

เลียนดายไม่ได้อ่าน

จาลวสคริปต์ฝั่งเซิร์ฟเวอร์
(Node.js ຂັບຍ່ອ)

ເລີມ 1

ประวัติการแก้ไข

ครั้งที่	วันที่	รายละเอียดการแก้ไข
1	30 ธ.ค. 2558	เริ่มสร้าง และเผยแพร่ผลงาน
2	1 ม.ค. 2559	แก้เนื้อหาที่ผิด
3	8 ม.ค. 2559	เพิ่มนื้อหา
4	27 ม.ค. 2559	แก้เนื้อหาที่ผิดตามคำแนะนำของคุณ Sarin Achawaranont
5	13 มี.ค. 2559	ปรับปรุงเนื้อหาใหม่
6	11 เม.ย. 2559	แก้ไขลึกน้อย
7	27 เม.ย. 2559	แก้ไขให้สอดคล้องกับ Node.js เวอร์ชัน 6 ที่ออกมาใหม่

ถ้าท่านดาวน์โหลดทิ้งไว้บ้าน แล้วเพิ่งมาเปิดอ่าน ก็ขอรบกวนให้โหลดใหม่อีกครั้งที่

http://www.patanasongsivilai.com/itebook_form.html

เพื่อผลอัพเดตแก้ไข pdf ตัวใหม่เข้าไป หรือใครเปิดดาวน์โหลดมาจากที่อื่น

ก็อาจพลาดเวอร์ชันใหม่ล่าสุดได้ครับ

และรบกวนช่วย**กรอกแบบสอบถาม** ตามลิงค์ข้างบนด้วยนะครับ

แอดมินโซ โอน้อยอก

(จตุรพัชร พัฒนทรงศิริวไล)

30 ธันวาคม 2558

ถ้าสนใจเกี่ยวกับเพจด้านไอที ก็ติดตามได้ที่ <https://www.facebook.com/programmerthai/>

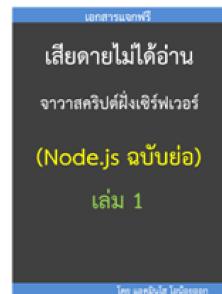
EBook เเละนี้ส่วนลิขสิทธิ์ตามกฎหมาย ห้ามมิให้ซื้อ นำไปเผยแพร่ต่อสาธารณะ เพื่อประโยชน์ในการค้า หรืออื่นๆ โดยไม่ได้รับความยินยอมเป็นลายลักษณ์อักษรจากผู้เขียน

คํานํา



สืบเนื่องมาจากผมได้แต่งหนังสือ **JAVA SCRIPPT (JavaScript)** มาตรฐานตัวใหม่ **ECMAScript 2015** หรือเรียกสั้น ๆ ว่า “**ES6**” หรือ “**ES6 Harmony**” ซึ่งประกาศออกมาราสุด เมื่อกลางเดือนมิถุนายน พ.ศ. 2558 โดยเล่นนี้ตีพิมพ์และจัดจำหน่ายโดยชีเอ็ด (ส่วนในปี 2559 ก็จะเป็น ES7 (เพิ่มฟีเจอร์ใหม่เข้ามาบ้างเดียว))

สำหรับหนังสือเล่มนี้ที่ท่านเปิดอ่าน ผมตั้งใจจะแจกจ่ายฟรี เพื่ออธิบายJAVA SCRIPPT ในอีกมุมมองหนึ่ง ซึ่งจะขยายเนื้อหาจากหนังสือที่ผมเขียนไว้ข้างต้น โดยจะแสดงให้เห็นว่า JAVA SCRIPPT ไม่ได้จำกัดแค่การทำงานอยู่บนเว็บเบราว์เซอร์ (Web browser) เท่านั้น แต่ยังสามารถอยู่ฝั่งเซิร์ฟเวอร์ได้ ด้วยการใช้ **Node.js** ไม่ต่างอะไรกับภาษาSCRIPPT ดัง ๆ เช่น PHP, ASP หรือ JSP เป็นต้น ...เจ้วป่าว



และถ้ามีเวลา ผมก็อยากเขียนในแบบอื่น เพื่อแสดงให้เห็นถึงความสามารถต่าง ๆ ที่ซ่อนเร้นอยู่เยอะมากในJAVA SCRIPPT เช่นเดือนเป็น “การเปิดโลกJAVA SCRIPPT เดอะชีรีส์” (ถ้าเป็นไปได้จะครับ)

หากเนื้อหามีอะไรผิดพลาดไป เช่น ให้ข้อมูลผิด สะกดอะไรผิดไป มุมแป็กบ้าง ขำบ้าง หรืออ่านแล้วมึนงงไป 7 วัน เป็นต้น ผมก็ขอภัยมา ณ โอกาสนี้ด้วย และถ้าคุณเข้าใจ ไม่เข้าใจยังไง ก็สามารถซื้อแบบได้ตลอดเวลา



ที่สำคัญผมลองเขียนเป็นน้ำจิ้มเล็กน้อยก่อน เป็นแค่ พื้นฐานเบื้องต้นเท่านั้น ทฤษฎีคงไม่ได้เจาะลึกอะไรมาก

...แต่ก็ตั้งใจพยายามเขียนเรื่อย ๆ พร้อมทั้งหมั่นอัพเดตเนื้อหา ขึ้นอยู่กับเวลา โอกาส และความสามารถจะอำนวย

หนังสือเล่มนี้เหมาะกับใคร

ก่อนจะอ่านหนังสือเล่มนี้ ผู้ต้องถามว่า ...คุณสนใจสิ่งเหล่านี้หรือไม่?

- คุณสนใจการสร้างเว็บแอพพลิเคชั่น (Web application) โดยใช้แค่ 3 ภาษาเท่านั้น ได้แก่ HTML, CSS และใช้ภาษาสคริปต์ หรือไม่?
- ไม่ใช้ภาษาสคริปต์ดัง ๆ ในฝั่งเซิร์ฟเวอร์ เช่น PHP, ASP และ JSP เป็นต้น
- เวลาจะรันสคริปต์ คุณไม่ต้องติดตั้งซอฟต์แวร์ที่เป็นเว็บเซิร์ฟเวอร์ (Web server) เช่น XAMPP, IIS และ Apache Tomcat เป็นต้น เพื่อใช้รันไฟล์สคริปต์ PHP, ASP และ JSP ตามลำดับ
- ต้องการเขียนเว็บแอพพลิเคชั่นแบบเรียลไทม์ พร้อมทั้งรองรับโหลดเยอะ ๆ ไม่ว่าจะเป็นการติดต่อเข้ามาของคลื่นออนไลน์จำนวนมาก รวมทั้งยุ่งเกี่ยวกับข้อมูลมหาศาล (Big Data)
- คุณไม่อยากปวดหัว เวลาเขียนโปรแกรมเพื่อแทกเทรด (Thread)
- และคุณอยากรู้ว่าทำไม Node.js ถึงทำให้ว่างการ Back-end **ต้องหันมากองดู ?**

“ถ้าคุณสนใจสิ่งเหล่านี้ ก็อ่านต่อได้เลยครับ”

ก่อนจะอ่านหนังสือเล่มนี้

- 1) คุณต้องมีพื้นฐานของคอมพิวเตอร์มาก่อน ...ก็จะง่ายซี เพราะเนื้อหาเล่มนี้ยังคงความลึก ๆ ถ้าไปให้มากก็หมายความว่า บัญชี สถาปนิก นักบิน พวกรู้สักส่วนหนึ่งแล้วนั่นเอง
- 2) ควรรู้ภาษาโปรแกรมที่ใช้เขียนหน้าเว็บ ได้แก่ CSS กับ HTML เพราะผมกล่าวถึงมันด้วย
- 3) ต้องมีพื้นฐานภาษาสคริปต์มาก่อน หรืออย่างน้อยก็ควรรู้ภาษา C, C++, C# และ Java เป็นต้น เพื่อจะได้เข้าใจโค้ดในหนังสือได้ไม่ยากเย็นอะไรนัก
- 4) ควรศึกษา ES6 (ภาษาสคริปต์ตัวใหม่) มาบ้าง เพราะอย่างน้อยผมก็พูดถึงนิดหน่อย
- 5) ไปที่ลิงค์ http://www.patanasongsivilai.com/itebook_form.html เพื่อดาวน์โหลด “วิธีติดตั้ง Node.js และ npm เป็นต้น” แล้วขอร้องเอกสารนี้ กรุณาอ่านมันก่อน มีประโยชน์เดียวคุณจะง่าย แล้วมาตอบนินทาผมในใจว่า ...เอ็งเขียนอะไรของมันฟะ ไม่รู้เรื่อง

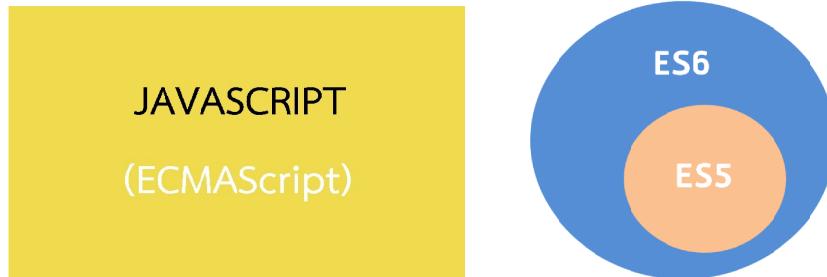
อธิบายเกี่ยวกับ ES6

ในปัจจุบันองค์กร Ecma International (องค์กรจัดการมาตรฐานแห่งยุโรป) จะเป็นผู้กำหนดมาตรฐานภาษาสคริปต์ ซึ่งมาตรฐานของมันจะมีชื่อเรียกว่า “ECMA-262” ส่วนตัวภาษาจาวาสคริปต์นั้น จะมีชื่อเรียกเต็มยศอย่างเป็นทางการว่า “ภาษา ECMAScript”

สำหรับจาวาสคริปต์ไม่ได้ออกเวอร์ชันล่าสุดนานนานเกือบ 6 ปี (เวอร์ชันเก่าคือ ES5) และล่าสุดเมื่อกลางเดือนมิถุนายน พ.ศ. 2558 องค์กร Ecma International ได้ออกมาตรฐานจาวาสคริปต์ตัวใหม่เป็น ECMAScript รุ่นที่ 6 **เป็นการเปลี่ยนแปลงครั้งใหญ่** ซึ่งชื่อเต็ม ๆ ของมันคือ ECMAScript 2015 แต่ส่วนใหญ่จะเรียกสั้น ๆ ไปเลยกว่า ES6 (มันยังมีชื่อเล่นอีกชื่อคือ ECMAScript Harmony หรือจะเรียกว่า ES6 Harmony ก็ได้เช่นกัน)

และแน่นอนครับ Node.js ก็ต้องรองรับ ES6 ได้ด้วยเช่นกัน แต่ผู้คงไม่ได้ลงลึกเกี่ยวกับ ES6 ในเล่มนี้ เพราะเนื้อหามักค่อนข้างเยอะ และผมก็แยกเขียนอีกเล่มหนึ่ง

*** ในช่วงที่ผมเขียนหนังสือนี้ ตัว Node.js (เวอร์ชัน 6) จะรองรับ ES6 ได้แค่ **93%** เท่านั้น



หมายเหตุ ES6 มันฟีเจอร์ใหญ่มาก จึงทำให้บางอย่างถูกเลื่อนออกไปใน ES7 (ECMAScript 2016) และเวอร์ชันต่อ ๆ ไป ประมาณว่าเป็นเวอร์ชันอัพเกรด เพิ่มเติมนิดเดียวจาก ES6 (เปลี่ยนเล็ก ไม่ได้เปลี่ยนใหญ่)

สำหรับ Node.js เวอร์ชัน 5 (เวอร์ชันเก่า) เวลารันจาวาสคริปต์ ต้องใช้แฟล็กเป็น **--use-strict** ดังภาพ

```
C:\ES6>node --use-strict server.js
Server running at http://127.0.0.1:3000/
```

ในหนังสือ "พัฒนาเว็บแอปพลิเคชันด้วย JavaScript" ที่ผมเขียน จะเห็นการใช้แฟล็กนี้

แต่ถ้าเป็น Node.js เวอร์ชัน 6 เวลารันจาวาสคริปต์ ไม่ต้องใช้แฟล็กเป็น **--use-strict** ดังภาพ

```
C:\ES6>node server.js ←
Server running at http://127.0.0.1:1337/
```

แนะนำ Node.js

ประวัติ Node.js

ประวัติของ Node.js มาจาก <https://en.wikipedia.org/wiki/Node.js> (ถ้าข้อมูลผิดพลาดทาง wiki นะ อิ ๆ)



Node.js ถูกสร้างขึ้น และเผยแพร่ให้ใช้งานบน Linux ในปี ค.ศ. 2552 โดยคุณ “Ryan Dahl” พร้อมเพื่อนทำงานของเขาร่วมกับบริษัท Joyent

Ryan Dahl ได้แรงบัลดาลใจหลังจากเห็นแบบสถาณะ progress bar บน Flickr เวลาอัปโหลดไฟล์ ซึ่งเว็บเบราว์เซอร์จะไม่รู้ว่าไฟล์กำลังถูกอัปโหลด และติดต่อกับเซิร์ฟเวอร์ แต่ทว่าเขาต้องการวิธีที่ง่ายกว่านั้น จึงเกิดเป็น Node.js ขึ้นมา และเขาได้นำ Node.js ไปพูดครั้งแรก ที่การประชุม JSConf เมื่อ 8 พ.ย. 2552 อีกด้วย ลองดูคลิปที่เข้าบรรยายได้เลยครับ

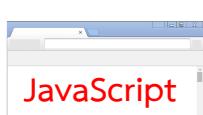
<https://www.youtube.com/watch?v=ztspvPYybIY>

รูปภาพคุณ **Ryan Dahl** ประบิดาผู้สร้าง Node.js ให้เกิดขึ้นมาบนโลกคอมพิวเตอร์

(ที่มาภาพ https://en.wikipedia.org/wiki/Node.js#/media/File:Ryan_Dahl.jpg)

Nodes.js คืออะไร

ให้คุณลืมประวัติ Node.js ไปก่อน เพราะไม่ค่อยสัมพันธ์กับความหมายมันเท่าไร



สำหรับเว็บเบราว์เซอร์ต่าง ๆ ไม่ว่าจะเป็น Internet Explorer (IE), Firefox และ Google Chrome เป็นต้น พวกมันจะมี JavaScript 引擎 (JavaScript engine) ติดตั้งอยู่ภายใน และใช้เป็นตัวแปลงภาษา และประมวลผลโค้ดที่เขียนด้วย JavaScript

ส่วน Node.js ก็สามารถรัน JavaScript ได้ เช่นกัน ไม่ต่างอะไรกับเว็บเบราว์เซอร์

พูดอีกนัยหนึ่งเรารัน JavaScript นอกเว็บเบราว์เซอร์ได้ด้วย Node.js (อยากรู้)



นับตั้งแต่ Node.js อุ๊ว อุ๊ว เกิดขึ้นมาบนโลกแห่งนี้ มันจึงเหมือนติดปีกให้ JavaScript บินได้ จริง ๆ เลยนะ

นิยาม

คราวนี้ผมจะให้ดูนิยาม Node.js อย่างเป็นทางการ จากเว็บไซต์ของมัน (<https://nodejs.org/en/>)

Node.js® is a JavaScript runtime built on **Chrome's V8 JavaScript engine**. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

ซึ่งผมได้พยายามแปล ให้มันสละสลวยเท่าที่จะทำได้ดังนี้

ไม่ใช่รายการ AF

หรือ “หู อะค่าเดมี่ แฟฟเนเกะเชีย”



Node.js คือตัวรันไทม์ (runtime) ของ **ภาษาจาวาสคริปต์** โดยมันถูกสร้างขึ้นมาบน V8 ซึ่งเป็นจากวาริปเตอร์เอนจินของ Google Chrome

สำหรับ Node.js จะทำงานแบบ event-driven และเป็น non-blocking I/O ด้วยเหตุนี้จึงทำให้มันเบาไว้ และเต็มเปี่ยมไปด้วยประสิทธิภาพ (lightweight and efficient)

จากนิยามที่ผมแปลมาให้ ไม่รู้ว่าคุณจะสงสัยคำศัพท์เหล่านี้หรือไม่ ?

- event-driven
- non-blocking I/O
- lightweight

???

