云函数

客户案例

产品文档

腾讯云

文档目录

客户案例
腾讯在线教育

最近更新时间：2023-06-05 16:33:35

本文分享了腾讯在线教育使用云函数的真实案例。腾讯在线教育团队是：
IMWeb 团队隶属腾讯公司，是中国领先的专业前端团队之一。
专注前端领域多年，负责过 QQ 资料、QQ 注册、QQ 群等亿级业务。
目前聚焦于在线教育领域，精心打磨腾讯课堂、腾讯企鹅辅导及 ABCmouse 三大产品。

技术方案的尝试

腾讯在线教育团队在传统的 Web 应用方向其实有众多技术方面的尝试，包括传统离线包、PWA 离线应用等，但每个技术栈都有其优点及缺点。目前团队技术方案的多维度对比如下图所示：

Scheme Name	Above-the-fold Time	Development Difficulty	Maintenance Costs	Iteration Costs	Network-Dependent	Client-Dependent	User Experience	SE Friendl
Async rendering	1.2s ±	Low	Low	Low	Yes	No	Poor	Avera
Offline package (async rendering)	900ms±	Low	Low	Low	No	Yes	Good	N/
SSR	700ms±	Medium-high	Medium-high	Medium-high	Yes	No	Average	Goo

通过对比可发现 SSR 在首屏渲染以及 SEO 等方面都较为突出。基于此结果，团队较倾向于 SSR 技术。在选择 SSR 技术方案时，可从以下两个方面进行考虑：

SSR 应用的性能

我们重点关注以下性能优化问题：
处理大量 CPU 密集型计算：类 React 的应用的 SSR 本质为：在服务端调用 React 的 renderToString 方法，将 React 组件渲染为 HTML 字符串。对于复杂的 SSR 应用来说，此过程可能存在大量的 CPU 密集型计算，且这不属于 Node 擅长的领域。

