1. 개요
In Nodejs.org Homepage,
-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.
-Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.
1)Server Side JavaScript
2)Built on Google Chrome V8 Runtime
3)Non-blocking I/O (Asynchronous)
4)Event-Driven
5)Module System
6)Native on Linux, Mac OS X, Windows
7)Enables Real-time Web
8)Created by Ryan Dahl in 2009.
-First presented version 0.1.16 at JSConf EU 2009(2009/11/7~8, 베를린)

2. 특징
1)개발언어 : JavaScript
-개발자는 Browser에서 사용하던 JavaScript 지식과 개발 경험을 그대로 사용해서 서버프로그래밍에 사용 가능
-Server Programming을 위해 새로운 언어를 배울 필요가 없다.
-JS devs already think asynchronously (Browsers + AJAX)
-JS is fast and getting faster
-JS quickly is becoming a compilation target
--https://github.com/jashkenas/coffee-script/wiki/List-of-languages-that-compile-to-JS
-JSON native and full application stack
--MongoDB, Node.js, Angular.js

2)Chrome V8 사용
-JavaScript의 해석과 동작을 위해 Chrome V8 엔진 사용
-2008년 9월 2일, Google Chrome 에 내장되어 공개
-JavaScript 엔진 중에서 가장 빠른 속도를 내기 때문에, 일정 수준의 성능도 보장된다.
-기본의 컴파일된 바이트코드를 실행시키거나 인터프리터로 해석하는 대신에 JavaScript를 컴파일하여 Native Machine Code로 변경한 후 실행할 수 있게 해준다.
-이렇게 컴파일된 코드는 코드의 실행 프로파일을 분석하여, 실행 시에 다시 한번 최적화된다.
-V8은 객체에 할당되는 메모리를 효율적으로 관리하며, 더 이상 필요 없는 객체는 수집하여 버린다(Garbage Collection).
-정확한 가비지 콜렉터는 V8의 고성능에 이바지하는 주요한 키 포인트이다.
-이러한 V8엔진은 웹 브라우저(크롬)에서 실행될 뿐 아니라, 독립형(Standalone)의 고성능 엔진으로 사용할 수 있고 웹 브라우저와는 전혀 상관없이 사용할 수 있다.
-웹 브라우저와 상관없이 독립형 JavaScript 언어로 개발을 가능하게 한 플랫폼이 node.js이다.
-V8 is a JavaScript engine specifically designed for fast execution of large JavaScript applications
-Used in Google's Chrome browser
-Written in C++
-Fast property access
-Dynamic machine code generation
-Efficient garbage collection
--https://developers.google.com/v8/
--https://developers.google.com/v8/intro
--http://blog.chromium.org/2012/05/better-code-optimization-decisions-for.html

3)Non-Blocking I/O
-작업이 끝날 때까지 대기해야 하는 방식은 Blocking 방식(작업이 끝날 때까지 대기하는 시간은 성능상의 낭비이다)이다.

-메모리를 계속 차지해서 자원 낭비가 심하다
-하지만 Non-blocking 방식은 IO 이벤트가 시작되어도 모듈을 반환하여 다른 작업이 가능한 준비상태로 전환된다.
-이러한 개발방식은 동기식보다 빠르고 메모리도 적게 차지하게 된다.
-IO is not free
  --L1: 3 cycles
  --L2: 14 cycles
  --RAM: 250 cycles
  --DISK: 41,000,000 cycles
  --NETWORK: 240,000,000 cycles
  --http://duartes.org/gustavo/blog/post/what-your-computer-does-while-you-wait/
-시간이 걸리는 IO
  --하드 디스크 접근
  --데이터베이스 서버
  --네트워크를 이용해서 다른 서비스 접근
-대형 할인 매장을 가면 사고 싶은 물건을 모두 선택한 다음 계산을 하기 위해 계산대 앞에 줄을 선다.
-평일 낮 시간이라 계산하는 점원이 1명밖에 없고, 3명의 손님이 차례를 기다리고 있다고 가정해보자.
-3명의 손님이 모두 계산할 때까지 기다려야 한다.
-1명의 손님이 계산이 끝나는데 평균 1분 정도 소용된다고 하면 본인은 3분 동안 줄이 줄어들기를 기다려야 한다.
-이것이 바로 Blocking이라고 한다.

