我为什么向后端工程师推荐Node.js

科普文一则，说说我对Node.js的一些认识，以及我作为前端工程师为什么会向后端工程师推荐Node.js。

3

> “Node.js 是服务器端的 JavaScript 运行环境，它具有无阻塞(non-blocking)和事件驱动(event-driven)等的特色，Node.js 采用V8引擎，同样，Node.js实现了类似 Apache 和 nginx 的web服务，让你可以通过它来搭建基于 JavaScript的Web App。”

3

我想不仅仅是Node.js，当我们要引入任何一种新技术前都必须要搞清楚几个问题:

1. 我们遇到了什么问题？
2. 这项新技术解决什么问题，是否契合我们遇到的问题？
3. 我们遇到问题的多种解决方案中，当前这项新技术的优势体现在哪儿？
4. 使用新技术，带来哪些新问题，严重么，我们能否解决掉？

我们的问题：Server端阻塞

Node.js被设计用来解决服务端阻塞问题。下面通过一段简单的代码解释何为阻塞：

Js代码：

```
1 //根据ID，在数据库中Persons表中查出Name
2 var name = db.query("selcect name from persons where id=1");
3 //进程等待数据查询完毕，然后使用查询结果。
4 output("name")
```

这段代码的问题是在上面两个语句之间，在整个数据查询的过程中，当前程序进程往往只是在等待结果的返回，这就造成了进程的阻塞。对于高并 发，I/O密集行的网络应用中，一方面进程很长时间处于等待状态，另一方面为了应付新的请求不断的增加新的进程。这样的浪费会导致系统支持QPS远远小于 后端数据服务能够支撑的QPS，成为了系统的瓶颈。而且这样的系统也特别容易被慢链接攻击(客户端故意不接收或减缓接收数据，加长进程等待时间)。

如何解决阻塞问题

可以引入事件处理机制解决这个问题。在查询请求发起之前注册数据加载事件的响应函数，请求发出之后立即将进程交出，而当数据返回后再触发这个事件并在预定好的事件响应函数中继续处理数据：

Js代码：

```
1 //定义如何后续数据处理函数
2 function onDataLoad(name){
3 output("name");
4 }
5 //发起数据请求，同时指定数据返回后的回调函数
6 db.query("selcect name from persons where id=1",onDataLoad);
```

我们看到若按照这个思路解决阻塞问题，首先我们要提供一套高效的异步事件调度机制。而主要用于处理浏览器端的各种交互事件的JavaScript，相对于其他语言，至少有两个关键点特别适合完成这个任务。

为什么JS适合解决阻塞问题

首先JavaScript是一种函数式编程语言，函数编程语言最重要的数学基础是λ演算(lambda calculus) -- 即函数对象可以作为其他函数对象的输入(参数)和输出(返回值)。

这个特性使得为事件指定回调函数变得很容易。特别是JavaScript还支持匿名函数。通过匿名函数的辅助，之前的代码可以进行简写如下：

Js代码：

```
db.query("selcect name from persons where id=1",function(name){
output(name);
});
```

还有另一个关键问题是，异步回调的运行上下文保持(本文暂称其为"状态保持")。我们先来看一段代码来说明何为状态保持：

Js代码：

```
01 //传统同步写法：将查询和结果打印抽象为一个方法
02 function main(){
03 var id ="1";
04 var name = db.query("selcect name from persons where id="+ id);
05 output("person id:"+ id +", name:"+ name);
06 }
07 main();
08
09    前面的写法在传统的阻塞是编程中非常常见，但接下来进行异步改写时会遇到一些困扰：
10
```

```
  Js代码:
//异步写法:
function main(){
var id ="1";
db.query("selcect name from persons where id="+ id,function(name){
output("person id:"+ id +", name:"+ name);//n秒后数据返回后执行回调
});
}
main();
```