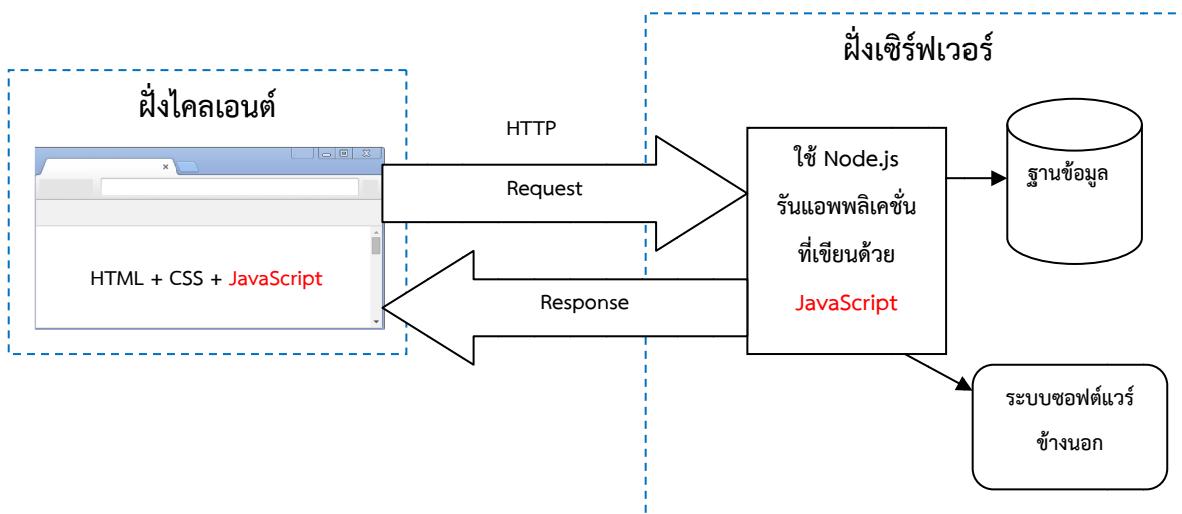
ชื่อเหมือนรุ่นของนักมวย

ถ้าคุณเข้าใจคำศัพท์ที่ผมยกมาให้ดู ก็จะเข้าใจคอนเซปต์ของ Node.js ทั้งหมด ซึ่งผมจะพยายามอธิบายคำพวกนี้ไปเรื่อยๆ และกัน... ก็แบบว่า ไม่รู้จะอธิบายยังไง ให้เข้าใจในตอนนี้ 555+

หมายเหตุ **V8** จะเป็นจาวาสคริปเตอร์เอนจินที่เป็นโอเพ่นซอร์ส ซึ่งถูกเขียนขึ้นด้วย C++ และมันก็ได้ถูกติดตั้งไว้ใน Google Chrome ...สำหรับ Node.js ก็ถูกสร้างขึ้นบน V8 ที่ว่านี้แหละ

แล้ว Node.js มันเอาไปใช้ทำงานอะไร

จริง ๆ แล้ว Node.js มันประยุกต์ใช้งานได้หลากหลายนะครับ แต่ที่ผมเห็นส่วนใหญ่จะนิยมนำไปพัฒนาเว็บแอ�플ิเคชัน (Web applications) โดยให้มันทำงานอยู่ฝั่งเซิร์ฟเวอร์ ตามภาพต่อไปนี้



ตามภาพในฝั่ง klient ซึ่งก็คือเว็บเบราว์เซอร์ จะต้องใช้ 3 ภาษาไฟต์บังคับ ในการพัฒนาหน้าเว็บไซต์ ซึ่ง 3 ภาษาี้นี้ มันเปรียบเสมือนเสาหลักค้ำฟ้าฝั่งหน้าเว็บ โดยจะประกอบไปด้วย

1. ภาษา HTML (ปัจจุบันเป็นเวอร์ชัน HTML5) เอาไว้แสดงหน้าเว็บให้ใช้
2. ภาษา CSS (ปัจจุบันเป็นเวอร์ชัน CSS3) เอาไว้ตกแต่งหน้าเว็บให้สวยงาม
3. จาสวสクリปต์ (ปัจจุบันเป็น ES6) จะทำให้เว็บมันไดนามิก (Dynamic) และดูยืดหยุ่น และมีชีวิตชีวา

ถ้าคุณมองดูฝั่งเซิร์ฟเวอร์ในภาพที่ผมวาดให้ดู ก็จะเห็นว่าแอ�플ิเคชันถูกพัฒนาด้วยจาสวสクリปต์ ที่ถูกรันด้วย Node.js โดยปราศจากภาษาสคริปต์ดัง ๆ เช่น PHP, ASP และ JSP เป็นต้น

สำหรับการพัฒนาเว็บแอ�플ิเคชันด้วยจาสวสクリปต์ล้วน ๆ เขาจะเรียกว่า “Full Stack JavaScript” และในเว็บฝรั่ง ผมเห็นเขาเรียกอีกคำหนึ่งว่า “Isomorphic JavaScript” ด้วยนั้ (Isomorphic ที่แปลว่า “ซึ่งมีรูปร่างสันฐานเหมือนกัน”)

...แต่ถึงกระนั้นก็ได้ การสร้างหน้าตาเว็บ (UI) ยังต้องพึ่งพาภาษา HTML กับ CSS อญี่ดี

หมายเหตุ Node.js สามารถทำงานตามลำพัง (Stand alone) เป็นแอ�플ิเคชันโดด ๆ โดยไม่ต้องทำเป็นเว็บแอ�플ิเคชันก็ได้นะ

บุจชา จำเป็นต้องใช้ Node.js ในฝั่งเซิร์ฟเวอร์หรือไม่?

คำตอบ ไม่จำเป็น เพราะมันมีภาษาทางเลือกเยอะ นอกจาจาวาสคริปต์ 😊

อีกทั้งตัว Node.js **ไม่ใช่แฟนทำแทนทุกเรื่องได้** เพราะมันไม่ได้แก้ปัญหาการเขียนโปรแกรมฝั่งเซิร์ฟเวอร์ได้ครอบจักรวาลหrogan แต่ Node.js จะตอบโจทย์กรณีที่งานหลังบ้านมีโหลดเยอะ ๆ เวลาติดต่อกับ IO เช่น โคลอนต์ติดต่อเข้ามาเยอะมาก ๆ ติดต่อฐานข้อมูลถี่ ๆ หรืออ่านเขียนข้อมูลจากฮาร์ดดิสก์หนัก ๆ เป็นต้น ซึ่งมันจะทำงานได้รวดเร็ว ให้ลื่น ปูดปำ ...จนเป็นเครื่องหมายการค้าเลยนะ



ด้วยเหตุนี้มันจึงเหมาะสมจะใช้สร้างเว็บแอพพลิเคชันแบบเรียลไทม์มากๆ (Real time web applications) หรือ แอพพลิเคชันเครือข่ายที่ขยายขนาดได้ (เทคโนโลยีที่ปรับเปลี่ยนตัวไปหน่อยจากคำว่า “scalable network applications”)

แต่ถึงกระนั้นก็ต้องเรียนรู้ Node.js ก็เหมือนเปิดโลกทัศน์ของภาษาจาวาสคริปต์ ในมุมมองที่คุณไม่เคยเห็นมาก่อน อย่างน้อยก็ “รู้ไว้ใช้ว่า ใส่บ่าแบกหาม” เพราะจาวาสคริปต์ในตอนนี้ คุณคงไม่ถี่ยงนะครับว่า ...มันผูกขาดการพัฒนาเว็บแอพพลิเคชันฝั่งโคลอนต์ เป็นที่เรียบร้อยรองเรียนจีน ...เลี้ยว แคม Node.js ก็ยังเกิดมาเพื่อสนับสนุนวงการ Backend อีกด้วย



“ถ้า ...ชนใดไม่มีดันตรีกាល ในสันดานเป็นคนชอบกลนัก
แล้วโปรแกรมเมอร์พัฒนาเว็บแอพพลิเคชัน
ไม่รู้จักจาวาสคริปต์ ก็เป็นคนชอบกลนัก ...เช่นเดียวกัน”

ขอบกระซิบnidหนึ่ง ตัว Node.js เวลาติดตั้งมันจะเล็กนิดเดียว และไม่ต้องติดตั้งซอฟต์แวร์ซึ่งทำหน้าที่เป็นเว็บเซิร์ฟเวอร์ให้เสียเวลาอีกด้วย ดังนั้นผู้สร้างเขาจึงกล้าบอกว่า Node.js มัน lightweight หรือเบาวินน์เอง



ถ้าถามว่า Node.js นิยมมากหรือไม่?

สำหรับในเมืองไทยก็มีการกล่าวถึงอย่างมาก และมีอพฟิศหลายแห่งใช้กันอยู่พอควร ส่วนในต่างประเทศดังมาก ๆ ครับ โดยภาพต่อไปนี้เป็นแค่ตัวอย่างบริษัท หรือหน่วยงานดัง ๆ ที่ใช้งานมั่น

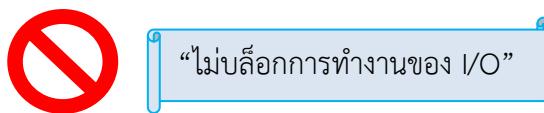


สาเหตุที่มันดังก็เพราะประสิทธิภาพตามที่กล่าวมาในหัวข้อก่อน ๆ ที่สำคัญมันเป็นโอเพ่นซอร์ส ใช้งานได้ฟรี มีไลบรารีต่าง ๆ ให้ใช้บน Node.js เยอะมาก ๆ และมีเฟรมเวิร์ค (Frame work) ตลอดจนมีหลายแพลทฟอร์ม (Platform) ที่นำ Node.js ไปต่อยอดเต็มไปหมด

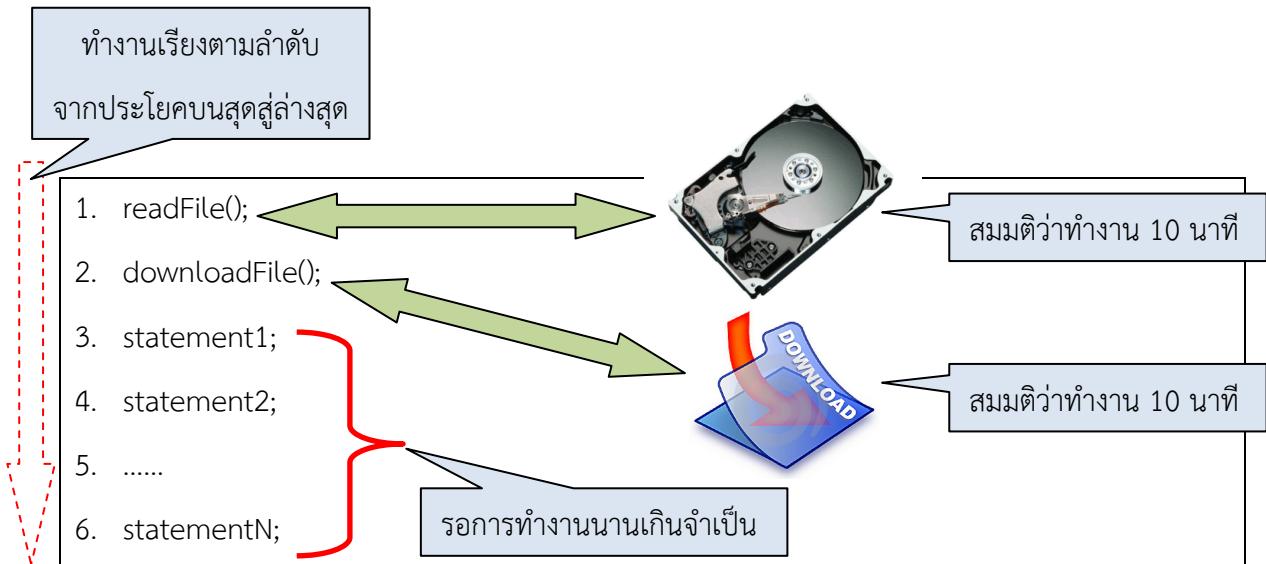
แนวคิดของ Node.js

ก่อนอื่นต้องพูดถึงคำว่า **I/O** ซึ่งเต็มมันคือ “Input/Output” ถ้าแปลเป็นไทยก็คือ “รับเข้า/ส่งออก” ซึ่งในที่นี้จะหมายถึง การรับเข้าและส่งออกข้อมูลจากโลกภายนอก เช่น การเปิดพอร์ตเพื่อรอให้คลาวน์ติดต่อเข้ามา การดาวน์โหลดไฟล์ การอ่านไฟล์จากฮาร์ดดิสก์ และการติดต่อฐานข้อมูล เป็นต้น

เวลาคุณ “คิดจะพัก คิดถึง คิทแคน” ... เยย์ไม่ช้าย คิดจะใช้งาน Node.js ต้องคิดแบบ **non-blocking I/O** ซึ่งประโยชน์นี้ถ้าแปลเป็นไทย จะได้ใจความว่า ...



จากนิยามดังกล่าว อาจฟังดูแล้วง ...แต่เดียวผมจะยกตัวอย่างปัญหา ที่ถือว่าใหญ่มาก ๆ ๆ เมื่อเกิดเหตุการณ์นี้ก็ต้องการการทำงานของ I/O ดังตัวอย่างโค้ดง่ายจ้าย ต่อไปนี้



ในโค้ดตัวอย่างที่เห็น มันจะทำงานเรียงตามลำดับ จากประโยชน์บนสุดสู่ล่างสุด โดยมีรายละเอียดดังนี้

- บรรทัดที่ 1 จะสมมติว่าอ่านไฟล์นาน 10 นาที (ติดต่อ I/O)
- บรรทัดที่ 2 จะสมมติว่าดาวน์โหลดไฟล์นาน 10 นาที (ติดต่อ I/O)
- สำหรับบรรทัดที่ 2 ต้องรอให้บรรทัด 1 ทำงานเสร็จก่อน ซึ่งเบ็ดเสร็จแล้วประโยชน์บรรทัดที่ 1 กับ 2 จะทำงานรวมกันทั้งสิ้น 20 นาที ...นานจนนั่งจีบหลับไปรอบหนึ่งได้

คุณจะเห็นว่าบรรทัด 1 มันบล็อกการทำงานของบรรทัด 2 (บล็อกการทำงานของ I/O) ซึ่งส่งผลทำให้ ชี ๆ ประโยชน์คำสั่งที่เหลือตามมาคือ statement1, statement2 จนถึง statementN ต้องรอนานขึ้นโดยไม่จำเป็น



ที่มาภาพ www.freedigitalphotos.net เครดิตรูปโดยคุณ digitalart

จากภาพที่เห็น ให้คุณลองนึกถึงการยืนเข้าแถวต่อคิว เมื่อแต่ละประโยชน์คำสั่งติดต่อกับ I/O ก็จะเหมือนการต่อคิว ต้องรอให้คนแรกทำเสร็จก่อน คิวถัดไปถึงทำงานได้

สำหรับการเขียนโปรแกรมแบบนี้ จะเรียกว่าทำงานแบบซิงโครนัส (Synchronous) ถ้าแปลตรงตัวก็ได้ว่า “ทำงานเป็นจังหวะสอดคล้องกัน” ซึ่งแต่ละประโยชน์คำสั่งจะมีจังหวะการทำงาน (ต้องทำให้เสร็จก่อน ถึงไปทำประโยชน์คัดไปได้)

แล้วจะแก้ปัญหารือการบล็อกการทำงานของ I/O อย่างไรดีละที่เนี่ย ?