提高首屏性能：由于离线包的环境依赖性（依赖 App），在传统的 Web 环境内是否可以有一套完整的解决方案来缓存相关的页面，从而提高首屏的性能。

SSR 的运维成本

服务的可用性：大部分前端工程师可能不擅长服务运维方面的工作，服务的可用性问题也成为了技术选型时的重要关注点之一。

根据以上两个方面，可将考虑因素总结为下图中的两点：

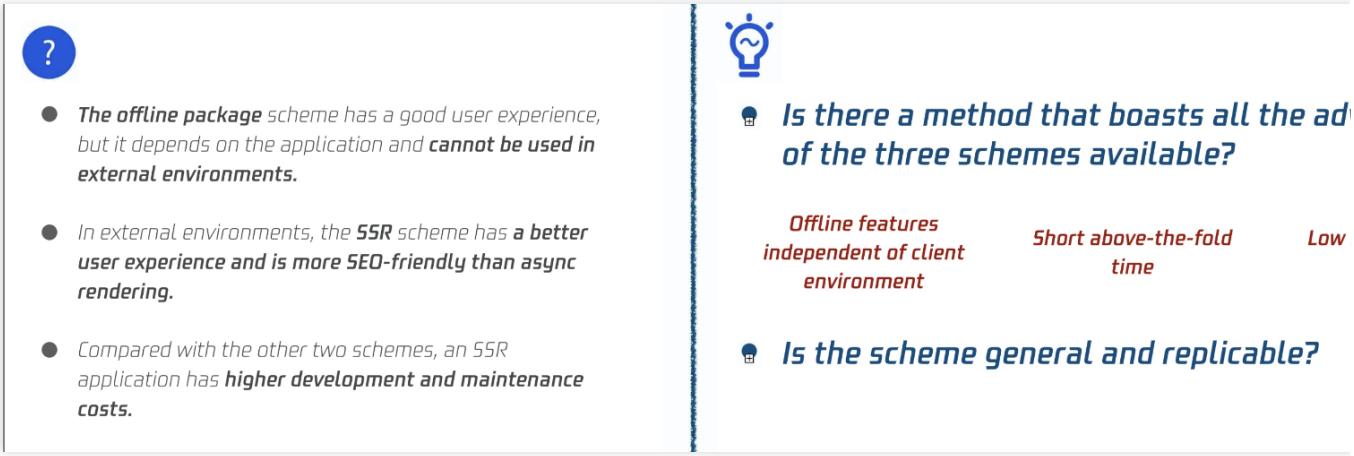
如何设计一种方式同时拥有这三种方案的优点？

不依赖客户端环境的离线功能。

首屏时间短，SEO 体验好。

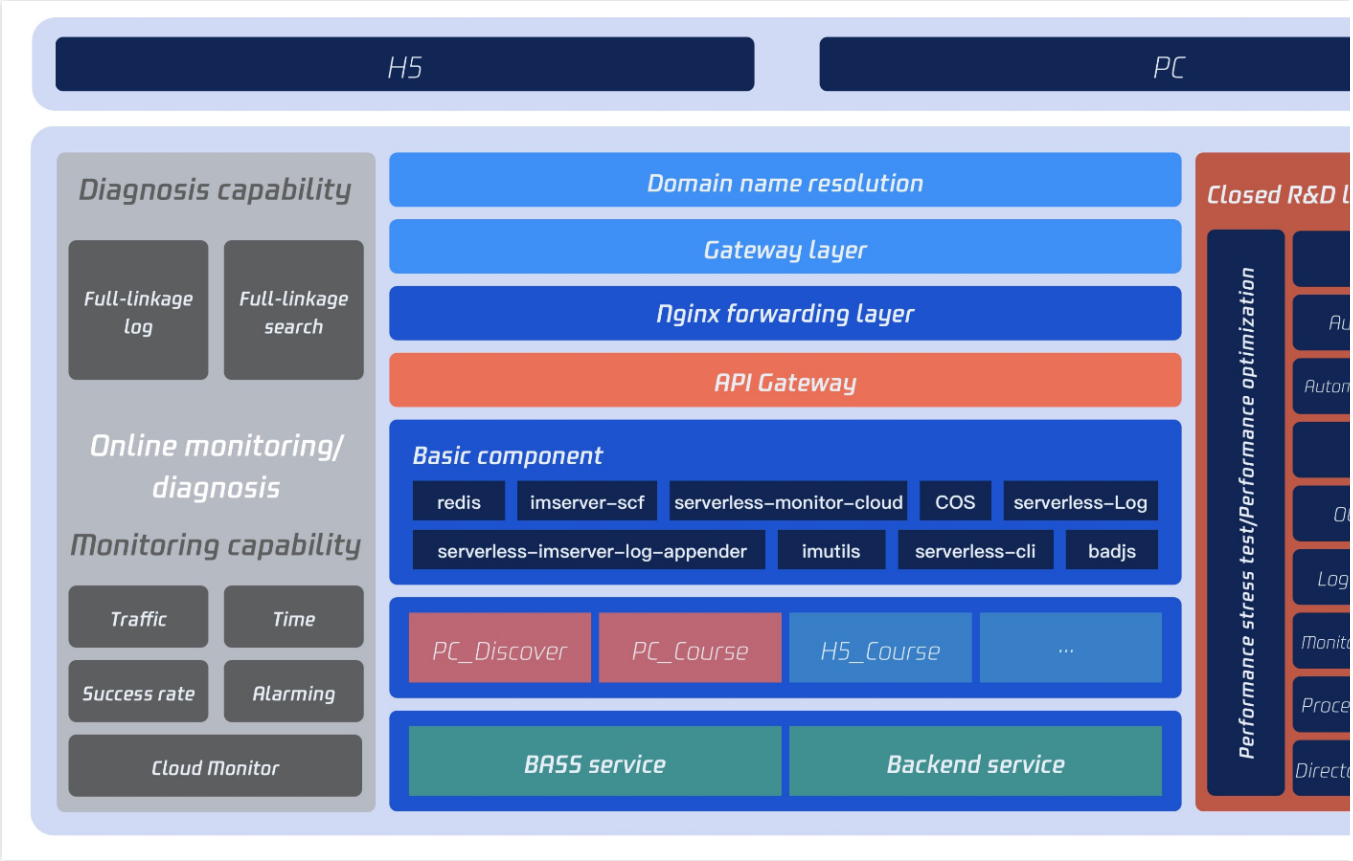
维护成本低，问题定位更方便。

方案是否通用且可复制？



腾讯在线教育团队 SSR 架构方案介绍

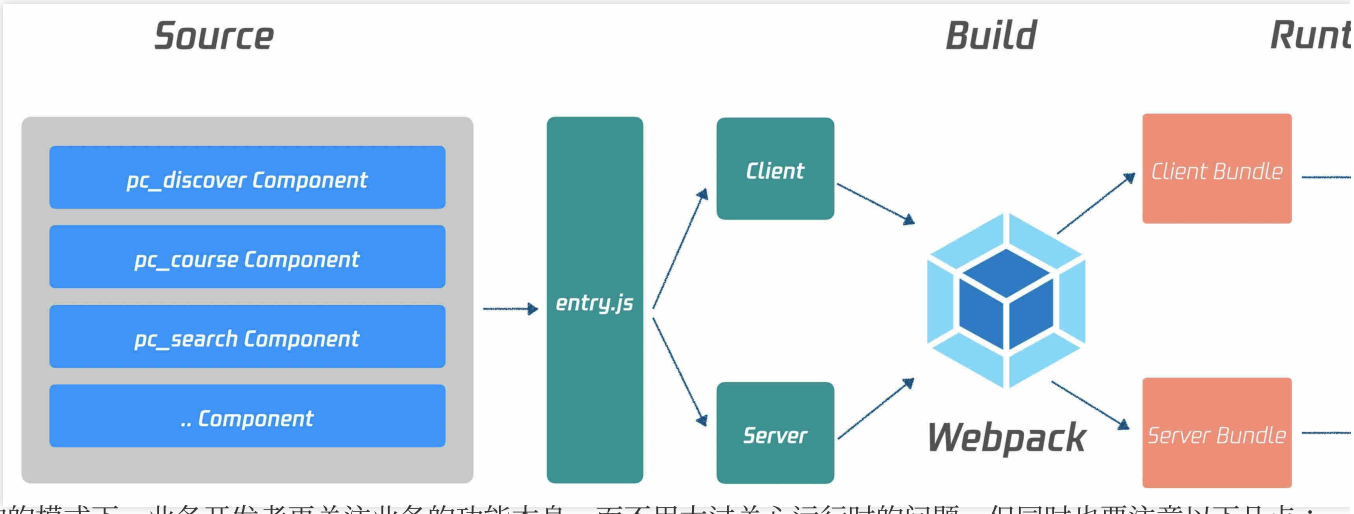
团队使用的 SSR 技术架构图如下所示：



接下来我们从代码组织、性能优化、运行上下文三个方面来详细讲解团队现有方案。

代码组织

在 PC/H5 项目中均采用了同构的模式来构建 SSR 应用。如下图所示：



在同构的模式下，业务开发者更关注业务的功能本身，而不用太过关心运行时的问题，但同时也要注意以下几点：

传统浏览器中的常量使用，例如 `window` 、 `document` 等。

HTTP 数据请求库必须同时支持服务端和客户端。

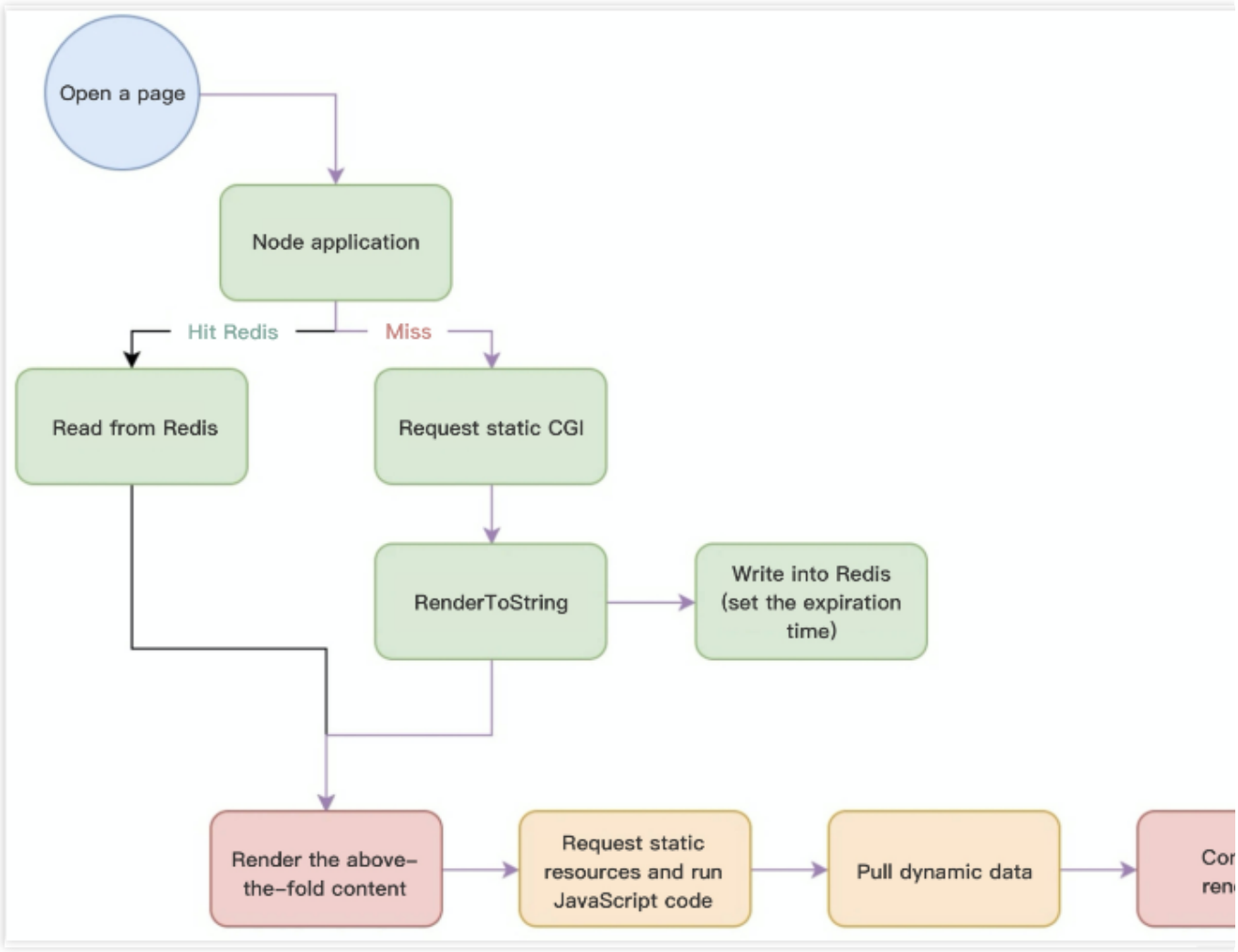
合理使用 React 应用的生命周期。

通过注入环境变量来区分当前运行时环境。

性能优化

接口动静分离

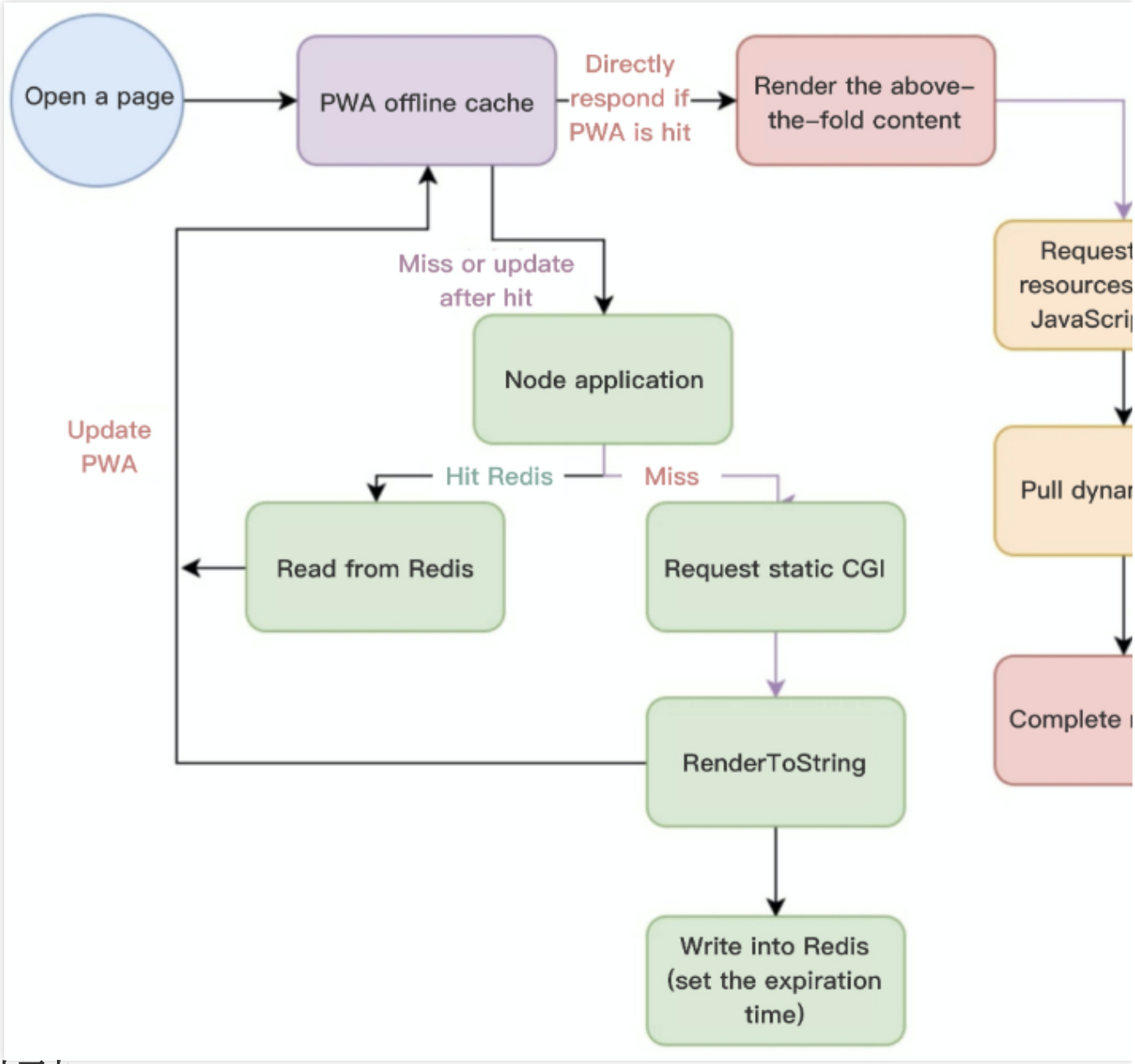
页面的渲染通常依赖后端的相关数据，而数据可以拆分为动态数据与静态数据两个部分：
静态数据：页面中不经常变更的数据。例如，企鹅辅导产品的课程标题、课程描述等。
动态数据：页面中与用户登录态相关的数据。例如，课程是否已经购买、当前课程的折扣等。
对接口做动静分离的意义在于，我们可以利用静态数据的时延性敏感度低的特性做缓存，在服务端利用静态数据渲染页面，之后在服务端利用动态数据做二次渲染。主要逻辑如下图所示：