细心的朋友可能已经注意到，当等待了n秒数据查询结果返回后执行回调时。回调函数中却仍然使用了main函数的局部变量"id"，而"id"似 乎应该在n秒前走出其作用域。为什么此时"id"仍然可以访问呢，这是因为JavaScript的另外一个重要语言特性：闭包(Closures)。接下 来我来详解闭包的原委。

在复杂的应用中，我们一定会遇到这类场景。即在函数运行时需要访问函数定义时的上下文数据(注意：一定要区分函数定义时和函数运行时两个不同的 时刻)。特别是在异步编程模型中，函数的定义和运行又分处不同的时间段，那么保持上下文的问题变得更加突出了。因为我们在任务执行一半时把资源交出去没有 问题，但当任务需要再次继续时我们必须还原现场。

在这个例子中，db.query作为一个公共的数据库查询方法，把"id"这个业务数据传入给db.query，交由其保存是不太合适的。但我 们可以稍作抽象，让db.query再支持一个需要保持状态的数据对象传入，当数据查询完毕后可以把这些状态数据原封不动的回传。如下：

Js代码：

```
function main(){
var id ="1";
var currentState = new Object();
currentState.person_id = id;
db.query("selcect name from persons where id="+ id, function(name,state){
output("person id:"+ state.person_id +", name:"+ name);
},currentState);//注意currentState是db.query的第三个参数
}
main();
```

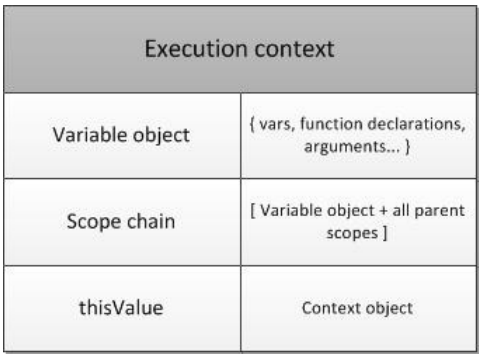
记住这种重要的思路，我们再看看是否还能进一步的抽象？可以的，不过接下的动作之前，我们还要了解在JavaScript中一个函数也是一个对 象。一个函数实例fn除了函数体的定义之外，我们仍然可以在这个函数对象实例之本身扩展其他属性，如fn.a=1;受到这个启发我们尝试把需要保持的状态 直接绑定到函数实例上：

Js代码

```
function main(){
var id ="1";
var currentState =new Object();
currentState.person_id = id;
function onDataLoad(name){
output("person id:"+ onDataLoad.state.person_id +", name:"+ name);
}
onDataLoad.state = currentState ;//为函数指定state属性，用于保持状态
db.query("selcect name from persons where id="+ id, onDataLoad);
}
```

我们做了什么？生成了currentState对象，然后在函数onDataLoad定义时，将currentState绑定给 onDataLoad这个函数实例。那么在onDataLoad运行时，就可以拿到定义时的state对象了。JavaScript的闭包特性就是内置了 这个过程而已。

在每个JavaScript函数运行时，都有一个运行时内部对象称为Execution Context，它包含如下Variable Object(VO,变量对象)， Scope Chain(作用域链)和"this" Value三部分。如图:



其中变量对象VO，包含了所有局部变量的引用。对于main函数，局部变量"id"存储在VO.id内。看起来用VO来代替我们的 currentSate最合适了。但main函数还可能嵌套在其他函数之内，所以我们需要ScopeChain，它是一个包含当前运行函数VO和其所有父 函数scope的数组。

所以在这个例子中，在onDataLoad函数定义时，就为默认为其绑定了一个[[scope]]属性指向其父函数的ExecutionContext的ScopeChain。而当函数onDataLoad执行时，就可以通过[[scope]]属性来访问父函数的VO对 象来找到id，如果父函数的VO中没有id这个属性，就再继续向上查找其祖先的VO对象，直到找到id这个属性或到达最外层返回undefined。也正 是因为这个引用，造成VO的引用计数不为0，在走出作用域时，才不会被垃圾回收。

