为什么使用框架？

前后端分离

在前后分离概念出现之前，大部分 web 项目都是后端人员又当爹又当妈的，前后端一起搞，导致质量和效率不是很好。而且对个人的发展也有影响，一个人你什么都会，也意味着你什么都不精，毕竟天才还是少数的。这也是社会趋势影响，大公司招聘，一般也都是需要某一方面很有研究的专才。

在互联网的洪流下，以前的那种方式越来越跟不上节奏，所以前后端分离应运而生。

前后端分离后，前端的任务也变得重要起来，web 前端开发慢慢趋于规范。

但是在 jQuery 称霸的时代里，并不能满足前端开发人员的需求。也慢慢暴露出了很多不好解决的问题：外部 js 引用太多，复用性低，开发周期太长，性能低，效率低等等，这些 jQuery 不好解决或者说解决不了的问题，也成为了前端开发的趋势。

使用框架解决了哪些问题

重复引用外部 js

在以前使用 jQuery 开发时，当项目越来越复杂和庞大的时候，可能会用到各种各样的第三方插件，而且不只是一个页面使用，所以会出现每个页面都要引用一遍相同的 js 文件，冗余大的问题。

这样不仅会使页面代码变得杂乱，而且会影响页面的打开速度，万一以后需要变更一下 js 文件的路径，还要一个一个去修改，对后期的维护也是很大的负担。

使用框架开发时（例如 Vue），一般都会搭配构建工具使用（例如 webpack），整个项目运行时会有一个入口文件，当你有多个组件都会用到某个文件或插件时，仅仅在这个入口文件引入一次，就可以在你所有组件中使用这个插件的方法，可以说是一劳永逸。就算后期文件位置有所变动，也只是修改入口文件中的引用路径就可以了。

组件化

组件是前端框架里非常强大的功能之一，它可以扩展你的 HTML，封装可以重用的代码块，比如你的轮播图、tab 切换、页面头部、页面底部等等。

这种独立的组件具有了结构（html），表现（css）和行为（js）完整的功能，很大程度的节省了代码量，提高了代码的复用性。根据不同的需求定制你自己的组件，在需要的页面引用即可。在团队合作开发中，相对独立开发不同的组件，效率上也有很大的提升。

开发周期长

jQuery 开发时，需要频繁的操作 DOM，几乎任何动态效果都需要去选择 DOM 来进行相应的操作，这使开发变得麻烦起来，很多的时间都用到了操作 DOM 上，项目的开发周期自然被延长。

使用框架开发，框架中封装许多的频繁使用的功能，例如 Angular 中的指令，指令功能有数据绑定，表单验证，数据格式化等等。这时前端的重点只需要放在数据逻辑部分，而不需要花费很大的精力去操作 DOM 完成功能，从而加快项目进度。

性能

很多 DOM 操作会引起回流和重绘，对于 jQuery 来说，大量的操作 DOM 虽然方便，但是很浪费页面性能。

框架和 jQuery 虽然都会操作 DOM，但是框架吧大量的 DOM 进行了处理和优化（例如 Vue 的虚拟 DOM），通过数据驱动，就能渲染出 DOM，大大提升了性能。

1.高效，开发变得快速，简单并且有效

2.成本，很多框架都是免费的，而且开发人员编写代码更快，客户成本自然而然就低了

3.安全，很多框架都拥有安全的实现

4.支持，跟其他发布工具类似，框架也有文档支持和团队支持

为什么前后端分离

一、前端

前后端分离已成为互联网项目开发的业界标准使用方式，通过 nginx+tomcat 的方式（也可以中间加一个 nodejs）有效的进行解耦，并且前后端分离会为以后的大型分布式架构、弹性计算架构、微服务架构、多端化服务（多种客户端，例如：浏览器，车载终端，安卓，IOS 等等）打下坚实的基础。这个步骤是系统架构从猿进化成人的必经之路。