我们利用 Redis 对静态数据渲染出来的页面做缓存，不仅可以加快 SSR 的渲染时间，同时可以提高单机的 QPS（`renderToString` 在一定意义上为 CPU 密集型操作）。

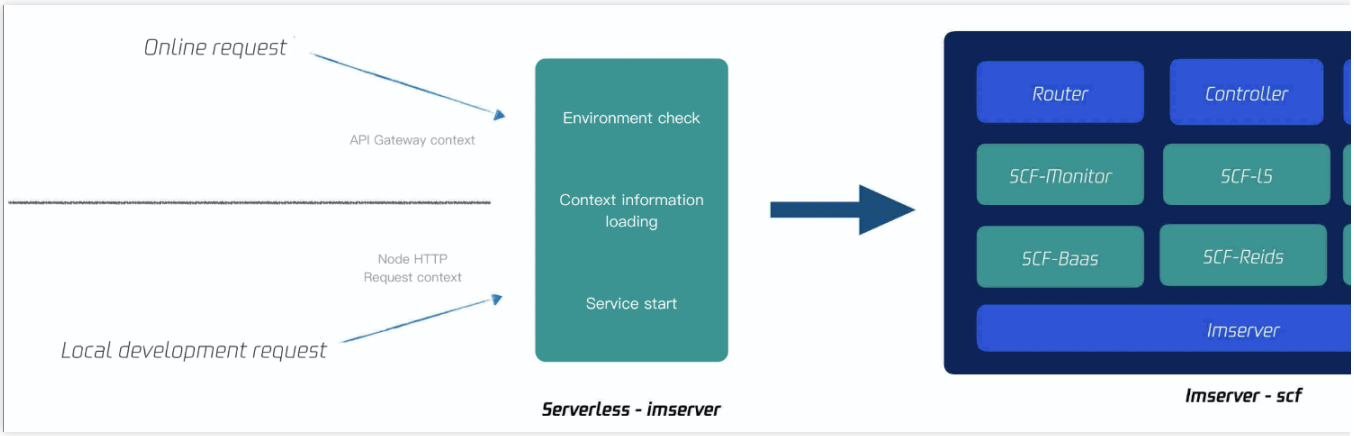
浏览器中利用 PWA 做离线缓存

在客户端中，我们可以利用 PWA 来做离线缓存，缓存静态数据直出的 HTML 页面，从而进一步的提高了直出页面的首屏性能。主要逻辑如下图所示：



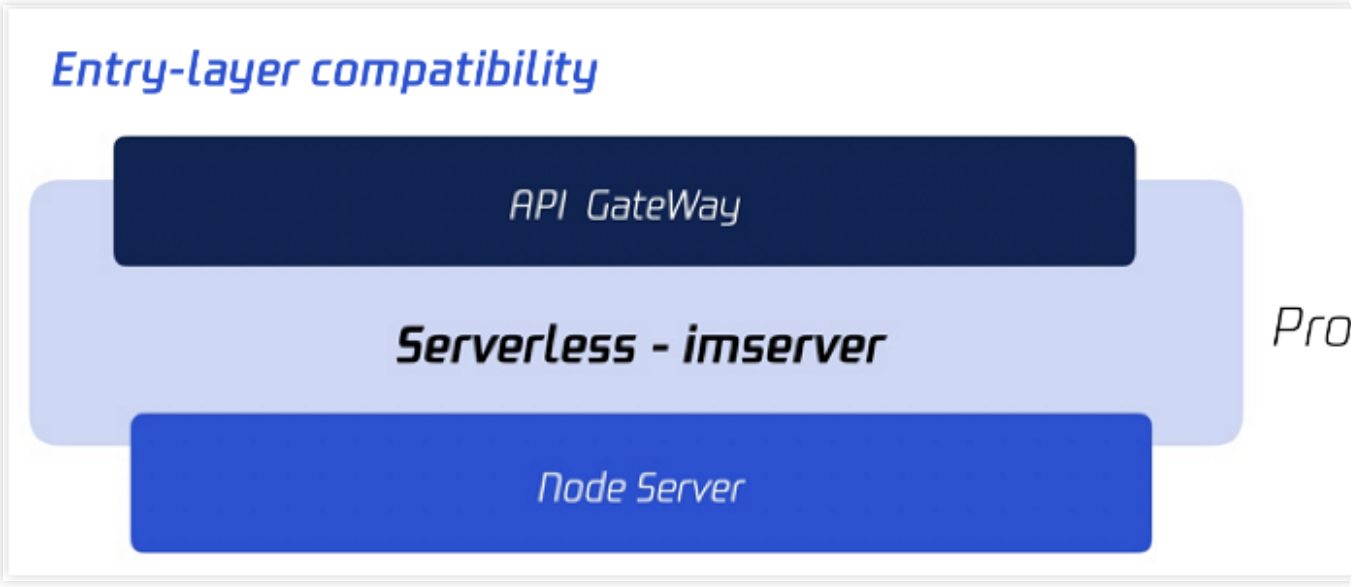
运行上下文

由于后端应用的运维复杂性、维护成本较高等问题，这里我们使用了 Serverless（腾讯云云函数 SCF） 来做直出应用的部署。得益于 Serverless 架构模式的天然优势，不用再关心服务的运维、服务的扩容等问题。



如上图所示，SSR 的应用本质为一个 Node 应用，SCF 的调用本质为一个 Event 事件。针对此问题我们采用了如下方法兼容这两种模式：

借助腾讯云 Serverless Cloud Framework 提供的很多标准化接口，给自研 Node 框架（imserver）增加了一层 Serverless 的封装。同时还在入口增加了 Event 到 Koa Request Context 的兼容。如下图所示：



SSR 的技术方案落地过程中的问题

问题1：云函数拆分

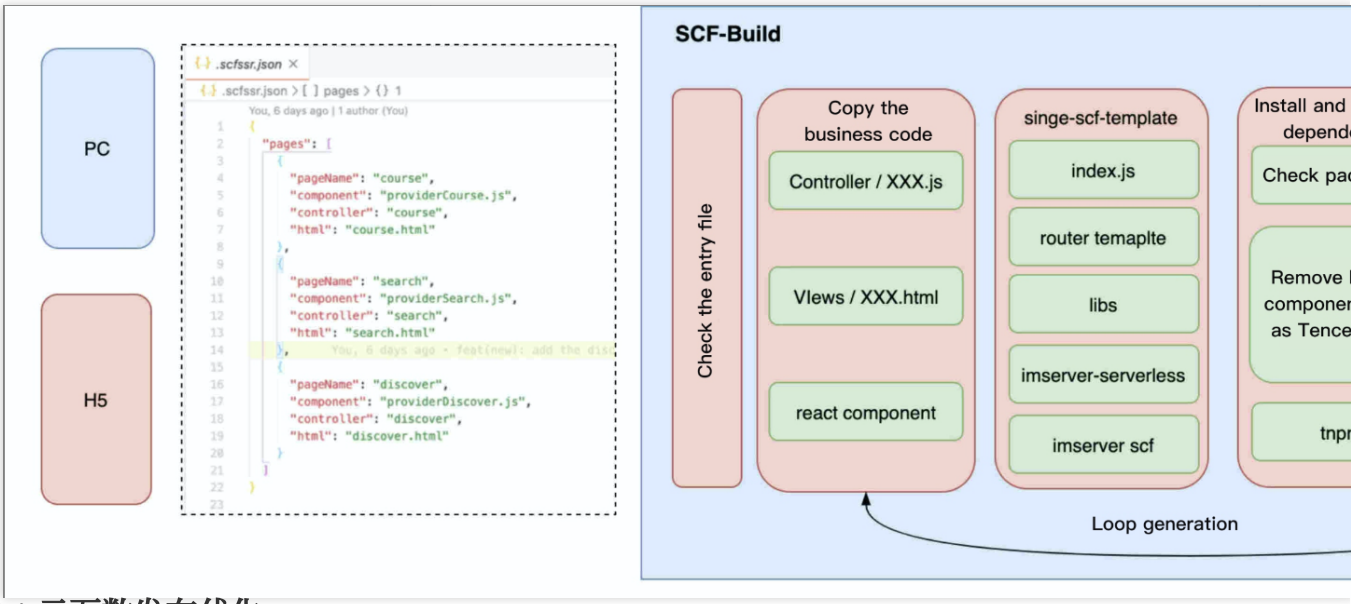
业务中有多个页面是通过 SSR 实现的，在采用了 SCF 实现 SSR 后，需确定是合并至一个云函数中（业务级），还是拆分为多个云函数（页面级）。推荐选择拆分为多个云函数（页面级），优点如下：