很多朋友觉得闭包较难理解，其实我们只要能明确的区分函数定义和函数运行两个时机，那么闭包就是让函数在运行时能够访问到函数定义时的所处作用域内的所有变量，或者说函数定义时能访问到什么变量，那么在函数运行时通过相同的变量名一样能访问到。

关于状态保持是本文的重点，在我看到的多数Node.js的介绍文章中并没有详解这里，我们只是知道了要解决阻塞问

题，但是JavaScript解决阻塞问题的优势到底在哪里，作为一名前端工程师，我想有必要花一些篇幅详细解释一下。

而之所以我叫它”状态保持”因为还有一个非常相似的场景可以类比：

用户从A页面提交表单到B页面，如果提交数据校验不通过，则需要返回A页面，同时保持用户在A页面填写的内容并提示用户修改不对的地方。从提交到校验出错再返回继续填写是一个包含网络交互的异步过程，这相当于填写表单这个任务过会儿再继续。

在传统网页开发中，用户的状态通过请求传递到服务端，交由后端状态保持(类似交给db.query的currentSate)。而使用Ajax 的网页，因为并未离开原页面，那么服务端只要负责校验用户提交的数据是否正确即可，发送错误，返回错误处相关信息即可，这就是所谓前端状态保持。可以看到 这个场景里边服务端做的事情变少了，变纯粹了。正如我们的例子中db.query不再存储转发第三个state参数，变得更在轻量。

我们看到通过JavaScript函数式语言特性，匿名函数支持和闭包很漂亮的解决了同步编程到异步编程转化过程中遇到的一系列最重要的问题。但JavaScript是否就是最好的？这就要回答我们引用新技术时需要考虑的最后一个问题了。

使用Node.js是否带来额外的困扰，如何解决？

Node.js性能真的是最好么？不用比较我们也可以得到结论，Node.js做无阻塞编程性能较难做到极致。何为极致？处理一个请求需要占用 多少内存，多少cpu资源，多少带宽，有丁点浪费就不是极致。阻塞式编程浪费了大量进程资源只是在等待，导致大量内存和cpu的浪费。在这方面 Node.js好很多，但也正是因为一些闭包等JavaScript内建机制也会导致资源的浪费，看下面的代码：

Js代码：

```
function main(){
var id = "1";
var str = "..."; //这里局部变量str存储一个2M的字符串
db.query("selcect name from persons where id="+ id,function(name){
output("person id:"+ id +", name:"+ name);//n秒后数据返回后执行回调
});
}
main();
```

至少整个数据查询过程中，变量str所使用的2M内存并不会被释放，而str保持下去可能并没有意义。前面已经解释过闭包的原理，闭包并没有智能到只包起来今后可能被访问到的对象。即使不了解闭包的原理，也可以通过一段简单脚本验证这点：

Js代码：

```
function main(){
var id ="1";
var str ="..."; //这里存储一个2M的字符串
window.setTimeout(function(){
debugger; //我们在这里设置断点
},10000)
}
main();
```

我们在回调函数当中只设置一个断点，并不指明我们要访问哪个变量。然后我们在控制台监视一下，id和str都是可以拿到的。

所以我来猜想一下，性能极端苛刻的场景，无阻塞是未来，但无阻塞发展下去，或者有更轻量的脚本引擎产生，或者JavaScript引擎可能要调整可以disable闭包，或者我们要通过给JS开发静态编译器在代码发布前自动优化我们的代码。

静态编译是如今JavaScript技术领域的又一个热点，我们都知道JavaScript是解释型脚本语言，在运行时自动编译。但是运行时编译只是将代码转为机器码执行，却并未覆盖传统编译型语言在编译阶段所做的任务。比如，语法检查，接口校验，全局性能优化等等。