核心思想是前端 html 页面通过 ajax 调用后端的 restuful api 接口并使用 json 数据进行交互。

在互联网架构中，名词解释：

Web 服务器：一般指像 nginx，apache 这类的服务器，他们一般只能解析静态资源。

应用服务器：一般指像 tomcat，jetty，resin 这类的服务器可以解析动态资源也可以解析静态资源，但解析静态资源的能力没有 web 服务器好。

一般都是只有 web 服务器才能被外网访问，应用服务器只能内网访问。

三、原始人时代（各种耦合）

几曾何时，我们的 JavaWeb 项目都是使用了若干后台框架，springmvc/struts + spring + spring jdbc/hibernate/mybatis 等等。

大多数项目在 java 后端都是分了三层，控制层，业务层，持久层。控制层负责接收参数，调用相关业务层，封装数据，以及路由&渲染到 jsp 页面。然后 jsp 页面上使用各种标签或者手写 java 表达式将后台的数据展现出来，玩的是 MVC 那套思路。

我们先看这种情况：需求定完了，代码写完了，测试测完了，然后呢？要发布了吧？你需要用 maven 或者 eclipse 等工具把你的代码打成一个 war 包，然后把这个 war 包发布到你的生产环境下的 web 容器里，对吧？

发布完了之后，你要启动你的 web 容器，开始提供服务，这时候你通过配置域名，dns 等等相关，你的网站就可以访问了（假设你是个网站）。那我们来看，你的前后端代码是不是全都在那个 war 包里？包括你的 js，css，图片，各种第三方的库，对吧？

好，下面在浏览器中输入你的网站域名（www.xxx.com），之后发生了什么？（这个问题也是很多公司的面试题）我捡干的说了啊，基础不好的童鞋请自己去搜。

浏览器在通过域名通过 dns 服务器找到你的服务器外网 ip,将 http 请求发送到你的服务器，在 tcp3 次握手之后（http 下面是 tcp/ip），通过 tcp 协议开始传输数据，你的服务器得到请求后，开始提供服务，接收参数，之后返回你的应答给浏览器，浏览器再通过 content-type 来解析你返回的内容，呈现给用户。

那么我们来看，我们先假设你的首页中有 100 张图片，此时，用户的看似一次 http 请求，其实并不是一次，用户在第一次访问的时候，浏览器中不会有缓存，你的 100 张图片，浏览器要连着请求 100 次 http 请求（有人会跟我说 http 长连短连的问题，不在这里讨论），你的服务器接收这些请求，都需要耗费内存去创建 socket 来玩 tcp 传输（消耗你服务器上的计算资源）。

重点来了，这样的话，你的服务器的压力会非常大，因为页面中的所有请求都是只请求到你这台服务器上，如果 1 个人还好，如果 10000 个人并发访问呢（先不聊服务器集群，这里就说是单实例服务器），那你的服务器能扛住多少个 tcp 连接？你的带宽有多大？你的服务器的内存有多大？你的硬盘是高性能的吗？你能抗住多少 IO？你给 web 服务器分的内存有多大？会不会宕机？

这就是为什么，越是大中型的 web 应用，他们越是要解耦。理论上你可以把你的数据库+应用服务+消息队列+缓存+用户上传的文件+日志+等等都扔在一台服务器上，你也不用玩什

么服务治理，也不用做什么性能监控，什么报警机制等等，就乱成一锅粥好了。但是这样就好像是你把鸡蛋都放在一个篮子里，隐患非常大。如果因为一个子应用的内存不稳定导致整个服务器内存溢出而 hung 住，那你的整个网站就挂掉了。
如果出意外挂掉，而恰好这时你们的业务又处于井喷式发展高峰期，那么恭喜你，业务成功被技术卡住，很可能会流失大量用户，后果不堪设想。（注意：技术一定是要走在业务前面的，否则你将错过最佳的发展期哟，亲~）
此外，你的应用全部都耦合在一起，相当于一个巨石，当服务端负载能力不足时，一般会使用负载均衡的方式，将服务器做成集群，这样其实你是在水平扩展一块块巨石，性能加速度会越来越低，要知道，本身负载就低的功能 or 模块是没有必要水平扩展的，在本文中的例子就是你的性能瓶颈不在前端，那干嘛要水平扩展前端呢？？？还有发版部署上线的时候，我明明只改了后端的代码，为什么要前端也跟着发布呢？？？
正常的互联网架构，是都要拆开的，你的 web 服务器集群，你的应用服务器集群+文件服务器集群+数据库服务器集群+消息队列集群+缓存集群等等。

