / 序列化 / 文本序列化（*json*、*serialize*、*xml*等）

优点：可读性

/技术点 / 序列化 / 二进制序列化，常见如*msgpack*、*protobuf*、*thrift*等

优点：数据长度缩减，传输速度提升

/技术点 / 序列化 / 二进制序列化，常见如*msgpack*、*protobuf*、*thrift*等 / *protobuf*一种平台无关、语言无关、可扩展且轻便高效的序列化数据结构的协议，可以用于网络通信和数据存储

谷歌的grpc通过.protocal定义数据格式，生成服务端rpc代码

/技术点 / *HTTP* / 状态码 / *5xx* / *503* / 服务不可用

因暂时超载或临时维护，您的 Web 服务器目前无法处理 HTTP 请求

/技术点 / *redis* / 数据结构 / *set*

集合的概念就是一堆不重复值的组合，可使用sort对其排序

/技术点 / *redis* / 数据结构 / *sorted-set*

和Set相比，Sorted Set是将set中的元素增加了一个权重参数score，使得集合中的元素能够按score进行有序排列

/技术点 / *redis* / 缓存 / 缓存穿透

指查询一个数据库一定不存在的数据

/技术点 / *redis* / 缓存 / 缓存击穿

缓存击穿，是指一个key非常热点，在不停的扛着大并发，大并发集中对这一个点进行访问，当这个key在失效的瞬间，持续的大并发就穿破缓存，直接请求数据库，就像在一个屏障上凿开了一个洞。
比如 电商项目中爆款的货物

/技术点 / *redis* / 缓存 / 缓存雪崩

指在某一个时间段，缓存集中过期失效

/技术点 / *redis* / 缓存 / 缓存雪崩 / 互斥锁

只允许一个请求去重建缓存，其他请求等待缓存重建执行完，重新从缓存中获取数据。

/技术点 / 系统 / 进程管理 / *supervisor*

进程管理，提供管理界面（restart、stop、start）

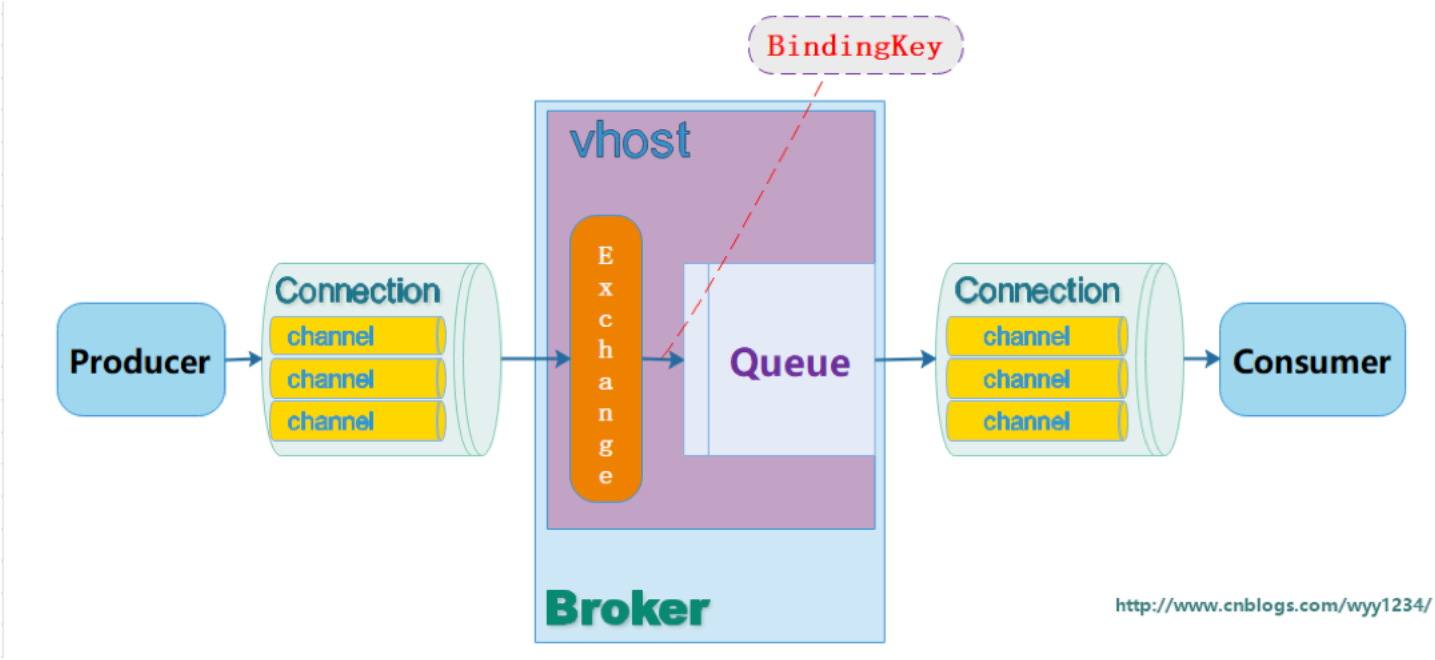
/技术点 / 系统 / 进程管理 / *supervisor* / *supervisord*

