Guide for

WEB FULLSTACK

workshop.benzen.io

BENZEN

안 내

본 자료는 벤젠(Benzen)이 서비스하는 웹서비스 풀스택 워크샵 (workshop.benzen.io)에서 제공하는 컨텐츠입니다. 무단 전재 및 복제를 금합니다.

2017년 06월 09일; 버전 0.9

저자 김동욱
서울대 컴퓨터공학 졸
벤젠 대표

kim@benzen.io
kim.benzen.io

목 차

1 기초 이론 ... 5

1.1 커리큘럼 소개 / 추상화 ... 6

1.2 컴퓨터 구조와 파일 ... 10

1.3 프로그램과 프로세스 ... 16

1.4 GUI/CLI, Shell, 파일 권한 ... 23

1.5 네트워크 ... 32

2 프로그래밍 연습 ... 43

2.1 프로그래밍 언어 ... 44

2.2 Node.js 설치 ... 48

2.3 기본 부품과 조합 ... 52

2.4 제어와 반복, 함수와 재귀, 에러 ... 58

2.5 명령형 프로그래밍, 스코프와 콜 스택 ... 69

2.6 객체지향 프로그래밍, 복사와 참조 ... 78

2.7 타입과 유추, 명명 규칙 ... 95

2.8 함수형 프로그래밍, 콜백과 클로저 ... 104

3 웹 프론트엔드 ... 115

3.1 웹 브라우저 ... 116

3.2 HTML ... 120

3.3 CSS ... 129

3.4 JavaScript ... 145

3.5 모델링 ... 158

3.6 이벤트 시스템 ... 172

3.7 jQuery ... 190

3.8 확장성있는 코드짜기 ... 195

4 웹 백엔드 ... 207

4.1 모듈, NPM ... 209

4.2 스트림, 표준입출력, 소켓 ... 218

4.3 HTTP 프로토콜 ... 224

4.4 웹 브라우저의 Request ... 231

4.5 정적 웹 서버의 Response ... 239

4.6 동적 웹 서버 ... 245

4.7 Express.js ... 254

4.8 쿠키와 세션, 인증 ... 262

4.9 동기와 비동기, Promise ... 272

4.10 Ajax, WebSocket ... 285

4.11 보안, Same Origin Policy ... 298

4.12 REST API, OAuth, SPA ... 306

5 데이터베이스 ... 319

5.1 메모리와 파일 ... 320

5.2 DB와 DBMS ... 327

5.3 MySQL과 SQL ... 338

5.4 Connector, SQL Injection, ORM ... 380

6 개발과 배포 ... 388

6.1 패키지 매니저, 자동화 도구 ... 389

6.2 버전 관리, Git, GitHub ... 406

6.3 호스팅, SSH, FTP ... *432*

6.4 DNS, 도메인, 메일 서버 (ing) ... *449*

6.5 암호화, 전자서명, 인증서와 SSL ... *451*

6.6 비밀번호 해싱 ... *470*

7 다른 플랫폼으로 ... *477*

7.1 다른 플랫폼들 (ing) ... *478*

7.2 GUI 프로그램 아키텍쳐, MVC 패턴 ... *480*

1 기초 이론

1. 커리큘럼 소개 / 추상화 ... 6

2. 컴퓨터 구조와 파일 ... 10

3. 프로그램과 프로세스 ... 16

4. GUI/CLI, Shell, 파일 권한 ... 23

5. 네트워크 ... 32

컴퓨터 구조, 운영체제, 파일, 프로그램과 프로세스, 네트워크(IP, 소켓, TCP/UDP, 프로토콜), GUI/CLI, Shell에 대해서 배웁니다. 발전 가능한 개발자가 되려면 언어는 단순한 도구라는 것을 이해하고, 특정 플랫폼에 종속되기보다 필요에 따라서 언어와 플랫폼을 선택 할 수 있어야 합니다.

1 기초 이론

1.1 커리큘럼 소개 / 추상화

1. 이론의 필요성
2. 추상화

> 수업의 진행 방향을 소개하고 IT의 핵심 원리인 추상화에 대해서 알아봅니다.

1. 이론의 필요성

<응용프로그램을 위한 거대한 피라미드>

피라미드를 전체를 볼 수는 있어야 하지만 꼭대기 층을 쌓아 올리기 위해서 아래 모든 층의 내부에 통달할 필요는 없습니다. 소프트웨어 개발에 필요한 상상력의 기반이 될 수 있는 토대를 공부합니다. 이론이 없는 코딩은 사상누각에 불과합니다.

1. 컴퓨터의 구조
2. 파일
3. 실행 파일(프로그램)과 프로세스
4. 운영체제
5. 네트워크

2. 추상화

<안드로이드 운영체제의 구조>

- 아키텍쳐

  x86, x64, ARM, MIPS, PowerPC, etc ...

- 운영체제

  Windows, Linux, OS X, iOS, Android, etc ...

- 플랫폼

  Java/Android, Swift/Objective-C, C/C++, .NET (c#), NodeJs, etc ...

- 프레임워크

  Java spring, Ruby on rails, PHP Laravel, NodeJs Express, etc ...

- 수 많은 라이브러리들

  오픈소스 ...

- 수 많은 언어들

  C/C++, C#, JavaScript, NodeJs, Java, Swift, Objective C, PHP, Ruby, Python, Lua, Go, ML, etc ...

- 네트워크 프로토콜

  HTTP/HTTPS, FTP, SMTP, SSH, DNS, .., TCP, UDP, IP, ..., ARP ..., Ethernet, WiFi, Bluetooth, etc ...

- 기업, 대학 등에서 전문화된 분야들

  인공지능, 데이터마이닝, AR/VR, 클라우드 컴퓨팅, 병렬처리, 그래픽스, 임베디드, 네트워크, 데이터베이스 etc ...

- 빠른 발전과 변화

추상화는 위처럼 복잡한 IT의 계층 구조에서 아랫층의 **복잡한 것을 단순하게 표현**하면서 윗층에서 반복적으로 재사용 할 수 있도록 하는 IT의 핵심적인 원리입니다.

1 기초 이론

1.2 컴퓨터 구조와 파일

1. 컴퓨터 구조
2. 파일
3. 파일 포맷
4. 텍스트 파일과 인코딩
5. 실행 파일

> 컴퓨터의 구조와 텍스트 파일과 인코딩, 실행 파일(프로그램)에 대해서 알아봅니다.

1. 컴퓨터 구조

<컴퓨터 구조>

- 중앙 처리 장치

  CPU

- 주 기억 장치

  RAM (메모리)

- 보조 기억 장치 (주변기기)

  HHD/SSD, USB, CD-ROM 등

- 기타 주변 기기

  카메라, 스피커, 마이크, 디스플레이, 키보드, 마우스, 터치패드, 프린터, 블루투스/와이파이 모듈, 랜카드 등

CPU는 기본적으로 메모리에 있는 명령들을 순차적으로 실행하는 역할을 담당합니다. 이 때의 명령은 단순한 산술 계산에서 부터 메모리나 보조기억장치 또는 주변 기기에 데이터를 읽고 쓰기(IO: Input Output)를 제어하기 까지 다양합니다. CPU는 사람, 메모리는 사람 앞의 책상, 보조 기억 장치는 책상 옆

에 책이 빼곡하게 꽂힌 책장으로 생각 할 수 있겠습니다.

2. 파일

기호	파일의 종류
d	디렉토리 (폴더)
p	FIFO (IPC 용도)
s	소켓 (IPC 용도)
b	블록 장치 (HDD, CD-ROM, 디스켓 등)
c	문자 장치 (프린터, USB, Serial 장치 등)
-	일반 파일

<UNIX 기반 운영체제의 파일 시스템>

컴퓨터에서 데이터는 보조 기억 장치에 파일이라는 형태로 저장됩니다. 일반적인 파일과 디렉토리부터 IPC (프로세스간 통신)을 위한 FIFO, 소켓 같은 특수한 파일, 그리고 주변 기기까지도 파일이라는 형태로 추상화 되어있습니다. 이렇게 컴퓨터가 기억하고 입/출력 하는 모든 종류의 정보는 결국 01010101.., 이진(binary) 데이터로 이루어져 있습니다. 컴퓨터의 운영체제는 파일 시스템을 설계하여 특수한 용도의 파일들을 정의하고 일반 파일들을 계층화하여 관리하고 있습니다.

3. 파일 포맷

우리가 접하는 일반 파일들에는 *.exe, .txt, .hwp, .doc, .xls, .html, .js, .java ..* 등 셀 수 없이 많은 확장자가 있습니다. 하지만 이러한 확장자는 단순히 파일명 뒤에 붙어 파일의 종류를 강조하는 용도일 뿐, 파일명의 일부분에 불과합니다. 때문에 확장자가 없어도 문제 없습니다. 또한 위에서 얘기한 것처럼 모든

파일은 결국 01010101..로 가득 할 뿐 그 자체로서는 의미를 갖지 못합니다. 파일에 기록된 **이진 데이터가 의미를 가질 때는 관련된 소프트웨어(프로그램)이 파일을 읽고 쓸 때**입니다. 이때 파일이 의미를 가지려면 이진 데이터를 **특정한 약속**에 맞추어 나열해야 합니다.

<어느 PDF 파일의 내용물>

이러한 특정한 약속을 파일 포맷이라고 합니다. 대표적인 파일 포맷으로는 **문서, 프로그램, 영상, 사운드, 이미지, 압축 파일** 등이 있겠습니다. 파일들을 열어(*텍스트 편집기가 아니라 **binary** 편집기로 열어서*) 내용물 010101..을 살펴보면 보통 맨 앞부분에 **파일 시그니쳐**(*수십-수백 자리의 고유한 **0101..***)를 나열하면서 시작됩니다. 그 뒤의 데이터는 그 파일 포맷의 규칙대로 의미를 갖는 0101..이 나열 될 것입니다. 무슨 파일이든 간에 해당 파일 포맷의 구체적인 약속을 모르고서야 단순한 0101..로 보일 수 밖에 없겠습니다.

4. 텍스트 파일과 인코딩

이때 소프트웨어들이 정하는 약속인 파일 포맷에 앞서서, 컴퓨터 초창기 때부터 0101..을 통해서 정수와 소수, 아주 큰 수 등의 **수 체계를 표현하는 약속**이 구현되었습니다. 나아가서 사람이 읽을 수 있는 문자를 표현하기 위해서 **문자열을 표현하기 위한 ASCII**라는 약속을 마련했습니다.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

<ASCII 표>

> ... 아스키는 컴퓨터와 통신 장비를 비롯한 문자를 사용하는 많은 장치에서 사용되며, 대부분의 문자 인코딩이 아스키에 기초를 두고 있다. 아스키는 1967년에 표준으로 제정되어 1986년에 마지막으로 개정되었다. 아스키는 7비트 인코딩으로, 33개의 출력 불가능한 제어 문자들과 공백을 비롯한 95개의 출력 가능한 문자들로 이루어진다. 제어 문자들은 역사적인 이유로 남아 있으며 대부분은 더 이상 사용되지 않는다. 출력 가능한 문자들은 52개의 영문 알파벳 대소문자와, 10개의 숫자, 32개의 특수 문자, 그리고 하나의 공백 문자로 이루어진다. ... (위키백과)

위의 ASCII 처럼 수와 문자열을 1대 1로 맵핑한 것을 인코딩이라고 합니다. 기본적인 ASCII에는 0부터 127까지의 수에 128개의 문자가 각각 맵핑되어있습니다. ASCII는 이렇게 1byte 인코딩으로 그 중에서 7bit만으로 문자 128개를 표현하고 있습니다. 나아가 ANSI라는 인코딩은 ASCII 인코딩에서 남은 1bit를 활용해 128부터 255까지의 수를 다른 언어로 확장한 1byte 인코딩입니다. 예로 ANSI-949는 남은 128~255를 한글에 맵핑한 인코딩입니다. 나아가 전세계의 다양한 언어권의 문자를 인코딩하기 위해서 Unicode라는 표준이 자리를 잡았습니다.

> 유니코드(Unicode)는 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다룰 수 있도록 설계된 산업 표준이며, 유니코드 협회(Unicode Consortium)가 제정한다. 이 표준에는 ISO 10646 문자 집합, 문자 인코딩, 문자 정보 데이터베이스, 문자들을 다루기 위한 알고리즘 등을 포함하고 있다. ... (위키백과)

유니코드에는 여러 인코딩이 있으며 그 중 대표적인 UTF-8은 전 세계의 문자를 1byte에서 4byte까지의 가변 길이로 표현합니다. 브라우저나 텍스트 편집기에서 문자, 특히 한글이 깨지는 경우는 저장 할 때(서버에서 웹페이지를 줄 때)의 인코딩과 읽을 때(브라우저에서 웹페이지를 해석 할 때)의 인코딩이 서로 일치하지 않는 경우입니다. 서로 다른 약속 체계로 소통하려하니 문제가 되는 것입니다.

5. 실행 파일

<윈도우즈의 PE 파일 포맷>

프로그램 혹은 실행 파일(executable)도 일반 파일입니다. 프로그램은 운영체제에 종속되어서 실행되기 때문에 윈도우즈의 실행 파일(.exe 파일 등)은 PE라는 파일 포맷을, 유닉스 기반 운영체제들은 ELF라는 파일 포맷을 또 MacOS의 경우는 Mach-O라는 파일 포맷을 따르고 있습니다. 윈도우즈에서 실행되는 파일이 리눅스에서 실행되지 않는 이유는 이처럼 실행파일들이 운영체제에 종속적인 파일 포맷을 따르고 있기 때문입니다. 프로그램의 구조와 실행 원리에 대해서는 다음 장에서 다룹니다.

[1] CPU, Central Processing Unit
[2] RAM, Random Access Memory
[3] Auxiliary Memory
[4] Peripheral Device
[5] IO, Input Output
[6] IPC, Inter Process Communication
[7] ASCII, American Standard Code for Information Interchange
[8] ANSI, American National Standards Institute
[9] Unicode

1 기초 이론

1.3 프로그램과 프로세스

1. 운영체제
2. 프로그램
3. 프로세스
4. 요약

> 아키텍쳐, 컴파일러, 프로그램과 프로세스, 운영체제에 대해서 알아봅니다.

1. 운영체제

<리눅스 운영체제의 부팅 과정>

운영체제, OS란 윈도우즈, 리눅스, 안드로이드, iOS, MacOS 같은 일종의 거대한 프로그램입니다. 이러한 프로그램을 응용프로그램과 구분하여 시스템프로그램이라고 말하기도 합니다. 컴퓨터에 전원을 넣으면 메인보드에 저장된 BIOS라는 프로그램이 HDD/SSD/USB 등의 보조기억장치에서 부트로더라는 프로그램을 찾아 실행합니다. 이때 부트로더는 운영체제를 실행하고.. 디스플레이에 예쁜 UI가 뜨게 됩니다. 운영체제가 하는 주요한 일은 다음과 같습니다.