-만일 내가 커피숍에 가서 작업을 하려고 한다고 가정하자
-주말에 커피숍은 언제나 사람들로 북적거린다.
-도착하자마자 나는 카운터에 가서 커피를 주문하고 동그란 무선 호출기를 받는다.
-커피가 준비되면 알람을 울려 커피가 준비되었음을 알려주는 호출기이다.
-호출기를 받고, 구석진 자리를 선택해 노트북의 전원 플러그를 꼽을 수 있는지 확인한다.
-노트북을 세팅하고, 가져온 참고 문서를 꺼내고, MP3 플레이어 및 웹 서핑 용도의 태블릿 PC를 노트북 옆에 세워 놓는다.
-그리고 창밖의 멋진 풍경을 잠시 감상하고 있다.
-갑자기 호출기가 붉은 빛을 토하면서 울린다.
-아까 주문한 커피가 준비되었다고 울리는가보다.
-나는 카운터로 가서 주문했던 커피를 받아서 자리로 돌아온다.
-커피를 주문한 다음 준비가 될 때까지 전혀 간섭받지 않고 하고 싶은 수행했다.
-블로킹을 당하지 않았다. Non-Blocking

4)Event Driven 기반의 프로그래밍
  -사용자 또는 외부 환경이 만들어낸 이벤트를 기반으로 어떤 처리를 할지 지정해주는 개발 방식
  -몇몇 웹서버는 데이터가 들어올 때까지 무한정 기다리는데, Node.js를 통해 Event-driven으로 개발하게 되면 특정 이벤트에 대한 처리만하여 웹서버와 연결시키면 되기 때문에 자원을 최소화하게 된다.
  -Efficient (if used asynchronously)
  -Only one stack
  -No memory overhead
  -Simpler model (no deadlocks, no race conditions ...)
  -카운터의 종업원은 필자가 주문하는 이벤트, 그리고 커피 준비가 완료되었다는 두 가지 이벤트를 받았다.
  -종업원은 해당 이벤트가 발생했을 때 본인이 해야 할 일을 했다.
  -이것이 바로 이벤트 주도(Event-driven)이다.
  -여기서 나에게 커피가 완료되었다고 알려준 호출기가 중요한 포인트이다.
  -이를 프로그래밍에서는 이벤트 콜백(Event callback)이라고 한다.

5)Single Thread
  -Each Thread takes memory
  -Threads introduce additional complexity
    --Race conditions
    --Deadlock
    --Additional setup overhead

6)아래 사이트의 그림 참조 : Node.js의 동작 원리 설명
-http://www.aaronstannard.com/intro-to-nodejs-for-net-developers/

7)The C10K Problem
-C10K refers to the problem of optimizing a web server to handle a large number of clients at the same time.
-C = CONCURRENT
-Apache uses one thread per connection
-NGINX doesn't use multiple threads but instead uses an event loop
-NGINX and Node.js are similar with respect to utilizing an event loop to achieve high concurrency on low latency
-http://www.kegel.com/c10k.html
-https://blog.webfaction.com/2008/12/a-little-holiday-present-10000-reqssec-with-nginx-2/

3. Server-side JavaScript History
1)Netscape LiveWire (1996)
2)Rhino (1997) [for Java]
3)Aptana Jaxer (2008)
4)RingoJS [for Java]
5)Narwhale
6)Node.js (2009)
7)IronJS [for .NET]
8)단순히 <script runat='server'...>

4. The node.js Darkside
1)Still relatively young
-실제 서비스에 적용된 레퍼런스가 많지 않다.
-다양한 상황에 대한 검증이 이뤄지지 않았다.
2)Bad idea to do raw computation in an event loop. Use node-webworker
3)Debugging is hard but will significantly improve in future versions
4)Callbacks (not really an issue)

```
doA( argA, function() {
  doB(argB, function() {
    doC(argC, function() {
      // etc.
    });
  });
});
```

5)Scale-up으로 성능이 크게 향상되지 않는다
-Single Thread를 사용하는 노드는 CPU 갯수나 메모리 용량에는 큰 영향을 받지 않는다.