云函数互相独立。假设页面 A 云函数调用失败，并不会影响到业务 B 的云函数。

云函数包的大小会降低。由于云函数的冷启动过程，代码的包的大小对函数的冷启动时间也有一定的影响。

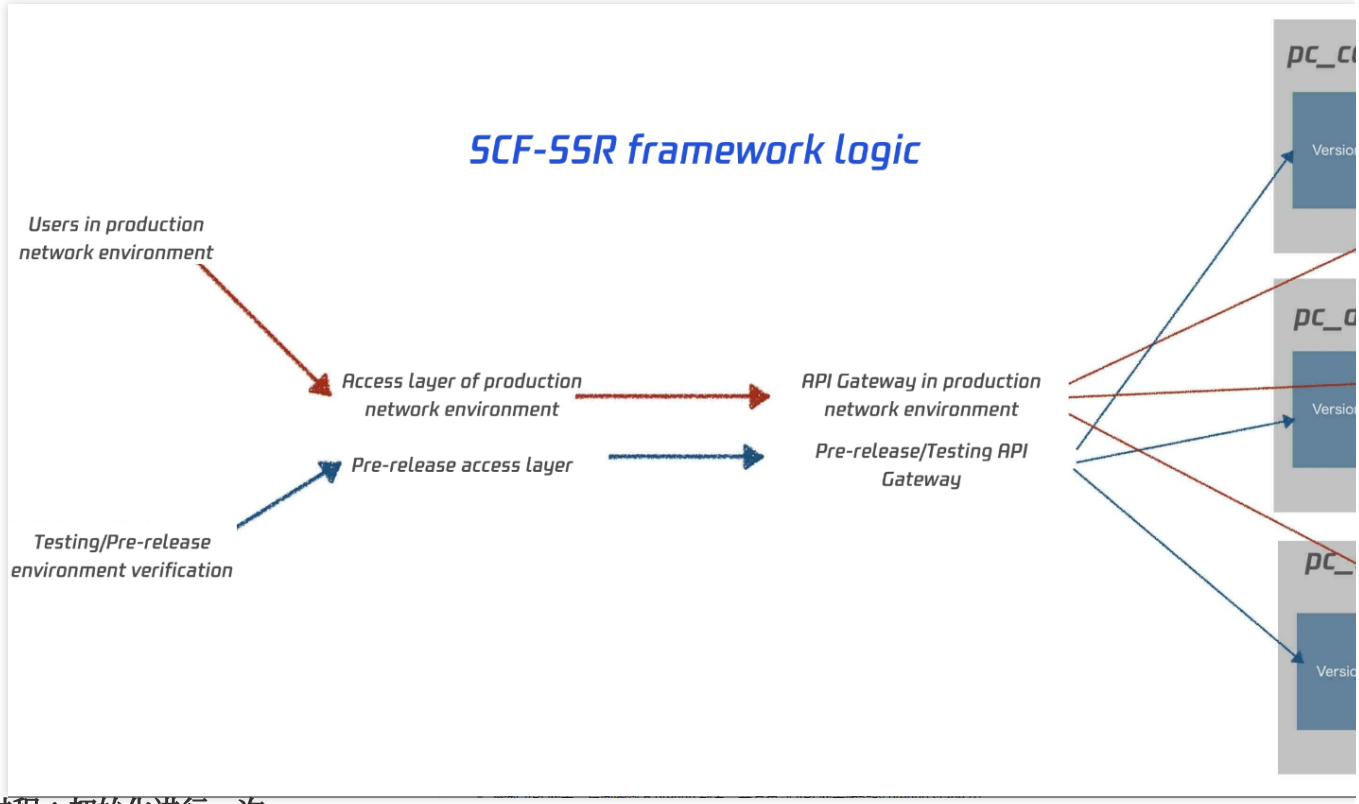
此处基于当前项目实现了云函数的自动化构建，通过 `.scfssr.json` 的配置文件自动生成相应的云函数，对现有的开发工作无任何影响，仅在构建时生成多个云函数，降低了应用的维护成本及开发成本。

同时基于云函数的构建过程，可使单个云函数代码更简练。通过分析 `package.json` 中的依赖，移除云函数容器中已经内置的工具包，并对云函数所依赖的第三方包做相应的引入分析，去除冗余。如下图所示：



问题2：云函数发布优化

下图为我们设计的基于 SCF 的多云函数直出方案逻辑，可查看当有版本更新时，发布流程及步骤是较为复杂的。



配置过程：初始化进行一次

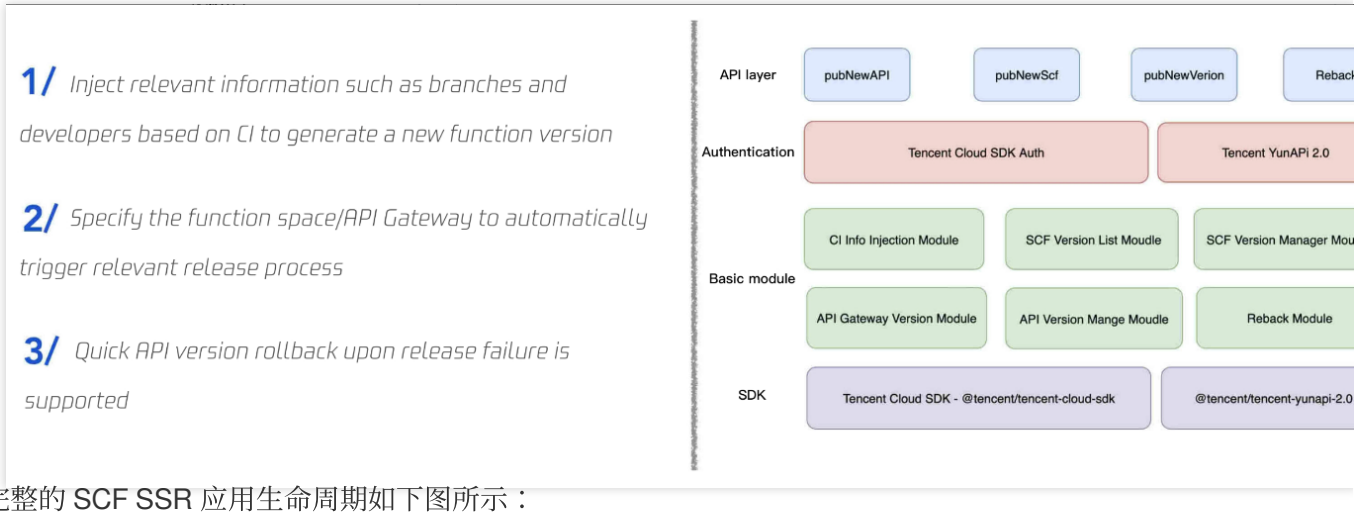
在函数中创建 release、prohub 别名，可预先指向 `$LATEST` 版本。