1. 컴퓨터의 메인보드에 접속된 하드웨어들을 상호 연결하고 제어함
2. 응용프로그램이 사용 할 수 있는 서비스/라이브러리들을 제공
    1. 보조 기억장치 및 주변기기를 추상화한 파일시스템을 조작 할 수 있도록 응용프로그램에 API를 제공
    2. 네트워킹을 추상화하여 응용프로그램에 API를 제공
    3. 공통적인 GUI를 쉽게 만들 수 있게 추상화하여 응용프로그램에 API를 제공
3. 응용프로그램의 실행, 멀티태스킹을 위한 스케쥴링
4. 컴퓨터의 보안과 사용자 계정을 관리

2. 프로그램

Assembly vs. machine code

Machine code bytes	Assembly language statements
	foo:
B8 22 11 00 FF	movl $0xFF001122, %eax
01 CA	addl %ecx, %edx
31 F6	xorl %esi, %esi
53	pushl %ebx
8B 5C 24 04	movl 4(%esp), %ebx
8D 34 48	leal (%eax,%ecx,2), %esi
39 C3	cmpl %eax, %ebx
72 EB	jnae foo
C3	retl

Instruction stream

```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

프로그래밍 언어가 등장하기 이전, 컴퓨터 발전의 초창기에는 CPU의 기능들(산술 연산 및 입출력 등)을 숫자 하나 하나에 맵핑하여두고 이러한 숫자들, 명령어 또는 기계어 (Instruction)를 목적에 맞게 순서대로 나열한 것이 프로그램이었습니다. 하지만 이진수로 코딩을 하는게 쉽지 않기 때문에, 명령어들에 문자열을 부여하여 조금 더 사람이 이해하기 쉽게 한 것을 어셈블리어라고 합니다.

<기계어에서 High Level Language까지>

```c
#include<stdio.h>

int no_value_glob_int;
int init_glob_int = 5;

char *string ="This is a string in .data section!";
char *dynamic_val;

main()
{
    int local_int = 5;
    int no_value_local_int;
    dynamic_val = (char*)malloc(24);
    strcpy(dynamic_val, "U.U.U Workshop");

    return 0;
}
```

<C언어로 작성한 소스코드>

추상화는 IT의 모든 곳을 관통하고 있습니다. 어셈블리어에서 한단계 더 나아가 High Level Language라고 불리는 사람이 이해하기 쉬운 C언어 같은 프로그래밍 언어들이 등장합니다. 이때 프로그래밍 언어로 작성된 소스코드는 단순한 텍스트 파일입니다. 따라서 소스코드는 특정 인코딩을 따라 문자들을 나열한 것에 불과합니다.

<컴파일 과정>

이 단순한 텍스트를 다시 기계어로 표현해야만 CPU가 이해 할 수 있는 프로그램이 됩니다. 이렇게 소스코드를 다시 기계어로 번역하는 과정을 컴파일이라고 합니다. 그리고 이러한 번역을 담당하는 프로그램을 컴파일러라고 합니다. 물론 프로그래밍 언어에 따라 저마다의 컴파일러가 필요합니다.

```
080482e0 <_start>:
 80482e0:       31 ed                   xor    %ebp,%ebp
 80482e2:       5e                      pop    %esi
 80482e3:       89 e1                   mov    %esp,%ecx
 80482e5:       83 e4 f0                and    $0xfffffff0,%esp
 80482e8:       50                      push   %eax
 80482e9:       54                      push   %esp
 80482ea:       52                      push   %edx
 80482eb:       68 04 84 04 08          push   $0x8048404
 80482f0:       68 74 82 04 08          push   $0x8048274
 80482f5:       51                      push   %ecx
 80482f6:       56                      push   %esi
 80482f7:       68 c4 83 04 08          push   $0x80483c4
 80482fc:       e8 cb ff ff ff          call   80482cc <__libc_start_main@plt>
 8048301:       f4                      hlt
 8048302:       90                      nop
 8048303:       90                      nop
```

<위의 C언어 소스코드를 번역한 (좌)기계어 (우)어셈블리>

이 때 생각 해볼 점은 CPU를 생산하는 기업이나 제품이 한가지가 아니라는 점입니다. Intel은 x86, x64 등의 아키텍쳐를 설계하여 CPU를 생산하고 있습니다. 또 모바일에서 강세를 보이는 ARM CPU 제품군은 ARM이라는 아키텍쳐를 쓰고 있습니다. 등등.. 이렇게 **프로그램은 CPU의 아키텍쳐에 종속**됩니다.

> **명령어 집합(영어: instruction set) 또는 명령어 집합 구조(영어: Instruction set architecture, ISA)**는 마이크로프로세서가 인식해서 기능을 이해하고 실행할 수 있는 기계어 명령어를 말한다. 마이크로프로세서마다 기계어 코드의 길이와 숫자 코드가 다르다. **명령어의 각 비트는 기능적으로 분할하여 의미를 부여하고 숫자화한다.** 프로그램 개발자가 숫자로 프로그램하기가 불편하므로 기계어와 일대일로 문자화한 것이 어셈블리어이다. ... (위키백과)

<프로그램은 운영체제에 종속>

프로그램이 CPU의 아키텍쳐에 종속된다. 여기서 나아가 또 하나 생각해볼 점은 프로그램을 메모리에 복사하면서 로드하고, 프로그램에 필요한 공유 라이브러리를 로드하고, 멀티태스킹을 위해 CPU와 메모리 등의 자원을 분배하는 등의 다양한 일들은 운영체제가 담당하고 있습니다. 따라서 운영체제들은 프로그램의 실행을 위해 저마다의 파일 포맷을 정해 놓고 있습니다. 여기서 또 **프로그램은 운영체제에 종속**됩니다. 리눅스에서 컴파일한 프로그램은 윈도우즈에서 실행되지 않습니다. 하지만 리눅스에서 작성한 소스코드를 윈도우즈에서 컴파일한다면 잘 작동합니다.

3. 프로세스

<리눅스 프로세스의 구조>

보조기억장치에 저장된 프로그램이 운영체제를 통해 실행되면 프로그램의 기계어들과 데이터 등이 메모리상에 복사되면서 CPU의 처리 대상이 됩니다. 이렇게 숨결이 들어간 프로그램을 프로세스(Process)라고 부릅니다. 프로그램을 개발하는 과정에선 프로세스의 메모리에 이 순간엔 무엇이 있고 또 다음 순간엔 무슨 일어날지를 상상하면서 코드를 작성하게 됩니다. 메모리상에 프로세스의 구조에 대해 간단히 알아 보겠습니다.

- text

  기계어로 표현된 프로그램의 로직들

- data

  프로그램을 작성할 때 지정된 값들

- heap

  런타임(Runtime)에 동적으로 생성되는 값들

- stack

  함수라고 불리는 프로그램의 반복적인 로직이 실행되고 소멸되는 공간

4. 요약

- 아키텍쳐, 명령어 집합

  CPU가 처리 할 수 있는 단순한 명령어(기계어)들의 집합으로 하드웨어 수준에서 설계 됨

- 소스코드

  사람이 이해하기 쉬운 프로그래밍 언어로 프로그램의 작동을 묘사한 텍스트 파일

- 컴파일, 컴파일러

  소스코드를 기계어로 번역하는 과정, 그리고 번역을 담당하는 프로그램

- 프로그램

  운영체제와 CPU에 종속적인 실행 가능한 파일

- 프로세스

  운영체제가 메모리에 프로그램을 복사하여 CPU가 처리중인 (실행중인) 프로그램

[1] OS, Operating System
[2] BIOS, Basic Input Output System
[3] Boot Loader
[4] API, Application Programming Interface
[5] Scheduling
[6] Compile
[7] Process
[8] Runtime

1 기초 이론

1.4 GUI/CLI, Shell, 파일 권한

1. GUI
2. CLI
3. Shell
4. Shell 명령어
5. 환경 변수
6. 파일 권한

> GUI와 CLI로 불리는 UI에 대해서 알아보고, Shell 명령어와 파일 권한에 대해 알아봅니다.

1. GUI

<Windows의 explorer.exe>

GUI (Graphical User Interface)는 그래픽을 통해 사용자와 프로그램이 상호작용하는 환경을 일컫는 말입니다. GUI의 제어는 마우스, 터치패드, 디스플레이 등 다양한 주변기기와의 입출력(IO)을 통해 이루어집니다. 최종 사용자(End User)가 이용하는 운영체제나 응용프로그램은 대부분 GUI를 통해 상호작용합니다. 대표적인 프로그램을 예로 들어보면 윈도우즈의 explorer.exe (바탕화면, 탐색기, 작업표시줄 등을 제공), 웹 브라우저, 게임 등이 있습니다.

<MacOS의 Finder>

이때 생각해 볼 점은 운영체제 별로 응용프로그램들의 GUI가 비슷하다는 점입니다. 윈도우즈 응용프로그램의 GUI는 서로 비슷한 모습이고, MacOS 응용프로그램의 GUI도 서로 비슷한 모습이고, Android, iOS 등 운영체제에 종속적인 응용프로그램들의 GUI는 서로 비슷한 모습을 하고 있습니다. 이는 운영체

제가 프로그램 개발자가 운영체제의 테마에 맞는 공통적인 GUI를 만들 수 있도록 관련 API를 제공하고 응용 프로그램은 이를 활용하여 개발되었기 때문입니다.

2. CLI

<MacOS의 Terminal>

CLI (Command Line Interface)는 텍스트 입출력(IO)을 통해 사용자와 프로그램이 상호작용하는 환경을 일컫는 말입니다. 키보드 입력만으로 운영체제와 프로그램을 제어해야 하기에 최종 사용자나 초보자에게는 익숙하지 않은 환경입니다. 개발자가 CLI 환경에 익숙해져야 하는 이유는 개발자용 프로그램이나 최종 사용자가 이용하지 않는 내부적인 시스템 프로그램은 대부분 CLI를 통해 제어되기 때문입니다. 또한 CLI는 키보드 입력만으로 운영체제와 프로그램을 제어하므로 익숙해지기만 한다면 주로 마우스를 이용하는 GUI 보다 더 좋은 생산성을 낼 수도 있습니다.

<CLI와 GUI는 인터페이스의 차이일 뿐>

프로그램은 이용 대상과 목적에 따라서 GUI를 제공 할 수도, CLI를 제공 할 수도 있습니다. 또는 두 방식의 인터페이스를 모두 제공 할 수도, 또는 아무런 인터페이스를 제공하지 않을 수도 있습니다. 둘의 차이는 단순히 상호작용 방식의 차이 일뿐 근본적인 프로그램의 구조나 실행 원리는 동일합니다.

3. Shell

<Git Bash>

운영체제의 기능을 이용하고 또 프로그램을 실행하기 위한 프로그램을 셸(Shell)이라고 합니다. 셸은 CLI/GUI 두가지 방식으로 제공 됩니다. 윈도우즈의 경우 GUI 셸은 explorer.exe (바탕화면이나 탐색기)로 볼 수 있고 CLI 셸은 cmd.exe (명령프롬프트)로 볼 수 있습니다. 제어판에서 운영체제의 설정을 변경하고 탐색기나 바탕화면에서 폴더나 파일을 만들고 프로그램을 실행하는 등 GUI 셸은 직관적으로 사

용하기에 좋습니다. 프로그래밍을 본격적으로 시작하기에 앞서서 CLI 셸에 익숙해 지도록 하겠습니다. 이때 윈도우즈 운영체제를 쓰시는 분들은 UNIX 기반 OS에 익숙해직 위해서 윈도우즈에서 UNIX 기반 OS의 셸(bash, sh 등)을 에뮬레이트하는(흉내내는) Git Bash라는 프로그램을 설치하고 사용하시면 좋겠습니다. UNIX 기반 OS에 익숙해 지려는 이유는 대부분의 서버 환경이나 오픈 소스 개발 진영이 UNIX 기반 OS 위에서 구축되고 있기 때문입니다.

[Git Bash 설치] https://git-for-windows.github.io (https://git-for-windows.github.io/)

4. Shell 명령어

프로그램 이름	용도	비고
pwd	현재 작업 중인 디렉토리의 경로를 출력	present working directory
ls -al	현재 작업 중인 폴더에 존재하는 파일들의 이름을 출력	list
cd 폴더명	현재 위치를 이동	change directory
clear	셸 화면을 청소	
mkdir 폴더명 [폴더명2, 폴더명3, ...]	폴더를 생성	make directory
touch 파일명 [파일명2, 파일명3, ...]	빈 파일을 생성	
rm -rf 파일명 [파일명2, 파일명3, ...]	파일 및 폴더를 삭제	remove
mv 파일명 [파일명2, 파일명3, ..] 목적 파일명	파일 및 폴더를 옮기거나 이름을 변경	move
cp 파일명 [파일명2, 파일명3, ..] 목적 파일명	파일 및 폴더를 복사	copy
cat -n 파일명 [파일명2, 파일명3, ..]	파일의 내용을 텍스트로 읽어서 출력	concatenate

<자주 사용할 프로그램들>

프로그램의 실행

Git Bash나 Terminal 종류의 CLI 셸을 이용해 Linux/Unix 환경의 셸의 대표적인 프로그램들 대해 알아보겠습니다. 셸(shell)에서 프로그램을 실행시키는 방법은 일반적으로 파일명 [-추가 옵션] [실행시 전달할 인자들] 입니다.

경로

셸에서 디렉토리를 돌아다니면서 작업을 하기에 앞서서 경로에 대한 이해가 필요합니다. 경로는 현재 디렉토리를 기준으로 표현 할 수도 있고(상대 경로), 최상위 디렉토리(root)를 기준으로 표현(절대 경로) 할 수도 있습니다. 최상위 디렉토리는 윈도우즈의 경우에는 C:\, D:\ 등이 될 수 있고 UNIX 기반 OS에서는 / 가 됩니다. 절대 경로는 /Some/Path/From/Root 와 같은 식으로 최상위 디렉토리에서 부터 파일 및 폴더의 경로를 표현합니다. 상대 경로는 . 으로 대변되는 현재 위치에서 출발하여 ./Some/path/../From/Here와 같이 경로를 표현합니다. . 은 현재 위치를 대변하고, ..은 현재 위치의 상위 디렉토리를 대변합니다. 추가로 ~는 운영체제에서 지정해준 사용자의 홈 디렉토리를 대변합니다.

자동 완성과 히스토리

셸에서 명령어를 어느 정도 입력하다가 탭키를 누르면 자동완성을 도와주는 경우가 많습니다. 또한 ↑와 ↓를 이용해서 입력했던 명령어들을 다시 불러올 수 있습니다. 탭과 화살표를 잘 쓰면 생산성을 높힐 수 있습니다.

연습

```
pwd
ls
mkdir test_folder
touch test_file
ls
ls -al
rm test_file
rm -r test_folder
ls
cd ..
ls
cd ~
touch a
ls
touch b c
cp c d
mv d e
ls
clear
```

5. 환경 변수

환경 변수 또는 시스템 변수는 운영체제에 등록된 일련의 값들입니다. 이런 값들은 프로세스에서 공통적으로 이용됩니다. 대표적으로 $PATH와 같은 환경 변수는 셸 프로그램이 실행할 프로그램을 찾는데 도움을 줍니다. ls와 같은 프로그램은 실제로 /bin/ls에 저장되어 있지만 셸에서 프로그램을 실행 할 때 현재 경로에 프로그램이 없으면 $PATH에 지정된 경로에서 프로그램을 차례대로 찾아보기 때문에 ls를 입력하더라도 /bin/ls를 실행 할 수 있습니다. 제 컴퓨터의 $PATH는 다음과 같습니다.