最常见的JavaScript静态编译就是脚本压缩工具，在代码发布到线上之前，我们通过各种压缩工具，将代码压缩，达到减少网络传输量的问 题。而在这个时间点，已经有越来越多的事情可做，比如：Google利用ClouserComplier提供的系列编译指令，让JavaScript更好 的实现OO编程。也有GWT，CoffeeScript这样的项目，将其他语言编译为JavaScript。在淘宝我们在代码静态编译阶段来解决因 JavaScript细粒度模块化改造引入各种性能问题，也用来对第三方提供JavaScript代码进行一定的安全检查。

我们期待前面的代码经过静态编译器编译后变成如下结果：

Js代码：

```
function main(){
var id = "1";
var str = "..."; //这里局部变量str存储一个2M的字符串
db.query("selcect name from persons where id="+ id,function(name){
output("person id:"+ id +", name:"+ name);
});
str = ""; //通过这一行，及时释放不必要的内存占用。
}
main();
```

除了性能方面的担忧，使用Node.js进行编程增加了代码编写的复杂度。因为我们习惯于阻塞式编程的写法，切换到异步模式编程，往往对于太多多层次的callback函数嵌套弄得不知所措。老赵最近开发了项目JSCEX正是要解决这个问题，它让大家在遵守一些小的约定后，能够仍然保持同步编程的写法进行代码开发。写完的代码同样通过静态编译器编译成异步回调式模式的代码再交给JavaScript引擎执行。

Node.js还要解决什么问题

说了这么多，无阻塞编程要做的还远不止这些。首先需要一个高效的JS引擎，高效的事件池和线程池。另外几乎所有和Node.js交互的传统模块如文件系统，数据访问，HTTP解析，DNS解析都是阻塞式的，都需要额外改造。

Node.js作者极其团队，正是认清问题所在以及JS解决这些问题方面的优势。基于Google开源的高效JavaScript引擎V8，贡献了大量的智慧和精力解决上述大部分问题后才有Node.js横空出世。

当前Node社区如此火热，千余开源的Node.js模块，活跃在WebFramework，WebSocket，RPC，模板引擎，数据抓取服务，图形图像几乎所有工程领域。

后记

本文主要的信息来自Node.js作者在JSConf09，JSConf10上的分享。 而作为前端开发，着重讲了函数式编程，闭包对于无阻塞开发的重要意义。我期待这篇文章能够给前端和后端工程师都带来收获。

同样作为前端开发，不得不再插几句，说说服务端JS能够解决的另一个问题：当前的Web开发前后端使用不同的语言，很多相同的业务逻辑要前后端 分别用不同语言重复实现。比如越来越多重度依赖JavaScript的胖客户端应用，当客户浏览器禁用JavaScript时，则需要使用服务端语言将主 业务流程再实现一次，这即是前端常说的”渐进增强”。

当我们拥有了服务端JavaScript语言，我们自然就会想到能否利用Node.js做到”一次开发，渐进增强”。解决掉这个为小量用户，浪费大量时间的恼人的问题。这方面的实践，YAHOO仍然是先驱，早在一年多前开始YAHOO通过nodejs-yui3项目做了很多卓越的贡献，而淘宝自主开发的前端框架Kissy也有服务端运行的相关尝试。

JavaScript在诞生之时就不仅仅是浏览器端工具，如今JavaScript能够再一次回到服务端展示拳脚，感谢V8，感谢NodeJS作者，团队和社区的诸多贡献者，祝Node好运，JavaScript好运。

关于作者

李穆，前端工程师，就职于淘宝广告技术部架构组，淘宝花名：李牧，专注淘宝广告引擎和业务系统前端开发。

来自：http://www.infoq.com/cn/articles/why-recommend-nodejs

Node.js

最后修改时间 **2011-11-28 11:34** 提问时间 **2011-11-28 11:33**

jopen
51 3 7
答案被采用率: 0%