API 网关中创建服务 A，配置 API 网关指向函数 B release 别名，并发布到 API 服务的 release stage 中。

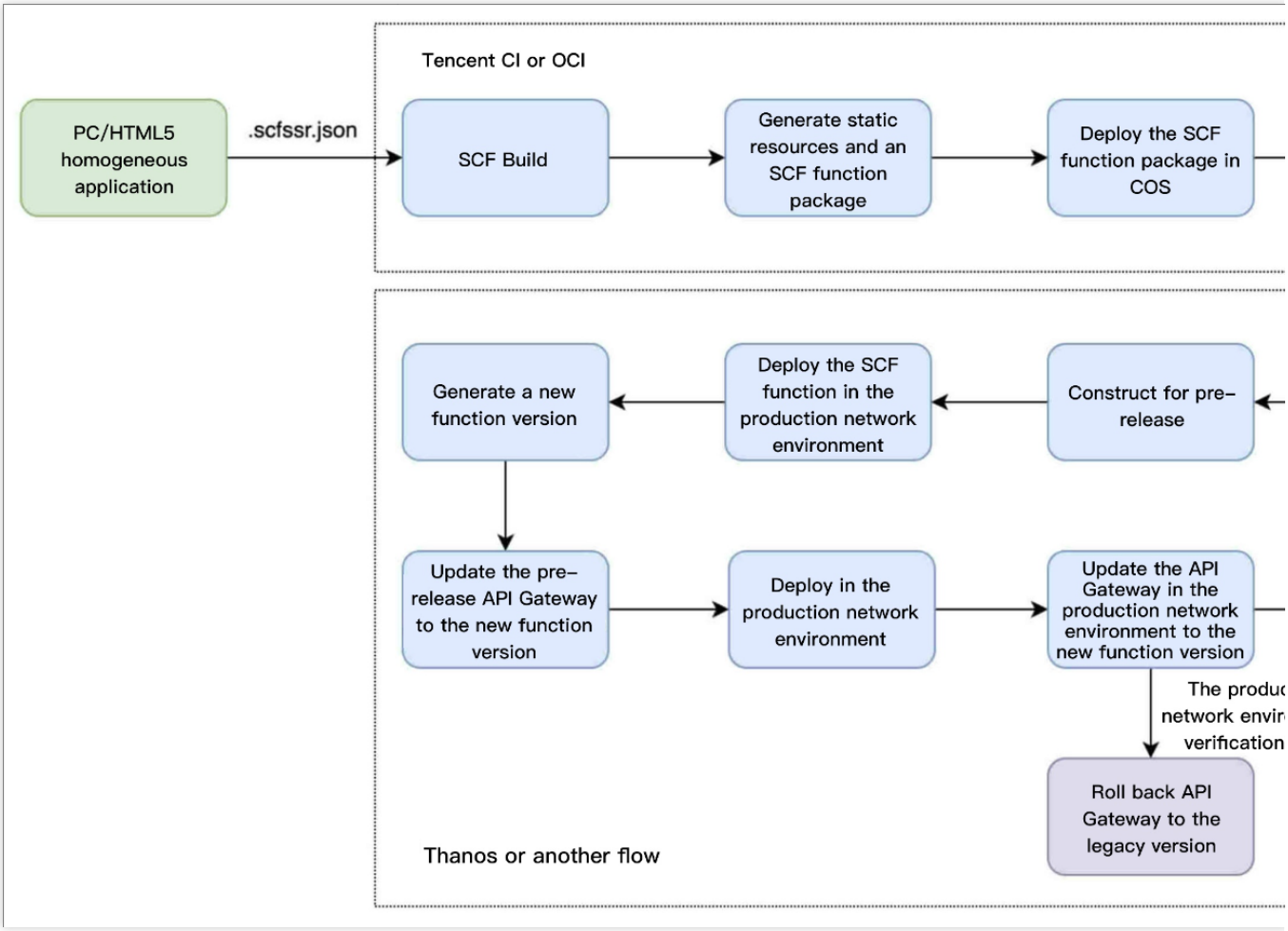
修改 API 网关，指向函数 B prehub 别名，并发布到 API 网关服务的 prehub stage 中。
修改 API 网关，指向函数 B 的默认流量，并发布到 API 网关服务的 dev stage 中。
至此 API 网关的配置完成，后续无需在 API 网关上再次修改及发布配置。

开发测试发布过程：持续开发测试发布上线

在函数上持续开发，一次发布版本 1、2、3。
需开发并测试最近版本时，配置 `$DEFAULT` 别名指向 `$LATEST` 版本，可基于此版本持续开发修改，修改完成后发布版本。
版本3可进入预发布环境时，配置 prehub 别名指向版本3，在预发布环境可进行测试和体验。
版本2已经在预发布层完成体验，可上线，将 release 别名灰度从版本1切换至版本2。
通过监控及日志查看灰度过程，版本2的流量是否正常上涨，版本1的流量是否正常下降，及发布过程中的各版本错误情况以及总体错误情况。
为优化云函数的发布流程，我们基于腾讯云 Serverless 所提供的 Node SDK 做了一键发布 SCF 的工具。如下图所示：



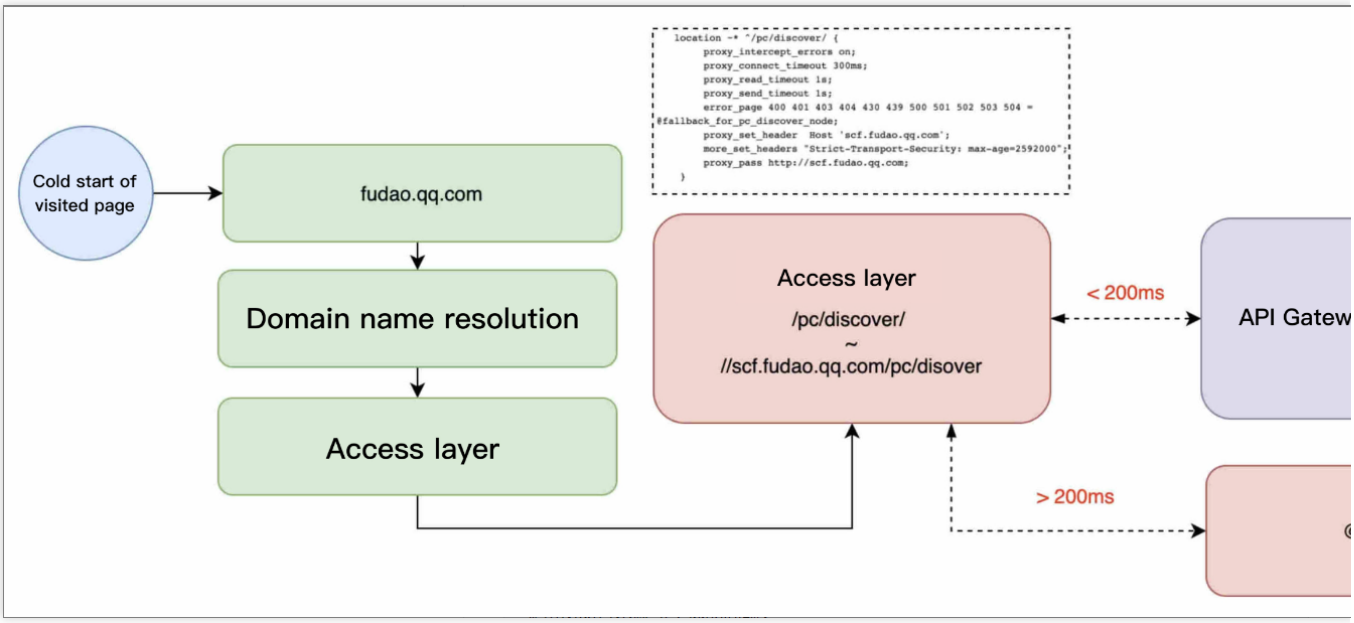
一个完整的 SCF SSR 应用生命周期如下图所示：



腾讯云 Serverless SSR 方案的优点及后续规划

使用基于 SCF 的 SSR 方案，节省了众多的服务运维成本。基于腾讯云 Serverless 的日志系统，所有的单个 SSR 应用请求在日志平台都有完整的链路，定位问题与处理问题的速度都有了质的提升。

Serverless 的架构模式存在冷启动时间较长的问题，SCF 针对此问题已经进行了技术优化，例如预启动容器等。我们在实际业务方面，也可尝试进行优化。例如，在接入层做了服务的降级优化。如下图所示：



后续的优化方案还可以从灰度、多维降级等方面来做改进。

使用 SSR 技术的建议

如果想追求更好的用户体验，建议针对核心业务做 SSR 优化，搭配 Serverless 来做服务的部署与运维。有了 Serverless 的支持，我们可以不用像以前一样关心机器的运维和扩容，可进一步提高团队生产力。
使用 SSR 后，建议可进一步完善自己业务的 devops 流程，将整个研发链路打通，从开发到测试再到部署都可高效进行。
推荐使用业务接入层来做服务降级，提高 SSR 应用的可用性。

在线教育行业案例
音视频转码最佳实践

最近更新时间：2022-05-20 18:49:25

客户介绍

某国内在线教育企业，于2006年在美国纽约证券交易所上市，总部位于中国北京市海淀区中关村，是一家综合性教育集团，同时也是教育培训集团。该企业业务包括外语培训、中小学基础教育、学前教育、在线教育、出国咨询、图书出版等各个领域。旗下还有多家教育子品牌。