```
echo $PATH
/Users/dehypnosis/.rbenv/shims:/usr/local/apache-ant-1.9.4/bin/:/Users/dehypnosis/Li
```

6. 파일 권한

운영체제의 파일 시스템에는 파일 권한(File Permission)이라는 요소가 있습니다. 운영체제는 각 파일과 디렉토리에 설정된 권한을 기반으로 같은 머신을 이용하는 사용자들의 접근 권한을 제어합니다. 이 파일 권한은 단일 머신에서 사용자간의 파일 영역을 분리해주거나, 외부에서 업로드된 파일들의 실행 권한에 대한 제어하는 등 보안 이슈와 직결되어 있습니다.

```
dehypnosis-mac:~ dehypnosis$ ls -al
total 792
drwxr-xr-x+   96 dehypnosis  staff    3264 Mar  6 13:14 .
drwxr-xr-x     5 root        admin     170 Oct  6 18:37 ..
drwxr-xr-x     3 dehypnosis  staff     102 Aug  3  2016 .AndroidStudio2.1
drwxr-xr-x     6 dehypnosis  staff     204 Dec  1 15:15 .AnySign4PC
-r--------     1 dehypnosis  staff       7 Oct  6 18:45 .CFUserTextEncoding
-rw-r--r--@    1 dehypnosis  staff   18436 Mar  3 12:36 .DS_Store
drwxr-xr-x     3 dehypnosis  staff     102 Mar 28  2015 .IdeaIC14
drwxr-xr-x     3 dehypnosis  staff     102 Apr  6  2015 .IntelliJIdea14
drwx------     3 dehypnosis  staff     102 Feb 11  2015 .MacOSX
drwx------   328 dehypnosis  staff   11152 Mar  4 22:17 .Trash
drwxr-xr-x     4 dehypnosis  staff     136 Apr  8  2015 .Trash-1000
drwxr-xr-x     3 dehypnosis  staff     102 Feb 13  2015 .WebStorm9
-rw-------     1 dehypnosis  staff     259 Oct 11 22:52 .Xauthority
drwxr-xr-x     4 dehypnosis  staff     136 Dec 31  2014 .adobe
drwxr-xr-x    20 dehypnosis  staff     680 Aug  4  2016 .android
drwxr-xr-x    14 dehypnosis  staff     476 Oct 17 14:06 .atom
drwxr-xr-x     4 dehypnosis  staff     136 Jan 18  2016 .avidemux6
-rw-r--r--     1 dehypnosis  staff  140925 Dec  7 17:22 .babel.json
...
```

파일 권한은 일반 파일에 대해서 읽기(read), 쓰기/삭제(write), 실행(execute) 디렉토리에 대해서 하위 파일 읽기(read), 하위 파일 쓰기/삭제(write), 하위 파일 실행 및 디렉토리 접근(execute) 권한으로 이루어집니다.

여기에 더해서 유닉스 시스템에서 사용자들은 특정 그룹에 속해있기 때문에, 모든 파일은 파일 소유자(user), 파일 소유자와 같은 그룹(group), 이외(other)의 세 부류에 대해서 rwx의 세가지 권한, 총 9가지의 권한 설정을 갖습니다.

파일 권한은 3자리의 8진수로 표기 할 수 있습니다. 첫째 자리 숫자는 파일 소유자의 권한, 두번째 자리 숫자는 소유자와 같은 그룹의 권한, 세번째 자리는 나머지 유저에 대한 권한입니다. 각 숫자는 read(4), write(2), execute(1)의 합으로 표기됩니다. 예를 들어 7은 r+w+x, 5는 r+x, 1은 x, 0은 권한 없음을 의미합니다. 쉘에서 파일 권한을 변경하는 방법은 다음과 같습니다.

```
dehypnosis-mac:~ dehypnosis$ touch w x y z
dehypnosis-mac:~ dehypnosis$ ls -l
-rw-r--r--   1 dehypnosis  staff     0 Mar  6 13:28 w
-rw-r--r--   1 dehypnosis  staff     0 Mar  6 13:28 x
-rw-r--r--   1 dehypnosis  staff     0 Mar  6 13:28 y
-rw-r--r--   1 dehypnosis  staff     0 Mar  6 13:28 z

dehypnosis-mac:~ dehypnosis$ chmod 700 w
dehypnosis-mac:~ dehypnosis$ chmod 500 x
dehypnosis-mac:~ dehypnosis$ chmod 300 y
dehypnosis-mac:~ dehypnosis$ chmod 100 z
dehypnosis-mac:~ dehypnosis$ ls -l
-rwx------   1 dehypnosis  staff     0 Mar  6 13:28 w
-r-x------   1 dehypnosis  staff     0 Mar  6 13:28 x
--wx------   1 dehypnosis  staff     0 Mar  6 13:28 y
---x------   1 dehypnosis  staff     0 Mar  6 13:28 z

dehypnosis-mac:~ dehypnosis$ chmod 755 x y z w
dehypnosis-mac:~ dehypnosis$ ls -l
-rwxr-xr-x   1 dehypnosis  staff     0 Mar  6 13:33 w
-rwxr-xr-x   1 dehypnosis  staff     0 Mar  6 13:28 x
-rwxr-xr-x   1 dehypnosis  staff     0 Mar  6 13:28 y
-rwxr-xr-x   1 dehypnosis  staff     0 Mar  6 13:28 z
```

마지막으로 특정 프로그램을 최고 관리자(root) 권한으로 실행하는 명령어를 기억해둡니다. 일반 유저의 권한으로 각종 프로그램의 설치나 시스템 설정의 변경이 불가능 할 수 있습니다. 이 때는 sudo 명령어로 임시적으로 최고 관리자 권한을 획득하여 프로그램을 실행합니다.