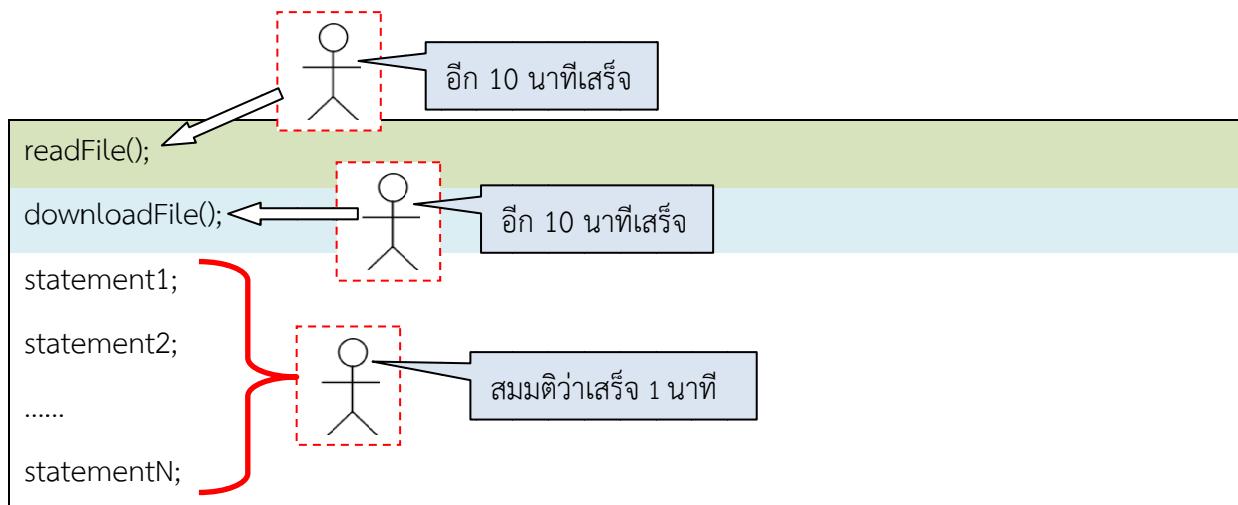
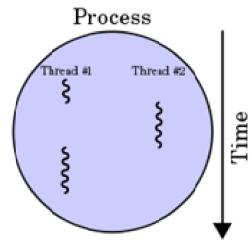
นานะปัญหาทุกอย่างมีทางออก ถ้าออกไม่ได้ก็ให้ไปทิ่ทางเข้า

...อ้ายล้อเล่น ให้คุณดูวิธีแก้ปัญหานี้หัวข้อถัดไปนะครับ 😊

แก้ปัญหาด้วยการใช้-thread

จากปัญหานี้ สามารถแก้ไขได้ด้วยการใช้-thread (Thread) ซึ่งก็คือprocess (Process) ย่อย ๆ โดยอาจเปรียบเทียบthread มันก็คือ คนงาน ส่วนโดยก็เหมือน งาน ที่ต้องมีคนงานมาทำอีกที

สำหรับบางภาษา เช่น Java และ C++ เป็นต้น สามารถเขียนให้มีthreadมากกว่า 1 ตัว (แตกthread) ซึ่งสามารถทำงานคู่ขนานพร้อม ๆ กันได้ ในเวลาเดียวกัน ดังตัวอย่างโค้ดต่อไปนี้

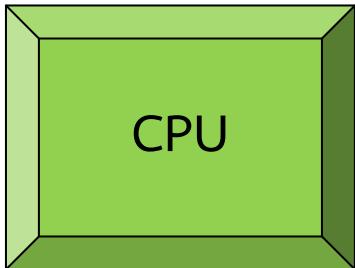


ในภาพจะมีthread 3 ตัว ซึ่งก็เหมือนคนงาน 3 คน ที่มาทำงานโดยคัด (ประมวลผล) ดังรายละเอียดต่อไปนี้

- เทรดคณ> จะประมวลผลประโยชน์ readFile();
- เทรดคณ> จะประมวลผลประโยชน์ downloadFile();
- เทรดคณ> จะประมวลผลประโยชน์ statement1, statement2 จนถึง statementN

โดยthreadทั้งสามตัวต้องกล่าว จะทำงานคู่ขนานไปพร้อม ๆ กัน ไม่ได้ทำงานเรียงตามลำดับ ซึ่งส่งผลทำให้ประโยชน์ statement1, statement2 จนถึง statementN ทำงานเสร็จก่อน (สมมติว่าเสร็จภายใน 1 นาที) ขณะที่ประโยชน์ readFile(); และ downloadFile(); ทั้งคู่จะทำงานเสร็จภายใน 10 นาทีให้หลัง (ไม่ใช่เวลารวม 20 นาที) ด้วยแนวคิดเช่นนี้จึงสามารถแก้ปัญหา จะไม่เกิดเหตุการณ์บล็อกการทำงานของ I/O

การที่เทรดมันทำงานคู่ขนานไปพร้อม ๆ กันได้ ก็ เพราะตัว CPU เป็นคนค่อยบริหารจัดการเทรดให้ทำงานคู่ขนาน ซึ่งจะเรียกความสามารถนี้ของ CPU ว่า “Multithreading” แต่รายละเอียดเบื้องหลังการทำงานคงต้องไปหาอ่านในวิชา OS (Operating System) แล้วละครับว่า ...มันทำงานยังไง แต่ถ้าคุณสนใจก็ลองดูได้ที่ http://www.no-poor.com/dssandos/os_ch03_process.htm



แต่การเขียนโปรแกรมให้มีหลาย ๆ เทรด ทำงานพร้อมกัน มันจะเขียน ยุ่งยากมากแบบยกกำลังสอง² นะซี่ ... เพราะมันเสี่ยงให้เกิดปัญหาหลาย ๆ อย่าง เช่น ปัญหาแต่ละเทรดแย่งชิงทรัพยากรกันเอง (Race condition) หรือแต่ละเทรดต่างรอค่อยการทำงานซึ่งกันและกัน (Deadlock) เป็นต้น

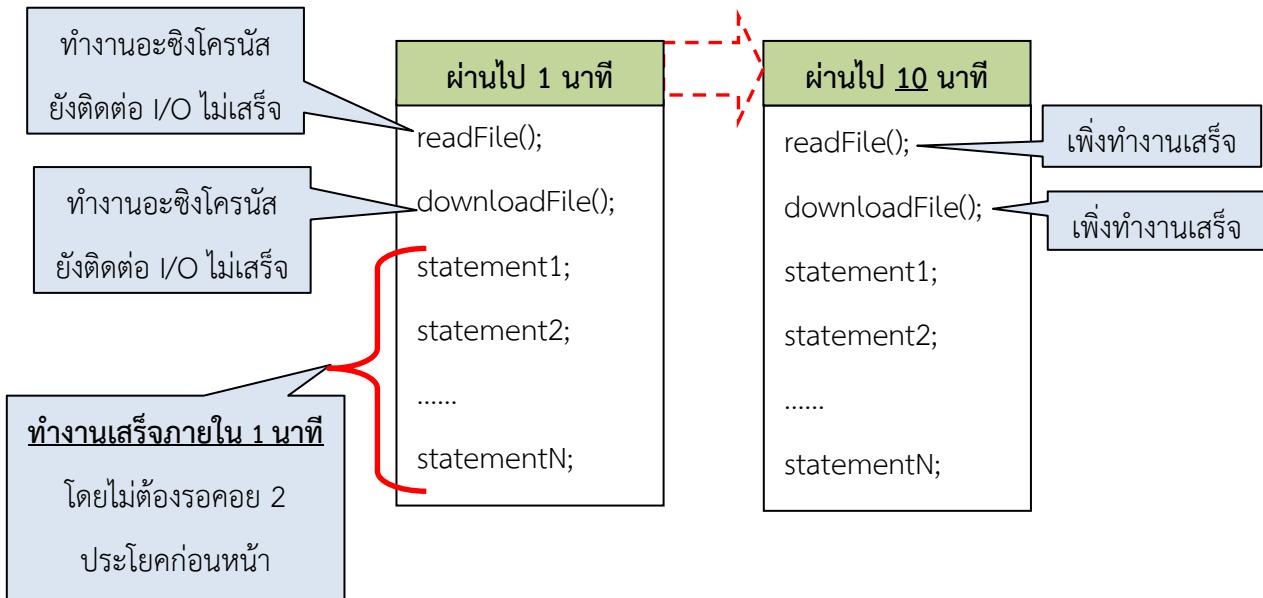
แต่ถ้าเป็น Node.js จะมีเพียงแค่ เทรดเดียว (Single Thread) ...ไม่มีการสร้างเทรด และแตกเทรด ให้ปวน เศียรเรียนเกล้าเหมือนภาษาอื่นเขา

คุณอาจสงสัยแล้วว่า ทำไมผมถึงอธิบายเทรดให้ยืดยาว เสียเวลาทำไม ? ในเมื่อ Node.js เขียนโปรแกรมแตกเทรดไม่ได้ ...จริง ๆ และผมต้องการสื่อถึงคนที่มาจากภาษาอื่น ที่เขามีเทรดใช้งานกัน เมื่อเปลี่ยนมาใช้ Node.js มันจะเป็น เขตปลดการแตกเทรด แต่ก็ทำให้เกิด non-blocking I/O ได้เหมือนกันนะครับแบบ

เปลี่ยนมุมมองการเขียนโปรแกรมเลี้ยงใหม่

เวลาเขียนโปรแกรมใน Node.js จะต้องเขียนแบบอะซิงโครนัส (Asynchronous programming) เพื่อไม่ให้เกิดการบล็อกการทำงานของ I/O (อย่าเพิ่งขวัดคิ้วนะครับ)

คำว่า “อะซิงโครนัส” อาจแปลตรงตัวได้ว่า “การทำงานที่ไม่พร้อมกัน” แต่ถ้าใช้ในความหมายของการเขียนโปรแกรมแบบอะซิงโครนัส ก็อาจหมายถึง การทำงานของโปรแกรม ที่ไม่ต้องรอค่อยให้ประโยชน์คำสั่งได้คำสั่งหนึ่งทำงานเสร็จก่อน ประโยชน์อื่นที่ตามมาทีหลัง สามารถทำงานได้ทันที ...ดังโค้ดตัวอย่างในหน้าถัดไป

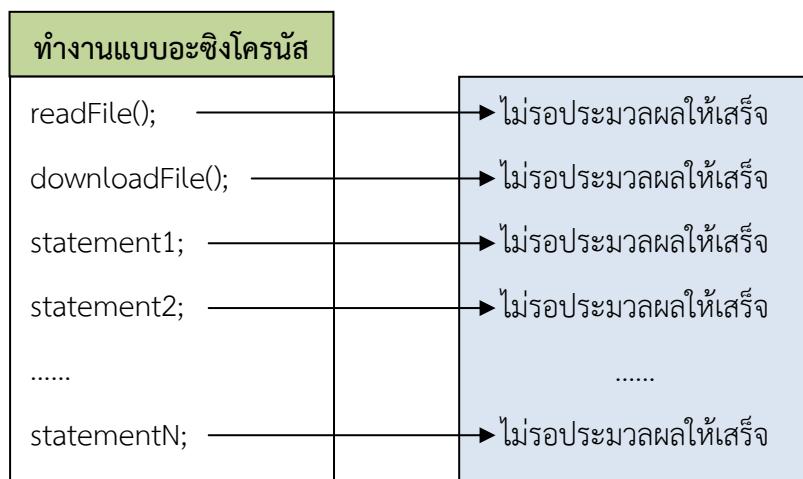
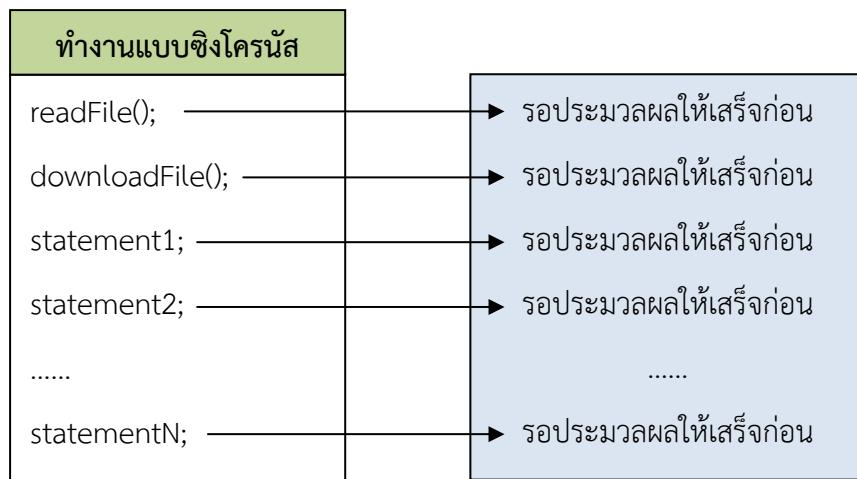


จากภาพโค้ดข้างบน (เมื่อตอนนี้ในหัวข้อก่อนหน้านี้) ถ้าเป็น Node.js จะทำงานแบบอะซิงโครอนส์ดังต่อไปนี้

1. คุณไม่ต้องรอให้ readFile(); ทำงานเสร็จก่อน หรือกล่าวอีกนัยหนึ่งไม่ต้องรอให้ฟังก์ชันติดต่อ I/O จนสำเร็จ แต่สามารถข้ามไปทำข้อ 2 ได้เลย (อีก 10 นาที จะทำงานเสร็จ)
2. ตรงประโยค downloadFile(); ก็ไม่ต้องรอให้ติดต่อ I/O เสร็จก่อนเช่นกัน จึงสามารถข้ามไปทำข้อ 3 ก่อนได้เลย (อีก 10 นาที จะทำงานเสร็จ)
3. ด้วยเหตุนี้ประโยค statement1, statement2 จนถึง statementN จึงสามารถทำงานเสร็จก่อนได้ภายใน 1 นาที (เป็นกรณีสมมติ)
4. พอยหลังจาก 10 นาทีผ่านพ้นไป ก็เพิ่งจะมาทำประโยคในข้อ 1 กับ 2 เสร็จทีหลัง

ห่วงว่าเมื่อถึงตอนนี้ คุณคงเห็นภาพการทำงานแบบอะซิงโครอนส์ของ Node.js มันจะไม่บล็อกการทำงานของ I/O นครับ ...ด้วยเหตุนี้คำว่า “non-blocking I/O” อาจเรียกใหม่เป็น “asynchronous I/O” ก็ได้เช่นกัน

เพื่อให้เห็นภาพความแตกต่างของโปรแกรม เวลา�ันทำงานแบบอะซิงโครอนส์ กับแบบอะซิงโครอนส์ ให้มันจะจ้างแจ้งแดงแจ้ง ผมจึงขอภาพตัวอย่างง่าย ๆ ในหน้าตัดไปมาให้ดูเล่น ๆ



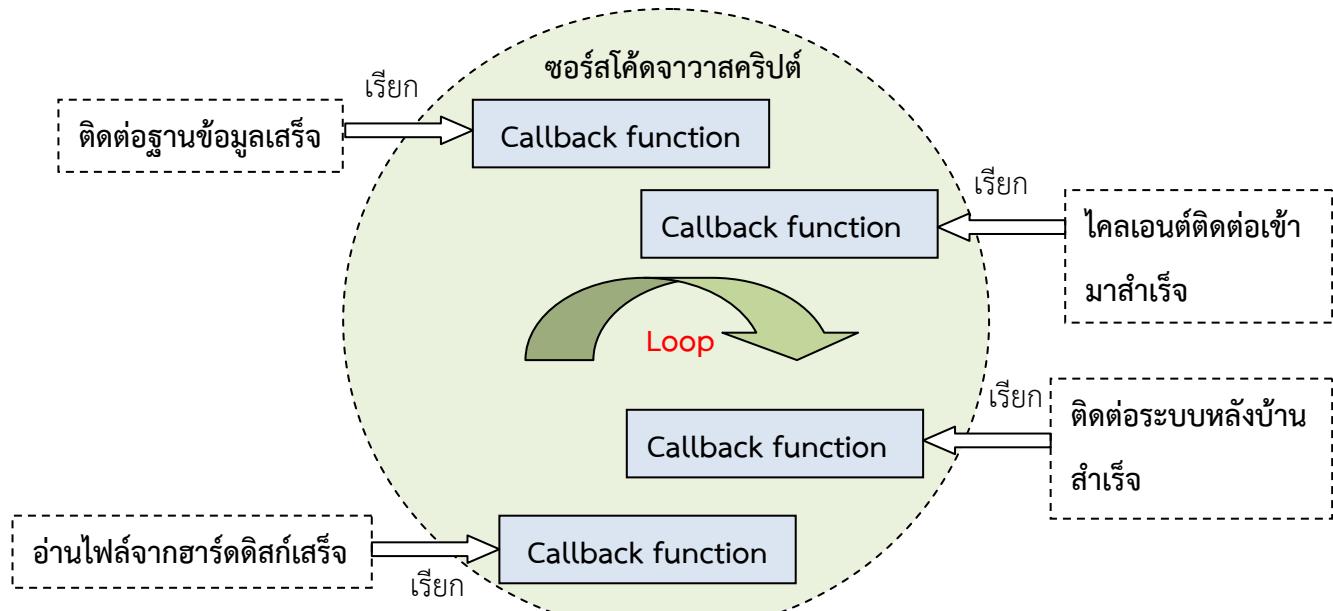
ถ้าจะเปรียบเทียบการทำงานแบบซิงโครนัส ก็เหมือนโทรศัพท์คุยกับเพื่อนที่คบกันใหม่ ๆ ห้ามวางสายทันที เดียวมันงอน ต้องคุยกับเพื่อน แล้วลึงจะไปเข้าห้องน้ำ กินข้าว ...บลา ๆ