5. Architecture
--refer to https://blog.zenika.com/2011/04/10/nodejs/
--refer to http://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop
--refer to https://www.youtube.com/watch?v=F6k8lTrAE2g
1)상위레벨 : JavaScript : Node Standard Library
2)Row Level : C
-Binding
-V8 Engine
-libev : Event
--다양한 기능을 가진 고성능 이벤트 루프 라이브러리

--libevent를 따라 느슨하게 모델링 되었지만, 이전에 존재했던 버그나 제한사항이 없다.
--프로세스 감시, 절대시각에 기반한 주기적 타이머, 그리고 epoll/kqueue/event ports/inotify/eventfd/signalfd 등 지원
--POSIX 라이브러리
-Libeio : IO
--이벤트 기반의 모든 게 비동기로 동작하는 C언어용 IO 라이브러리
--기본적으로 POSIX API에 기반을 두고 있으며 파일 처리 관련 작업을 한다.
--read, write, open, close, stat, unlink, fdatasync, mknod, readdir 등의 작업을 비동기로 처리
3)JavaScript로 작성한 Node.js 코드 동작을 위한 표준 라이브러리 부분과 C로 작성된 하위 부분으로 나뉜다.
4)Node.js 라이브러리와 인터페이스를 위한 Node Binding이 하위 레이어의 상단에 있다.

6. Node Package Manager(NPM)
1)NPM is a package manager for node.
2)You can use it to install and publish your node programs.
3)It manages dependencies and does other cool stuff.
4)http://npmjs.org/
5)http://search.npmjs.org/
-Express
-Request
-Socket.io (LearnBoost)
-Jade
-... 10k + more ...
6)Community-created Modules(express, socket.io, restify,...) --> Core Modules(http, net,...) --> Node Core(V8, Libev, Libeio,...)

7. REPL
1)Read Evaluate Print Loop (REPL)
2)Similar to Perl, Python, and Ruby
3)It's great for testing out and learning about Node.js
4)Since Node.js uses V8, Node REPL is an ideal place to easily try out and learn JavaScript

8. Node.js History
1)2010년 클라우드 서비스를 제공하는 회사인 Joyent가 자사의 클라우드 플랫폼 중 하나로 노드를 선택하면서 공식 스폰서로 나선다.
2)조이엔트는 라이언을 풀타임 노드 개발자로 고용하고, 몇 달 후 노드 커미터 중 한 명인 아이작 슐레터(Issac Z. Schlueter)도 고용하면서 전폭적으로 노드를 지원
3)Github에서 개발자가 관심 있는 프로젝트로 표시하는 Popular Watched 순위 (https://github.com/popular/watched)에서 노드는 만여 명의 watched를 받으며 jQuery와 Rails를 제치고 2위를 차지했다.
4)간단한 역사

[Node.js]	[io.js]
-2009 : Node.js 발표	
-2012 : v0.6(stable), v0.7(unstable)	
-2013 : v0.8(stable), v0.9	
-2014 : v0.10(stable), v0.11	2014.1 : io.js 발표
-2015 : v0.10.36(stable), v.0.11.15	2015 : v1.0
	2015.5 : v2.0
	2015.8 : v3.0

-2015.9 : v4.0
-2015.10 : v5.0
*2015.2 : Node.js 재단 설립 발표
*2015.5 : Node.js와 io.js 통합 결정
*Joyent 소속의 라이언 달이 1대 리더로서 Node.js의 개발을 이끌었다.
이후 2대 이후 3대 리더가 이끌면서 개발 속도가 더디게 됐다.
이에 Node.js 개발자들이 Node Forword 단체를 만들고 Nodejs를 포크한 io.js를 발표했고, Node.js보다 빠르게 새로운 기술을 대거 도입했다.
2015년에 Node.js재단이 설립되고, Node.js와 io.js를 통합하기로 결정했다.
이후에 출시된 Node.js는 io.js와 Node.js를 모두 계승한다.