```
dehypnosis-mac:~ dehypnosis$ touch /var/log/someLogFile
touch: /var/log/someLogFile: Permission denied
dehypnosis-mac:~ dehypnosis$ sudo touch /var/log/someLogFile
dehypnosis-mac:~ dehypnosis$ ls /var/log/someLogFile
/var/log/someLogFile
```

[1] GUI, Graphical User Interface

[2] End User

[3] CLI, Command Line Interface

[4] Shell

[5] File Permission

[6] sudo, substitute user do

1 기초 이론

1.5 네트워크

1. 네트워크의 구분
2. 통신의 원리
3. MAC 주소와 IP 주소
4. 사설망과 NAT
5. TCP/UDP와 포트
6. 응용 프로토콜
7. 서버-클라이언트, P2P

> 네트워크 계층, Ethernet, WiFi, IP, TCP,UDP, 프로토콜, 포트, 공인망과 사설망, NAT 등 네트워크 이론에 대해서 알아봅니다.

1. 네트워크의 구분

<LAN과 라우터>

웹 브라우저로 구글이나 페이스북 같은 웹페이지를 보는 것, 카카오톡으로 메세지를 주고 받는 것, 실시간 온라인 게임 등. 다양한 컴퓨터 통신의 기저에 있는 원리를 알아보고 다시 우리가 알아야 할 높은 층에 집중하겠습니다. 네트워크는 인터넷서비스제공업체(ISP)들, 또는 공기관, 사설기관의 유/무선망 및 중계 컴퓨터들을 통해서 전세계에 구축되어 있습니다.

- LAN

  Local Area Network: 집, 회사, 지역 등의 지엽적인 네트워크를 지칭

- WAN

  Wide Area Network: LAN 보다 큰 규모로 국가, 대륙를 아우르는 원거리 망을 지칭

<Intranet, Internet>

- Intranet

  집, 회사 등 폐쇄적으로 운영되는 네트워크

- Internet

  전세계에서 접근이 가능한 공개적인 네트워크

2. 통신의 원리

통신은 프로세스간에

통신은 호스트(컴퓨터)간에 이루어집니다. 맞는 말이지만 좀 더 정확하게는 통신은 프로세스 간에 이루어집니다. 호스트를 건물으로, 프로세스를 건물에 거주하는 사람이라고 생각하면 건물과 건물끼리 편지를 주고 받기보다는 집에사는 누군가 끼리 편지를 주고 받는 것과 동일합니다. 카카오톡 클라이언트 프로세스와 카카오사의 서버 프로세스와의 통신, 크롬 웹 브라우저와 구글사의 서버 프로세스와의 통신 등. 이처럼 통신은 프로세스간에 이루어집니다.

OSI 7 Layer

<OSI 7 Layer>

추상화의 원리는 네트워크에도 적용됩니다. 프로세스간의 메세지를 편지 봉투에 넣어고, 받는이와 보낸이를 적고, 네트워크 망을 따라 편지를 배달하는 일은 모두 단계별로 분리되어 있습니다.

- 1. Message

  프로세스가 전달하고자 하는 데이터

- 2. Segment/Datagram

  보내는/받는 프로세스를 식별하기 위해 부가 데이터(Port)를 추가

- 3. Packet

  보내는/받는 호스트를 식별하기 위해 부가 데이터(IP)를 추가

- 4. Frame

  실제로 데이터를 보내는/받는 장치인 유선랜(Ethernet) 카드, WiFi 및 Bluetooth 카드 등의 장치를 식별하기 위해 부가적인 데이터(MAC Address)를 추가

3. MAC 주소와 IP 주소

<Frame/Packet Header>

전송 매체(유선, 무선)를 따라서 데이터(Frame)를 송수신하는 장치에는 48bit의 고유한 물리적 주소인 MAC Address (Media Access Control Address)가 공장에서 제조 될 때 부터 할당되어 있습니다. 반면에 우리가 흔히 얘기하는 IP 주소는 일종의 가상 주소이며 불변하는 값이 아닙니다. MAC Address가 있음에도 불구하고 IP라는 주소 체계가 도입된 이유는 라우팅의 편리함에 있습니다. 라우팅(Routing)이란 데이터의 송신부터 수신에 이르는 배달 과정을 나타내는 말입니다. 비유를 하자면 MAC 주소로 라우팅을 하는 것은 주민등록번호로 편지를 보내는 것과 비슷합니다. 반면에 IP 주소는 우체국(전세계의 네트워크에 존재하는 수 많은 라우터들)에서 편지를 배달하기 좋도록 설계된 주소 체계라는 정도로 이해하고 넘어가겠습니다.

추가로 IP 주소는 하나의 호스트를 나타내는 주소지만 MAC 주소는 호스트에 연결된 물리적 통신 장치 하나를 나타내는 주소입니다. 예를 들어 유선랜(Ethernet)을 통해 인터넷에 연결하고 WiFi를 통해 인터넷을 공유하고 있는 상황을 생각해보면, 두 네트워크 장치는 각자의 MAC 주소로 통신을 하면서도 같은 IP 주소로 통신을 하게 될 것입니다.

4. 사설망과 NAT

IP 주소는 한정적인 자원

IPv4는 IP 주소 체계의 4번째 버전을 의미합니다. IPv4는 32bit의 길이로 설계 되었으며 따라서 2^32(약 43억)개의 주소를 갖습니다. 보통 사람이 읽기 쉽도록 8bit씩 4자리로 끊어서 10진수로 표기합니다. 즉 IPv4는 0.0.0.0 ~ 255.255.255.255의 범위를 갖습니다. 인터넷의 발전 속도에 비해서 43억

개의 주소가 부족한 숫자였나봅니다. 2015년 말에 IP 주소를 할당하는 최상위 기관 IANA에서 IPv4 주소의 고갈을 공식적으로 발표하였습니다. 지금은 128bit의 길이를 갖는 IPv6를 도입하고 IPv4와의 호환성 문제 등을 해결하며 IP 주소 체계의 전환을 앞두고 있습니다.

용도	대역	비고
사설망	10.0.0.0 ~ 10.255.255.255	
사설망	172.16.0.0 ~ 172.31.255.255	
사설망	192.168.0.0 ~ 192.168.255.255	
호스트 자기 자신 (localhost)	127.0.0.1	localhost
호스트에 접근 가능한 모든 아이피 (사설망, 공인망 모두)	0.0.0.0	
브로드캐스팅 (모두에게)	255.255.255.255	

<특수 IP 대역>

공인망과 사설망

인터넷이 발달하면서 IP 주소가 점차 부족해져 IP 주소의 할당이 힘들어졌습니다. 또한 IP 주소의 구입은 국가, 기관, 기업 수준에서 이루어지기 때문에 개인이나 민간 수준에서 IP 주소를 마련하고 네트워크를 구축해 인터넷에 연결하기도 힘들었습니다. 이에 IP 주소의 재사용을 위해서 사설망(Private Network)이라는 개념이 생겼습니다. 위의 표처럼 특정한 IP 대역의 주소는 공인망(Public Network)에 할당하지 않고 그 어떤 사설망에서라도 내부적으로 사용 할 수 있도록 정해두었습니다. 집에서 공유기를 쓰신다면 사설망을 구축하신 것입니다.

<NAT (Network Address Translation)>

NAT (Network Address Translation)

사설망 대역의 IP 주소는 수 많은 곳에서 재사용을 하기 때문에, 정작 고유한 호스트를 구분하기 위한 IP 주소 본연의 역할을 하지 못합니다. 때문에 사설망을 인터넷에 연결하기 위해서는 NAT (Network Address Translation)라는 기술이 필요합니다. 사설망을 벗어나 네트워크를 공인망(네트워크)에 연결하는 장치을 허브/게이트웨이/라우터/공유기 등으로 부릅니다. 이때 이 게이트웨이에는 공인 IP 주소가 할당되어 있고 이를 바탕으로 게이트웨이가 사설망의 여러 호스트들을 대표해서 본인의 IP 주소로 공인망에서 데이터를 송수신하게 됩니다. 이러한 기술을 NAT라고 합니다.

5. TCP/UDP와 포트

<Segment/Datagram Header>

네트워크가 발전하면서 굳어진 약속 체계들을 프로토콜(Protocol)이라고 합니다. 지금까지 Ethernet/Wifi/../IP 등의 저수준 프로토콜에 대해 알아보았습니다. 네트워크 상의 데이터는 IP를 통해서 자신이 출발한, 도착해야 할 호스트를 구분 할 수 있습니다. 하지만 실제 통신은 프로세스간에 일어나기 때문에 호스트에 도착한 데이터는 운영체제에서 실행되는 수 많은 프로세스 중에 특정 프로세스를 구분 할 필요가 있습니다. 이를 위해서 포트(Port)라는 개념이 도입되었습니다. 프로세스는 운영체제에게

특정한 포트를 사용하겠다고 요청하고 2^16(약 6만)개의 숫자 중에서 한개의 숫자를 할당 받습니다. 이를 포트라고 합니다. 프로세스는 고유한 포트번호, 호스트의 IP 주소, 장치의 MAC Address를 통해서 네트워킹하게 됩니다.

<TCP/UDP>

IP(Internet Protocol) 위에 TCP와 UDP라는 두가지 프로토콜이 있습니다. TCP/UDP는 모두 포트번호를 통해 프로세스를 구분하게 됩니다. 이 둘의 차이는 신뢰성있는 프로토콜(TCP)과 신뢰성 없는 프로토콜(UDP)입니다. TCP의 경우에는 보낸 데이터가 잘 도착했는지를 확인하고 오류시 이를 복구하는 등의 로직을 프로토콜 수준에서 구현하고 있습니다. UDP의 경우에는 단순히 데이터에 송/수신 프로세스의 포트 번호를 부착하는 프로토콜입니다. UDP의 경우는 데이터의 신뢰성보다 속도가 더 중요한 경우(미디어 스트리밍, 게임 등)에 쓰이곤 합니다. 이외의 경우는 보통 TCP를 사용하기 때문에 TCP/IP라는 단어는 인터넷 프로토콜을 의미하는 대표적인 단어가 되었습니다.

6. 응용 프로토콜

먼 길을 걸어 TCP/UDP 까지 도착했습니다. TCP/UDP 위가 최종적인 응용 프로토콜, 즉 우리가 프로그램을 작성하는 위치가 되겠습니다. 우리는 프로그램을 작성하면서 TCP나 UDP 둘 중 하나를 선택해 네트워킹을 설계 할 수 있습니다. 물론 그 주고 받는 데이터는 바이너리가 될 수도 텍스트가 될 수도 있습니다.

프로토콜	포트번호(서버)	용도
HTTP	80	웹 문서 전송
FTP	21	파일 전송
SSH	22	원격 셸
DNS	53	도메인 네임 서버
POP3, SMTP, HTTPS, etc ...		

<널리 쓰이는 응용 프로토콜>

이 때 우리가 단순히 독립적으로 쓰이는 프로그램을 만든다면 자신만의 프로토콜을 만들어도 무관합니다. 우리가 만든 프로그램들 사이에서만 서로 이해 할 수 있는 약속이면 충분하기 때문입니다. 하지만 그와 달리 인터넷이 발달하면서 전 세계적으로 쓰여온 프로토콜들을 쓰게되는 경우가 있습니다. 예를 들어 웹 서버나 웹 브라우저 프로그램을 만든다면 웹문서 전송 프로토콜인 HTTP라는 약속에 대해 공부 할 필요가 있습니다. 이처럼 널리 쓰이는 프로토콜들은 0~1023의 포트(Well-known Port)를 선점하고 있는데 운영체제로부터 0~1023 대역의 포트를 할당 받으려면 관리자 권한이 필요합니다.

7. 서버-클라이언트, P2P

<Server-Client / P2P>

마지막으로 프로세스간 네트워킹의 두가지 형태에 대해 알아보겠습니다.

Server-Client Network는 온라인 게임, 웹 서비스, 페이스북 앱, 카카오톡 앱, 음악/영상 스트리밍, 원격 쉘(SSH), FTP, 데이터베이스 등 대부분의 프로그램에서 사용하는 구조입니다. Server는 다수의 Client의 요청 데이터를 처리하고, 응답 데이터를 보내는 프로세스(웹 서버, 게임 서버, 카카오톡 서버 등) 입니다. Client는 중앙 Server에게 요청 데이터를 보내고, 응답 데이터를 받는 최종 사용자(End User)용 프로세스(웹 브라우저, 게임 앱, 카카오톡 앱 등, 보통 GUI를 제공) 입니다. 따라서 어떤 서비스를 서버-클라이언트 구조로 설계했다면 2개의 프로그램을 작성해야합니다.

(이전 표에서 Well-Known Port들은 모두 서버 측의 포트 번호를 의미하고 있습니다.)

P2P Network는 토렌트, 1:1 온라인 게임 등 Server, Client의 구분 없이 연결된 프로세스들이 동등한 작업을 수행하는 구조입니다. 따라서 어떤 서비스를 P2P 구조로 설계했다면 서버/클라이언트 역할을 모두 수행하는 단일 프로그램을 작성해야 합니다.

[1] ISP, Internet Service Provider
[2] LAN, Local Area Network
[3] WAN, Wide Area Network
[4] Intranet
[5] Internet
[6] OSI, Open Systems Interconnection
[7] Message
[8] Segment
[9] Datagram
[10] Packet
[11] Frame
[12] Ethernet
[13] WiFi
[14] Bluetooth
[15] MAC Address, Media Access Control Address
[16] IP, Internet Protocol
[17] Routing
[18] IANA, Internet Assigned Numbers Authority
[19] NAT, Network Address Translation
[20] Protocol
[21] TCP, Transmission Control Protocol

[22] UDP, User Datagram Protocol

[23] Well-known Port

[24] Server-Client Network

[25] P2P Network, Peer-to-Peer Network

2 프로그래밍 연습

1. 프로그래밍 언어 ... 44
2. Node.js 설치 ... 48
3. 기본 부품과 조합 ... 52
4. 제어와 반복, 함수와 재귀, 에러 ... 58
5. 명령형 프로그래밍, 스코프와 콜 스택 ... 69
6. 객체지향 프로그래밍, 복사와 참조 ... 78
7. 타입과 유추, 명명 규칙 ... 95
8. 함수형 프로그래밍, 콜백과 클로저 ... 104

언어에 국한되지 않는 프로그래밍의 기본적인 원리를 배웁니다. Node.js를 통해 기본적인 문법부터 객체 지향(OOP), 함수형 프로그래밍, 모듈화, 추상화 등에 대해서 배웁니다. 이를 통해 사람이 읽고 쓰기 좋은 코드, 재사용성과 확장성이 있는 코드의 구조를 체득합니다.

2 프로그래밍 연습

2.1 프로그래밍 언어

1. 프로그래밍 언어의 구분
2. 프로세스 가상머신
3. JavaScript, Node.js

> 프로그래밍 언어의 종류에 대해서 알아보고, JavaScript와 Node.js를 소개합니다.

1. 프로그래밍 언어의 구분

<컴파일 언어>

- 컴파일 언어

  소스코드를 컴파일 한 뒤 그 바이너리를 배포하는 언어. 예로써 C언어로 소스코드를 작성 한 뒤 컴파일하여 그 실행 파일을 배포합니다. 또는 Swift로 iOS 프로그램의 소스코드를 작성 한 뒤 컴파일하고 AppStore에 배포합니다.

- 스크립트 언어

  소스코드 자체를 배포하여 실행시에 동적으로 컴파일하는 언어. 이 때의 컴파일러를 특히 인터프리터(Interpreter)라고도 합니다. 예로써 웹 브라우저에서 실행되는 JavaScript나 게임에서 실행되는 Lua 등이 있습니다.

프로그래밍 언어는 프로그램을 작성하기 위한 도구입니다. 언어의 문법이나 라이브러리 등의 내용을 깊히 아는 것도 중요하지만, 우선 프로그래밍 언어가 도구라는 인식을 가지는 것이 먼저입니다. 때문에 구현하고자 하는 프로그램의 속성에 따라서 개발자가 능동적으로 언어와 플랫폼을 선택을 할 수 있어야 합니다.

2. 프로세스 가상머신

<JAVA의 컴파일>

(Java와 JavaScript는 이름은 비슷하지만 전혀 다른 언어입니다.)

Java나 C# 같이 특정 언어로 작성한 프로그램이 OS나 CPU에 종속되지 않는 경우가 있습니다. 즉 Java로 작성하고 컴파일 한 바이너리가 윈도우즈, Linux, Android, macOS에서도 동일하게 실행됩니다. 이러한 크로스플랫폼(Cross-Platform)은 Java 컴파일러가 소스코드를 OS나 CPU에 맞추어 컴파일하지 않고 Java만의 고유한 아키텍쳐를 따라 컴파일하기 때문에 가능합니다. 어느 OS에서 Java로 작성된 소스코드를 컴파일해도 Java의 바이너리는 Java ByteCode를 기반으로 동일한 프로그램을 생성합니다. 이후에 Java ByteCode가 특정 OS에서 실행될 때, 가상 머신이라는 프로그램을 거쳐 OS와 CPU가 이해 할 수 있도록 바이너리를 재컴파일 한 뒤 실행하게 됩니다. 이 때문에 Java로 만든 프로그램을 실행하려면 Java Runtime이라는 가상머신 프로그램이 필요하고, C#으로 만든 프로그램을 실행하려면 Common Language Runtime이라는 가상머신 프로그램이 필요합니다.

3. JavaScript, Node.js

```
/**
 * Receives array of numbers bigger than 0 and returns maximum value.
 * @param {array} incoming array of numbers
 * @return {number} The maximum value in array
 */

function getMaxValue(array){

    // Declare a new variable to hold maximum value
    var maxValue = 0;

    // Iterate over array's elements
    for (var index=0; index<array.length; index++){

        // Get array element at specific index
        var element = array[index];

        // In case element value is bigger than current maxValue - update current maxValue
        if (element > maxValue) {
            maxValue = element;
        }
    }

    // Return maxValue
    return maxValue;
}

var myArray = [1,25,3,94,5,12,4,5,1,123,524,123];
var maxValue = getMaxValue(myArray);
console.log("maxValue == " + maxValue);
```

<JavaScript>

JavaScript는 웹 브라우저 위에서 동작하는 스크립트 언어입니다. 이후 JavaScript의 문법을 차용하여 Node.js라는 OS 위에서 동작하는 언어가 개발되었습니다. 학습 난이도를 줄이기 위해서 서로 많이 닮은 두 언어를 이용해서 수업을 진행하기로 하였습니다.

- JavaScript

  웹 페이지에 사용자와의 동적인 상호작용을 도입 할 목적으로 개발되었습니다. JavaScript의 모든 모듈은 웹 브라우저 위에서 작동합니다.

- Node.js

  JavaScript와 거의 동일한 문법(Syntax)으로 이루어져있으나 Node.js의 모든 모듈은 OS 위에서 작동합니다.

예를 들어서 JavaScript 코드로는 웹 브라우저 창을 벗어나 그림을 그리거나, 하드디스크의 프로그램을 실행시키거나, 파일을 삭제하거나 등의 작업을 할 수 없습니다. 웹 브라우저라는 플랫폼에서 JavaScript 코드에게 이러한 접근을 허용하지 않기 때문입니다. 그러나 Node.js는 OS 위에서 바로 작동하기때문에 Node.js로 TCP/IP 네트워킹을 구현하거나, GUI를 구현하는 등의 좀 더 유연한 작업을 할 수 있습니다. 하지만 언어가 작동하는 플랫폼이 넓은 것이 무조건 최고라고 하긴 힘듭니다. 개발자의 목적에 맞는 적절한 플랫폼, 생태계를 선택해야 합니다.

[1] Interpreter
[2] Cross-Platform

2 프로그래밍 연습

2.2 Node.js 설치

1. Node.js 및 NPM 설치
2. IDE 설치
3. 환경 테스트

> Node.js와 NPM을 설치하고 IDE를 설치해 개발환경을 구성합니다.

1. Node.js 및 NPM 설치

<Node.js 공식 홈페이지>

Node.js 및 NPM(Node Package Manager)을 설치합니다. Node.js를 설치하면 NPM이 같이 설치됩니다. 노드 공식 홈페이지 (https://nodejs.org/ko/)에서 최신 버전을 설치합니다.

2. IDE 설치

<Visual Studio Code>

소스 코드는 텍스트 파일이기 때문에 메모장 같은 단순한 텍스트 에디터로도 작성이 가능합니다. 하지만 생산성을 높히기 위해서 컴파일러, 텍스트 에디터, 디버거 및 코드 자동 완성 도구 등이 함께 통합된 소프트웨어인 통합개발환경(IDE)을 사용합니다. IDE는 간단하게는 타이핑 오류를 교정해주고 언어의 문법 구조를 더 잘 볼 수 있게 Syntax Highlighting 등의 기능을 제공합니다. 아래에 몇가지 에디터를 소개하였습니다. 수업에서는 Sublime Text3를 사용합니다.

이름	특징	가격	링크
Visual Studio Code	Microsoft에서 개발 한 웹 개발용 에디터	무료	링크 (https://code.visualstudio.com/)
Brackets	다양한 확장 기능을 제공하는 웹 개발용 오픈소스 에디터	무료	링크 (http://brackets.io/)
Atom	Github에서 개발 한 오픈소스 에디터	무료	링크 (https://atom.io/)
Sublime Text3	다양한 확장 기능을 제공하는 에디터	유료	링크 (https://www.sublimetext.com/3)
IntelliJ/WebStorm	Android Studio의 기반이되는 IDE로 JetBrains사에서 개발	유료	링크 (https://www.jetbrains.com/webstorm/)

<상용 IDE>

3. 환경 테스트

새로운 프로젝트를 시작하면 플랫폼에 필요한 환경과 높은 생산성으로 개발 할 수 있는 개발 환경을 구성 할 필요가 있습니다. 설치한 IDE를 실행 시키고 새 파일을 작성하고 저장합니다.