ส่วนการทำงานแบบอะซิงโครนัส ก็เหมือนคุณส่งไลน์หาเพื่อน/หล่อน แล้วเราเก็บหน้าไปทำธุระส่วนตัวส่วนตัวก่อนได้ หลังจากนั้นค่อยมาเปิดอ่านไลน์ แล้วทักมันกลับไปอีกครั้ง

แนวคิดเรื่อง Event-driven ☀️

เพื่อให้ Node.js ทำงานแบบเชิงโครงสร้างได้ เขาจะใช้แนวคิด **Event-driven** ซึ่งได้รับอิทธิพล และคล้ายกับ Event Machine ของ Ruby หรือ Twisted ของ Python (ไม่ต้องซีเรียสกับคำศัพท์ที่พยายามมาก็ได้นะครับ)

Event-driven ถ้าแปลตรงตัวก็คือ “ถูกขับเคลื่อนด้วยเหตุการณ์” ซึ่งเหตุการณ์ที่ว่า ก็คือเหตุการณ์ที่ติดต่อกับ I/O ต่าง ๆ นั้นเองและครับ เพื่อให้เข้าใจมากขึ้น ผู้จะขอเปลี่ยนจากคำว่า “Event-driven” เป็น “Event loop” หรือ “วนลูป รับเหตุการณ์” ซึ่งไม่รู้จะงกันมากไปกว่านี้หรือไม่ แต่ยังไงก็ให้ดูภาพประกอบแล้วกันนะ



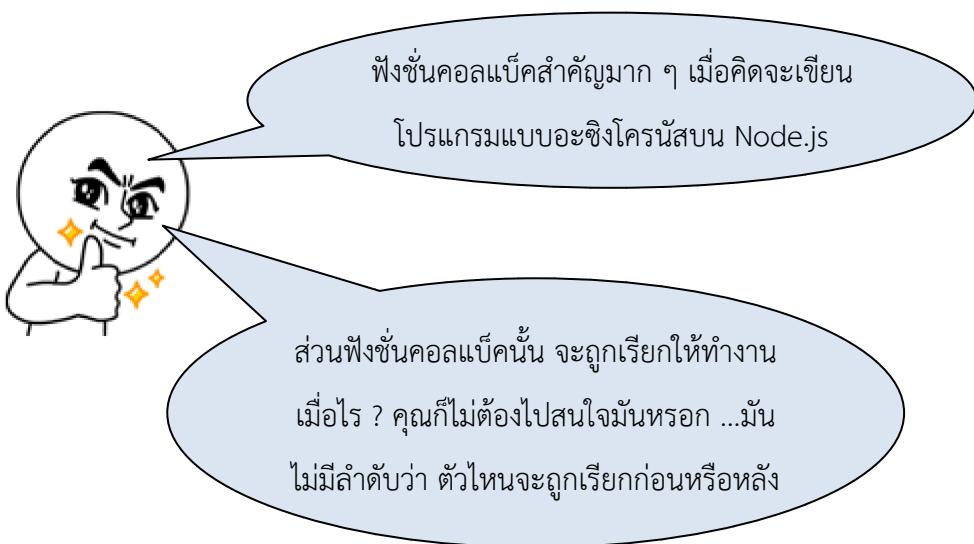
จากรูปภาพนี้ ให้คุณลองจินตนาการว่าเมื่อ Node.js มันประมวลผลไฟล์จาวาสคริปต์ ซึ่งเบื้องหลังการทำงาน มันจะมีเพียงเทรดเดียวเท่านั้น ที่มาอ่านและทำงานตามโค้ดที่เขียนไว้ ซึ่งเทรดดังกล่าวจะวนลูป (ตั้งแต่เริ่มแรก เลยครับ) เพื่อรับเหตุการณ์ที่ Node.js ต้องติดต่อกับ I/O ต่าง ๆ และเมื่อมันติดต่อเสร็จแล้ว ก็จะมาเรียกฟังชั่น คอลแบ็คที่อยู่ในโค้ดภายหลัง

คำถาม แล้วมันจะหยุดวนลูป เพื่อรับเหตุการณ์ต่าง ๆ เมื่อไร ?

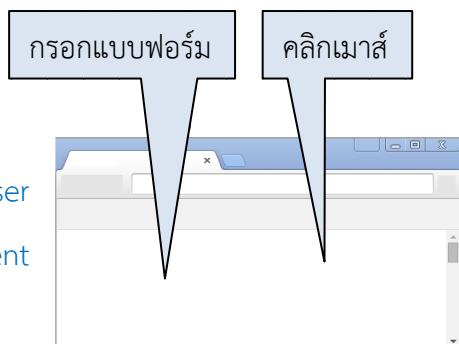
คำตอบ จะหยุดวนลูป เมื่อฟังก์ชันคอลแบ็คทุกตัวทำงานเสร็จเรียบร้อยแล้ว ไม่มีคอลแบ็คตัวใดหลงเหลือให้เรียกใช้

...และสิ่งหนึ่งที่คุณต้องรู้ก็คือ พังก์ชันคอลแบ็คใน Node.js คุณต้องเขียนเองน้า...

ผมอยากรู้คุณจำง่าย ๆ อย่างนี้แล้วกัน



จริง ๆ แล้ว ถ้าใครเคยเขียนโปรแกรมประเภท GUI (Graphic user interface) หรือใช้ภาษาสคริปต์ยุ่งเกี่ยวกับ DOM (Document Object Model) ... น่าจะเข้าใจตอนเชปท์แบบนี้ไม่ยาก



```
<!-- รองรับเหตุการณ์
เมื่อมีการคลิกมาส์บนปุ่มกดหน้าเว็บ-->
<button onclick="myFunction()">
  Click me
</button>
```

ผมจะให้คุณนึกถึงเวลาเขียนภาษาสคริปต์บนหน้าเว็บ (HTML) ตอนที่ผู้ใช้คลิกมาส์ กรอกแบบฟอร์ม และการกระทำต่าง ๆ บนหน้าเว็บ ก็จะเกิดเหตุการณ์ขึ้นมา (Event) ซึ่งจะส่งผลทำให้ฟังก์ชั่นคอลเบ็ค ถูกเรียกให้ทำงานภายหลัง

ซึ่งมันเป็นแนวคิด Event-driven เมื่อนั้นกับ Node.js ไม่ผิดเดียวเลยครับ ... แต่ว่ามันจะทำงานแบบอะซิงโครอนส์ในด้าน UI เป็นหลัก ส่วน Node.js จะทำงานอะซิงโครอนส์ในด้าน I/O เป็นหลัก

***หมายเหตุ ยังมีการทำงานแบบอะซิงโครอนส์ที่เหมือน ๆ กัน ทั้งในฝั่งหน้าเว็บ และฝั่ง Node.js เช่น การหน่วงเวลาด้วยฟังก์ชัน setTimeout() และ setInterval() เป็นต้น

บททวนฟังก์ชันคอลแบ็ค

ฟังก์ชันใน Java Script จะถือว่าเป็น **First-class functions** (แปลว่า “ฟังก์ชันเต็มขั้น” ล้อมาจากคำว่า “First-class citizen” หรือประชาชนเต็มขั้น) หมายความว่า ฟังก์ชันจะเป็นข้อมูลตัวหนึ่ง ที่สามารถกำหนดค่าให้กับตัวแปรได้

ฉันคือ function

ใช้เป็นข้อมูลได้นะ

*** หมายเหตุ ฟังก์ชันใน Java Script มักยังเป็นอีกตัวหนึ่งที่สามารถรับและส่งฟังก์ชันให้กับตัวอื่นได้ หรือจะเรียกว่า “Higher Order Functions” ดังต่อไปนี้

ด้วยเหตุนี้เราจึงสามารถส่งฟังก์ชัน ให้เป็นค่าอาภิเษนต์แก่ฟังก์ชันตัวอื่นได้ หรือจะเรียกตัวฟังก์ชันออกมาก็ทำได้ด้วย ...ซึ่งจะเรียกคุณสมบัติแบบนี้ว่า “Higher Order Functions” ดังต่อไปนี้

```
function readFile( callback ) {           // ประกาศฟังก์ชัน ให้มีพารามิเตอร์ชื่อ callback
    callback();                          // เรียกฟังก์ชันให้ทำงาน
}
// ประกาศตัวแปรแบบ let ใน ES6
let myFunction = function() {           // นำฟังก์ชันมากำหนดค่าให้กับตัวแปร myFunction
    console.log('Reading a file from json.txt');
};
readFile(myFunction);                  // แสดงผลลัพธ์เป็น "Reading a file from json.txt"
```

ในตัวอย่างนี้จะเห็นว่าฟังก์ชัน readFile(callback){...} จะมีตัวแปรพารามิเตอร์ callback ที่รับค่าอาภิเษนต์ เป็นฟังก์ชัน ซึ่งในกรณีคือ myFunction จึงทำให้มันถูกเรียกให้ทำงานภายใน readFile() ด้วยประโยชน์ของ callback(); ในบรรทัดที่สองได้

อย่างที่ทุกท่านทราบ ฟังก์ชันที่ถูกใช้เป็นค่าอาภิเษนต์ เขาจะเรียกว่า **ฟังก์ชันคอลแบ็ค** (Callback functions) หรือเรียกสั้น ๆ ไปเลยว่าคอลแบ็คก์ได้ (ชี้เกี้ยงเจียนนะ)

โดยคอลแบ็คก์อาจแปลตรงตัวได้ว่า “ฟังก์ชันที่ถูกเรียกกลับ” และในตัวอย่างดังกล่าว ก็ชัดเจนดีนะครับว่า

...ฟังก์ชัน myFunction() จะถูกฟังก์ชัน readFile() โทรเรียกกลับอีกที

บททวน Closures

ตัวอย่างต่อไปนี้ ผู้จะขอทบทวนเรื่อง Closures ใน Java Script

```
function readFile() { // ประกาศฟังก์ชัน
    Closure
    let filename = 'json.txt';
    // ถูม่องเห็น
    return function() {
        // รีเทิร์นฟังก์ชันภายใน (Inner functions)
        console.log(`Reading a file from ${filename}`);
    }
}

let myFunction = readFile();

myFunction(); // เรียกฟังก์ชันให้ทำงาน และแสดงผลลัพธ์เป็น "Reading a file from json.txt"
```

เนื่องจากฟังก์ชันก็คือชนิดข้อมูลตัวหนึ่ง จากตัวอย่างเมื่อเรียกฟังก์ชัน readFile(); ก็จะรีเทิร์นฟังก์ชันภายในที่สามารถมองเห็นตัวแปร filename ซึ่งประกาศอยู่ที่ฟังก์ชันตัวนอกสุด

...ด้วยเหตุนี้เมื่อเรียก myFunction(); ก็ยังคงเข้าถึงตัวแปร filename อย่างไม่มีปัญหาอะไรเลย

ต้องบอกอย่างนี้นะครับว่า ...ฟังก์ชันใน Java Script นอกจากประกาศอยู่ในฟังก์ชันตัวอื่นได้แล้ว มันยังจำเพื่ินที่ซึ่งมันเคยอาศัยอยู่ได้ (Context) และมองเห็นตัวแปรที่ประกาศอยู่ในฟังก์ชันข้างนอกได้ด้วย
สำหรับฟังก์ชันที่ซ้อนอยู่ในฟังก์ชันหลัก และจำเพื่ินที่ซึ่งมันอาศัยอยู่ เขาจะเรียกฟังก์ชันแบบนี้ว่า **Closures** ที่แปลว่า “การปิด” ...หมายความว่า ฟังก์ชันที่ซ้อนอยู่ภายใน แต่สามารถปิดล้อมตัวแปรที่อยู่นอกขอบเขตได้

สาเหตุที่ผมทบทวนคร่าว ๆ เกี่ยวกับฟังก์ชันคอลแบ็ค กับ closure ก็เพราะมันเป็น หัวใจหลักในการเขียนโปรแกรมด้วย Java Script บน Node.js นะซิ

... แต่ถึง closures จะแปลว่าปิด แต่ก็ห้ามปิดใจไม่ให้เรียนรู้ Java Script นะครับ อิ ๆ



เกร็ดความรู้ ★

Ryan Dahl เริ่มสร้าง Node.js ด้วยการใช้ภาษา C แต่เมื่อทำไปพบร่วมกันที่จะมาตกล่องปล่องชื่นกับภาษาสคริปต์ เนื่องจากมันมี closures กับ first-class functions ซึ่งมันพอเหมาะสมกับการเขียนโปรแกรมแบบซิงโครนัส ด้วยการใช้ Event-driven นั่นเอง

เกร็นนำมอดูล

“ศัพท์บัญญัติราชบัณฑิตยสถาน”
เข้าเขียนมอดูลแบบนี้จริง ๆ

มันเป็นมาตรฐานใน
การโหลดมอดูลต่าง ๆ

มอดูล (Modules) ใน Node.js จะใช้มาตรฐาน CommonJS ... เอาเป็นว่าให้คุณลองนึกถึงไลบรารีที่ใช้เขียน โปรแกรม (Library) โดยปกติเราจะใช้มันเก็บโค้ดอะไรก็ ตามที่ต้องเรียกใช้งานบ่อย ๆ ... และมอดูลก็มีแนวคิดเดียวกัน



สำหรับมอดูลใน Node.js ส่วนใหญ่จะเก็บ API ซึ่งเป็นฟังก์ชันที่ต้องติดต่อกับ I/O แบบซิงโครนัส ... ส่วน การโหลดสิ่งที่บรรจุอยู่ในมอดูลมาใช้งาน ก็จะให้ใช้ประโยชน์คำสั่งดังตัวอย่างต่อไปนี้

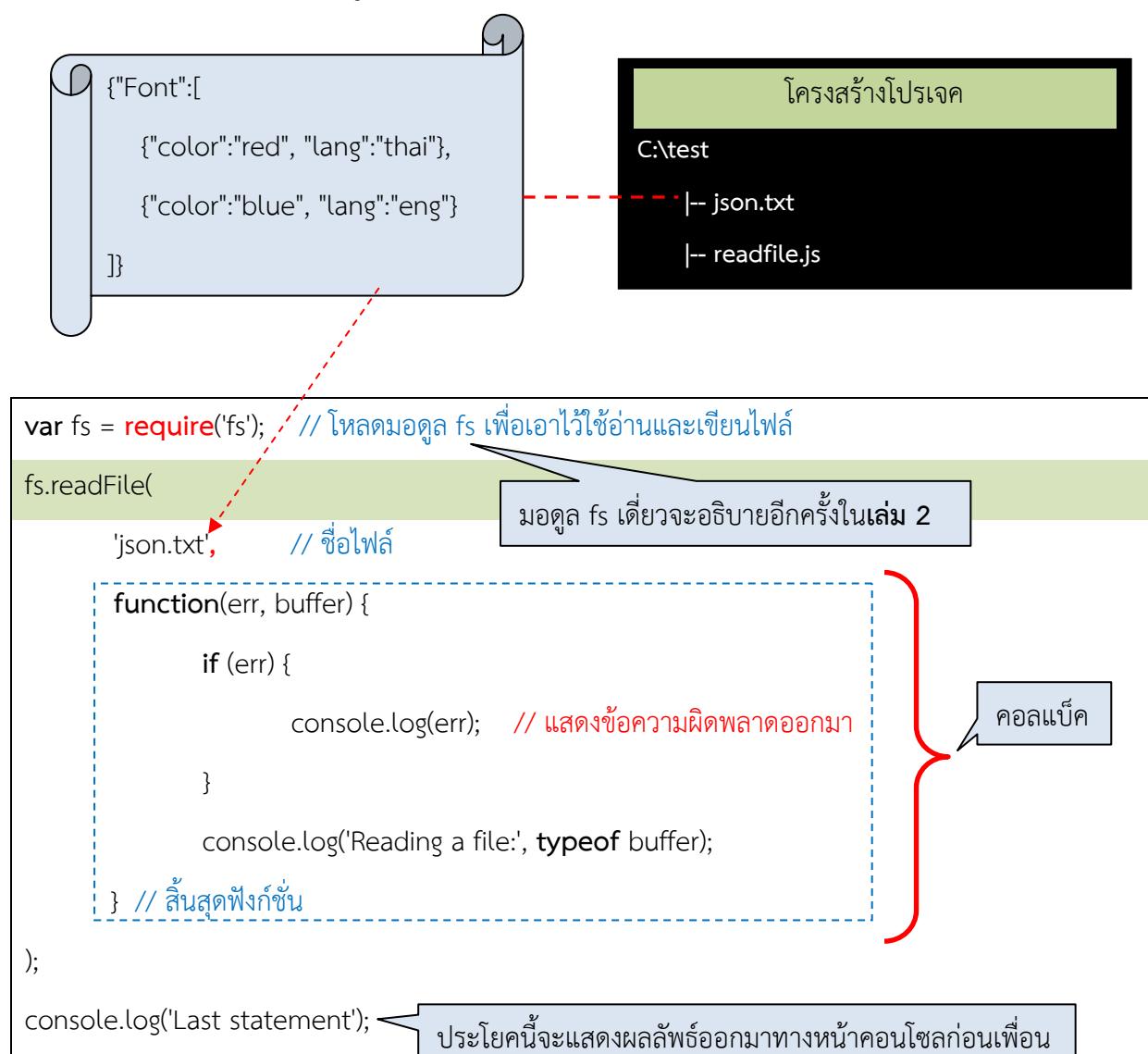
```
require('module_name');
// module_name คือชื่อมอดูล
```

require() ก็คือฟังก์ชัน โดยเราต้องระบุชื่อมอดูลเป็นค่าอักษรในตัวแปร แล้วเมื่อนั้น require() จะรีเทิร์นอ้อบเจกต์ ออกมา โดยค่าพร็อพเพอร์ตี้ (Property) ของมัน ก็คือสิ่งที่เราได้โหลดจากมอดูลเข้ามานั้นเอง