四、JSP 的痛点
以前的 javaWeb 项目大多数使用 jsp 作为页面层展示数据给用户，因为流量不高，因此也没有那么苛刻的性能要求，但现在是大数据时代，对于互联网项目的性能要求是越来越高，因此原始的前后端耦合在一起的架构模式已经逐渐不能满足我们，因此我们需要需找一种解耦的方式，来大幅度提升我们的负载能力。
1、动态资源和静态资源全部耦合在一起，服务器压力大，因为服务器会收到各种 http 请求，例如 css 的 http 请求，js 的，图片的等等。一旦服务器出现状况，前后台一起玩完，用户体验极差。
2、UI 出好设计图后，前端工程师只负责将设计图切成 html，需要由 java 工程师来将 html 套成 jsp 页面，出错率较高（因为页面中经常会出现大量的 js 代码），修改问题时需要双方协同开发，效率低下。
3、jsp 必须要在支持 java 的 web 服务器里运行（例如 tomcat，jetty，resin 等），无法使用 nginx 等（nginx 据说单实例 http 并发高达 5w，这个优势要用上），性能提不上来。
4、第一次请求 jsp，必须要在 web 服务器中编译成 servlet，第一次运行会较慢。
5、每次请求 jsp 都是访问 servlet 再用输出流输出的 html 页面，效率没有直接使用 html 高（是每次哟，亲~）
6、jsp 内有较多标签和表达式，前端工程师在修改页面时会捉襟见肘，遇到很多痛点。
7、如果 jsp 中的内容很多，页面响应会很慢，因为是同步加载。
8、需要前端工程师使用 java 的 ide（例如 eclipse），以及需要配置各种后端的开发环境，你们有考虑过前端工程师的感受吗。
基于上述的一些痛点，我们应该把整个项目的开发权重往前移，实现前后端真正的解耦!

七、前后分离的优势
1、可以实现真正的前后端解耦，前端服务器使用 nginx。前端/WEB 服务器放的是 css，js，图片等等一系列静态资源（甚至你还可以 css，js，图片等资源放到特定的文件服务器，例如阿里云的 oss，并使用 cdn 加速），前端服务器负责控制页面引用&跳转&路由，前端页面异步调用后端的接口，后端/应用服务器使用 tomcat（把 tomcat 想象成一个数据提供者），加快整体响应速度。(这里需要使用一些前端工程化的框架比如 nodejs，react，router，react，redux，webpack)
2、发现 bug，可以快速定位是谁的问题，不会出现互相踢皮球的现象。页面逻辑，跳转错

误，浏览器兼容性问题，脚本错误，页面样式等问题，全部由前端工程师来负责。接口数据出错，数据没有提交成功，应答超时等问题，全部由后端工程师来解决。双方互不干扰，前端与后端是相亲相爱的一家人。
3、在大并发情况下，我可以同时水平扩展前后端服务器，比如淘宝的一个首页就需要 2000+台前端服务器做集群来抗住日均多少亿+的日均 pv。（去参加阿里的技术峰会，听他们说他们的 web 容器都是自己写的，就算他单实例抗 10 万 http 并发，2000 台是 2 亿 http 并发，并且他们还可以根据预知洪峰来无限拓展，很恐怖，就一个首页。。。）
4、减少后端服务器的并发/负载压力。除了接口以外的其他所有 http 请求全部转移到前端 nginx 上，接口的请求调用 tomcat，参考 nginx 反向代理 tomcat。且除了第一次页面请求外，浏览器会大量调用本地缓存。
5、即使后端服务暂时超时或者宕机了，前端页面也会正常访问，只不过数据刷不出来而已。
6、也许你也需要有微信相关的轻应用，那样你的接口完全可以共用，如果也有 app 相关的服务，那么只要通过一些代码重构，也可以大量复用接口，提升效率。（多端应用）
7、页面显示的东西再多也不怕，因为是异步加载。
8、nginx 支持页面热部署，不用重启服务器，前端升级更无缝。
9、增加代码的维护性&易读性（前后端耦在一起的代码读起来相当费劲）。
10、提升开发效率，因为可以前后端并行开发，而不是像以前的强依赖。
11、在 nginx 中部署证书，外网使用 https 访问，并且只开放 443 和 80 端口，其他端口一律关闭（防止黑客端口扫描），内网使用 http，性能和安全都有保障。
12、前端大量的组件代码得以复用，组件化，提升开发效率，抽出来！