运行 Supervisor 时会启动一个进程 supervisord，它负责启动所管理的进程，并将所管理的进程作为自己的子进程来启动，而且可以在所管理的进程出现崩溃时自动重启。

/技术点 / 系统 / 进程管理 / *supervisor* / *supervisorctl*

是命令行管理工具，可以用来执行 stop、start、restart 等命令，来对这些子进程进行管理。

/技术点 /队列 / *RabbitMQ*



/技术点 /队列 / *RabbitMQ* / *Connection*

连接，对于RabbitMQ而言，其实就是一个位于客户端和Broker之间的TCP连接。

/技术点 /队列 / *RabbitMQ* / *Channel*

信道，仅仅创建了客户端到Broker之间的连接Connection后，客户端还是不能发送消息的。需要在Connection的基础上创建Channel，AMQP协议规定只有通过Channel才能执行AMQP的命令，一个Connection可以包含多个Channel。之所以需要Channel，是因为TCP连接的建立和释放都是十分昂贵的

/技术点 /队列 / *RabbitMQ* / *Exchange*

接受生产者发送的消息，并根据Binding规则将消息路由给服务器中的队列。ExchangeType决定了Exchange路由消息的行为，例如，在RabbitMQ中，ExchangeType有Direct、Fanout、Topic和Header四种，不同类型的Exchange路由规则是不一样的。

/技术点 /队列 / *RabbitMQ* / *Queue*

消息队列，用于存储还未被消费者消费的消息，队列是先进先出的，

/技术点 /队列 / *RabbitMQ* / *Message*

就是消息，由Header和Body组成，Header是由生产者添加的各种属性的集合，包括Message是否被持久化、由哪个Message Queue接受、优先级是多少等，Body是真正传输的数据，内容格式为byte[]

/技术点 /队列 / *RabbitMQ* / *Broker*（*Server*）

接受客户端连接，实现AMQP消息队列和路由功能的进程，我们可以把Broker叫做RabbitMQ服务器

/技术点 /架构 /微服务 / *RPC*

远程过程调用 可以是TCP、HTTP、UDP，TCP对内性能更佳

/技术点 /架构 /微服务 /服务注册中心

注册系统中所有服务的地方

/技术点 /架构 /微服务 /服务注册

服务提供方将自己调用地址注册到服务注册中心，让服务调用方能够方便地找到自己

/技术点 /架构 /微服务 /服务发现

服务调用方从服务注册中心找到自己需要调用服务的地址。

/技术点 /架构 /微服务 /负载均衡

服务提供方一般以多实例的形式提供服务，使用负载均衡能够让服务调用方连接到合适的服务节点

/技术点 /架构 /微服务 /服务容断

通过熔断器等一系列的服务保护机制，保证服务调用者在调用异常服务时能快速地返回结果，避免大量的同步等待。

/技术点 /架构 /微服务 /服务降级

某个服务熔断之后，服务器将不再被调用（或者设置delayTime），此时客户端可以自己准备一个本地的fallback回调，返回一个缺省值，这样做，虽然服务水平下降，但好歹，比直接挂掉要强。

/技术点 /架构 /微服务 /服务网关

也称为API网关，是服务调用的唯一入口，可以在这个组件中实现用户鉴权、动态路由、灰度发布、负载限流等功能

/技术点 /架构 /微服务 /配置中心

将本地化的配置信息注册到配置中心。 单体应用时，我们可以把常用的配置项保存在本地配置文件中。但是当我们的服务实力运行在成百上千的节点中的时候，人工维护这些配置的正确的复杂度则会变得很高。这时候我们需要引入配置中心来统一管理所有服务的配置。开源的配置中心实现有很多，如K8S的ConfigMap，携程开源的Apollo、Spring Cloud Config等等。