มอดูล ถ้าแปลตรงตัวก็จะได้ว่า

“หน่วยที่นำมาประกอบ เป็นสิ่งหนึ่งสิ่งเดือนมา”

จะขอยกตัวอย่างการใช้งานมอดูลดังนี้



จากโค้ดตัวอย่างดังกล่าว จะสมมติว่าบันทึกไว้เป็นไฟล์ชื่อ “readfile.js” เมื่อนำไปรันบน Node.js ด้วยการพิมพ์คำสั่งบนคอมมานไลน์เป็น “node readfile.js” ก็จะได้ผลลัพธ์ดังนี้

```
c:\test>node readfile.js
```

Last statement

Reading a file: object

สำหรับตัวอย่างดังกล่าว จะเป็นการใช้งานมอดูล ซึ่งภายในโค้ดจะมีรายละเอียดที่น่าสนใจดังนี้

- ประโยค `var fs = require('fs');` จะทำการโหลดมอดูลที่ชื่อ “fs” ด้วยการใช้ฟังก์ชัน `require` ซึ่งจะรีเทิร์นออบเจกต์ออกมาย โดยมีตัวแปร `fs` มาอ้างอิงออบเจกต์ดังกล่าวอีกด้วย

- ประযุค `fs.readFile('json.txt', ...)` จะเป็นการเรียกเมธอด `readFile` (พรือพเพอร์ตของ `fs`) ให้อ่านไฟล์ที่ชื่อ "json.txt" (ติดต่อกับ I/O) ซึ่งการทำงานจะเป็นแบบอะซิงโครอนส์ จึงไม่ต้องรอให้ประยุคนี้ทำงานเสร็จก่อน แต่สามารถข้ามไปทำประยุค `console.log('Last statement');` ที่อยู่บรรทัดสุดท้ายก่อนได้เลย
- เมื่ออ่านไฟล์ `json.txt` เสร็จเรียบร้อยแล้ว (เกิดเหตุการณ์อ่านไฟล์เสร็จ) ก็จะมาเรียกคอลแบ็ค ซึ่งเป็นค่าอาภิเษนต์ตัวที่สองของ `fs.readFile()` ให้ทำงานภายหลังนั้นเอง

***หมายเหตุ ปกติแล้วฟังก์ชันของอืบเจ็กต์ จะเรียกว่า “เมธอด (method)”

ถ้าคุณเห็นโค้ดของ Node.js ไปเรื่อย ๆ เวลาโหลดมودูลมาใช้งาน จะเห็นว่าอีพากเมธอดที่ต้องติดต่อกับ I/O แบบอะซิงโครอนส์ เมธอดพวกลักษณะนี้จะประกาศพารามิเตอร์ที่มีหน้าตาสวยงามเหมือน ๆ กัน ประมาณนี้

```
apiObject.method(param1, param2, ...., [callback_function]);
```

สำหรับเมธอดที่เราโหลดมาใช้งาน มันจะรับค่าอาภิเษนต์ได้กี่ตัว ก็ขึ้นอยู่กับว่าประกาศไว้จำนวนเท่าไร แต่ค่าอาภิเษนต์ตัวสุดท้าย ส่วนใหญ่แล้วจะเป็นคอลแบ็ค ...จริง ๆ นะ ไม่ได้มีด้วย และตัวคอลแบ็คเราจะเขียนหรือไม่เขียนก็ได้เช่นกัน (เป็นออบชั่น)

เมื่อถึงตอนนี้คุณน่าจะมองภาพออกนั่นรับว่า ...ถ้าจะเปรียบเทียบการใช้ฟังก์ชัน `require()` กับภาษาอื่น ก็อาจเทียบได้ดังตัวอย่างต่อไปนี้

- ใน Java คือประยุค `import`
- หรือใน C# คือประยุค `using`
- หรือใน C และ C++ คือประยุค `#include`

สำหรับรายละเอียดการใช้งานมอดูล เดียวจะมาอธิบายต่ออีก แต่ตอนนี้ขอตัดจบก่อนครับ (จะงี้น)

คำแนะนำเรื่องโค้ด

โอม นะจีะ นะจัง
...จะเข้าใจหลักการเพี้ยง

เวลาคุณจะอ่านโค้ดใน Node.js ก็ขอให้เข้าใจหลักการ 3 ข้อที่ผมสรุปมาให้ได้แก่

- ★ 1. โค้ดมักจะเริ่มต้นด้วยการใช้ `require()` เพื่อโหลดมอดูลที่เหมาะสมกับงาน แล้วเรียกใช้งานมันให้เป็น
- ★ 2. เข้าใจหลักการทำงานแบบซิงโครนัส โดยใช้กลไก Event-driven (Event-loop)
- ★ 3. เข้าใจคอนเซปท์ฟังก์ชั่นคอลแบ็ค (รวมทั้งความเป็น first-class functions) กับคอนเซปท์ closure

การอ่านโค้ดบน Node.js จริง ๆ นะไม่ยากเลย ขอแค่รู้จักเปิดคู่มือศึกษา API แต่ละมอดูลที่เราโหลดเข้ามา (หลักการข้อ 1) พร้อมทั้งเข้าใจหลักการข้อ 2 กับ 3 ให้ดี ๆ ก็สามารถประยุกต์ใช้งาน Node.js แบบพลิกแพลงหลายร้อยกระบวนการท่าได้แล้ว ...แต่ต้องรู้จัavariable ตัวยันะ มันจะดีมาก ๆ

// โค้ดส่วนใหญ่ใน Node.js ก็จะประมาณนี้

```
var module = require('module_name'); // มักจะเริ่มต้นด้วยการโหลดมอดูลให้เหมาะสมกับงาน
```

/* สำหรับโค้ดที่ตามมา

...ขอให้เราเรียกใช้งานมอดูล ตามคู่มือให้เป็น

...ต้องเข้าใจหลักการทำงานของโปรแกรม ...แบบซิงโครนัส โดยใช้กลไก Event-driven

...ที่สำคัญคุณต้องเขียนจาวาสคริปต์แบบ Function Programming ไม่ใช่แบบ OOP (ต้องเข้าใจหลักการพอก first-class functions, ฟังก์ชั่นคอลแบ็ค และ closure)

*/

ยิ่งใครมาจากภาษาอื่น และเป็นแฟนพันธุ์แท้เรื่องการเขียนโปรแกรมเชิงวัตถุ หรือเรียกว่า OOP (Object oriented Programming) อาจไม่ค่อยชอบโค้ดบน Node.js เลย เพราะจะเหมือนอยู่คนละกาแล็กซี่โลก

Object oriented Programming

VS

Functional Programming

เพราะทุกอย่างโค้ดใน Node.js จะเต็มไปด้วยฟังก์ชัน ที่เห็นมันอยู่ได้หมดทุกแห่งหน ทั้งเป็นค่าอาเกิร์เมนต์ หรือช้อนอยู่ในฟังก์ชันตัวอื่น หรือถูกเรียกจากฟังก์ชันตัวอื่น จะมีเรื่อง closure อีก แฉมแต่ละฟังก์ชัน คอลแบ็คจะถูกเรียกให้ทำงานตอนไหนก็ยังไม่รู้ เพราะมันทำงานแบบอะซิงโครอนส์ ตามเหตุการณ์ที่เกิดขึ้น

....แน่นอนโค้ดใน Node.js อาจดูแล้วตาลาย สำหรับผู้ไม่คุ้นชินครั้งแรก ห้า ห้า ๆ

ด้วยเหตุนี้ ผู้จึงแนะนำว่าให้คุณเลิกมองโค้ดแบบ OOP แต่ให้มองเป็นแบบ Functional Programming หรือ การเขียนโปรแกรมเชิงฟังก์ชันแท้ ๆ จะดีกว่า

สำหรับแนวคิด Functional Programming เรียกสั้น ๆ ว่า “FP” ใน Java Script เรื่องนั้นๆ เลยขอไม่อธิบายนะครับ ... เพราะมันเกินขอบเขตของหนังสือเล่มนี้พอควร

หมายเหตุ Java Script ยังไม่สนับสนุนแนวคิด FP แบบแท้ ๆ 100 % นะครับ

สไตล์การเขียนโค้ดใน Node.js

การเขียนโค้ดใน Node.js จะมีสไตล์ที่คุณควรรู้ไว้ ดังต่อไปนี้

```
// function callback (param) {           // ประกาศฟังก์ชันแบบนี้ได้เหมือนกัน
    var callback = function(param) {
        console.log('to do something:', param);
    }

    function myFunction(param, func) { // ค่าอาเกิร์เมนต์ตัวที่สอง จะรับค่าเป็นฟังก์ชันคอลแบ็ค
        func(param);                // เรียกคอลแบ็คให้ทำงาน
    }
}

myFunction('say', callback);           // แสดงผลลัพธ์เป็น "to do something: say"
```

myFunction() ในตัวอย่าง เวลาเรียกใช้งานจะรับค่าอาเกิร์เมนต์ตัวที่สองเป็นตัวแปร callback ซึ่งมีค่าเป็นฟังก์ชัน ...ซึ่งโค้ดดูปกติธรรมดามากมีอะไร

แต่โค้ดส่วนใหญ่ใน Node.js นักจะนิยมเรียก myFunction() พร้อมทั้งประกาศฟังก์ชันไร้ชื่อ (anonymous functions) ในตำแหน่งที่เป็นค่าอา กิว เมนต์ที่เดียวจบเลย (inline) ...ถ้าไม่ก้าวไปอีก ก็ต้องย่อไปเป็น

```
function myFunction(param, func) {
    func(param);
}

myFunction('say', function(param){
    console.log('to do something:', param);
});

// แสดงผลลัพธ์เป็น "to do something: say"
```

ประกาศฟังก์ชันไร้ชื่อ
ในตำแหน่งที่เป็นค่าอา กิว เมนต์

ในตัวอย่างจะเรียกฟังก์ชัน myFunction() ให้ทำงาน พร้อมทั้งประกาศฟังก์ชันไร้ชื่อ ในตำแหน่งที่เป็นค่าอา กิว เมนต์ตัวที่สอง ...ซึ่งการเขียนแบบนี้คุณจะเห็นยอดมาก จนแทบจะตาแฉะ

อีกอย่างหนึ่งที่คุณควรรู้ไว้ ...Node.js จะเขียนในสไตล์ที่เรียกว่า **Continuation-passing style (CPS)** มันคือวิธีเขียนให้คอลแบ็คทำงานอย่างต่อเนื่อง และผูกจักรภัยด้วยการยกตัวอย่างได้ดี (ทำงานจริงไม่ได้นะ)

```
http.request(options, function(response) {
    response("data", function(data) {
        console.log("display", options); // บรรทัด 3
        console.log("some data from the response", data);
    });
});
```

ปกติแล้วการเรียกฟังก์ชันปกติธรรมดា เมื่อทำงานเสร็จจะรีเทิร์นค่าอกมาด้วยประโยชน์ return แต่วิธี CPS เมื่อฟังก์ชันทำงานเสร็จ มันจะเรียกคอลแบ็คให้ทำงานเป็นลำดับสุดท้ายแทน

ในตัวอย่าง เมื่อฟังก์ชัน http.request(options,...) ทำงานเสร็จ คอลแบ็คหมายเลข 1 จะถูกเรียกให้ทำงาน แล้วไปเรียกฟังก์ชัน response("data",...) เมื่อมันทำงานเสร็จ ก็จะเรียกคอลแบ็คหมายเลข 2 ให้ทำงาน

คงพอนึกภาพอุก奴รักว่า คอลแบ็คใน Node.js มันจะเขียนช้อน ๆ กันแบบนี้ และเวลาทำงานก็จะถูกเรียกต่อเนื่องไปเรื่อย ๆ ...เท่านั้นยังไม่พอ ในบรรทัดที่ 3 ในตัวอย่างก่อน ยังเห็นตัวแปร options ด้านนอกอีกด้วย

เมื่อถึงตรงนี้คิดว่าคนที่ชอบ OOP บางคน อาจไม่ชอบวิธีเขียนแบบนี้ เพราะดูซุ่มๆ ว่ามันพิลึกดี จัดไม่เป็นระเบียบเหมือน OOP ...แต่ก็ทำได้เนอะ คุณต้องทำใจ เมื่อคิดนอกใจนอกกาวย์ เปลี่ยนจากภาษาอื่นมาใช้ Node.js

