1. [About NodeJS](https://nodejs.org/about/)
   1. 原文

About Node.js®

As an asynchronous event driven framework, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection the callback is fired, but if there is no work to be done Node is sleeping.

var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World\n');

}).listen(1337, "127.0.0.1");

console.log('Server running at http://127.0.0.1:1337/');

This is in contrast to today's more common concurrency model where OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node are free from worries of dead-locking the process—there are no locks. Almost no function in Node directly performs I/O, so the process never blocks. Because nothing blocks, less-than-expert programmers are able to develop scalable systems.

Node is similar in design to and influenced by systems like Ruby's Event Machine or Python's Twisted. Node takes the event model a bit further, it presents the event loop as a language construct instead of as a library. In other systems there is always a blocking call to start the event-loop. Typically one defines behavior through callbacks at the beginning of a script and at the end starts a server through a blocking call like EventMachine::run(). In Node there is no such start-the-event-loop call. Node simply enters the event loop after executing the input script. Node exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript -— the event loop is hidden from the user.

HTTP is a first class citizen in Node, designed with streaming and low latency in mind. This makes Node well suited for the foundation of web library or framework.

Just because Node is designed without threads, doesn't mean you cannot take advantage of multiple cores in your environment. You can spawn child processes that are easy to communicate with by using our child\_process.fork() API. Built upon that same interface is the cluster module, which allows you to share sockets between processes to enable load balancing over your cores.

* 1. 翻译

作为一个异步事件驱动的框架，Node.js被设计用来构建可伸缩的网络应用。在下面的“hello,world”例子中，很多连接能够给同时被处理。在每一连接建立之后出发它的回调函数，但是如果没有工作去做时，Node便会等待。

var http = require('http');

http.createServer(function(req, res){

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello, world!');

}).listen(1222, '127.0.0.1');

console.log('Server running at <http://127.0.0.1:1333>');

这与现今大部分OS使用的并发模型的思路相反。基于线程的网络设计是相对低效并且难以使用。因此，Node的用户不用担心进程的死锁，因为压根就没有锁。在Node中机会没有任何函数执行 I/O，所以进程永远不会阻塞。因为不会阻塞，所以没有丰富开发经验的开发人员也能开发可灵活的系统。

Node在设计上受Ruby的Event Machine或者Python的Twisted，与它们相似。Node让事件模型更上一层楼，它提出将事件循环作为语言结构而不是一个库。在其他的系统中，总是存在一个阻塞的调用来开始事件循环。一个有代表性的例子是：在脚本的开始处通过回调函数定义行为，在最底部通过一个阻塞调用比如EventMachine::Run()来启动一个服务。在Node中，没有像这样的“启动事件循环”的调用。执行完输入的脚本之后，Node就已经进入了事件循环。当没有更多回调函数来执行的时候，Node依旧会宝轮这个事件循环。这个行为就像是浏览器中的JavaScript——事件循环对于用户而言是不可见。

记住，HTTP在Node中是一个一等公民，采用流（streaming）和低延迟（low latency）的设计方案，这使得Node非常适合作为网站库或者框架的基础。如果只是因为Node设计中没有线程，这并不意味你不能利用你系统环境中的多核的优势。使用child\_process.fork()API来创建子进程可以很轻松的通信。Node内建的类似接口——cluster模块可以帮助你在线程之间共用socket，以此来保证内核的负载平衡。

1. AngularJS
   1. 原文

What Is Angular?

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

Angular is what HTML would have been, had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in what do I have to do to trick the browser into doing what I want?

The impedance mismatch between dynamic applications and static documents is often solved with:

a library - a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.

frameworks - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., durandal, ember, etc.

Angular takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. Angular teaches the browser new syntax through a construct we call directives. Examples include:

Data binding, as in {{}}.

DOM control structures for repeating, showing and hiding DOM fragments.

Support for forms and form validation.