八、注意事项
1、在开需求会议的时候，前后端工程师必须全部参加，并且需要制定好接口文档，后端工程师要写好测试用例（2 个维度），不要让前端工程师充当你的专职测试，推荐使用 chrome 的插件 postman 或 soapui 或 jmeter，service 层的测试用例拿 junit 写。ps：前端也可以玩单元测试吗？
2、上述的接口并不是 java 里的 interface，说白了调用接口就是调用你 controler 里的方法。
3、加重了前端团队的工作量，减轻了后端团队的工作量，提高了性能和可扩展性。
4、我们需要一些前端的框架来解决类似于页面嵌套，分页，页面跳转控制等功能。（上面提到的那些前端框架）。
5、如果你的项目很小，或者是一个单纯的内网项目，那你大可放心，不用任何架构而言，但是如果你的项目是外网项目，呵呵哒。
6、 以前还有人在使用类似于 velocity/freemarker 等模板框架来生成静态页面，仁者见仁智者见智。
7、这篇文章主要的目的是说 jsp 在大型外网 java web 项目中被淘汰掉，可没说 jsp 可以完全不学，对于一些学生朋友来说，jsp/servlet 等相关的 java web 基础还是要掌握牢的，不然你以为 springmvc 这种框架是基于什么来写的？
8、如页面上有一些权限等等相关的校验，那么这些相关的数据也可以通过 ajax 从接口里拿。
9、对于既可以前端做也可以后端做的逻辑，我建议是放到前端，为什么？因为你的逻辑需要计算资源进行计算，如果放到后端去 run 逻辑，则会消耗带宽&内存&cpu 等等计算资源，你要记住一点就是服务端的计算资源是有限的，而如果放到前端，使用的是客户端的计算资源，这样你的服务端负载就会下降（高并发场景）。类似于数据校验这种，前后端都需要做！
10、前端需要有机制应对后端请求超时以及后端服务宕机的情况，友好的展示给用户。

十、总结

前后端分离并非仅仅只是一种开发模式，而是一种架构模式（前后端分离架构）。千万不要以为只有在撸代码的时候把前端和后端分开就是前后端分离了，需要区分前后端项目。前端项目与后端项目是两个项目，放在两个不同的服务器，需要独立部署，两个不同的工程，两个不同的代码库，不同的开发人员。前后端工程师需要约定交互接口，实现并行开发，开发结束后需要进行独立部署，前端通过 ajax 来调用 http 请求调用后端的 restful api。前端只需要关注页面的样式与动态数据的解析&渲染，而后端专注于具体业务逻辑。

asp

ASP 即 Active Server Pages，是 Microsoft 公司开发的服务器端脚本环境，可用来创建动态交互式网页并建立强大的 web 应用程序。当服务器收到对 ASP 文件的请求时，它会处理包含在用于构建发送给浏览器的 HTML（*Hyper Text Markup Language，超文本置标语言*）网页文件中的服务器端脚本代码。除服务器端脚本代码外，ASP 文件也可以包含文本、HTML（*包括相关的客户端脚本*）和 com 组件调用。[1-2]

ASP 简单、易于维护 ， 是小型页面应用程序的选择 ，在使用 DCOM （*Distributed Component Object Model*）和 MTS（*Microsoft Transaction Server*）的情况下， ASP 甚至可以实现中等规模的企业应用程序。