```
console.log("hello");
```

셸을 실행하고 cd 명령어로 파일이 저장된 디렉토리로 이동 한 뒤 node 파일명을 실행합니다. node가 파일을 읽은 뒤 컴파일하고 실행하면 셸에 날짜가 출력됩니다.

[1] IDE, Integrated Development Environment

2 프로그래밍 연습

2.3 기본 부품과 조합

1. 프로그래밍 언어의 핵심 요소
2. 변수와 주소값
3. 값과 연산자
4. Array

> 변수, 연산자, 값, 타입과 같은 프로그래밍 언어의 공통적 부품들과 이를 조합하는 방법에 대해 알아봅니다.

1. 프로그래밍 언어의 핵심 요소

```
a = 1.5;
b = "hello";
c = a + b;
d = a * 2;
```

- 주소

  프로세스 메모리의 특정 위치를 가리키는 번지 수 (32bit 또는 64bit)

- 값

  프로세스 메모리에 기록된 데이터

- 타입

  값 들의 그룹(숫자인지 텍스트인지 등)

- 이름

  특정 주소를 가리키는 소스코드 상의 심볼

- 로직 (연산자 및 제어문, 함수)

  CPU에 산술 명령을 내리거나, 코드의 분기나 반복을 제어

모든 프로그래밍 언어는 기본 부품과 부품들의 조합으로 이루어져 있습니다. 당장은 Node.js를 배우지만 다른 언어의 구성도 크게 다르지 않습니다. 필요와 원리를 기억하면 새로운 언어를 배우는 일도 간단해집니다.

2. 변수와 주소값

프로그래밍 언어에서 주소를 가리키기 위한 이름을 변수라고 합니다. 변수는 메모리 상의 특정한 주소를 가리키고 있습니다. 변수에 값을 할당하는 것은 변수가 가리키는 주소 공간에 값을 저장하는 것입니다.

예시	설명
x = 10;	x가 가리키는 곳에 10이라는 값을 저장
y = x;	y가 가리키는 곳에 x의 값을 복사해서 저장

<변수>

3. 값과 연산자

타입	설명	예시
Number	정수, 소수, 무한대, 잘못된 숫자를 포함한 숫자 그룹	77, -100.55, Infinity, -Infinity, NaN (Not a Number)
String	문자열을 나타내는 그룹	"", "a", "abc", "abc def ghi", 'wyz', `ufc`, "\""
Boolean	참과 거짓을 나타내는 그룹	true, false
(타입이 아님)	빈 값	null
(타입이 아님)	빈 값도 아니고 애초에 정의가 되지 않은 값	undefined

<타입과 값>

연산자	설명	예시
+	더하기	3+10, "hel"+"lo"
*	곱하기	3*10
/	나누기	3/10 (0.3)
%	나머지	13%10 (3)
**	제곱	3**2 (9)
++	증가	3++ (4)
--	감소	3-- (2)

<산술 연산자>

연산자	설명	예시
=	대입	x = 10, x = "xyz", x = y+z
+=	x = x+1	x += 1, x += "hello"
-=	x = x-1	x -= 1
*=	x = x*2	x *= 2
/=	x = x/2	x /= 2
%=	x = x%2	x %= 2

<할당 연산자>

연산자	설명	예시
==	느슨하게 같음	1 == 1 (true), 1 == true (true)
===	엄격하게 같음	1 === true (false)
!=	느슨하게 다름	true != 1 (false)
!==	엄격하게 다름	true !== 1 (true)
>, <	초과, 미만	1 > 0.9 (true), 0.9 < 1 (true)
>=, <=	이상, 이하	1 >= 1 (true), 1 <= 1="" (true)
(Boolean) ? (A) : (B)	삼항연산자	(2 > 1) ? "o":"x" ("o"), (10 == 5) ? 1:2 (2)

<비교 연산자>

연산자	설명	예시
!	NOT	!true (false), !false (true)
&&	AND	true && true (true), true && false (false)
||	OR	true || true (true), true || false (true)

<논리 연산자>

타입 캐스팅 (형 변환)

서로 다른 타입 간에 연산이 일어날 때는 타입 캐스팅(Type Casting) 또는 형 변환이라고 하는 연산이 내부적으로 일어납니다. 예를 들어 3 + "abc" 같은 경우는 Number 3 이 String "3"으로 변환 되고 "3"+"abc" 연산이 일어나 "3abc" 가 됩니다. 가능한 형 변환이나, 형 변환시 어떤 타입을 따를지에 대한 우선 순위 등은 언어별로 차이가 있을 수 있습니다.

Node.js는 형 변환에 너그러운 언어입니다.

예시	결과
true+"abc"	"trueabc"
!""	true
"7"*3	21
"ab"*3	NaN
"ab"*3	NaN

<타입 캐스팅>

연산자	설명	예시
typeof	값의 타입을 String으로 리턴	typeof true ("boolean"), typeof xyz ("undefined")
instanceof	값이 특정 타입에 속하는지를 리턴 (뒤에서 다시 다룸)	[] instanceof Array (true)

<타입 연산자>

값과 연산자로 조합 가능한 식

```
false || !false;
true && 3 > 1;
123 % 11;
(10*10 - 5) % 5;
35 == 7*5;
"abc" + 1 + "def" + 35*6 + "ghi" + false;
typeof 123;
a = 5;
a++;
++a;
a -= 10;
a += 30;
a--;
a *= 2;
b = "abcdefg";
c = b;
bool = (b == "abcdefg");
```

4. Array

프로그래밍 언어에는 보통 값들을 한개의 이름 아래에 묶어두기 위해서 Collection이라고 불리는 자료 구조들을 제공합니다. Node.js에서는 Array(배열)라고 하는 Collection을 제공합니다.

```
// 배열 생성
numbers = [1,2,3];
emptyList = [];

// 값 읽기 (순서가 0부터 시작)
numbers[0]; // 1
numbers[1]; // 2
numbers[2]; // 3

// 값 할당
numbers[0] = 100;

// 값 추가
numbers.push(10);
```

주어진 배열에 Apple을 더하고 첫번째 요소를 Banana로 바꾼뒤 출력

```
fruits = ["Grape", "Peach"];
fruits.push("Apple");
fruits[0] = "Banana";
console.log(fruits);
```

[1] Type Casting

2 프로그래밍 연습

2.4 제어와 반복, 함수와 재귀, 에러

1. 제어문
2. 조건문
3. 반복문
4. 함수
5. 재귀
6. 에러

> 프로그램의 논리적인 흐름을 제어하는 제어문과 반복문, 함수와 재귀에 대해서 알아봅니다.

1. 제어문

<반복문과 조건문>

프로그램은 기본적으로 소스코드의 **첫째 줄부터 마지막 줄까지 차례대로 실행**되지만 때로는 실행 순서를 조절해야 할 필요가 있습니다. 어느 **조건(Boolean)을 만족하는 경우에만 코드를 실행 (조건문)**해야 하는 경우도 있고, 어느 **조건(Boolean)을 만족하는 경우에 특정 코드를 반복 (반복문)**해야 하는 경우도 있습니다. 또는 **초기 값의 변화를 주면서 특정한 코드를 반복해야 하는 경우 (함수)**도 있습니다.

2. 조건문

조건이라 함은 계산했을 때 Boolean이 되는 식이나 변수, 값을 의미합니다. 물론 논리 연산자(&&, ||, !)를 조합해서 조건을 표현 할 수도 있습니다. 또 복잡한 조건들의 조합은 괄호를 이용해서 표현할 수도 있습니다.

IF-ELSE 문

```
if (조건) {

} else if (조건) {

} else if (조건) {

} else {

}
```

x라는 숫자가 담긴 변수가 1~10 사이 일때, 10~20 사이일 때 20~ 일때로 분기

```
x = 5;
if (1 <= x || x < 10) {
    // x가 1~10
} else if (10 <= x || x < 20) {
    // x는 10~20
} else {
    // x는 20 이상 또는 1 미만
}
```

SWITCH 문

```
switch (값) {
    case 값1:
        // 값1인 경우
    break;
    case 값2:
        // 값2인 경우
    break;
    default:
        // 값1도 값2도 아닌 경우
}

x = "a";
switch(x) {
    case "a":
        // ...
    break;
    case "b":
        // ...
    break;
    default:
        // ...
}
```

3. 반복문

특정 작업을 반복적으로 처리를 해야할 일을 처리할때 FOR문 WHILE문을 사용합니다. 배열의 각 요소들에 대해서 같은 처리를 반복하는 경우를 생각해 볼 수 있습니다.

FOR문

```
// 어떠한 반복적인 코드를
arr = [1,2,3,4,5, ..., 100];
arr[0] += 10;
arr[1] += 20;
arr[2] += 30;
arr[3] += 40;
arr[4] += 50;
...
arr[99] += 1000;

// 위 코드를 for문으로 표현하면
for(i=0; i < arr.length; i++) {
    arr[i] += (i+1)*10;
}

// for문의 구조
for(초기화; 반복 조건; 반복 후) {
    // ..

    if (...)
        break; // 루프를 완전히 종료

  if (...)
    continue; // 이번 루프는 여기서 종료하고 다시 루프 조건을 확인하고 진행
}
```

FOR문을 이용해서 구구단을 출력해보기

```
for(i=1; i < 10; i++){
    for(j=1; j < 10; j++){
        console.log(i+"*"+j+"="+i*j);
    }
}
```

FOR문을 이용해서 구구단중 1, 3, 5, 6, 7, 9단만 출력해보기

```
for(i=1; i < 10; i++){
    if(i%2==0) continue;
    for(j=1; j < 10; j++){
        console.log(i+"*"+j+"="+i*j);
    }
}
```

WHILE문은 FOR문과 달리 루프 조건만을 명시합니다. 하지만 WHILE문은 FOR문은 쓰이기에 따라서 똑 같은 일을 할 수 있습니다. 상황에 맞게 편리한 문법을 쓰면 됩니다.

```
i = 0;
while (i < 10) {
   // 마찬가지로 break; continue; 사용 가능
   // ...

   i++;
}
```

WHILE문을 이용해서 구구단 중 2, 5, 8 단만 출력해보기

```
i = 1;
while(i < 10){
    if(i==2 || i==5 || i==8){
        j=1;
        while(j < 10){
            console.log(i+"*"+j+"="+i*j);
            j++;
        }
    }
    i++;
}
```

4. 함수

함수(Function)는 초기 값(인자)의 변화를 주면서 특정한 코드를 반복하는 경우에 사용합니다. 함수를 사용하면 코드의 재사용성을 높힐 수 있습니다. 나아가서 함수는 말이되는 코드, 짜임새있는 프로그램의 구조를 구성하는 기초가 됩니다. 함수를 사용하지 않는다면 특정 기능이 필요한 곳마다 같은 코드를 반복해서 작성해야 합니다. 이 때의 문제점은..

1. 기능이 필요 할 때마다 전체 코드를 매번 작성해야 하는 수고
2. 기능의 공통적인 부분에 변화가 필요할 때 모든 코드를 수정해야 하는 수고
3. 코드의 기능이 눈에 쉽게 읽히지 않음

함수는 초기 값(인자)들을 받아서 특정 코드를 수행하고 결과 값을 리턴(반환)해줍니다. 이때 리턴 값을 명시하지 않으면 undefined를 리턴합니다.

함수 정의하기

```
sum = function (a, b){ // 함수를 정의하고 sum이라는 이름을 붙힘
    return a + b;
};
result = sum(1, 3); // 함수를 실행 할 땐 함수명(인자들...)
```

함수 정의하기2

```
function sum(a, b){ // 이름을 붙히면서 정의하기
   return a + b;
}
result = sum(1, 3);
```

함수를 정의하는 것과 함수를 실행하는 것은 다릅니다. 실행하기 위해서 함수 이름 뒤에 인자 묶음 **(...)**을 붙힙니다.

```
sum = function(a,b){
    return a+b;
};
sum(1,2); // 3
sum(1,2,3); // 3
sum(1); // ???
sum(); // ???
sum2 = sum;
sum2(4,5); // 9
```

구구단을 x단 부터 y단까지 출력하는 함수를 정의

```
function print99(x, y) {
    for(var i=x; i < y+1; i++){
        for(var j=1; j < 10; j++){
            console.log(i+"*"+j+"="+i*j);
        }
    }
}
print99(3, 6);
```

5. 재귀

재귀(Recursion) 함수는 함수 내부에서 자기 자신을 다시 호출하는 함수를 말합니다. 재귀 함수를 사용하면 반복문과 조건문만으로는 표현하기 어려운 작업을 구현 할 수 있습니다.

<팩토리얼 함수의 호출>

fac(n) = n*fac(n-1)을 만족하는 팩토리얼 함수를 구현

```
function fac(n){
    if(n==0 || n==1) // 종료 조건
        return 1;
    else
        return n * fac(n-1); // 재귀 호출
}
```

재귀 함수에는 반복적인 호출이 **종료되는 조건**이 존재해야 합니다. 또한 재귀 호출을 하는 n*fac(n-1) 부분을 작성할 때는 **fact(n-1)이 제대로 된 값을 리턴해준다고 가정**하고 구현합니다. 호출 구조를 상상해 보면서 함수가 계산 범위를 줄여나가면서 종료 조건에 도달하는지 확인합니다.

<피보나치 함수의 호출>

fibo(n) = fibo(n-1)+fibo(n-2)를 만족하는 피보나치 함수를 구현

```
function fibo(n){
    if(n==1 || n==2) // 종료 조건
        return 1;
    else
        return fibo(n-1) + fibo(n-2); // 재귀 호출
}
```

재귀함수를 사용하면 수학적인 문제를 쉽게 해결할 수 있습니다. 예) 하노이의 탑 문제 (https://ko.wikipedia.org/wiki/%ED%95%98%EB%85%B8%EC%9D%B4%EC%9D%98

6. 에러

프로그램의 에러는 크게 두 종류로 볼 수 있습니다. 첫째는 컴파일 타임 에러(Compile time error)로, 흔히 문법적인 오류로 인해 컴파일 자체가 되지 않는 경우입니다. 이러한 에러는 컴파일러가 주는 힌트를 통해 디버깅하면 됩니다. 둘째는 런 타임 에러(Run time error)로, 프로그램 실행 중에 구조적인 또는 논리적인 결함으로 오류가 발생하는 경우입니다. 이때 에러는 프로그램을 중지시킬 수도 있고, 단순히 예상치 않은 동작을 할 수도 있습니다. 이 때는 로직을 일일이 점검해야 하기에 디버깅에 골치가 아플 수 있습니다.
위의 에러들은 의도치 않은 에러지만, 프로그램을 작성하다보면 의도된 에러가 필요 할 수 있습니다.

에러 발생시키기

```
// x를 y로 나누는 함수
function devide(x, y){
    if (y == 0) {
        var err = new Error("Can't devide by zero.");
        throw err; // 에러를 발생시키면 함수가 바로 종료됨
    }

    return x/y;
}

var a = devide(10, 5); // 2
var b = devide(10, 0); // VM348:6 Uncaught Error: Can't devide by zero.(…)

// 이때 12번 줄에서 발생한 에러는 devide 함수를 종료시키고, 나아가서 12번줄 밑의 프로그램도 중지시킵니다.
console.log('program stopped..'); // 출력 안됨
```

위처럼 Uncaught Error는 프로그램을 종료시킵니다.

에러 제어하기

```
// x를 y로 나누는 함수
function devide(x, y){
    if (y == 0) {
        var err = new Error("Can't devide by zero.");
        throw err; // 에러를 발생시키면 함수가 바로 종료됨
    }

    return x/y;
}

try {
    var a = devide(10, 5); // 2
    var b = devide(10, 0); // Error!
} catch(e) {
    console.log(e);
    console.log("Try again..!");
}

// try, catch문으로 제어된 에러는 프로그램을 종료시키지 않습니다.
console.log('program not stopped..');

/**
Error: Can't devide by zero.(…)
Try again..!
program not stopped..
**/
```

이렇게 에러를 던지고 잡는 제어문을 통해서 프로그램의 확장성을 높힐 수 있습니다. devide 함수에서 0으로 나누는 오류를 임의로 처리하지 않고 throw문을 통해 에러만을 발생시킵니다. 이렇게 오류에 대한 처리를 다른 코드에 위임함으로써 devide 함수는 더욱 다양하게 사용 될 수 있습니다.

의도된 에러는 유용합니다

```
var fs = require('fs');
try {
  // 존재하지 않는 파일을 읽으려고 함
  var data = fs.readFileSync('invalid-file-path');
  console.log(data);

} catch(err) {
    // 에러 발생시
  console.log(err);

} finally {
    // 에러에 관계 없이 항상 처리
    // catch에서 다시 에러가 발생하더라도 처리됨
}
```

try...catch의 문법 예시

(https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/try...

[1] Function

[2] Recursion

[3] Compile time error

[4] Run time error

2 프로그래밍 연습

2.5 명령형 프로그래밍, 스코프와 콜 스택

1. 스코프
2. 콜 스택
3. 명령형 프로그래밍

> 스코프와 콜 스택에 대해서 알아보고, 가계부 프로그램을 만들며 명령형 프로그래밍에 대해 알아봅니다.

1. 스코프

코드상에서 이름(변수)들에 접근 가능한 유효범위을 스코프(Scope)라고 합니다.