客户痛点

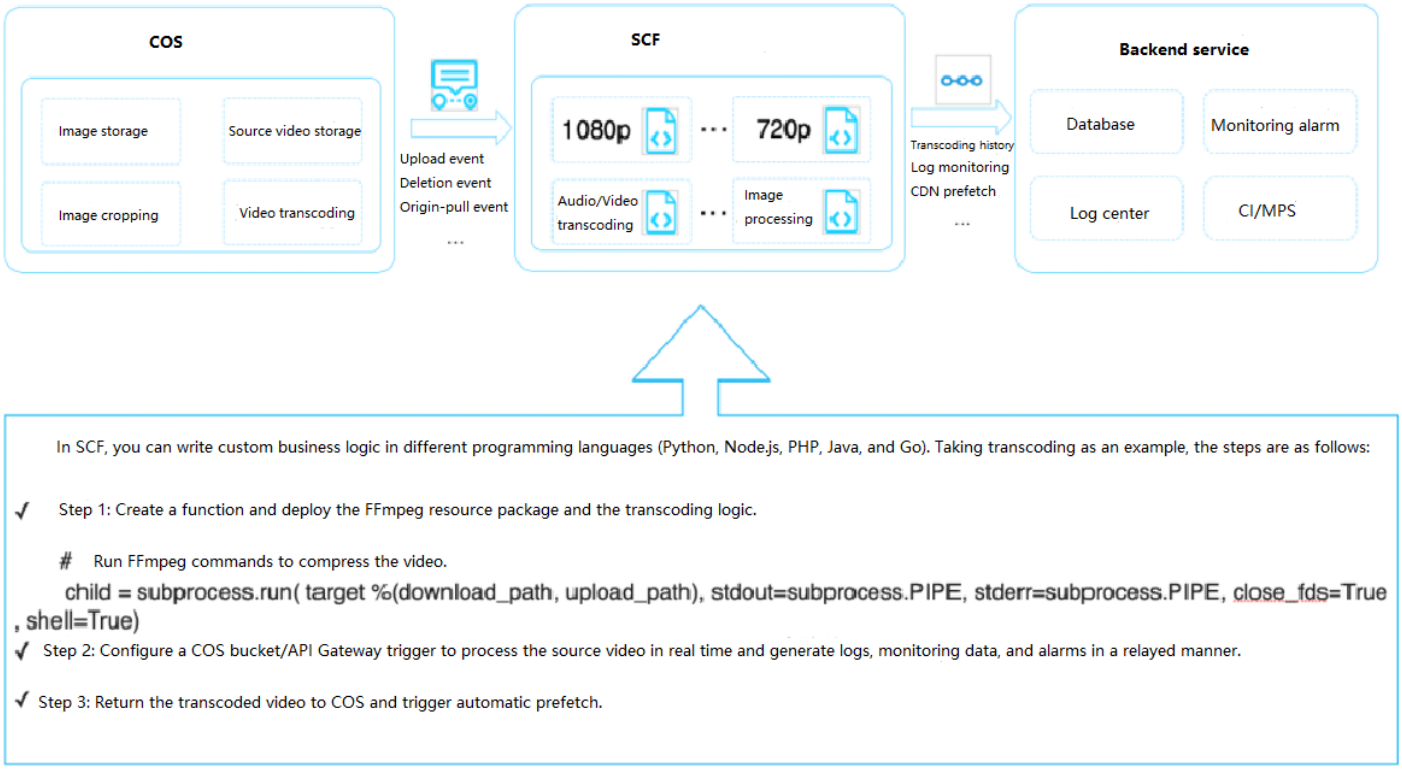
在每年暑期时，会有大量学生在企业平台上学习。在此之前都是在自建的机房里基于服务器和 NFS 来实现音视频课程的存储和转码逻辑。但由于暑期流量比较大，IDC 里的服务器不一定能满足计算需求，同时自建服务的硬件采购周期较长，于是期望寻找一种弹性方法，既能够支持快速业务部署，又能高效的完成转码功能。

在视频应用、社交应用等场景下，用户上传的图片、音视频的总量大、频率高，对处理系统的实时性和并发能力都有较高的要求。传统的容器服务，需要用户自己维护容器集群，弹性伸缩效率较低。

Serverless 解决方案

腾讯云 Serverless 云函数支持自定义转码函数，帮助企业快速搭建定制化任务处理能力，弥补当前单独云服务的功能盲点，将 ffmpeg 业务方便地从物理机、云主机或容器中移植到云函数。

使用云函数 + ffmpeg 和 COS 联动做音视频转码的运行原理如下图：



技术方案上，在云上采用云函数 + COS 的方式，可以支持弹性伸缩，即使将本地流量全部切到云上，也能全部承载。新的业务流程，会加入任务调度模块，当业务流量进入时，可以自动或者手动将流量分别导入自研服务和云上服务，同时在流程里加入了很多高可用技术，例如通过任务 TraceID 进行全链路追踪、云端计算失败本地自动重试等。新的方案里，云端服务开发简单，且无需投入太多运维精力，同时具有更优的成本优势。按量计费（用多少付多少）的云函数计费方式，降低了大量的资源成本。

Serverless 应用价值

使用腾讯云 Serverless 云函数实现音视频转码服务的优势：

- 云函数提供标准运行环境，并且保障资源的高可用和弹性伸缩，无需专人维护。
- 云函数基于实际业务消耗收费，不存在资源浪费。
- 云函数的开发调试流程效率会更加高效，依赖和业务解耦，可以分别单独更新，支持实时热更新。
- 运行环境隔离，单次请求失败不影响其他请求的正常执行。

当现有业务引入云函数时，需要注意以下两点：

- 云函数的引入，需要对接现有 CI/CD 流程，开发方式上有一定的转变。
- 现有业务代码需要做一定的改造，主要改造是将 ffmpeg 集成到函数代码中，与代码文件一起部署到 SCF。

游戏聊天系统

最近更新时间：2021-12-30 17:25:22

客户简介

社交，是游戏文玩家的一项基本需求。在游戏中，成熟稳定的聊天系统担负着玩家交流的重要使命。本文分享了江娱互动使用腾讯云云函数的真实案例，介绍如何通过云函数升级游戏中的聊天系统。

江娱互动在《世界争霸》和《农场小镇》两款产品中都使用了自研聊天系统。随着在线人数逐渐增多，系统的稳定性和成本面临着更多考验。需进一步升级技术栈，以保障性能为前提，同时尽量减少人力和资源成本的花费。最终选择了使用腾讯云云函数，原因如下：

- 云函数无需服务器，仅需关注业务逻辑代码，省去运维烦恼。
- 按量付费的计费模式，避免业务低谷期的资源浪费，减少了成本开支。非常适用于游戏聊天系统 API 这种复杂度低的中小型需求。

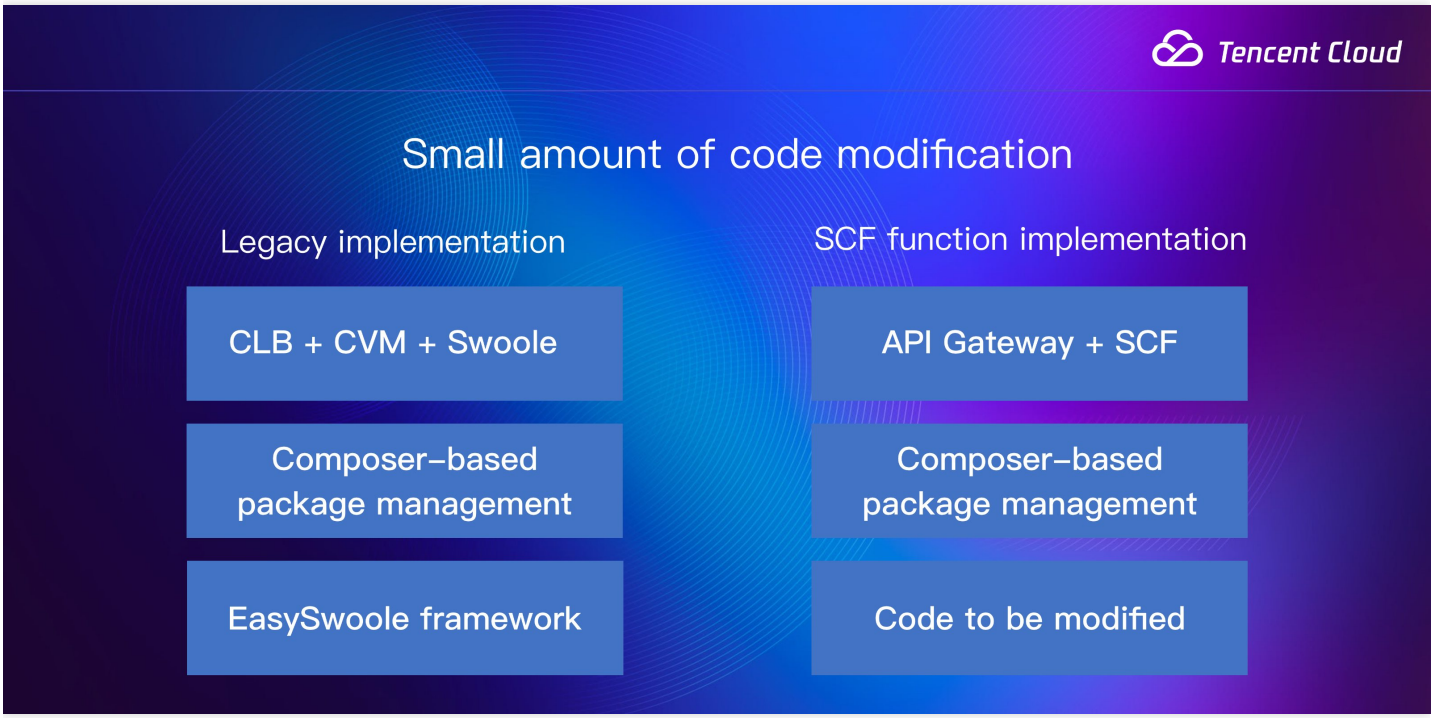
接下来我们只需关注如何将现有系统无缝迁移，即云函数如何满足目前所有的特定需求。

客户需求

需求1：仅修改少量代码

原有的部分 API 采用 swoole 作为底层扩展，部署在腾讯云云服务器 CVM 上，并使用腾讯云负载均衡接收外部请求。代码层面使用了 composer 进行包管理，HTTP 业务框架采用了开源框架 easywoole。