CGI

CGI 是 Web 服务器运行时外部程序的规范,按 CGI 编写的程序可以扩展服务器功能。CGI 应用程序能与浏览器进行交互,还可通过数据库 API 与数据库服务器等外部数据源进行通信,从数据库服务器中获取数据。格式化为 HTML 文档后，发送给浏览器，也可以将从浏览器获得的数据放到数据库中。几乎所有服务器都支持 CGI,可用任何语言编写 CGI,包括流行的 C、C ++、VB 和 Delphi 等。CGI 分为标准 CGI 和间接 CGI 两种。标准 CGI 使用命令行参数或环境变量表示服务器的详细请求，服务器与浏览器通信采用标准输入输出方式。间接 CGI 又称缓冲 CGI,在 CGI 程序和 CGI 接口之间插入一个缓冲程序，缓冲程序与 CGI 接口间用标准输入输出进行通信。[1]

公共网关接口

CGI(Common Gateway Interface) 是 WWW 技术中最重要的技术之一，有着不可替代

的重要地位。CGI 是外部应用程序（CGI 程序）与 WEB 服务器之间的接口标准，是在 CGI 程序和 Web 服务器之间传递信息的过程。CGI 规范允许 Web 服务器执行外部程序，并将它们的输出发送给 Web 浏览器，CGI 将 Web 的一组简单的静态超媒体文档变成一个完整的新的交互式媒体。

Common Gateway Interface，简称 CGI。在物理上是一段程序，运行在服务器上，提供同客户端 HTML 页面的接口。这样说大概还不好理解。那么我们看一个实际例子：现在的个人主页上大部分都有一个留言本。留言本的工作是这样的：先由用户在客户端输入一些信息，如评论之类的东西。接着用户按一下“发布或提交”（到目前为止工作都在客户端），浏览器把这些信息传送到服务器的 CGI 目录下特定的 CGI 程序中，于是 CGI 程序在服务器上按照预定的方法进行处理。在本例中就是把用户提交的信息存入指定的文件中。然后 CGI 程序将执行结果返回给服务器（webServer），然后服务器将结果返回给客户端，表示请求的任务已经结束。此时用户在浏览器里将看到“留言结束”的字样。整个过程结束。

功能

绝大多数的 CGI 程序被用来解释处理来自表单的输入信息，并在服务器产生相应的处理，或将相应的信息反馈给浏览器。CGI 程序使网页具有交互功能。

处理步骤

(1)通过 Internet 把用户请求送到 web 服务器。

(2)web 服务器接收用户请求并交给 CGI 程序处理。

(3)CGI 程序把处理结果传送给 web 服务器。

(4)web 服务器把结果送回到用户。

优点

CGI 可以为我们提供许多 HTML 无法做到的功能。比如 a.一个记数器 b.顾客信息表格的提交以及统计 c.搜索程序 d.WEB 数据库，用 Html 是没有办法记住客户的任何信息的，就算用户愿意让你知道。用 Html 也是无法把信息记录到某一个特定文件里的。要把客户端的信息记录在服务器的硬盘上，就要用到 CGI。这是 CGI 最重要的作用，它补充了 Html 的不足。是的，仅仅是补充，不是替代。

使在网络服务器下运行外部分应用程序（或网关）成为可能。CGI-BIN 目录是存放 CGI 脚本的地方。这些脚本使 Web 服务器和浏览器能运行外部程序，而无需启动另一个程序。

它是运行在 Web 服务器上的一个程序，并由来自于浏览者的输入触发。CGI 是在 HTTP 服务器下运行外部程序（或网关）的一个接口，它能让网络用户访问远程系统上的使用类型程序，就好像他们在实际使用那些远程计算机一样。

CGI 能够让浏览者与服务器进行交互，如果你曾经遇到过在网络上填表或者进行搜索，就很有可能就是用的 CGI。

尽管 CGI 易于使用，但是当大批人同时使用一个 CGI 应用程序是会反应较慢，网络服务器 速度也会受到很大 影响。CGI 应用程序的优点是可以独立运行。

CGI 应用程序可以由大多数的编程语言编写，如 Perl（Practical Extraction and Report Language)、C\C++、Java 和 Visual Basic 等。不过对于那些没有太多编程经验的网页制作人来说，实在是一个不小的难题。