```
a = [1, 2, 3];
function b() {
    return 1;
}
console.log(123);
```

지금까지 작성하던 방식으로 생성된 a나 b와 같은 이름 또는 console.log와 같은 이미 정의된 이름들은 기본적으로 전역 스코프(global scope)에 존재하게 됩니다. 전역 스코프는 모든 코드 위치에서 접근할 수 있으며, 명시적으로 제거하지 않는 이상은 그 값이 프로세스가 종료 될 때까지 메모리에서 해제되지 않습니다. 또한 전역 스코프에 존재하는 이름을 전역 변수라고 부르기도 합니다.

전역 변수와 변수 해제

```
a = [1,2,3];
function printA() {
    console.log(a); // 함수 안에서도 전역 변수에 접근 할 수 있습니다.
}

printA(); // [1,2,3]

delete a; // 또는 a = null; a = undefined; 를 통해서 명시적으로 값을 해제 할 수 있습니다.

printA(); // 오류! Uncaught ReferenceError: a is not defined
```

그리고 특정 코드 위치에서만 지엽적으로 접근 할 수 있는 스코프를 지역 스코프, 그 변수를 지역 변수라고 부릅니다. 어떤 변수가 지역 스코프에 위치하게 되는가는 프로그래밍 언어별로 미세한 차이가 있을 수 있습니다. JavaScript에서는 함수 안에서 var 키워드를 통해 정의된 변수가 지역 스코프에 위치하게 됩니다. 또한 일반적으로 지역 변수는 함수가 종료됨과 동시에 해제됩니다.

지역 변수

```
function inner() {
    var a = 10; // inner함수 안에서만 접근 할 수 있는, 지역 변수를 생성합니다.
    b = 20; // 함수 안에서도 var 키워드 없이 변수에 접근하면 전역 변수로 처리됩니다.
}
d = 30; // 전역 변수로 처리됩니다.
var c = 40; // 함수 밖에서 var 키워드로 정의된 변수도 다를 바 없이 전역 변수로 생성됩니다.

inner();
console.log(b); // 20
console.log(a); // 오류! Uncaught ReferenceError: a is not defined
```

왜, 지역 변수라는 개념이 필요할까? 라고 물으시면 함수의 재사용성을 살리기 위해서, 메모리를 효율적으로 이용하기 위해서 라고 말 할 수 있겠습니다. 전역 스코프, 지역 스코프의 개념 없이 단일한 전역 스코프만 존재한다면, 수 많은 이름들이 함수가 실행될 때마다 생성되고, 덮어 씌워지고는 등, 이름들이 서로 뒤섞이게 될 것입니다.

전역 변수만을 이용한다면...

```
function aa() {
    a = 30;
    b = 40;
    return a + b;
}
function bb(x, y) {
    a = x + y;
    b = x - y;
    return a*b;
}

a = 10;
b = 20;
c = aa();
bb(a, b);
```

전역 스코프만으로는 프로그램을 안전하게 작성 할 수가 없습니다. 또한 코드의 흐름을 읽기도 힘이 듭니다. 이는 서로 할 일을 나누어서 자기일만 확실히 한다는 추상화, 캡슐화 원칙에도 어긋납니다. 또한 함수의 인자로 쓰이는 이름이 전역 스코프에 이미 존재하는 이름이라면 어떨까요?

```
a = 50;
function inc(a) {
    return a + 1;
}
inc(100);
```

전역 스코프만 존재한다면 inc 함수가 실행되면서 인자 a와 밖에서 이미 정의된 a라는 이름이 충돌하게 됩니다. 함수의 실행과 지역 스코프에 대해서 자세히 알아보도록 하겠습니다.

2. 콜 스택

<JavaScript 콜 스택>

플랫폼을 떠나서 모든 프로그램에는 진입점(Entry Point)이라는 위치가 존재합니다. C나 Java의 main, Node.js나 JavaScript의 스크립트 첫 라인 등, 플랫폼마다 그 차이가 존재 할 수 있지만, 운영체제가 실행할 프로세스의 첫번째 코드 위치를 진입점이라고 합니다. 프로세스가 실행되면, 프로세스는 진입점에서부터 순차적으로 코드들을 수행하면서 메모리의 DATA 영역에 할당된 값들을 조작해 나갑니다. 이 때 이 DATA 영역을 전역 변수들이 존재하는 메모리 공간, 즉 전역 스코프라고 생각하셔도 좋겠습니다.

프로세스 메모리 모식도

```
function x() { ... }
a = 10;
b = "안녕";
for (i=0; i<10; i++) a++;

/*
STACK-------
...

HEAP--------
...

DATA-------
a=10, b="안녕", i=0

CODE--------
function x ...
for ...
*/
```

> JavaScript는 스크립트 언어이면서 또 고수준으로 추상화된 언어이기 때문에, 실제 메모리 상태는 일반적인 프로세스를 모식한 위 그림과는 큰 차이가 있습니다. 실제로 JavaScript의 함수 콜에 따른 실행 문맥(Execution Context)의 변동과 스코프라는 개념은 저수준에서 작동하지 않습니다. 여기에서는 저수준의 개념을 이해할 수 있도록 스코프를 메모리의 스택 프레임에 비유하여 설명합니다. 자세한 내용이 궁금하신 분들은 고급 프로그래밍 언어의 컴파일, JavaScript의 실행 문맥에 대해서 더 공부해보시길 바랍니다.

그러던 도중에 함수를 실행하는 구문을 만나면, 즉 함수를 콜하게 되면, 메모리의 STACK 영역에 함수의 **인자로 전달된 값**들을 복사하며 **스택 프레임(Stack Frame)**이라는 영역을 할당합니다. 실제 스택 프레임에는 인자들 뿐만 아니라 **함수가 종료(return)되면 다시 돌아갈 코드의 메모리 번지**에 대한 정보 또한 포함되어 있습니다. 이 때 생긴 **스택 프레임을 바로 지역 스코프**라고 생각하셔도 좋겠습니다. 이렇게 함수가 실행되면서 인자로 전달되는 값들이 새로운 스택 프레임에 복사되기 때문에 **함수 안에서의 a와 함수 밖의 a는 서로 다른 메모리 번지**를 가리키게 됩니다.

<Scope Chaining>

코드에서 변수를 참조하는 경우엔 **코드가 실행되는 문맥에서 가장 가까운 스코프의 변수**를 가리키게 됩니다. 이 때문에 변수명이 중복되는 경우엔 가장 가까운 지역 변수를 참조하게 되며, 변수명이 존재하지 않는 경우엔 이름을 찾을 때까지 스코프를 거슬러 올라갑니다. 최종적으로 전역 스코프에도 그 이름이 존재하지 않으면 오류가 발생합니다. 이러한 메커니즘을 **스코프 체이닝(Scope Chaining)**이라고 합니다.

함수의 실행과 새로운 스택 프레임 생성

```
function x(a, b, c) {
    > console.log(a+10, b, c+20);
}
a = 10;
b = "안녕";
for (i=0; i<10; i++) a++;
> x(a, b, 100);

/*
STACK-------
x: a=20, b="안녕", c=100
console.log: 30, "안녕", 120

DATA--------
a=20, b="안녕", i=10
*/
```

그리고 함수가 종료되면서 함수가 실행된 순서의 역순으로 스택 프레임이 해제됩니다. 이를 통해서 지역 스코프의 변수 또한 해제됩니다.

> 사실 JavaScipt와 같은 고수준 언어에서는 지역 변수를 해제하는 과정에 조금 더 복잡한 메커니즘을 갖고 있습니다. 추후에 프로세스의 메모리를 관리하는 Garbage Collector라는 개념에 대해서 다룹니다.

3. 명령형 프로그래밍

프로그래밍은 기본적으로 값(상태)을 변경시키는 명령문의 순차적인 나열입니다. 여기에 조건문 반복문 등의 제어 구문을 통해서 복잡한 로직을 표현 할 수 있습니다. 또한 초기 값을 다르게 반복되는 코드는 함수를 통해 추상화 할 수 있습니다.

> ..., 명령형 프로그래밍(Imperative Programming)은 ... 프로그래밍의 상태와 상태를 변경시키는 구문의 관점에서 연산을 설명하는 프로그래밍 패러다임의 일종이다. 자연 언어에서의 명령법이 어떤 동작을 할 것인지를 명령으로 표현하듯이, 명령형 프로그램은 컴퓨터가 수행할 명령들을 순서대로 써 놓은 것이다... (위키백과)

지금까지 배운 내용을 토대로 가계부를 만들어 보겠습니다. 가계부는 잔액과 입출금 내역을 표시 할 수 있어야하고, 입금과 출금을 기록 할 수 있는 기능이 있습니다. 현재 있는 금액과 입출금 내역은 변수에 저장해야 합니다. 입출금 내역은 계속 늘어나므로 배열로 구현하고, 입금과 출금, 출력 기능은 함수로 선언한 변수에 접근하도록 구현해 보겠습니다.

```
var amounts = [];
var names = [];
var total = 0;

function deposit(amount, name) {
  if (amount + total < 0) {
    throw new Error(`Not enough balance for ${name}`);
  }

  amounts.push(amount);
  names.push(name);
  total += amount;
}

function print() {
  var result = '';
  for(var i=0; i < amounts.length; i++) {
    console.log(`${amounts[i] > 0 ? '입금' : '출금'}\t${names[i]}\t${amounts[i]}`);
  }
  console.log(`잔액:\t${total}`);
}

try {
  deposit(100, "월급");
  deposit(200, "용돈");
  deposit(-150, "월세");
  deposit(-300, "보험료");
} catch (e) {
  console.log(e);
}

print();

/**
Error: Not enough balance for 보험료 ...
입금 월급 100
입금 용돈 200
출금 월세 -150
잔액: 150
**/
```

사용자 인터페이스(CLI 또는 GUI)를 제공하진 않았지만, 간단히 가계부를 흉내내는 프로그램을 작성하였습니다. 이 코드의 문제점이 무엇일까요?

1. 가계부를 사용자마다 사용 할 수 있게 여러개로 늘려야 한다면?
2. 또 사용자마다 은행 계좌별로 가계부를 쓸 수 있게 한다면?
3. 입출금 내역에 입출금자를 추가하고 싶다면?
4. 등등...

목적에만 급급하게 짜내려간 코드는 추상화가 부족하기 때문에 **확장성과 재사용성이 부족**합니다. 또한 **전역 스코프(Scope)가 지저분**해지기 쉬우며, 코드 자체가 **"말"이 되지 않기** 때문에 사람이 쉽게 읽고 이해하기가 힘듭니다. 사람이 이해하고 작성하기 쉬운 코드로 나아가기 위해서 다음 파트에서 **객체 지향 프로그래밍(OOP)**에 대해서 알아보겠습니다.

[1] Scope
[2] Entry Point
[3] Execution Context
[4] Stack Frame
[5] Scope Chaining

2 프로그래밍 연습

2.6 객체지향 프로그래밍, 복사와 참조

1. 객체 지향 프로그래밍
2. 복사와 참조
3. 클래스와 인스턴스
4. 클래스 정의하기
5. 상속

> 객체지향 프로그래밍과 모델링에 대해 알아봅니다. 또한 값의 복사와 참조를 배우고, 언제 복사와 참조가 일어나는지 구분 할 수 있도록 합니다.

1. 객체 지향 프로그래밍

객체 지향 프로그래밍(OOP: Object Oriented Programming)은 코드를 작성하는 일종의 방법론, 패러다임입니다. OOP에서 **객체는 어떤 물건이나 추상적인 개념**을 의미합니다. 예를 들어 지난번 가계부 프로그램의 **장부, 입출금 내역**을 객체로 볼 수 있습니다. **객체는 속성(값)들의 집합**으로 표현 됩니다. 예로 강아지를 객체로 표현한다면 그 객체는 견종, 나이, 이름 등의 속성과 짖는다, 달린다 등의 속성을 가질 수 있습니다. 현실의 물건이나 개념을 객체로 옮기는 과정을 모델링이라고 합니다. 이때는 **현실의 요소들을 추상화하여 프로그램에 필요한 핵심적인 요소만을 코드로 옮기게** 됩니다. 작성하는 코드가 말이 되는지 (Human-Readable)를 생각하며 모델링하면 짜임새 있는 코드를 작성 할 수 있습니다.

객체는 속성(값)들의 집합으로 표현 됩니다

```
// 객체의 속성은 키-값의 쌍으로 이루어집니다.
var computer = {
  OS: "macOS",
  CPU: "Intel CPU",
  RAM: "8GB",
  created: "2013-10-21"
};

// 키는 "" 문자열로 표현해도 무관합니다.
var dog = {
  "name": "mong",
  "age": 6,
  "color": "black and white"
};
```

객체의 속성에 접근하는 법

```
var obj = {
  number: 10,
  "some-key": "abc" // some-key: "abc" 의 경우는 some - key : "abc"로 인식해 오류가 발생합니
};

obj.number = 20;
console.log(obj.number); // 20

var name = "some-key";
console.log(obj[name]); // "abc"
console.log(obj["some-key"]); // "abc"

// obj.some-key 로 접근하면 obj.some - key 로 인식합니다.
// 이런 경우에는 obj["some-key"] 를 이용합니다.
```

객체는 속성으로 원시값을 가질 수도, 함수를 가질 수도 또는 다른 객체를 가질 수도 있습니다

```
var x = { a:1, b:"2" };

// 속성으로 값이나 함수를 가질 수 있습니다.
x.c = 10;
x.d = "abc"
x.e = function(){};

function someFunc(){}
var anotherFunc = function(){}

x.someName = someFunc;
x.anotherName = anotherFunc;

// 물론 속성으로 배열도 가질 수 있습니다.
x.z = [1,2,3];

// 속성으로 또 다른 객체도 가질 수 있습니다.
// 즉, 객체는 값으로 취급됩니다.
x.f = {
  g: 10,
  h: {
    deep: {
      deep2: {
        deep3: {
          x: 10
        }
      }
    }
  },
  i: function(){}
};

// 객체가 값이기 때문에 배열에 객체를 담을 수도 있습니다.
x.j = [
    {a:1,b:2},
    {a:2,b:3},
    4,
    5,
    "abc"
  ];
```

2. 복사와 참조

이름은 메모리의 주소를 가리키는 심볼입니다.