若变更为云函数方案，则非代码层面将由腾讯云 API 网关及云函数来提供服务，代码层依然需使用 composer 进行包管理。而原有基于 swoole 的 HTTP 框架将无法继续使用，修改代码的重点则集中在框架部分。如下图所示：



主要修改思路如下：

1. 确定逻辑入口，确保用一个云函数来处理所有请求。
   入口其实是一个路由，我们需要定义一种简单的路由格式，在云函数入口代码处得到需要的信息，转给原有的类进行处理并返回特定的内容。以下是一个简单的 `url` 格式示例：

   ```
   https://url/controller/action?query
   ```

   只需要解析云函数给出的 path，就能得到 controller 和 action，做出判断后调用相应类的方法后返回。基于这样的入口，原有的逻辑处理类就可以被调用到了。
2. 处理原来的逻辑处理类的父类，弃用框架后需要实现一个基本功能的父类。例如，获取 qerystring 内容、解析 body、返回统一格式的返回值等。
   下图中以 PHP 为示例，不同的开发语言思路类似。代码还需进一步修改，例如，增加数据库配置信息（可使用云函数中的环境变量来传递）及原有耗时任务的异步操作等。

   ```
   date_default_timezone_set('Asia/Shanghai');
   require_once __DIR__ . "/vendor/autoload.php";
   function run($event, $context)
   {
   $path = $event->path;
   //解析path
   list($controller, $action) = parsePath($path);
   $controllerClassName = "\\App\\HttpController\\" . ucwords($controller);
   if(!class_exists($controllerClassName)){
   return return404();
   ```

```
}
$controllerClass = new $controllerClassName($event, $context);
//避免在 HttpController 目录下放置的不应该被外部调用的类被调用
if(!controllerClassName instanceof \App\BaseController){
return return404();
}
if(!method_exists($controllerClass, $action)){
return return404();
}
try{
return $controllerClass->$action();
}catch(Throwable $e){
return return500();
}
}
```

需求2：快速发布

快速发布的能力是不可或缺的，在迁移过程中，会反复进行各种测试。在进行迁移时云函数的本地测试功能尚未支持 PHP，所以在使用 API 网关与云函数组合时，发布流程为：

1. 开发代码
2. 部署云函数 `$LATEST` 版本
3. 基于 `$LATEST` 版本生成新版本号
4. API 网关对应路径切换版本
5. API 网关发布测试版本
6. API 网关线上使用版本切换

发布的过程较为复杂，在迁移开始时，步骤3尚未支持 API 调用，无法实现自动化部署，目前已支持了该能力。但也可以使用 API 网关直接指向云函数的 `$LATEST` 版本，再部署云函数的方案。该方案较为简单，但仅适用于测试阶段，不适用于线上阶段的发布。
我们采用了两者结合的方式，稳定的功能使用稳定版本的发布流程，而新功能则新建一个 API 路径并指向 `$LATEST` 版本，随时进行发布也不会影响线上功能。

问题及解决方案

在发布 API 网关时，有时会遇到资源超限的情况，查明是由于云函数的并发实例数量限制，当发布新的 API 版本时，会请求进入新的实例而旧的实例未释放。两种实例数量和超过限制，此时需向腾讯云申请提高限额。

需求3：内网互通

此次为系统迁移，但原有的系统中仍然有部分内容需要继续使用。因此需要云函数可以与原有的 CVM 内网进行通信，结合云函数本身支持部署到已有的内网中，内网互通很容易实现。

问题及解决方案

由于我们的 API 服务需要向外发出请求，而内网云函数在系统迁移时不具备直接访问外网的能力，通过 NAT 网关实现了云函数访问公网的功能。详情请参见 在私有网络中配置 NAT 网关。

> 注意：
> 通过 NAT 网关的解决问题时，建议给云函数单独分配一个子网。因为使用已有的子网绑定 NAT 网关，会导致出口 IP 变化。若该子网下的机器 IP 恰好在某些白名单下，则会造成影响。

需求4：日志查询

原有的日志采用直接落盘，定期压缩转储的方式。而系统迁移到云函数后，需采用云函数的日志机制。云函数可直接投递日志到腾讯云日志服务中，任何输出的信息都会直接作为日志被投递，需要根据实际需求规划日志内容。

我们把函数入口的原始信息、URL 路径、客户端 IP、解析后的参数以及业务日志等都进行了输出，方便快速定位和查询。另外，不需要单独输出一次返回值，云函数可自行打印。

> 注意：
> 在开启日志投递后，需打开索引才可查看日志。若日志内容包含索引分词符，则设置索引时需从分词符中删除对应关键字，否则内容会被分割。

在系统迁移时，云函数日志部分还存在着不足之处，例如云函数与 API 网关的日志是分离的。HTTP 的原始入口是 API 网关，这就导致一些问题跟进较困难。目前了解到云函数和 API 网关日志的 RequestId 已经打通，可以很方便地通过同一个 ID 查询到同一次请求的日志。

需求5：耗时任务处理

原有的方案是使用 swoole 的 task 处理耗时任务，由于云函数的 PHP 环境支持 swoole，初步尝试了如下图所示的改造方案：

```
$p = "1";
$process = new \swoole_process(function (\swoole_process $worker) use ($p){
echo $p;
sleep(3);
echo $p;
});
$pid = $process->start();
```

但通过该方案制造出的进程不能完全掌控。经测试发现，打印出的日志属于其他请求，同时也无法确定进程的计时及生命周期，所以采用了更为保险的“消息队列”方案。

选用了腾讯云的消息队列服务 CKafka，通过封装一个通用结构的消息体并发给 CKafka，CKafka 会触发另一个云函数（运行着耗时任务的代码）。采用通用结构时，可以忽略消息队列的主题，若有任何想要异步操作的任务，只需写在被 CKafka 触发的云函数中，再把需要触发的云函数名和参数发给 CKafka 即可。如下图所示：

问题及解决方案

Ckafka 主题默认仅创建一个分区，如消费速度不理想，可尝试新增分区。新增后，云函数侧需删除触发器再重新添加。

需求6：配置文件更新

该系统中的配置文件是需要经常更新的大文本。例如，聊天服务中会涉及到的屏蔽词库，该文件容量大且会频繁更新。

原有方案为：配置文件单独具备 git 库，策划提交后执行 jenkins，再由 jenkins 上传文件到 CVM 并进行 reload。
系统迁移到云函数后，无法单独上传配置文件，仅支持将文件放置在代码中。方案也更新为：策划提交 git，jenkins 从 git 获取文件后上传至 COS，云函数再从 COS 拉取。如下图所示：

问题及解决方案

性能问题，云函数拉取 COS 有一定耗时，因此不能每一个请求就拉取一次文件。意味着需要把每一次拉取的内容存放在内存中，但无法统一管理云函数的内存，无法保证实时变更。
采用了折中方案，比较内存中保存文件内容和上一次拉取时间，如果超过5分钟，则重新拉取。可以保证相对的实时性和性能，也可满足目前的需求。如下图所示：

```
private static $fileContent = null;
private static $lastTime = 0;
public static function refresh(){
self::$fileContent = self::readFromCos("words.txt");
```

```
}
public static function getFileContent(){
if(time() - self::$lastTime > 300 || empty(self::$fileContent)){
self::refresh();
self::$lastTime = time();
}
return self::$fileContent;
}
```

客户价值

至此，系统迁移过程中的需求都已实现，迁移工作得以顺利推进。迁移到云函数之后的优势如下：

- 无需维护 API 服务器，无需考虑 CPU、内存的问题，请求量增加时也无需考虑是否增加 CVM。
- 监控内容比较详细，可以更好地查看整体的运行效率。例如，查看是否存在慢请求、访问趋势，是否有错。
- 使用消息队列拆分后，解耦彻底，且可确保消息不会丢失。消息队列触发云函数的方法适用于不断累积形式的慢任务。
- 版本管理功能改进后可随时切换版本，无需再重新拉取代码分支发布。