*Node.js 재단(https://nodejs.org/en/foundation/)
-Node.js 플랫폼과 관련된 모듈 개발 지원하는 협업 오픈 소스 프로젝트
-Open Govenance Model
-기술 결정 위원회(Technical Streering Committee)
-주요 멤버(Platinum)
IBM, Intel, Joyent, Microsoft, PayPal, Redhat

9. 버전 구성과 지원
1)Node.js버전을 두 단계로 지원
2)기존 : 짝수버전(Stable), 홀수 버전(Unstable)
3)4.x 이후 : Stable, LTS
4)LTS : 짝수 버전 Stable 6개월 이후 LTS로 전환
-LTS(Long Term Support)
-LTS : 호환성이 깨지는 변경 없음
-LTS 18개월 지원, 그 후 Maintain 상태(12개월)
-매년 새로운 메이저 버전의 LTS 시작
5)홀수버전은 Stable만 진행되고 LTS로 전환되지 않는다. LTS는 매년 새로운 짝수 버전으로 LTS를 시작한다.
6)서비스를 운영할 때 LTS 버전에 맞춰서 운영하면 약 3년 동안 호환성 문제 없이 운영할 수 있다.

10. 적용 사례
1)전 세계적으로 모바일을 포함한 다양한 장치를 통해서 동시에 접속하는 대량 트랜젝션에도 빠르고 안정적인 처리량 및 응답속도를 제공해야 하는 경우
2)제한된 물리적인 리소스에도 대량 처리량을 지속적으로 처리해야 하는 경우
3)개발 플랫폼을 변경해야 하는 시점에 개발자들이 이미 알고 있는 개발언어인 JavaScript 를 사용함으로써 Learning Curve를 최소화해야 하는 경우
4)Front-end 의 소스와 Back-end 소소를 한 가지 언어로 통일시킴으로써 전사 표준 개발 프레임워크를 클라이언트/서버 구분 없이 단순하게 가져가고, 개발 조직 역시 구분하지 않게 프로젝트를 수행해야 하는 경우
5)eBay : http://www.ebaytechblog.com/2011/11/30/announcing-ql-io/ 참조
6)Linkedin : https://engineering.linkedin.com/testing/continuous-integration-mobile 참조
7)Yahoo's Cocktails -> Mojito(https://developer.yahoo.com/cocktails/mojito/docs/intro/)
8)Yammer : https://developer.yammer.com/

11. 권장 분양
1)실시간 Social Network
2)Data IO가 많은 분야
3)IoT 분야 등.

12. References
1)API

-http://nodejs.org/api
2)SRC
-https://github.com/joyent/node
3)How To Node
-http://howtonode.org
4)StackOverflow
-http://stackoverflow.com/questions/tagged/node.js
5)Projects & Companies using Node.js
-http://goo.gl/Wcqtj
6)Introduction to Node.js with Ryan Dahl
-http://youtu.be/jo_B4LTHi3I
7)The Node Beginner Book
-http://www.nodebeginner.org
8)Ryan Dahl - History of Node.js
-http://youtu.be/SAc0vQCC6UQ
9)What is Node? (Free)
-http://shop.oreilly.com/product/0636920021506.do
10)Ryan Dahl's 2009 slides
-http://s3.amazonaws.com/four.livejournal/20091117/jsconf.pdf
11)Node.js community wiki
-https://github.com/nodejs/node/wiki
12)Node.js를 배우는 방법
-http://mobicon.tistory.com/224