/技术点 / *PHP* / *swoole* /进程模型 / *Master- Manager-Worker-Task* / *Master*进程 / *MainReactor* 主线程

主线程会负责监听server socket，如果有新的连接accept，主线程会评估每个Reactor线程的连接数量。将此连接分配给连接数最少的reactor线程，做一个负载均衡。

/技术点 / *PHP* / *swoole* /进程模型 / *Master- Manager-Worker-Task* / *Master*进程 / *Reactor*线程组

Reactor线程负责维护客户端机器的TCP连接、处理网络IO、收发数据完全是异步非阻塞的模式。
swoole的主线程在Accept新的连接后，会将这个连接分配给一个固定的Reactor线程，在socket可读时读取数据，并进行协议解析，将请求投递到Worker进程。在socket可写时将数据发送给TCP客户端。

/技术点 / *PHP* / *swoole* /进程模型 / *Master- Manager-Worker-Task* / *Manager*进程 / *Task*进程组 / *task*

swoole_server->task函数是非阻塞函数，任务投递到task进程中后会立即返回，即不管任务需要在task进程内处理多久，worker进程也不需要任何的等待，不会影响到worker进程的其他操作。
但是task进程却是阻塞的，如果当前task进程都处于繁忙状态即都在处理任务，你又投递过来100个甚至更多任务，这个时候新投递的任务就只能乖乖的排队等task进程空闲才能继续处理。
如果投递的任务量总是大于task进程的处理能力，建议适当的调大task_worker_num的数量，增加task进程数，不然一旦task塞满缓冲区，就会导致worker进程阻塞，这将是我们不期望的结果。

/技术点 / *PHP* / *swoole* /协程

协程可以理解为纯用户态的线程，其通过**协作**而不是抢占来进行切换。相对于进程或者线程，协程所有的操作都可以在用户态完成，创建和切换的消耗更低
Swoole可以为每一个请求创建对应的协程，根据IO的状态来合理的调度协程

/技术点 /缓存 /浏览器缓存

将请求的页面存储在客户端缓存中，当访问者再次访问时，浏览器直接从客户端中获取数据，减少了对服务器的访问，加快了网页的加载速度。

/技术点 /缓存 /浏览器缓存 /强缓存

用户发送的请求，直接从科幻段缓存读取，不请求服务器

/技术点 /缓存 /浏览器缓存 /协商缓存

用户发送的请求，发送给服务器，由服务器判定是否使用客户端缓存

/技术点 /缓存 /文件缓存 /数据文件缓存

更新频率低，读取频率高的数据，缓存成文件。

/技术点 /缓存 /文件缓存 /全站静态化

全站设置为静态化html，不必请求数据库，编译，渲染

/技术点 /缓存 /文件缓存 / *CDN*缓存

CDN内容分发网络。用户访问网站时，自动选择就近的CDN节点内容，不需要请求服务器，加快了网站的打开速度。

/技术点 /缓存 / *NoSQL*缓存 / *Memcached*缓存

高性能的分布式缓存服务器

/技术点 /缓存 / *NoSQL*缓存 / *Redis*缓存

高性能的k/v数据库

/技术点 /缓存 / *NoSQL*缓存 / *MongoDB*缓存

基于分布式文件存贮的数据库。

/技术点 /缓存 /*Web*服务器缓存 /*Nginx*缓存

利用expire参数，指定缓存的过期时间，可缓存html、iamge、js、css等。

```
//图片为例
location ~\.(gif|jpg|png|jpeg)$ {
    root html;
    expires 1d; //指定缓存时间
}
```

/技术点 /缓存 /*Opcode*缓存

Opcode 操作码。
Php解释器完成对脚本代码的分析后，便将生成可以直接运行的中间代码（操作码），对其缓存避免重复编译，减少CPU和内存开销

/技术点 /缓存 /*Opcode*缓存 /*OPchache*

OPcache　通过将 PHP 脚本预编译的字节码存储到共享内存中来提升 PHP 的性能， 存储预编译字节码的好处就是 省去了每次加载和解析 PHP 脚本的开销。