Attaching new behavior to DOM elements, such as DOM event handling.

Grouping of HTML into reusable components.

A complete client-side solution

Angular is not a single piece in the overall puzzle of building the client-side of a web application. It handles all of the DOM and AJAX glue code you once wrote by hand and puts it in a well-defined structure. This makes Angular opinionated about how a CRUD (Create, Read, Update, Delete) application should be built. But while it is opinionated, it also tries to make sure that its opinion is just a starting point you can easily change. Angular comes with the following out-of-the-box:

Everything you need to build a CRUD app in a cohesive set: Data-binding, basic templating directives, form validation, routing, deep-linking, reusable components and dependency injection.

Testability story: Unit-testing, end-to-end testing, mocks and test harnesses.

Seed application with directory layout and test scripts as a starting point.

Angular's sweet spot

Angular simplifies application development by presenting a higher level of abstraction to the developer. Like any abstraction, it comes at a cost of flexibility. In other words, not every app is a good fit for Angular. Angular was built with the CRUD application in mind. Luckily CRUD applications represent the majority of web applications. To understand what Angular is good at, though, it helps to understand when an app is not a good fit for Angular.

Games and GUI editors are examples of applications with intensive and tricky DOM manipulation. These kinds of apps are different from CRUD apps, and as a result are probably not a good fit for Angular. In these cases it may be better to use a library with a lower level of abstraction, such as jQuery.

* 1. 翻译

AngularJS是一个针对动态web的结构型框架。它能够帮助使用html作为模板语言同时清晰地简单的拓展HTML的语法来表现你的应用组件。Angular的数据绑定和依赖注入减少了很多代码的书写。并且它的所有都发生在浏览器的内部，这让它成为了服务端技术的完美搭档。

Angular是针对应用而设计的。对于静态文档来说，HTML是一个非常好的声明语言。但是HTML没有太多用来创建应用的方法，而创建一个web应用就是一个“我要做些什么来让浏览器听我的话”的过程。

在动态应用和静态页面之间的矛盾（阻抗失配）常常通过这些来解决：

一个库

一个框架

Angular 采用了另一种方法。它试图将二者之间的矛盾最小化，Angular通过一个叫做指令（directive）的结构让浏览器接受新的语法，例如：

数据绑定， {{ }}

DOM控制结构，用以重复，显示和隐藏DOM碎片

支持表单和表单验证

为DOM元素附上新的行为，比如DOM事件绑定

将HTML聚合成可以重复使用的组件

一个完善的客户端端（client-side）解决方案

在整个构建web应用程序的客户端的难点中，Angular并不是一个单一的解决方案。它还处理了你曾经手写的所有的DOM和AJAX的胶水代码（glue code），并且定义了很好的结构。这使得Angular在如何构建CRUD(Create， Read， Update， Delete)应用程序的问题上固执己见。虽然如此，但是它也试着确保他的选项只是一个出发点，你可以很容的修改。Angular从下面这些创造性的特点开始：

创建一个CRUD应用你所需要做的事情只有这些：数据绑定，基础模板指令，表单验证，路由，深链接，可复用的组件和依赖注入

测试方案：单元测试，端对端测试，mocks测试和测试装置

种子应用程序目录布局和测试脚本作为起点

AngularJS的可爱之处

AngularJS通过为开发者呈现一个更高层次的抽象来简化应用的开发。如同其他的抽象技术一样，这也会损失一部分灵活性。换句话说，并不是所有的应用都适合用AngularJS来做。AngularJS主要考虑的是构建CRUD应用。幸运的是，至少90%的WEB应用都是CRUD应用。但是要了解什么适合用AngularJS构建，就得了解什么不适合用AngularJS构建。

如游戏，图形界面编辑器，这种DOM操作很频繁也很复杂的应用，和CRUD应用就有很大的不同，它们不适合用AngularJS来构建。像这种情况用一些更轻量、简单的技术如jQuery可能会更好。