13. Book References
1)T Academy, Node.js 프로그래밍
2)실무 환경에 맞춘 Node.js 프로그래밍, 조인석/황수빈, 혜지원, 2014
3)Node.js 프로그래밍, 변정훈, 에이콘, 2012
4)제대로 배우는 Node.js 프로그래밍(Learning Node), 셜리 파워즈, 안재우 역, O'Reilly, 한빛미디어, 2013
5)Node.js 인 액션, 마이크 캔텔론 외, 정승희 외, 위키북스, 2014
6)실무에 바로 적용하는 Node.js, 아자트 마르단, 테크 트랜스 그룹 T4 역, 제이펍, 2015
7)Do it! Node.js 프로그래밍 [전면개정판], 정재곤, 이지스퍼블리싱, 2017
8)풀스택 개발자를 위한 MEAN 스택 입문, 애덤 브레즈 외, 박재호 역, 한빛미디어, 2015

14. Node.js's Trend & Future
1)업계 동향 실시간 차트
-http://langpop.corger.nl/
2)트랜디스킬스 닷컴
-https://trendyskills.com/
3)구글 트랜드
-https://trends.google.com/trends/explore?q=node.js,angularjs

15. Programming Model
1)Asynchronous Model
-동기모델이 함수 호출과 같은 작업이 있을 때, 해당 작업이 끝나기를 기다려야 하는 것과 달리 비동기 모델은 함수 호출 후 즉시 반환되고, 이후 함수의 작업 결과를 이벤트 혹은 콜백을 통해 전달받는다.
-동기식 함수 구현

```
function add(x, y){
  return x + y;
}
var result = add(3, 4);
```

```
304     console.log('Result = ' + result);
305
306   2)Lab1 : syncdemo.js
307     var fs = require('fs');
308     var contents = fs.readFileSync("syncdemo.js", "utf8");
309     console.log(contents);
310     console.log("Reading File...");
311    ---------------------------------------------
312     var fs = require('fs');
313     var contents = fs.readFileSync("syncdemo.js", "utf8");
314     console.log(contents);
315     console.log("Reading File...");
316     Reading File...
317
318    -비동기식 함수 구현
319     function add(x, y, callback){
320       var result = x + y;
321       callback(result);
322     }
323     add(1,2, function(result){
324       console.log('Result = ' + result);
325     });
326
327   3)Lab2 : asyncdemo.js
328     var fs = require('fs');
329     var contents = fs.readFile("asyncdemo.js", "utf8", function(error, contents){
330       if(error)
331         return console.log(error);
332       console.log(contents);
333     });
334     console.log("Reading File...");
335    ---------------------------------------------
336     Reading File...              <--비동기방식이기 때문에, 먼저 출력됨.
337     var fs = require('fs');
338     var contents = fs.readFile("asyncdemo.js", "utf8", function(error, contents){
339       if(error)
340         return console.log(error);
341       console.log(contents);
342     });
343     console.log("Reading File...");
344
345   4)Lab3 : readfile.js
346     var fs = require('fs');
347     var content = fs.readFile("readfile.js", encoding="utf-8", function(error, content){
348         if(error)
349             return console.log(error);
350         console.log(content);
351     });
352     console.log("파일의 내용 : ");
353
354   5)Callback
355    -callback함수는 다른 함수로 전달되는 파라미터처럼 호출될 함수를 알려주어, 다른 프로그램 또는 다른 모듈에서 특정 함수
       를 호출하게 하는 방법이다.
356
```

6)Lab : callbackdemo.js

```
var fs = require('fs');
var fcontens;
function readingFile(callback){
    fs.readFile("callback.js", "utf-8", function(error, contents){
        fcontents = contents;
        if(error) return console.error(error.stack);
        callback(contents);
    })
}
function mycontent(){
    console.log(fcontents);
}
readingFile(mycontent);
console.log("Reading File...");
```

```
Reading File...                <--비동기방식이기 때문에, 먼저 출력됨.
var fs = require('fs');
var fcontens;
function readingFile(callback){
    fs.readFile("callback.js", "utf-8", function(error, contents){
        fcontents = contents;
        if(error) return console.error(error.stack);
        callback(contents);
    })
}
function mycontent(){
    console.log(fcontents);
}
readingFile(mycontent);
console.log("Reading File...");
```