อีกทั้งถ้าเขียนโค้ดใน Node.js ไม่ดี ก็อาจเกิดคอลแบ็คแครก (Callback Hell) เนื่องจากมีการเรียกใช้คอลแบ็คต่อเนื่องกันเป็นลูกโซ่หลายชั้นกัน จนโค้ดอ่านยากมาก ดังตัวอย่าง

```
var express = require('express');           // มอดูล express จะอธิบายอย่างละเอียดอีกทีในเล่ม 2
var app = express();
var fs = require('fs');

app.get('/', function(req, res) {          1
    res.send('<h1>Hello world </h1>');
}

fs.writeFile('msg.txt', 'callback hell', function (err) { 2
    if (err) { console.log(err); }

    fs.readFile('msg.txt', function(err, buffer) {          3
        if (err) { console.log(err); }

        console.log(buffer.toString()); // เข้าถึงข้อมูลในไฟล์ json.txt
    }); // สิ้นสุด fs.readFile(...)

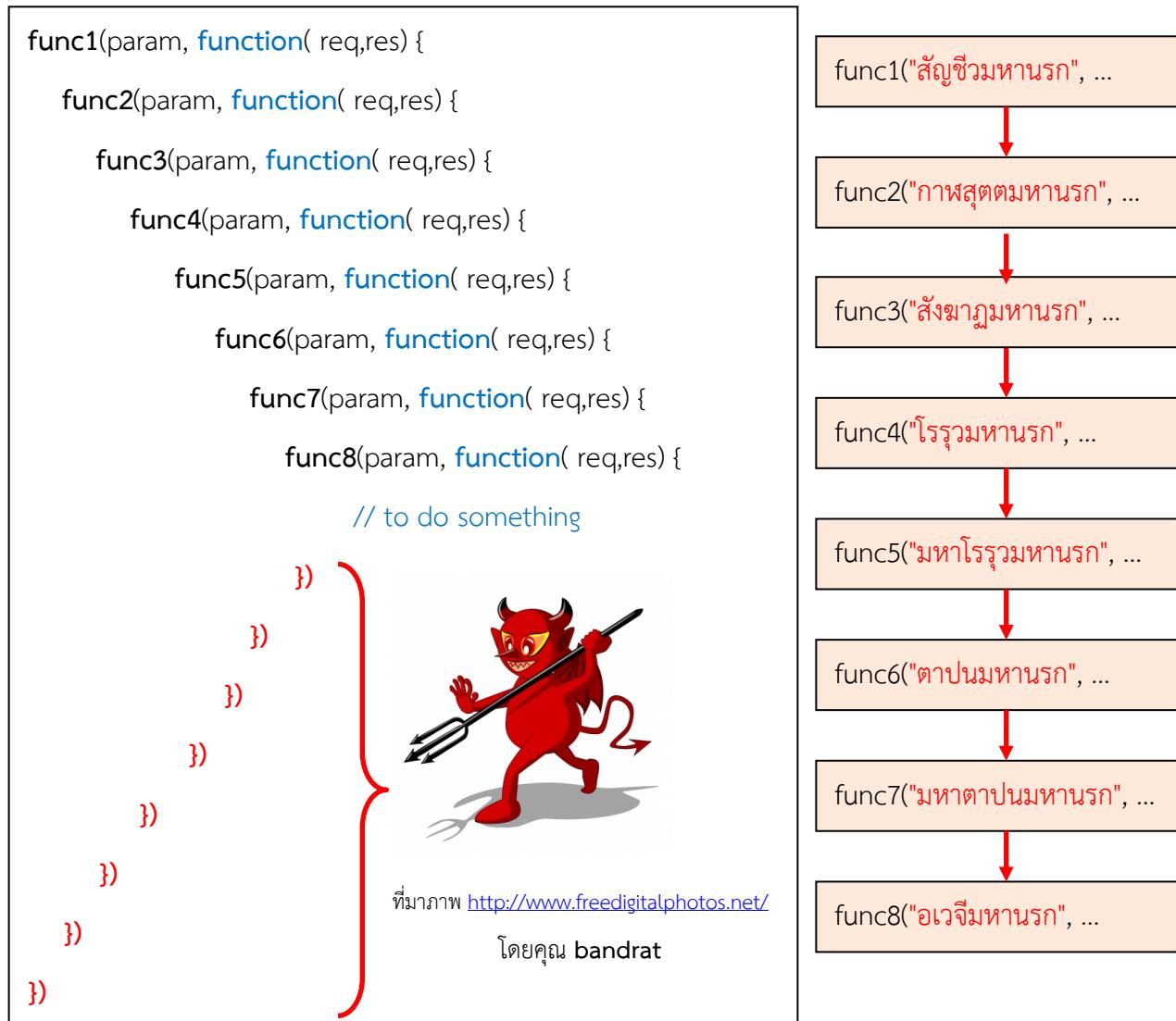
}); // สิ้นสุด fs.writeFile(...)

}); // สิ้นสุด app.get(...)

app.listen(8080, function() {
    console.log('Server running at http://localhost:8080/');
});

console.log("Start server");
```

ตัวอย่างในหน้าก่อน จะมีเพียงคอลแบ็ค 1, 2, 3 ซ้อนกันเท่านั้น ซึ่งอาจดูไม่น่ากลัวอะไร ยังพอໄล่อ่านได้แต่คุณลองคิดดูซิว่าถ้าเกิดมีคอลแบ็คตัวที่ 4, 5, 6 ...N ซ้อนกันไปเรื่อย ๆ ก็จะบากบานมาก จนกลายเป็นคอลแบ็คหนราก แบบตกชุม茄เวจี เพราะโค้ดจะอ่านยากมาก จนตาลาย ...ซึ่งเราต้องระมัดระวังให้ดี ดังตัวอย่าง



จากตัวอย่างที่เห็น func1, func2, ... func8 จะเรียกใช้งานกันต่อเนื่องเป็นลำดับ ซึ่งอาจทำให้คุณกุ่มมับปวดหัวจนต้องกินยาพารา เวลาอ่านโค้ด ซึ่งเราควรหลีกเลี่ยงวิธีเขียนแบบนี้ใน Node.js

สำหรับการปรับโค้ดจากคอลแบ็คหนราก ให้กลายมาเป็นคอลแบ็คสوارร์ค มันเป็นเรื่องของการบำรุงรักษาโค้ด เลยนอกประเด็นของหนังสือเล่มนี้พอกครว เลยจะไม่กล่าวถึง (ถ้ามีเวลาและโอกาส ก็จะจะเขียนอธิบายอีกที)

หมายเหตุ ใน ES6 จะมีฟีเจอร์ **Promise** เพื่อช่วยเขียนโค้ดแบบซิงโครนัส แม้ยังจะหลีกเลี่ยงการเกิดคอลแบ็คหนรากได้อีกด้วย

โค้ดที่เป็นอะซิงโครนัส กับซิงโครนัส

ผมจะขอยกตัวอย่างโค้ด Node.js ต่อไปนี้ เพื่อให้คุณรับมือระหว่างตัว มีอะไรบ้าง

```
var fs = require('fs');

console.log('Before while loop'); // แสดงออกมา แค่ข้อความเดียว

while(true); // เส้นทางการทำงานของโปรแกรม
  fs.readFile( // จะติดอยู่ที่ลูปนี้ชั่วกับปั๊กกลับ
    'test.txt', // ชื่อไฟล์
    function(err, buffer) { // คอลแบ็คไม่เคยถูกเรียก (เส้นทางการทำงานของโปรแกรม มาไม่ถึง)
      console.log('After while loop');
    }
);

```

ในตัวอย่างนี้จะบันทึกไว้เป็นไฟล์ “readfile.js” เมื่อสั่งให้มันรัน ก็จะมีรายละเอียดที่น่าสนใจดังนี้

- จะมีแค่ข้อความเดียวถูกแสดงออกมา ได้แก่ 'Before while loop'

```
c:\test>node readfile.js
```

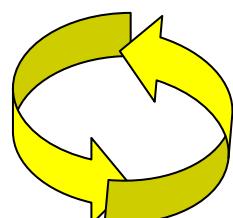
```
Before while loop
```

- ส่วนข้อความ 'After while loop' จะไม่ถูกแสดงออกมา

ซึ่งคุณต้องเข้าใจนะครับว่า Node.js มันมีแค่เทรดเดียว และตรงประโยชน์ค while(true); ก็คือการเขียนจาวา สคริปต์ธรรมดาก็ไม่ได้ทำงานแบบอะซิงโครนัสแต่อย่างใดเลย ...อ้าว (ร้อนหรือ)

...คือใน Node.js นะ โค้ดบางส่วนจะทำงานแบบซิงโครนัส จนเสร็จเรียบร้อยไปก่อน

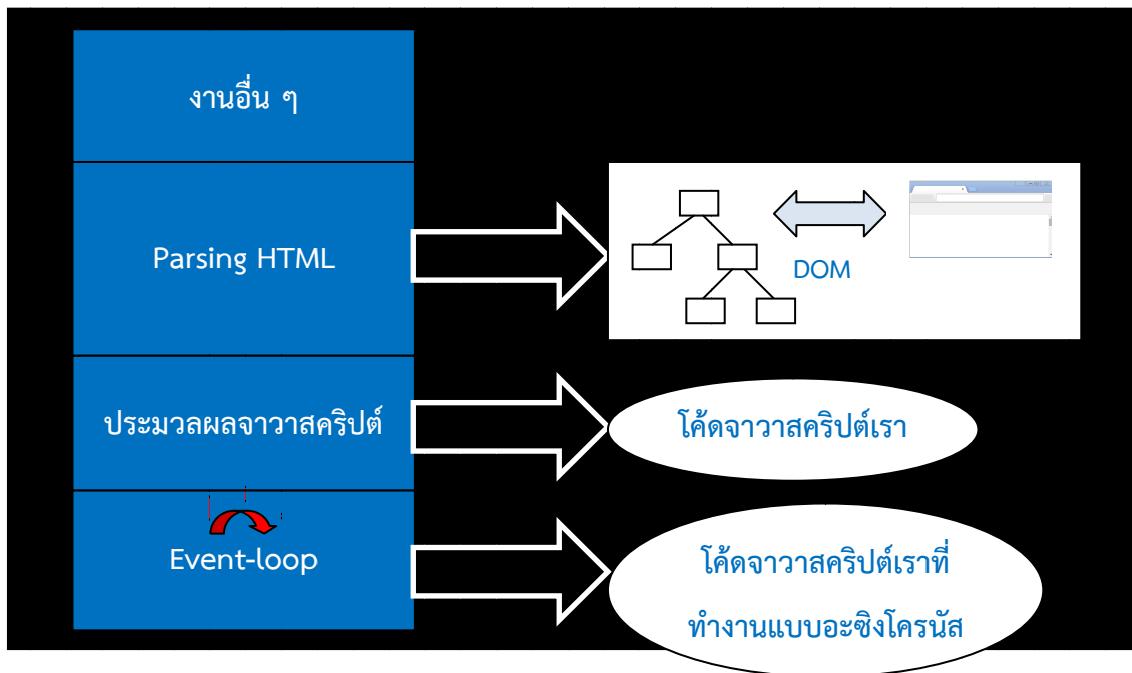
...ส่วนโค้ดที่ทำงานแบบอะซิงโครนัสของตัวอย่างนี้ ก็คือคอลแบ็คของ fs.readFile() เท่านั้นเองแหละ



ด้วยเหตุนี้ ประโยชน์คำสั่ง while(true); จึงวนติดลูปตลอดกาลชั่วฟ้าดินสลาย ทราบได้ที่เราไม่ยอมปิดคอมมานไลน์ หรือกด Ctrl +C เพื่อยกเลิกการรัน

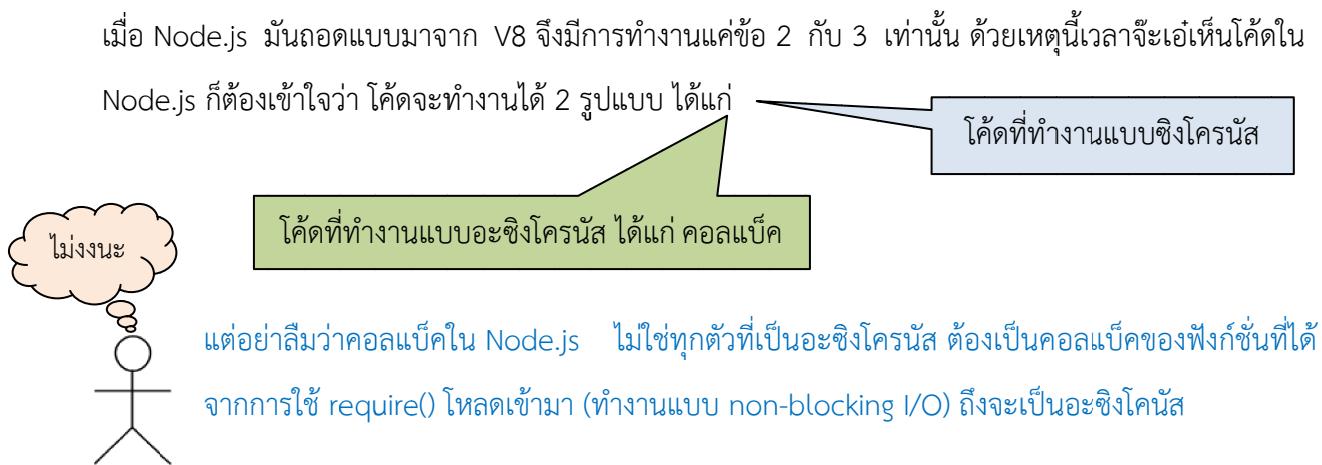
เบื้องหลังของ Node.js

ก่อนจะใช้งาน Node.js ควรเข้าใจเบื้องหลังการทำงานของเว็บเบราว์ส่วนใหญ่ ซึ่งจะมีแค่เทรดเดียว เมื่อ มันเริ่มทำงานด้วยการอ่านไฟล์ HTML ก็จะมีรายละเอียดตามภาพ



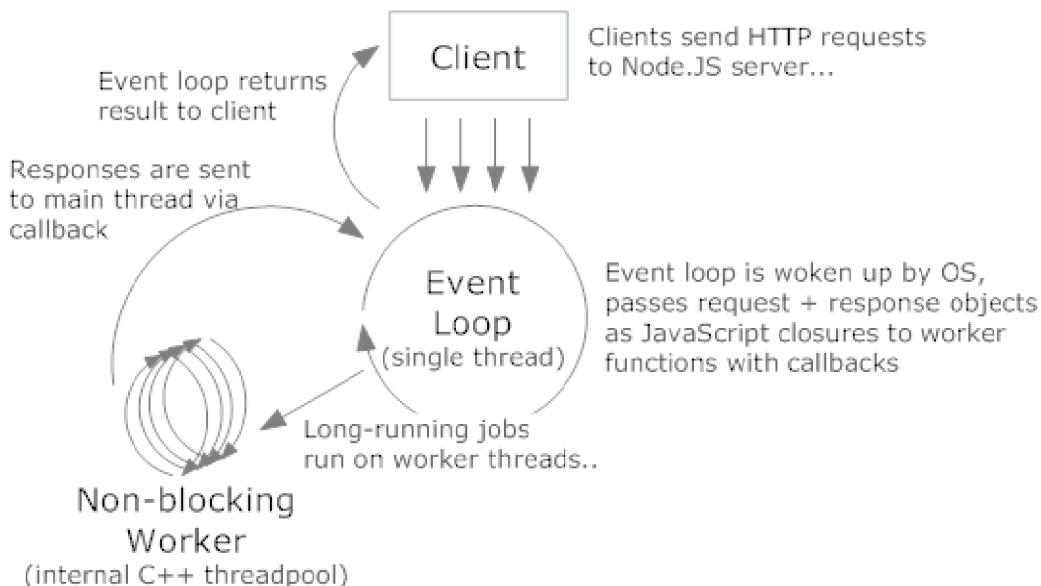
เว็บเบราว์ส่วนใหญ่ ก็จะทำงานตามภาพที่ผู้สอนมาให้ดูประมาณนี้แหละ โดยสามารถอธิบายการทำงานที่สำคัญดังนี้

- 1) **Parsing HTML** คือการเอาไฟล์ HTML มาแปลงให้กลายเป็นโครงสร้างเอกสารแบบลำดับชั้นที่เรียกว่า DOM (Document Object Model) พร้อมทั้งแสดงผลออกมายังหน้าจอเว็บเบราว์ส
- 2) หลังจากนั้น **จา瓦สคริปต์** บนจินในเว็บเบราว์ส จะมาประมวลผลโค้ดจา瓦สคริปต์เรา (แท็ก `<script>....</script>`) ให้เสร็จครั้งเดียวไปเลย ซึ่งการทำงานจะเป็นแบบอะซิงโครนัส
- 3) แต่ทว่า **จา瓦สคริปต์** บนจินยังทำงานไม่เสร็จดีนะครับ เพราะยังมี Event-loop ที่รอรับเหตุการณ์ต่างๆ ในหน้าเว็บ เช่น คลิกเม้าส์ พิมพ์ดีดบนหน้าเว็บ เป็นต้น ซึ่ง **จาวาสคริปต์** บนจินจะมาเรียกคอลแบ็ค ที่ตั้งกับเหตุการณ์นั้น ๆ ให้ทำงาน ...แน่นอนนะครับ คอลแบ็คคือคังกล่าวจะทำงานแบบอะซิงโครนัส



เพื่อทดสอบความเข้าใจ ให้หนักหัวเล่น ๆ ก็จะแสดงภาพดังต่อไปนี้

Node.JS Processing Model

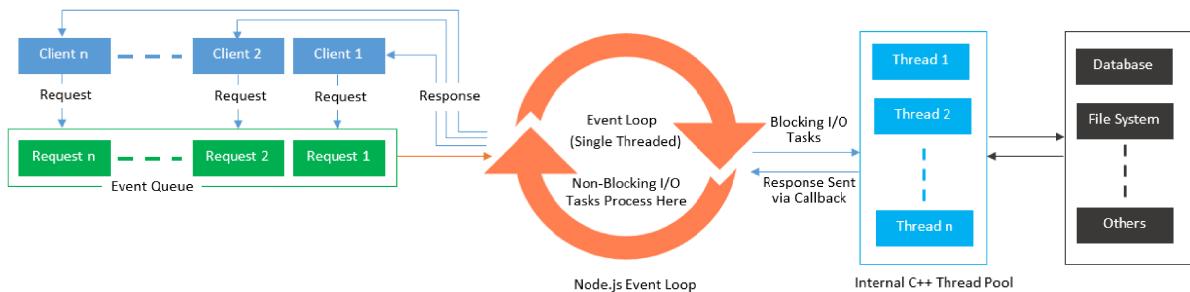


ภาพข้างบนที่เห็นนำมาจากเว็บ <http://kikobeats.com/synchronously-asynchronous/> ซึ่งผมจะขอ อธิบายสั้น ๆ เกี่ยวกับกลไก Event-loop ใน Node.js ดังต่อไปนี้แล้วกันนะ (แปลและเรียบเรียงมาอีกทีนะ)