```
1  var x = 10; // 값의 할당
2  var y = x; // 이름의 할당
3  console.log(y); // 10
```

위 코드 2번 줄에서 y라는 이름에 x라는 이름을 할당하고 있습니다. 이때는 x라는 이름이 가리키는 주소의 값이 y라는 이름이 가리키는 주소에 복사됩니다.

```
1  var x = 10;
2  var y = x;
3  y++;
4  console.log(y, x); // 11, 10
```

메모리 상에 x의 값과 y의 값은 별개로 존재합니다.

값 바꾸기?

```
1  function swap(a,b){
2    var temp = a; // a를 잠시 담아두고
3    a = b; // a는 b의 값을 할당
4    b = temp; // b는 원래 a의 값을 할당
5  }
6
7  var x = 5;
8  var y = 10;
9  swap(x,y);
10 console.log(x,y); // 5,10 ???
```

swap이라는 함수가 실행될 때 인자로 받아온 a,b는 x,y의 주소가 아니라 스택 위에 복사된 x,y의 복사본을 가리키고있습니다.

배열 복사하기?

```
1  var x = [1,2,3];
2  var y = x;
3  y.push(4);
4
5  console.log(x); // [1,2,3,4] ???
```

y는 x가 가리키는 배열을 복사한 것이 아니라 x가 가리키는 배열을 그대로 가리키고있습니다. 이를 참조라고 합니다. 이처럼 할당에는 값을 메모리에 복사하여 가리키는 복사(Copy by value)와 원본의 주소를 그대로 가리키는 참조(Copy by reference)의 두가지 방식이 있습니다. 언어별로 복사와 참조를 명시적으로 제어 할 수 있는 경우(대표적으로 C언어의 포인터)도 있고, JavaScript처럼 암묵적으로 경우에 따라 처리하는 경우도 있습니다.

- 복사

  JavaScript에서 원시값이라 불리는 "abc", 123, NaN, null, true, undefined 등에 대해서는 항상 복사가 일어납니다.

- 참조

  JavaScript에서 원시값 외의 모든 객체에 대해서는 항상 참조가 일어납니다.

배열 복사하기!

```
var x = [1,2,3];
var y = [];
for(var i=0; i < x.length; i++) {
  y.push(x[i]); // push를 호출할 때 Number 값의 복사가 일어남
};
y.push(4);

console.log(x, y); // [1,2,3], [1,2,3,4]

// 이때는 x, y는 별개의 배열이 됩니다.
```

값 바꾸기!

```
function swap(a,b){ // 여기서 a,b는 아래의 x,y와 동일한 주소를 가리키고 있습니다.
  var temp = a.val; // 복사
  a.val = b.val; // 복사
  b.val = temp; // 복사
}

var x = {val: 5};
var y = {val: 10};
swap(x,y); // x,y는 참조
console.log(x.val,y.val); // 10,5
```

3. 클래스와 인스턴스

우리가 지금까지 이용했던 String, Number, Array와 같은 값들도 모두 객체입니다. 각 객체들은 저마다 이런 저런 속성들을 갖고 있습니다.

타입	예시	속성	속성(함수)
Object	{a:1, b:2}	.a .b ...	.toString() ...
String	"abc"	.length [0] ...	.toString() .indexOf("a") .replace("a","A") .split("b") .substr(0,2) ...
Number	123.55	...	.toString() .toFixed(2) ...
Array	[1,2]	.length [0] ...	.toString() .concat([3,4]) .indexOf(1) .push(3) .pop() .splice(0,1) .forEach(function(a){ .. }) .sort(function(a,b){ .. }) ...
Function	function(a,b){ return a+b; }	.length .name ...	.toString() .apply(null, [1,2]) .call(null, 1, 2) ...

<주요 내장 객체와 그 속성>

JavaScript에서는 위 표처럼 Object 타입을 뼈대로 여러개의 타입들을 설계해두었습니다. 이 때 특정 타입의 객체를 생성하기 위한 설계도를 클래스(Class)라고 하고, 설계도를 바탕으로 만들어진 객체를 그 클래스의 인스턴스(Instance)라고 합니다. 언어마다 클래스를 표현하는 방식은 다양하지만 클래스는 기본적으로 생성자라고 불리는 함수를 뼈대로 갖고 있습니다. 클래스의 생성자를 통해서 다양한 초기 값을 가진 인스턴스들을 만들 수 있습니다.

타입이 바로 클래스 생성자

```
// Array는 Array 타입의 인스턴스를 만들기 위한 생성자 함수입니다.
// 즉 Array()를 통해서 실행할 수 있는 함수입니다.
// 하지만 인스턴스를 만들기 위해서 생성자 함수를 실행할 때는 new 키워드를 앞에 붙혀줍니다.

var x = new Array(1, 2, 3);
var x2 = [1, 2, 3]; // Array 인스턴스를 간략하게 생성하는 축약 문법(리터럴)입니다.

var y = new Object();
y.a = 1;
y.b = 2;
var y2 = {a:1, b:2}; // Object 인스턴스를 간략하게 생성하는 축약 문법(리터럴)입니다.
```

클래스	객체	설명	리터럴
Object		모든 객체의 기본이 되는 객체의 클래스	{a:1,b:2}
Function		함수 객체의 클래스	function(){}
Array		배열 객체의 클래스	[1,2]
String		문자열 객체의 클래스	"abc"
Date		날짜 및 시간을 다루는 객체의 클래스	
Number		숫자 객체의 클래스	123
Boolean		논리적 참과 거짓을 다루는 객체의 클래스	true
RegExp		정규표현식을 다루는 객체의 클래스	/([a-z]*)/ig
	Math	수학 관련 값과 함수들이 담긴 단일 객체	
	JSON	JSON 포맷의 텍스트를 다루는 함수들이 담긴 단일 객체	

<표준 내장 객체>

JavaScript의 표준 내장 객체 (https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects) 중 자주 사용되는 일부는 위 표와 같습니다. 객체들의 주요한 속성들은 앞으로 Node.js와 JavaScript를 공부해 나가면서 천천히 눈에 익히도록 하겠습니다.

4. 클래스 정의하기

이제 우리가 직접 클래스와 생성자 함수를 정의하는 방법을 알아보겠습니다. 추상화하고자 하는 객체의 공통적인 속성과 인스턴스 별로 달라질 수 있는 초기값에 대해 생각해봅니다. 클래스의 이름은 첫글자를 대문자로, 인스턴스의 이름은 첫글자를 소문자로 하는 것이 이름을 구분하기에 좋습니다.

커스텀 클래스 정의하기

```
// 클래스 정의
class User{
  // 생성자
  constructor(name, type, email){

    // 인스턴스의 속성(값)
    this.name = name; // this는 생성될 인스턴스를 의미합니다.
    this.type = type;
    this.email = email;

    // 인스턴스의 속성(함수)
    this.speak2 = function(){
      console.log("I am "+this.name); // this는 이 함수를 호출하는 인스턴스를 의미합니다.
    }
  }

  // 인스턴스의 속성(함수)
  speak(){
    console.log("My name is "+this.name); // this는 이 함수를 호출하는 인스턴스를 의미합니다.
  }
}

// 인스턴스 생성
var user1 = new User('kim', 'admin', 'kim@benzen.io');
var user2 = new User('son', 'normal', 'son@benzen.io');

console.log(user1, user2);
user1.speak(); // My name is kim
user1.speak2(); // I am kim
user2.speak(); // My name is son
```

class 키워드는 최신 버전의 JavaScript인 ES6에서 등장한 키워드입니다.
Node.js에서는 사용해도 문제 될 일이 없지만, 추후 JavaScript 파트에서는 웹 브라우저 호환성을 위해서 사용하지 않도록 하겠습니다.

메소드

위의 user1.speak와 user1.speak2의 차이점에 대해 생각해봅시다. user1.speak2는 생성자 안에서 인스턴스에 직접 할당되는 함수이고, user1.speak는 클래스 안에서 정의된 유일한 함수입니다. 이때 user1.speak2와 user2.speak2는 메모리 상에서 각각 공간을 차지하는 코드가 됩니다. 반면에

user1.speak와 user2.speak는 유일한 함수를 인스턴스끼리 공유하게 됩니다. 즉 표면적인 차이는 크게 없지만 .speak 가 .speak2 에 비해서 효율이 높은 방식입니다. 또 이런 공통적인 인스턴스의 속성(함수)을 메소드(Method)라고 부릅니다.

클래스 속성과 인스턴스 속성

```
class Article{
  constructor(title, contents){
    // 인스턴스 속성(값)
    this.title = title;
    this.contents = contents;
    this.created = new Date();

    // 클래스 속성(값)인 list 배열에 생성된 인스턴스를 넣기
    Article.list.push(this);
  }

  // 인스턴스 속성(함수)
  print(){
    console.log(this.title, this.contents, this.created);
  }

  // 클래스 속성(함수)
  static printAll(){
    // 여기서 this는 인스턴스가 아닌 Article 클래스를 의미합니다.
    console.log("Total "+ this.list.length + " articles");
    for (var i=0; i < this.list.length; i++){
      var article = this.list[i];
      article.print();
    }
  }
}

// 클래스 속성(값)
Article.list = [];

// 인스턴스를 생성하는 부분
var article1 = new Article('title1', 'contents1');
var article2 = new Article('title2', 'contents2');
article1.print();
Article.printAll();

/**
title1 contents1 Tue Oct 18 2016 16:47:35 GMT+0900 (KST)
Total 2 articles
title1 contents1 Tue Oct 18 2016 16:47:35 GMT+0900 (KST)
title2 contents2 Tue Oct 18 2016 16:47:35 GMT+0900 (KST)
**/
```

인스턴스에 값이나 메소드를 할당하는 것은 자연스러워 보입니다. 이를 인스턴스 속성, 인스턴스 메소드라고 합니다. 그런데 클래스 자체에 값이나 메소드를 할당 할 수도 있습니다. 이는 static 속성, static 메소드, 클래스 속성, 클래스 메소드 등으로 부릅니다. 클래스 속성은 해당 클래스 전체와 관련된 기능을 담당하도록 구현하는 것이 적절하고, 인스턴스 메소드는 해당 인스턴스의 고유한 속성과 관련된 기능을 구현하는 것이 자연스럽습니다.

예를 들어, 게시물을 추상화한 Article이라는 객체의 클래스가 있고 그 인스턴스인 article1과 article2가 있습니다. 이때 지금 생성된 모든 게시물 객체들을 전부 출력하는 기능이 필요하다면 article1.printAll 보다는 Article.printAll이 자연스럽습니다. 반면 특정 게시물 한개를 출력하는 기능은 Article.print 보다는 article1.print가 자연스럽습니다. N번째 게시물을 출력하는 .printByIndex(N)는 Article.printByIndex(2)이 자연스럽습니다.

이처럼 인스턴스 속성은 개별 인스턴스에서 참조 할 수 있는 속성이고, 클래스 속성은 인스턴스 없이 참조 할 수 있는 속성입니다. 필요와 목적에 따라서 자연스럽게 구현하면 됩니다.

모델링 연습

```
class AccountBook {
  constructor(name, author) {
    this.name = name;
    this.author = author;
    this.list = [];
    this.total = 0;
    AccountBook.instances.push(this);
  }

  deposit(comment, amount) {
    if (this.total + amount < 0) {
      throw new Error(`Not enough balance for ${this.name}`);
    }
    this.total += amount;
    this.list.push({
      comment: comment,
      amount: amount,
    });
  }

  print() {
    var result = `===${this.name} by ${this.author}===\n`;
    for(var i=0; i < this.list.length; i++) {
      var item = this.list[i];
      result += `${item.amount < 0 ? '출금' : '입금'}\t${item.comment}\t${item.amount}
    }
    result += `===${this.total}===\n`;
    console.log(result);
  }
}
AccountBook.instances = [];
AccountBook.printAll = function() {
  for(var i=0; i < AccountBook.instances.length; i++) {
    var accountBook = AccountBook.instances[i];
    accountBook.print();
  }
};

var ac1 = new AccountBook('장부1', '김씨');
ac1.deposit('월급', 300);
ac1.deposit('집세', -150);
var ac2 = new AccountBook('장부2', '박씨');

AccountBook.printAll();
```

5. 상속

상속(Inheritance)은 다른 클래스의 속성들을 이어 받는 것입니다. 하위 클래스에서 어떤 상위 클래스를 상속 받게 되면 상위 클래스의 속성들을 그대로 사용 할 수도 또 덮어 쓸 수도 있습니다. 상속은 객체간의 상하 관계를 표현하고 비슷한 클래스들을 묶어 추상화하고 재사용성을 높히기 위해서 사용합니다.

예를 들어 여행 정보와 관련된 프로그램에서 버스와 비행기, 선박의 운송 수단이 있다고 할 때, 이 셋 모두 운송 수단이라는 공통점이 있습니다. 이 때 운송 수단이 갖는 공통적인 속성들을 기반으로 운송 수단이라는 상위 클래스를 모델링하면 견고한 추상화를 꾀할 수 있습니다.

상속과 다형성

```
class Transportation {
  constructor(id, capacity){
    this.id = id;
    this.capacity = capacity;
  }

  getPrice(){
    return 0;
  }
}

class Airplane extends Transportation { // extends 키워드를 이용합니다.
  constructor(id, capacity, seatClass){
    super(id, capacity); // super는 Transportation 클래스의 생성자를 의미합니다.
    this.seatClass = seatClass;
  }

  getPrice(){ // 상속받은 메소드를 덮어 쓰기
    var price = Airplane.prices[this.seatClass];
    if (!price) { // price가 undefined거나 0이거나 null이거나
      return -1;
    } else {
      return price;
    }
  }
}
Airplane.prices = {
  I: 100,
  B: 300,
  F: 500
};

class Ship extends Transportation {
  constructor(id, capacity, isCruise){
    super(id, capacity);
    this.isCruise = isCruise;
```