腾讯云云函数给江娱互动带来了众多优势及便利，江娱互动也在规划还有哪些功能可以继续使用：

- 无状态的 HTTP 服务。例如，客服消息接收、支付回调接口。
- 无须返回的异步任务。例如，微信小游戏上报玩家排名。
- 定时任务。例如，定期给玩家推送相关的活动信息。

腾讯互娱国际（IEGG）

最近更新时间：2021-12-06 16:02:05

客户介绍

腾讯互动娱乐事业群（IEG）旗下的腾讯游戏主打游戏开发和运营、网络游戏社区的机构。在游戏上云的道路上，腾讯互娱一直在不断探索、不断突破。2021年03月，腾讯互娱针对国际业务推出了在线游戏开发平台 PGOS，PGOS 提供：

- **后端服务平台：**PGOS（Proxima Game Online Service）是一种游戏在线服务解决方案，旨在降低游戏后端开发和维护难度，同时降低成本，从而使开发者专注于游戏玩法与核心逻辑开发。
- **全面托管服务：**借助完整的后端解决方案，消除了大规模构建，管理和运行服务器的挑战。即时自动扩缩容的专用服务器，为实时游戏提供低延迟和高可靠性。
- **一站式控制面板：**开发者及运维人员可以在 PGOS Web 门户上检索玩家信息及查看日志流水，监控实时数据并编辑相关服务配置。
- **跨平台 SDK：**提供开箱即用的 C++ SDK 和 UE4 插件，方便开发者在其游戏客户端和专有服务器中使用 PGOS 服务。
- **灵活的匹配规则：**为玩家提供在实时战斗中快速准确地与其他玩家进行匹配的能力。
- **扩展性和灵活性：**整个系统采用了微服务架构模式，而非具有不同层级的整体结构，使得开发人员可以添加和修改相应服务之间的交互。

Serverless 解决方案

腾讯云 Serverless 天然支持上述 PGOS 提供的能力特性，PGOS 选择使用腾讯云 Serverless 技术提供底层计算支持，能够更好助力团队快速上云。

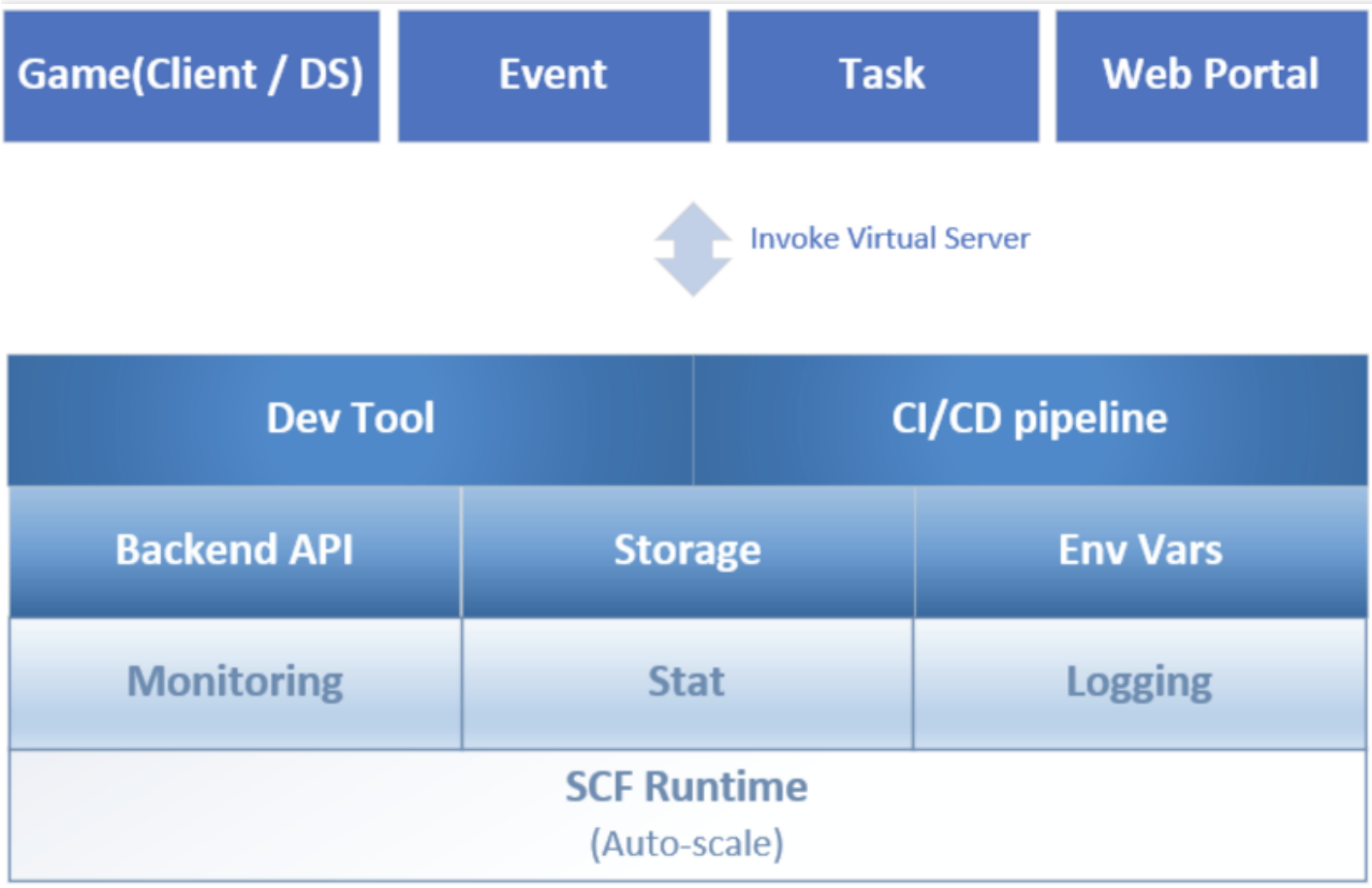
- **开箱即用：**用户无需额外购买、搭建和配置服务器，可完全专注于业务代码。这种架构方式不仅加快游戏发行和迭代速度，同时可降低运维成本，用户无需关注底层资源，腾讯云 Serverless 保障业务的稳定、安全和资源的可用。
- **动态扩缩容：**Serverless 的另一大特点是自动扩缩，轻松应对流量洪峰。在访问量突增时，自动扩容保障业务的正常运行。在流量低谷，自动缩容以节约成本。
- **实时监控：**腾讯云 Serverless 提供实时日志、监控面板，研发人员、管理人员可以实时监控业务运行状态，并且对接腾讯云云监控服务，提供运行时间、状态异常等多维度告警能力，使得问题可以在最短时间内被捕捉并且通知到用户。
- **扩展性和灵活性：**FaaS 的原子特性，天然支持业务灵活扩展。不同的云函数可支持独立的功能，既可支持函数间的相互调用又可独立更新和部署。同时支持函数代码在线编辑功能，从业务开发到部署再到监控，腾讯云

Serverless 提供了一站式的解决方案。

- **多种事件触发**：腾讯云 Serverless 支持约10种事件触发方式，包括定时触发器、API 网关触发器、对象存储触发器等等，满足用户多种触发场景的需求。

Serverless 为游戏上云提供算力支持的技术原理

腾讯云 Serverless 可以为国际业务 PGOS 提供底层运算支持，一个虚拟服务器（Virtual Server）对应一个或多个云函数，用户创建 Virtual Server 并编写对应的业务逻辑。PGOS 依赖 Serverless 提供完善的监控、日志能力，并对接后端服务，PGOS 更进一步封装 DevOps 工具，为用户提供全托管、自动构建和部署的功能。



PGOS 提供多种驱动方式，底层对应不同的函数触发器来实现触发 Virtual Server 中业务的运行。

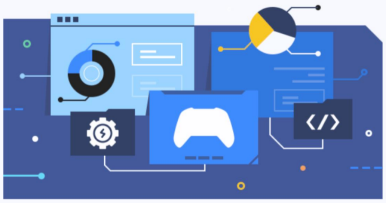
- **定时器驱动**：游戏可以在 Web Portal 上配置一个定时任务，定时触发 Virtual Server 的指定接口。
- **事件驱动**：Virtual Server 可以监听特定事件，当事件发生时自动触发 Vritual Server。
- **游戏驱动**：Game Client 或 DS 可以主动调用 Extension Interface，通过 Gateway 触发 Virtual Server 的指定接口。
- **手动驱动**：通过 Web Portal 手动运行/触发 Vritual Server 的指定接口。

Full Managed Services

Eliminate the challenges of building, managing and running servers on a large scale with a complete backend solution and DS management service enable server scheduling.

Cross Platform SDK

We have built a C++ SDK and UE4 Plugin that can be used “out of the box” for developers to easily utilize the services in their game clients and dedicated servers.



One-Stop Control Panel

Developers can retrieve player profiles, monitor live data and edit service configurations on PGOS web portal.

Extensions & Flexibility

The entire system is packed with various services, and a microservices architecture pattern is adopted instead of using a monolithic structure with different layers.