เบื้องหลังการทำงานของ Node.js จริง ๆ แล้ว มันจะใช้ V8 เพื่อแปลงโค้ดจากวัสดุสคริปต์ ให้กลายมาเป็นโค้ดในภาษา C++ และจึงแทรก-threadอยู่ ๆ ออกมา (ในรูปคือ Non-blocking Worker) เพื่อจัดการเรื่อง non-blocking I/O โดยใช้ไลบรารี libuv (มีไว้เพื่อจัดการเรื่องอะซิงโครนัส I/O โดยเฉพาะ) ส่วนthreadที่กล่าวมา นี่คือทำงานเสร็จจะส่งレスポンส์ (Response) ไปเรียกคอลแบ็คในโค้ดเรา ที่ตรงกับเหตุการณ์นั้น ๆ ให้ทำงาน

***ถ้าท่านสนใจเกี่ยวกับ libuv ก็สามารถอ่านได้ที่ <https://github.com/libuv/libuv>

ลองดูวิธีการที่อธิบายการทำงานของ Node.js ดังนี้ครับ

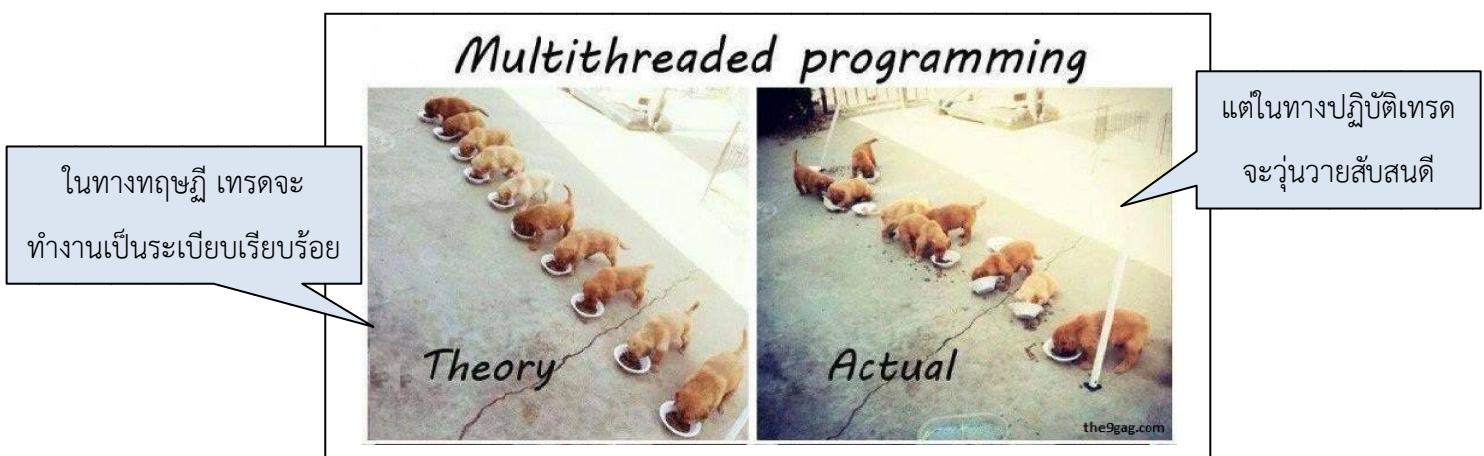


ถ้าสนใจรายละเอียดเรื่องรูปนี้ ก็ลองดูคำอธิบายต่อที่ลิงค์ด้านล่างแล้วกันนะครับ (ผมเอารูปมาจากลิงค์นี้แหละ) ...และขออนุญาตตัดจบแคนนี้แล้วกันนะ FFE ๆ

- <http://www.dotnet-tricks.com/Tutorial/nodejs/QF14301215-Exploring-Node.js-Code-Execution-Process.html>



ເອົາເປັນວ່າຄ້າຄຸນອຍກປວດຫວຸ່ງຢາກຍກກຳລັ້ງ² ດ້ວຍການແຕກທຽດເລີ່ມເອງ ເພື່ອຈັດການ
ດ້ານ I/O ມີຫັນສິ່ງເລີ່ມຕົ້ນໃຈໜີ່ໄມ່ຕ້ອບໂຈທີ່ຢູ່ຢູ່ ແຕ່ຄ້າໄມ່ອ່າຍຄິດເຮືອງທຽດໃຫ້ນັ້ນກະ
ສົມອງ ອຍກມີເວລາໄປໂຟກສໂຄ້ດທີ່ຕ້ອງການທຳມາດຈິງ ຖ່ານັ້ນ (Business logic) ການ
ໃຊ້ງານ Node.js ຄື່ອ 1 ໃນຄຳຕອບດີທີ່ສຸດໃນຂໍ້ມູນນີ້ครັບ (ຂ່າຍທີ່ພົມແຕ່ງຫັນສຶກນະ)



ภาพล้อเลียนการเขียนโปรแกรมเพื่อແຕກທຽດ ຮະຫວ່າງການຄຸນກັບປົກປົກ

(ที่มาภาพ <http://www.funnymeme.com/2015/06/15/dog-memes-multithreaded-programming/>)

ແຕ່ເວັບກະຈົບນິດໜີ່ ແນວ່າ Node.js ມີຄູກອອກແບບໃໝ່ທຽດເດືອຍ ແຕ່ຄຸນກີ່ສາມາດໃໝ່ມອດຸລ “cluster”
ເພື່ອແຕກໂພຣເຊກສາກທຳມາດຍ່ອຍໄດ້ເຊັ່ນກັນ (ທຳງ່າຍດ້ວຍ)

มодูล

ปกติแล้วถ้าเราจะนำเข้า (Import) ไฟล์ *.js ตัวอื่น ๆ ในหน้าเว็บ HTML เราอาจจะใช้โค้ดตั้งตัวอย่างนี้

```
<script src="lib.js"></script>
```

ส่วนในจาวасคริปต์ฝั่ง Node.js ก็สามารถนำเข้าไฟล์ *.js ตัวอื่นเข้ามาได้ (แต่ผู้ช่วยเรียกว่า “**โหลด**”) ด้วย ประโยชน์ดังนี้

```
var module = require('module_name');
```

โค้ดนี้คุณเคยเห็นแล้วแหล่ะ เวลาโหลดมอดูลในบทก่อน ๆ ด้วย require() แต่ในบทนี้ผมจะอธิบายให้ลึกลงไปอีก จากที่เคยค้างค้างเอาไว้ 😊

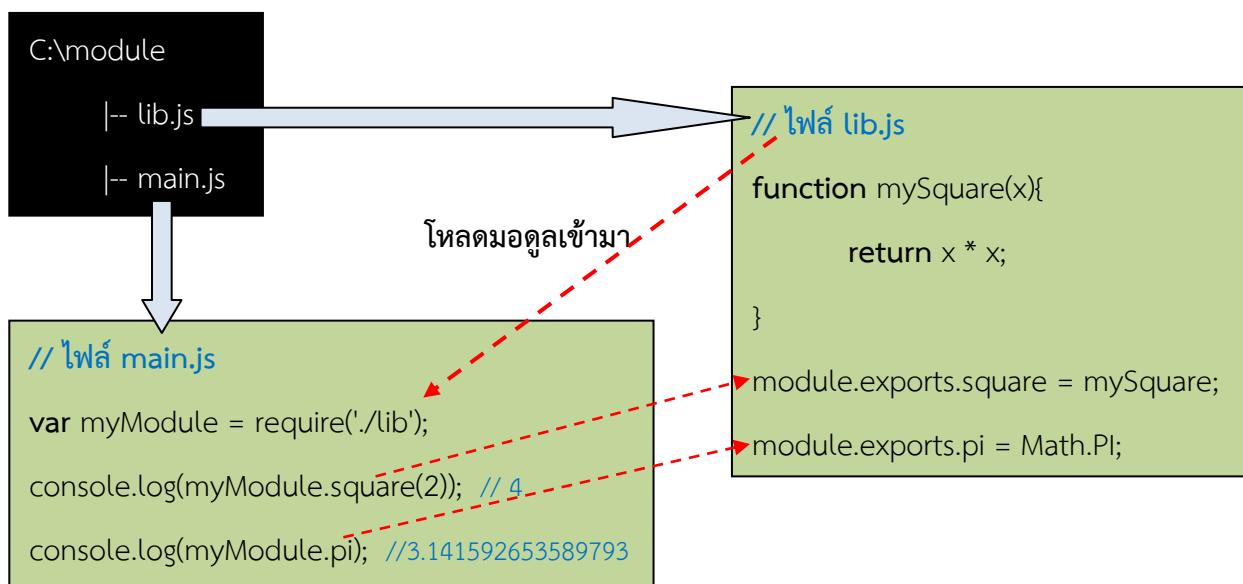
ในบทก่อนหน้านี้ คุณคงเคยเห็นนามอดูลชื่อ “fs” ซึ่งจะเป็นมอดูลหลักของ Node.js ที่มีมาให้อยู่แล้ว แต่ถ้าเป็นมอดูลเสริม (third-party) คุณต้องโหลดมาติดตั้ง ด้วยคำสั่ง npm บนคอมมานไลน์ดังนี้

```
npm install <ชื่อมอดูล>
```

ข้อควรระวัง ถ้าคุณไม่ดาวน์โหลด “**วิธีติดตั้ง Node.js และ npm เบื้องต้น**” มาอ่าน ...เนื้อหาต่อจากนี้ไปอาจจะงงได้นิดหน่อย

วิธีสร้างมอดูลใช้งานเอง

วิธีสร้างมอดูลขึ้นมาใช้งาน จะอธิบายง่าย ๆ โดยให้คุณดูโครงสร้างโปรเจค และโค้ดข้างล่างต่อไปนี้



ถ้าคุณลองรันคำสั่ง "node main.js" ก็จะได้ผลลัพธ์ดังนี้

```
C:\module>node main.js
4
3.141592653589793
```

จากตัวอย่างที่ผมยกมาให้ดู จะเป็นการสร้างมодูล “lib” (ไฟล์ชื่อ lib.js) ซึ่งคุณน่าจะเห็นตอนเชป์การสร้าง มодูลมันไม่ยากเลย โดยสามารถสรุปวิธีการสร้างได้ดังนี้

- มودูลจะต้องเขียนเป็นไฟล์ภาษาสคริปต์ ที่มีนามสกุล *.js
- สำหรับสิ่งที่ส่งออกไป หรือ export ออกไป (ให้ไฟล์ตัวอื่นมาโหลดไปใช้งาน) จะเป็นอะไรก็ได้ เช่น พัฟ์ชั่น ตัวแปร คลาส และอื่น ๆ โดยจะมีรูปแบบประโยคคำสั่งดังต่อไปนี้

```
module.exports.xxx = yyy;
// โดยที่ yyy คือสิ่งที่จะส่งออกไป
// แต่เมื่อไฟล์ภาษาสคริปต์ตัวอื่น มาโหลด yyy ไปใช้งาน ก็จะให้เข้าถึงโดยใช้ชื่อ xxx แทน

// หรือจะเขียนเป็น
module.exports = object; // เมื่อ object คืออ็อบเจกต์ในภาษาสคริปต์
```

วิธีโหลดมอดูล

ถ้าเป็นมอดูลหลัก (Core modules) ที่มีมาให้แล้วใน Node.js เรา ก็โหลดมาใช้งานได้เลย โดยไม่ต้องระบุชื่อ path (path) เช่น

```
var http = require('http'); // http คือมอดูลหลัก
```

แต่ถ้ามอดูลที่โหลดเข้ามานั้น เราเขียนขึ้นมาใช้เอง ก็ต้องโหลดมาเป็นไฟล์ .js โดยการระบุชื่อพาร์忒์ ๆ ไป เลย ดังนี้

```
var myModule = require('/module/lib'); // ไม่มีนามสกุล .js ต่อท้ายชื่อไฟล์
// var myModule = require('/module/lib.js'); // จะมี .js ต่อท้ายชื่อไฟล์ก็ได้
```

ประโยคคำสั่งนี้จะโหลดมอดูล “lib” ที่ผมเขียนขึ้นเองกับมือ (ในหัวขอก่อน) โดยมีรายละเอียดที่น่าสนใจดังนี้

- เนื่องจากผู้ใช้งานบนวินโดวส์ และตัว Node.js ก็ติดตั้งอยู่ที่ไดร์ฟ C: (ไฟล์ติดตั้งอยู่ที่ “C:\Program Files\nodejs\”) ...ด้วยเหตุนี้พาร์ชีชื่อ '/module/lib' ก็คือ 'c:\module\lib.js' นั้นเองจะครับ
- เราสามารถระบุชื่อมодูลให้เป็นชื่อไฟล์ โดยมีนามสกุล .js ต่อท้าย หรือไม่มี .js ต่อท้ายก็ได้ ... เพราะ Node.js จะมองเห็นเป็นชื่อโฟลเดอร์ก่อน เล้าถ้าค้นหาไม่เจอ ก็จะเติม .js ต่อท้ายชื่อเข้าไป (เดียวต่อไปคุณจะเห็นเองว่า เราสามารถโหลดมอดูลเป็นโฟลเดอร์ได้ด้วย)

แต่ถ้าเราอ้างพาธแบบ Relative ก็ให้ใช้ “./” นำหน้า เพื่อบอกตำแหน่งได้เรียบง่ายๆ จุดที่ไฟล์กำลังทำงานอยู่ ซึ่งในตัวอย่างเดิม ถ้าไฟล์ที่โหลดมอดูล lib.js มันทำงานอยู่ที่ c:\module ก็อาจเขียนใหม่ได้เป็น

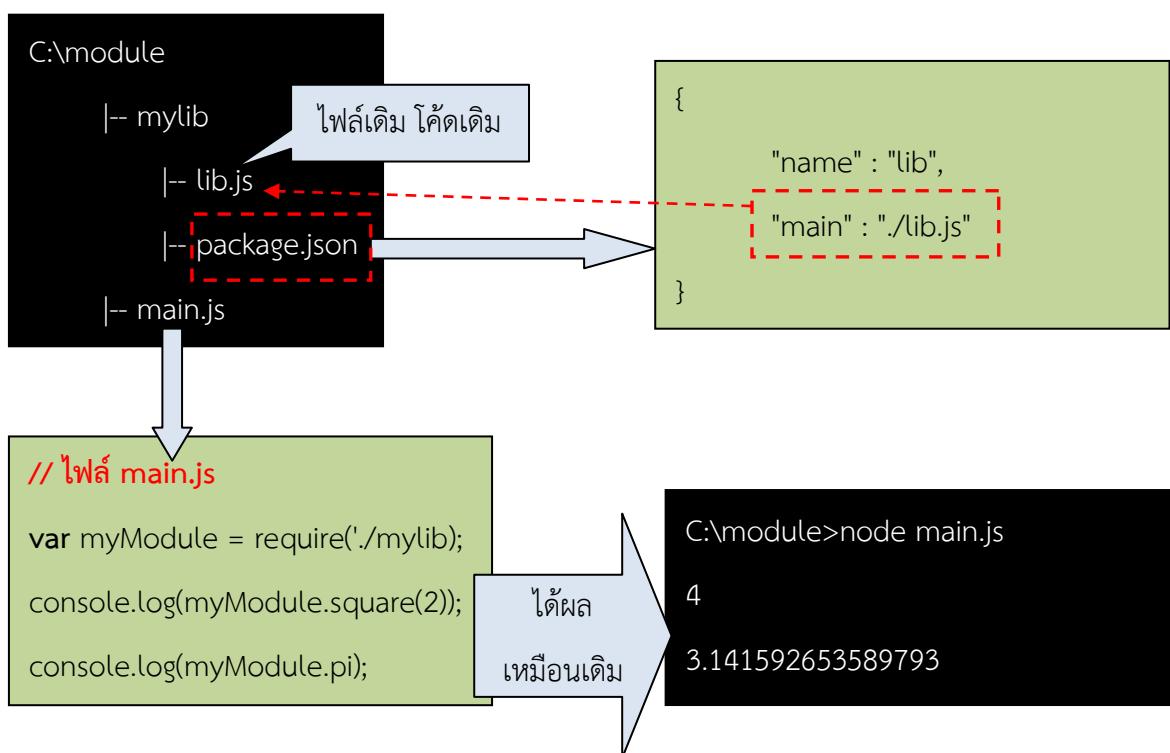
```
var myModule1 = require('./lib');
var myModule2 = require('./lib');
var myModule3 = require('../module/lib'); // บรรทัดที่ 3
```

```
C:\module
  |-- lib.js
```