工作原理

1.浏览器通过 HTML 表单或超链接请求指向一个 CGI 应用程序的 URL。

2.服务器收发到请求。

3.服务器执行指定 CGI 应用程序。

4.CGI 应用程序执行所需要的操作，通常是基于浏览者输入的内容。

5.CGI 应用程序把结果格式化为网络服务器和浏览器能够理解的文档（通常是 HTML 网页）。

6.网络服务器把结果返回到浏览器中。

ASP

ASP（Active Server Pages）：活动服务器页面，就是一个编程环境，在其中，可以混合使用 HTML、脚本语言以及组件来创建服务器端功能强大的 Internet 应用程序。如果你以前创建过一个站点，其中混合了 HTML、脚本语言以及组件，你就可以在其中加入 ASP 程序代码。通过在 HTML 页面中加入脚本命令，你可以创建一个 HTML 用户界面，并且，还可以通过使用组件包含一些商业逻辑规则。组件可以被脚本程序调用，也可以由其他的组件调用。

ASP 的工作原理:

当在 Web 站点中融入 ASP 功能后，将发生以下事情:

1、用户调出站点内容，默认页面的扩展名是.asp。

2、浏览器从服务器上请求 ASP 文件。

3、服务器端脚本开始运行 ASP。

4、ASP 文件按照从上到下的顺序开始处理，执行脚本命令，执行 HTML 页面内容。

5、页面信息发送到浏览器。

因为脚本是在服务器端运行的，所以 Web 服务器完成所有处理后，将标准的 HTML 页面送往浏览器。这意味着，ASP 只能在可以支持的服务器上运行。让脚本驻留在服务器端的另外一个益处是：用户不可能看到原始脚本程序的代码，用户看到的，仅仅是最终产生的 HTML 内容。