ในตัวอย่างนี้ คุณจะเห็นว่าประโยชน์คำสั่งในบรรทัดที่ 3 เราสามารถใช้ “../” เพื่อยื้บได้เรียบง่ายๆ มากหนึ่งระดับ ...ซึ่งมันเป็นการอ้างพาธที่เราคุ้นเคยกันดีนี้แหละเนอะ

โหลดมอดูลจากโฟลเดอร์

เราสามารถโหลดมอดูลด้วยการระบุเป็นชื่อโฟลเดอร์ก็ได้ โดยก่อนอื่นจะขอจัดโครงสร้างโปรเจคใหม่ ดังนี้



ในตัวอย่างที่ยกมาในหน้าก่อน คุณจะเห็นประโยชน์ค้าสั้ง  `var myModule = require('./mylib');`

มันคือการอ้างถึงชื่อโฟลเดอร์เป็น "mylib" โดยทั้งนี้ Node.js จะเข้าไปอ่านไฟล์ "package.json" (ในโฟลเดอร์ mylib) ซึ่งข้างในไฟล์จะระบุพื้นที่เป็น "main" : "./lib.js" เพื่อบอกชื่อมодูลที่จะถูกโหลดเข้ามาว่า ...มันชื่อเป็น lib.js (อยู่ในไดเรคเทอรีปัจจุบัน mylib)

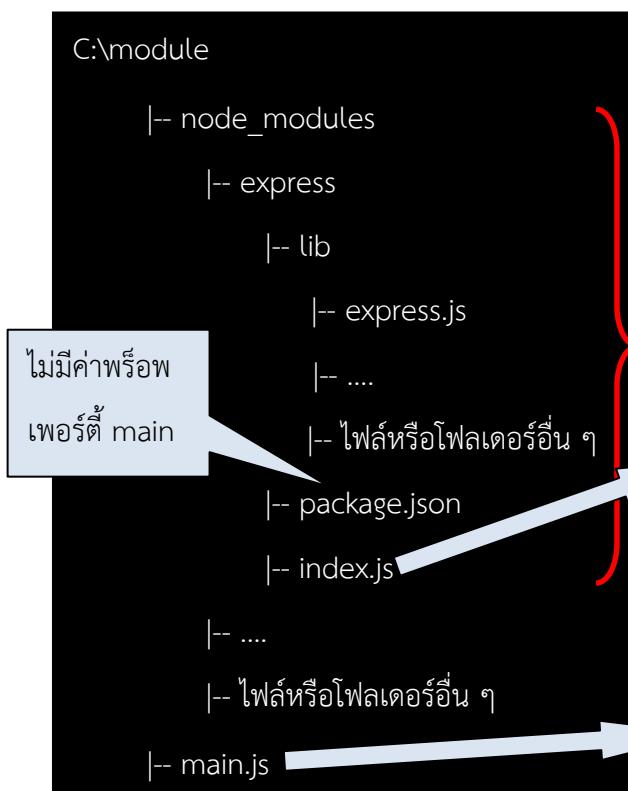
แต่ถ้าไฟล์ package.json ไม่มีชื่อพื้นที่เป็น "main" ละก็ ...โดยดีฟอลต์แล้ว ตัวฟังก์ชัน require() จะโหลด 모ดูลที่มีชื่อไฟล์เป็น "index.js" เข้ามาแทน

สรุป ฟังก์ชัน require() เมื่อเจอโฟลเดอร์ ก็จะเข้าไปอ่านไฟล์ package.json เพื่อหาที่อยู่ของไฟล์มودูลที่จะโหลดเข้ามา ถ้าไม่มีระบุไว้ในค่าพื้นที่ main ก็จะใช้ไฟล์ชื่อ index.js แทนครับผม

โหลดมอดูลจากโฟลเดอร์ node_modules

ก่อนจะอธิบายการโหลดมอดูลด้วยวิธีนี้ ผู้จะโหลดมอดูลเสริมเข้ามา ด้วยคำสั่งบนคอมมานไลน์ดังนี้

```
npm install express --save
```



ในตัวอย่างนี้คำสั่ง npm จะทำงานอยู่ในไดเรคเทอรี C:\module ...เมื่อติดตั้งมอดูลเสริมเสร็จแล้ว ก็จะเห็นว่ามีโฟลเดอร์ node_modules โผล่ขึ้นมา

โดยจะมีโครงสร้างโปรเจคดังนี้

// ไฟล์ index.js

```
'use strict';
module.exports = require('./lib/express');
```

// main.js เจียนโค้ดขึ้นมาใหม่เป็น

```
var express = require('express');
```

ในตัวอย่างนี้ไฟล์ main.js จะโหลด module ที่อยู่ในโฟลเดอร์ node_modules ด้วยการเขียนโค้ดเป็น

```
var express = require('express');
```

เราต้องเข้าใจอย่างนี้นะครับ ถ้าไม่ได้โหลด module หลัก และไม่ได้ระบุชื่อ path มันจะวิ่งไปหา module ที่พารคือ “./node_modules/” และในตัวอย่างนี้จะหา module “express” ที่พารคือ “C:\module\node_modules\”

ซึ่งมันจะหาโฟลเดอร์ express เจอ แต่ทว่าไฟล์ package.json ที่อยู่ในโฟลเดอร์ ไม่ได้ระบุค่าพร็อพเพอร์ตี้ main ด้วยเหตุนี้ฟังก์ชัน require() จึงโหลด index.js เข้ามาแทน ซึ่งข้างในไฟล์จะบอกให้ไปโหลดไฟล์ module ‘./lib/express.js’ เข้ามา เพราะมันใช้ประโยชน์คำสั่ง

```
module.exports = require('./lib/express');
```

...แต่ยังไม่จบ ถ้าสมมตินะครับ ถ้าไม่มีโฟลเดอร์ node_modules ในไดเรกเทอร์ปัจจุบัน ฟังก์ชัน require() ก็จะวิ่งไปหาที่โฟลเดอร์แม่ ได้แก่ “./node_modules/” และถ้าไม่เจออีก ก็จะไปค้นหาอย่างไดเรกเทอร์ซึ่งได้ติดตั้ง npm (ไดเรกเทอร์ที่เป็นรูท)

ในวินโดวส์ก็จะเป็น “C:\Users\username\AppData\Roaming\npm”

(เมื่อ username คือชื่อโฟลเดอร์ของผู้ใช้งานบนเครื่อง)

แต่ถ้าเป็น MacOS หรือ Linux ตัว npm ก็จะติดตั้งอยู่ที่ “/usr/local/share/npm”

ถ้าจะสรุปวิธีการมองหา module ที่อยู่ในโฟลเดอร์ node_modules ก็จะเป็นดังนี้

1. จะมองหา module ที่อยู่ใน node_modules ปัจจุบันก่อน แต่ถ้าไม่เจอ ก็จะไปข้อ 2
2. จะมองหา module ที่ไดเรกเทอร์แม่ ได้แก่ ..\node_modules/ แต่ถ้าไม่เจออีก ก็ไปข้อ 3
3. จะมองหา module ที่ไดเรกเทอร์รูท เป็นลำดับสุดท้าย

สรุปวิธีโหลดมอดูลที่กล่าวมาทั้งหมด ก็จะมีดังนี้

1. โหลดจากมอดูลหลัก (มอดูลมาตรฐาน)
2. โหลดมอดูลด้วยการระบุ파รเป็นชื่อไฟล์ *.js
3. โหลดมอดูลด้วยการระบุ파รเป็นชื่อโฟลเดอร์
4. โหลดมอดูลจากโฟลเดอร์ node_modules



เกร็็ดความรู้ เมื่อโหลดมอดูลมาครั้งแรกด้วยฟังก์ชัน require() มันจะเก็บไว้ในหน่วยความจำ (Cached) และเมื่อโหลดมอดูลซึ่งเดิมครั้งต่อ ๆ มา มันก็จะใช้ตัวเดิมที่อยู่ในหน่วยความจำ เช่น

```
var http1 = require('http'); // บรรทัด 1
var http2 = require('http'); // บรรทัด 2
var http3 = require('http'); // บรรทัด 3
```

ในตัวอย่างนี้แม้ว่าจะโหลดมอดูล http สามครั้งก็จริง แต่มันจะใช้งานมอดูลในหน่วยความจำที่เดียวกัน

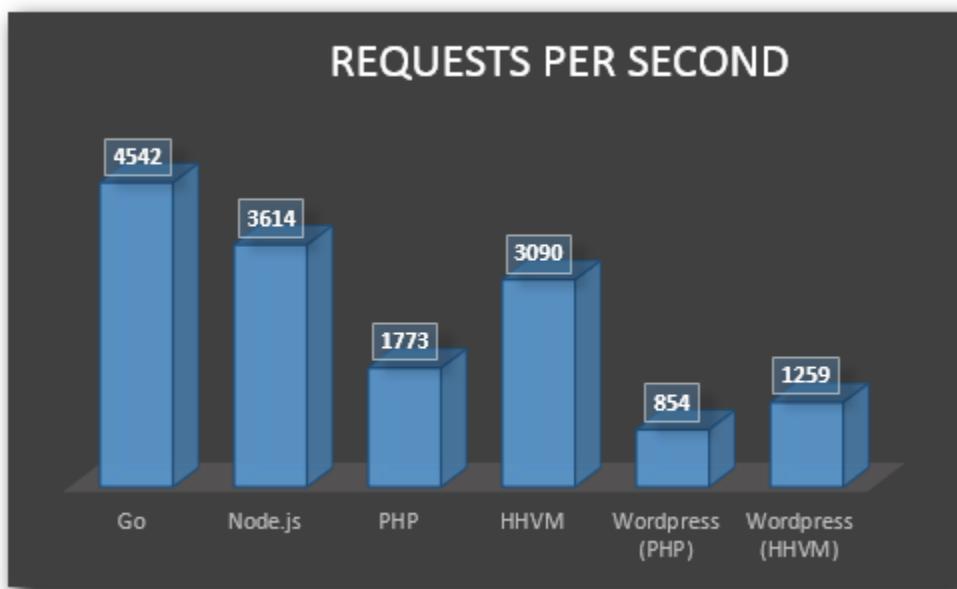
ผลการทดสอบ Benchmarks

เพื่อไม่ให้เชียร์ Node.js จนออกหน้าอกตาเกินไปนัก ผู้เขียนพยายามทดลองวัดประสิทธิภาพของมัน เมื่อเทียบกับภาษาอื่น ๆ ซึ่งมีคนวัดผลเอาไว้เรียบร้อยแล้ว ตามลิงค์ข้างล่าง

- <http://dan.hersam.com/2015/02/25/go-vs-node-vs-php-vs-hhvm-and-wordpress-benchmarks/>

วิธีการทดลองเขาจะเขียนโปรแกรมอยู่ในฝั่งเซิร์ฟเวอร์ เมื่อมี Requests ติดต่อเข้ามา ก็จะส่งผลลัพธ์ตอบกลับไป (Response) เป็นข้อความ "Hello world" อย่างง่าย ๆ ซึ่งโปรแกรมจะเขียนขึ้นด้วย 3 ภาษา ได้แก่ Go, Node.js และ PHP

หลังจากนั้นเขาจะใช้เครื่องมือ Test ก็คือ “Apache HTTP server benchmarking” ทำการส่ง Requests หลาย ๆ อัน ไปทดสอบโปรแกรมทดลอง แล้วดูว่า ...แต่ละโปรแกรมจะรับ Request ภายใน 1 วินาที ได้มากสุดเท่าไร? จากนั้นจึงนำผลมาเปรียบเทียบกัน ซึ่งจะได้ตามรูปข้างล่าง



สภาพแวดล้อมของการทดลอง
 เชิร์ฟเวอร์เป็น Vultr
 RAM ขนาด 768MB
 CPU ตัวเดียว 3.4MHz
 และใช้ nginx 1.6.2

ผลการวัดของแต่ละโปรแกรม ซึ่งจะมีหน่วยวัดเป็นจำนวน Requests ที่รับได้ต่อวินาที [9]

จากรูปจะเห็นว่าภาษา Go ประสิทธิภาพนำมารอันดับ 1 ชนภาษาอื่นหลุดลุย ส่วนอันดับ 2 คือ Node.js ส่วน PHP นี้เหมือนจะแพ้ชาวบ้านเขา ...แต่เดียวกว่าก่อนเมื่อใช้ HHVM อย่างเดียว จำนวน Request ที่รับได้ (3090 ครั้งต่อวินาที) จะใกล้เคียงกับ Node.js (3614 ครั้งต่อวินาที)

หมายเหตุ

- ภาษา Go เจ้าของคือ Google
- ส่วน HHVM (HipHop Virtual Machine) เจ้าของคือ Facebook ซึ่ง HHVM จะทำหน้าที่คอมไพล์ภาษา PHP (รวมทั้งภาษา Hack ด้วย) ให้กลายมาเป็น bytecode นั่นเอง ด้วยเหตุนี้จึงไม่น่าแปลกใจที่มันจะทำงานเร็วใกล้เคียงกับ Node.js
- แต่ Node.js ที่เข้าทำการทดลองเมื่อปี 2558 มันยังเป็นเวอร์ชันอันก่าอยู่ ถ้าเป็น Node.js เวอร์ชัน 6 (ล่าสุด) น่าจะมีประสิทธิภาพดีกว่านี้ (เดาคาดรับ)
- แต่ทว่าผลการวัดที่นำมาให้ดู มันบอกแค่ประสิทธิภาพของแต่ละภาษาว่า ...รับ Request ที่ติดต่อเข้ามา เก่งมากน้อยแค่ไหน? แต่เมื่อได้วัดความสามารถในการติดต่อ I/O หนัก ๆ ว่าเป็นอย่างไร? ...จึงยังไม่ได้วัดประสิทธิภาพของแต่ละภาษาครบทุกด้าน

สรุป ผลการวัดที่นำมาให้ดู เพื่อจะบอกว่า Node.js ยังไงก็ไม่ใช่เครื่องมือที่เจ๋งสุดในทุกด้าน ดังนั้นการใช้งานก็ควรใช้ให้ถูกต้อง และเหมาะสมกับงานนั้น ๆ จะดีกว่า

สังท้าย

“สำหรับเล่ม 1 ผู้จบแคนเน่นะครับ”

สำหรับ เล่ม 2 จะเป็นแนวลอนหัวรือใช้งาน Node.js เน้นพากิจกรรมซื้อขาย ๆ เพื่อให้มองเห็นภาพรวมในการใช้งานมากกว่า ซึ่งเล่มนี้ก็สามารถโหลดได้ตามลิงค์ข้างล่างเช่นเคย

- http://www.patanasongsivilai.com/itebook_form.html

*** ทั้งนี้อนาคตเนื้อหาอาจปรับปรุงเปลี่ยนแปลงได้ ตามเทคโนโลยีที่มุนเร็วไปติดจรวด

และถ้าพูดกันตามตรง ผู้ใดไม่แนะนำให้คุณใช้งาน Node.js ดิบ ๆ เถื่อน ๆ หรอก เพราะเวลาคุณทำงานจริง มันมีเครื่องมือช่วยเขียน ไม่ว่าจะเป็น เฟรมเวิร์ค แพลทฟอร์ม และอื่น ๆ ที่จะทำให้ชีวิตเขียนโปรแกรมคุณดีกว่านี้ครับ

อ้างอิง

หนังสือ

[1] Pedro Teixeira, “Professional Node.js Building Javascript Based Scalable Software”, John Wiley & Sons, Inc., 2013.

เอกสารจากเว็บไซต์ เข้าถึงล่าสุด 30 ธ.ค. 2558

[1] <https://en.wikipedia.org/wiki/Node.js>

[2] <https://nodejs.org/en/>

[3] <http://expressjs.com/>

[4] <http://getbootstrap.com/>

[5] <http://ejs.co/>

[6] <http://expressjs.com/en/starter/generator.html>

[7] <http://kikobeats.com/synchronously-asynchronous/>

[8] <https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/>

[9] <http://dan.hersam.com/2015/02/25/go-vs-node-vs-php-vs-hhvm-and-wordpress-benchmarks/>