CGI：（Common Gateway Interface）Http 服务器与后端程序（如 PHP）进行交互的中间层。

工作原理及处理方式（fork-and-execute 模式）：

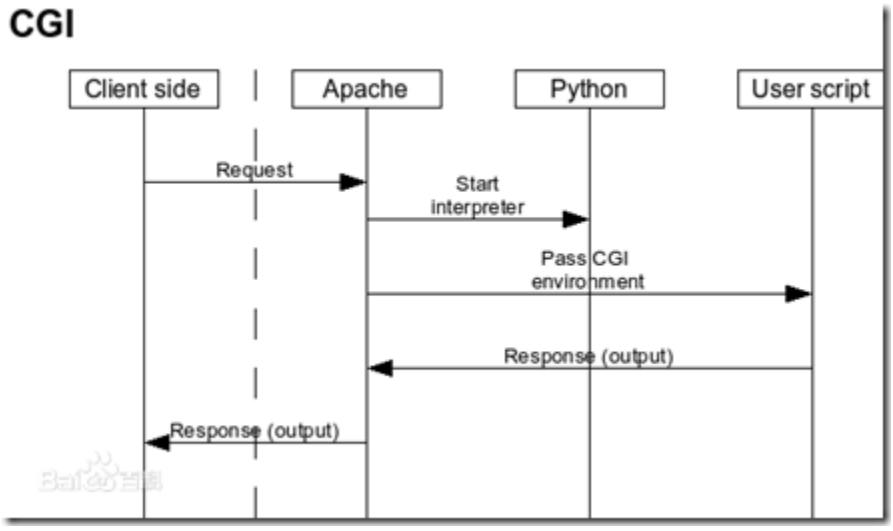
1.当 Web Server 有 Request 到达

2.fork 一个 CGI 进程或线程（配置管理，环境初始化）

3.执行后台脚本

4.将结果返回 Web 服务器。

5.Web 服务器将结果返回给用户。



FastCGI：常驻型（long-live）CGI 形式，经过激活后，不会每次都要花时间去 fork。

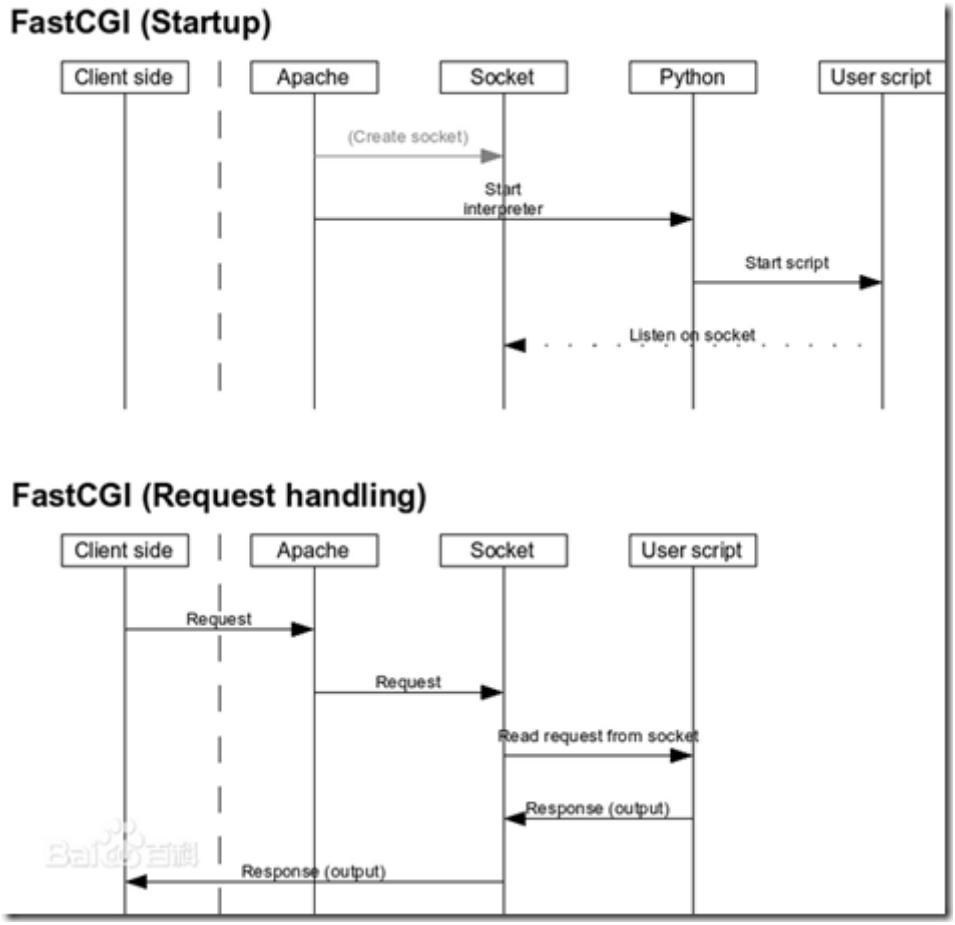
工作原理及处理方式：

1.Web Server 启动时载入 FastCGI 进程管理器（IIS ISAPI 或 Apache Module）

2.FastCGI 进程管理器自身初始化，启动多个 CGI 解释器进程（可见多个 php-cgi 进程），并等待来自 Web Server 的连接

3.当有客户端请求到达 Web Server 时，FastCGI 进程管理器选择并连接到一个 CGI 解释器；Web Server 将 CGI 环境变量和标准输入发送到 FastCGI 子进程 php-cgi。

4.FastCGI 子进程完成处理后将标准输出和错误信息返回 Web Server。当 FastCGI 子进程关闭连接时，请求便告知处理完成。子进程继续响应来自 FastCGI 进程管理器分配的其他请求。



PHP-FPM：只用于 PHP 的 PHP FastCGI 进程管理器。

PHP5.3.3 以后的版本已经集成了 PHP-FPM 了。

php-fpm 提供了更好的 PHP 配置管理方式，可以有效控制内存和进程、可以平滑重载 php 配置。

./configure php 源码的时候，加—enable-fpm 参数可开启 PHP_FMP。

Spawn-FCGI：一个普通的 FastCGI 进程管理器。