```
  }

  getPrice(){ // 상속받은 메소드를 덮어 쓰기
    return (this.isCruise) ? 200 : 50; // 삼항 연산자
  }
}

var air1 = new Airplane("747", 20, "F");
var air2 = new Airplane("747", 50, "B");
var ship1 = new Ship("Cruise88", 300, true);
var ship2 = new Ship("Ship39", 150, false);

var list = [air1, air2, ship1, ship2];
for(var i=0; i < list.length; i++) {
  // trans는 각기 Ship 혹은 Airplain 클래스의 인스턴스이지만, 모두 다 Transportation의 인스턴스는
  var trans = list[i];
  console.log(trans.id, trans.capacity, trans.getPrice());
}

/**
747 20 500
747 50 300
Cruise88 300 200
Ship39 150 50
**/
```

위의 .getPrice()를 Airplane과 Ship에서 다르게 구현한 것처럼, 내부적인 로직은 다르더라도 표면적인 형태(함수의 인자나 리턴 값의 타입)가 동일한 성질을 다형성(Polymorphism)이라고 합니다.

모델링 연습

```
class Figure {
  getSize() {
    return 0;
  }
}

class Oval extends Figure {
  constructor(radius1, radius2){
    this.radius1 = radius1;
    this.radius2 = radius2;
  }

  getSize() {
    return this.radius1 * this.radius2 * Math.PI;
  }
}

class Circle extends Oval {
  constructor(radius) {
    super(radius, radius);
  }
}

class Rect extends Figure {
  constructor(width1, width2, height) {
    this.width1 = width1;
    this.width2 = width2;
    this.height = height;
  }

  getSize() {
    return ((this.width1 + this.width2) / 2 ) * this.height;
  }
}

class Rectangle extends Rect {
  constructor(width, height) {
    super(width, width, height);
  }
}

class Square extends Rect {
  constructor(width) {
    super(width, width);
  }
}
```

대부분의 고급(High-Level) 언어에서는 OOP를 지원하고 있습니다. 또한 기본적인 클래스와 상속의 개념에서 발전해서, 언어별로 이런 저런 키워드를 통해 추상 클래스, 인터페이스, 어댑터, Trait 등의 추가적인 개념을 제공하고 있습니다. 여기서 OOP를 더 깊게 다루지는 않습니다. 하지만 **모든 원리는 추상화**

와 재사용성에 있습니다. 추상화와 재사용성의 원리에 입각해서 모델링을 하다가, 또 다른 도구의 필요성을 느끼신다면 그 때 OOP의 추가적인 개념들에 대해서 공부하셔도 충분하실 것으로 생각합니다.

[1] OOP, Object Oriented Programming
[2] Copy by value
[3] Copy by reference
[4] Class
[5] Instance
[6] Method
[7] Inheritance
[8] Polymorphism

2 프로그래밍 연습

2.7 타입과 유추, 명명 규칙

1. 타입
2. 명명 규칙
3. 타입 유추

> 타입을 유추해 코드를 파악하는 연습을 해봅니다. 이름을 짓는 관습에 대해 알아봅니다.

1. 타입

지금까지 이름은 주소를 가리키고, 그 주소가 가리키는 메모리에는 특정 값이 있으며, 그 값은 특정 타입(Type)에 속해있다고 했습니다. 타입은 메모리에 위치하고 있는 값이 구성되어 있는 방식이라고 할 수 있습니다. OOP적인 관점에서는 메모리에 위치한 값이 어떤 설계도, 즉, 어떤 클래스의 인스턴스인지 라고 생각 할 수 있겠습니다.

<Type>

코딩에 있어서 타입에 신경써야 하는 이유는, 우리는 코드를 읽으면서 단순히 변수의 이름 뿐만 아니라 그 이름이 가리키고 있는 숨겨진 문맥을 읽을 수 있어야하기 때문입니다. 타입을 읽을 수 있다면, 어떤 이름이 가리키는 객체가 어떤 속성들을 지녔는지를 알 수 있고, 어떤 이름이 함수라면 그 함수가 무슨 타입의 인자들을 원하는 지 알 수 있습니다. 이를 통해서 생산성을 높힐 수 있고, 주어진 함수를 잘못 사용하는 오류를 방지 할 수 있습니다.

<Typed Languages>

언어/플랫폼에 따라서 이러한 타입을 다루는 관점이 크게 다를 수 있습니다. 이는 크게 두가지 방식으로 구분 할 수 있습니다.

Strongly vs. Weakly Typed Language

```
-- Strongly
"1" + 1; // Error!
(Number)"1" + 1;


-- Weakly
"1" + 1;
```

먼저 특정 연산에서 **형 변환이 암묵적으로(컴파일러가 알아서) 이루어 질 수 있는가**의 여부에 따라서 강하게, 약하게 타입이 정해진 언어라고 구분 할 수 있습니다. Strongly Typed 경우에는 형 변환이 필요한 경우엔 항상 개발자가 명시적으로 코드를 작성해야 합니다.

Statically vs. Dynamically Typed Language

```
-- Statically
Number x = 10;
String y = "ten";
Array<Number> z = [1,2,3,4];
Number function sum(Number a, Number b) {
  return a + b;
}
Number result = sum(x, y); // Error!


-- Dynamically
var x = 10; // Number
var y = "ten";
var z = [1,2,3,4];
function sum(a, b) {
  return a + b;
}
var result = sum(x, y); // No Error!
```

또한 이름 자체에 타입이 고정되어 있는가에 따라서 정적, 동적 타입 언어로 구분 할 수 있습니다. 동적 타입 언어는 개발의 편리나 속도를 향상 시킬 수 있지만, 컴파일러가 타입 체킹을 완벽히 하지 못하기 때문에 잠재적인 런타임 오류의 위험이 있습니다. 반면에 정적 타입 언어는 컴파일 타임에서 타입 오류를 검증 할 수 있어 안정적이지만 코드량이 늘고, 로직이 유연하지 못하다는 단점이 있습니다.

이러한 관점에서 보면 JavaScript는 Weakly, Dynamically Typed Language 라고 할 수 있습니다. 또한 언어마다 타입을 다루는 차이는 언어의 좋고 나쁨, 쉽고 어려움을 의미하지 않습니다. 단순한 도구의 차이임을 이해하고, 필요에 따라 익히고 쓰시면 되겠습니다.

2. 명명 규칙

명명 규칙(Naming Convention)은 영문 이름을 작성하는 관례를 말합니다. 크게 캐멀케이스(camelCase), 스네이크_케이스(snake_case), 파스칼케이스(PascalCase) 세가지 방식이 있습니다. 이 표기법은 여러 단어로 이루어진 이름의 경우 띄어쓰기 대신에 언더바(_)를 삽입하거나 단어의 첫글자를 대문자로 표기하는 방식입니다. 예를 들어 set color, append child 같은 단어를 setColor, appendChild로 표기합니다. JavaScript에서 함수나 메소드, 속성, 변수들의 이름은 이처럼 camelCase를 따르는 것이 관습적입니다. 그리고 클래스의 이름이나 단일 객체의 이름은 보통

PascalCase를 씁니다.

또한 함수의 이름은 setColor, appendChild 처럼 동사형을 쓰고, 값이나 객체의 이름은 color, children 처럼 적절한 명사형을 쓴다면 좀 더 코드를 유추하는데 도움이 됩니다.

관례에 맞게 이름 짓기

```
// 아래처럼 작성해도 문법상의 오류는 전혀 없습니다.
class user{
    constructor(Name){
        this.Name = Name;
    }

    PrintUserName(){
        console.log(this.Name);
    }
}
var User = new user("abc");
User.PrintUserName();

// 다만 위처럼 쓰기 보다는 아래처럼 씁니다.
// 이런 관습을 지킴으로써 타인의 코드를 유추하기가 더 쉬워집니다.
class User{
    constructor(name){
        this.name = name;
    }

    printUserName(){
        console.log(this.name);
    }
}

var user = new User("abc");
user.printUserName();

// 고정된 값(상수)은 모두 다 대문자인 SNAKE_CASE로 쓰는 경우도 있습니다.
var SOME_CONSTANT = "./files/path/fixed";
```

3. 타입 유추

협업을 할 때나, 라이브러리의 API 문서를 읽는 등 타인의 코드를 읽고 이해해야 하는 경우는 적지 않습니다. 그렇기 때문에 남이 작성한 코드를 읽고 이해하는 능력은 중요합니다.

또한 수 많은 프로그래밍 언어가 있지만 언어별로 문법(Syntax)의 차이는 크지 않습니다. 이름과 값, 객체, 함수, 제어문 등 기본 부품을 근거로 처음보는 코드도 해석 할 수 있어야 합니다.

1. 이름이 대문자면 클래스이거나 단일 객체일 확률이 높다.
2. 이름이 소문자면 함수, 값이거나 인스턴스일 확률이 높다.
3. 이름(...) 꼴은 어떤 함수다.
4. new 이름(...) 꼴은 어떤 클래스다.
5. 이름 뒤에 .someThing 또는 ["someThing"]이 붙으면 어떤 객체다.
6. 이름 뒤에 [Number]이 붙으면 Array 또는 String이다.
7. ...등등

코드 유추하기

```
1  var obj1 = new Something("a","b");
2  // Something은 클래스고, obj1은 Something 타입의 객체다.
3  // 또 Something의 생성자는 인자로 String 2개를 받을 수 있다.
4
5  obj1.doThing();
6  // Something 타입 객체는 doThing이라는 인자를 받지 않는 인스턴스 메소드를 갖는다.
7
8  obj1.asyncTask(function(data){
9    console.log(data+1);
10 });
11 // Something 타입 객체는 asyncTask라는 인스턴스 메소드를 갖는다.
12 // 또 asyncTask 메소드는 인자로 콜백 하나를 받고, 그 함수는 Number와 + 연산이 가능한 인자 하나를 받는다
13
14 obj1.doSomething().doThing().attr = 12;
15 // Something의 doSomething 인스턴스 메소드는 doThing이라는 함수를 속성으로 갖는 객체를 리턴한다.
16 // 그런데 5번 줄에서 doThing이 obj1의 메소드였으니 doSometing은 아마도 호출한 인스턴스를 다시 리턴하는
17 // 또 doThing이라는 함수는 attr라는 Number를 속성으로 갖는 객체를 리턴한다.
```

위 14번줄 코드처럼 메소드를 실행하고 다시 바로 메소드를 실행하는 패턴을 메소드 체이닝(Method Chaining)이라고 합니다. 어떤 객체에 대해 연속적인 처리를 할 때, 가독성을 높히기 위해서 구현 할 수 있는 방식입니다.

메소드 체이닝

```
class Ball{
  constructor(x, y){
    this.x = x;
    this.y = y;
  }

  moveRight(amount){
    this.x += amount || 1; // amount가 undefined거나 0이면 1 아니면 amount
  }

  moveLeft(amount){
    this.x -= amount || 1;
  }

  moveUp(amount){
    this.y += amount || 1;
  }

  moveDown(amount){
    this.y -= amount || 1;
  }
}

// 이 때 공을 왼쪽으로 10, 위로 10 옮기려면
var ball = new Ball(0,0);
ball.moveLeft(10);
ball.moveUp(10);


// 메소드 체이닝을 구현하려면
class Ball2{
  constructor(x, y){
    this.x = x;
    this.y = y;
  }

  moveRight(amount){
    this.x += amount || 1;
    return this; // 인스턴스를 다시 리턴한다.
  }

  moveLeft(amount){
    this.x -= amount || 1;
    return this; // 인스턴스를 다시 리턴한다.
  }

  moveUp(amount){
    this.y += amount || 1;
    return this; // 인스턴스를 다시 리턴한다.
  }

  moveDown(amount){
    this.y -= amount || 1;
    return this; // 인스턴스를 다시 리턴한다.
```

```
    return this; // 인스턴스를 다시 리턴합니다.
  }
}

// 다시 공을 왼쪽으로 10, 위로 10 옮기려면
var ball2 = new Ball2(0,0);
ball2.moveLeft(10).moveUp(10);
```

사전 지식 없이 코드 읽기

```
var express = require('express');
var router = express.Router();
var fs = require('fs');
var path = require('path');
var bodyParser = require('body-parser');

router.get('/', (req, res) => {
  res.render('bbs/list', {
    posts: loadPosts()
  });
});

// GET /bbs/write
router.get('/write', (req, res) => {
  res.render('bbs/write', {
  });
});

// POST /bbs/write
router.post('/write', bodyParser.urlencoded(), (req, res) => {
  if (req.body.title == "") throw new Error("No Title");
  else if (req.body.password != "1234") throw new Error("Invalid Password");

  var id = savePost({
    title: req.body.title,
    subtitle: req.body.subtitle,
    contents: req.body.contents,
    created: new Date().toString()
  });

  res.redirect('/bbs/'+id);
});

module.exports = router;
```

[1] Type

[2] Naming Convention

[3] camelCase

[4] snake_case

[5] PascalCase

[6] Method Chaining

2 프로그래밍 연습

2.8 함수형 프로그래밍, 콜백과 클로저

1. 함수도 값이다
2. Array의 메소드
3. 콜백
4. 클로저

> 함수를 값처럼 사용하고 함수의 응용을 강조하는 프로그래밍 패러다임에 대해 알아보고, 콜백(Callback), 클로저(Closure)에 대해서 알아봅니다.

1. 함수도 값이다

<합성 함수>

함수를 값으로 다루기

```
var sum = function(a,b){ return a+b; };
func1(1);
func2("abc", 123);
func3(sum);
func4("abc", sum);
func5("xxx", func3, function(){ ... });
func6(123, function(){ ... }, function(){ ... });
```

JavaScript에서는 함수 역시 객체, Function 타입의 인스턴스입니다. 변수에 할당을 할 수도 있고, 다른 함수를 실행할 때 인자로 넘길 수도 있습니다. 함수형 프로그래밍은 이런식으로 함수의 응용을 강조하는 일종의 패러다임입니다.

좀 더 이론적으로 말하자면, 함수형 프로그래밍은 입력 값을 목적 값을 향해서 순차적으로 변화시키기 (명령형 프로그래밍) 보다는, 함수의 합성을 응용하여 입력 값으로 부터 목적 값을 생성하는 프로그래밍 패러다임을 말합니다.

함수형 프로그래밍은 수학적인 사고방식을 표현하기에 적합하며, 복잡한 로직을 단순하게 표현 할 수 있고, 또 일반적으로 입력 값에 대한 변화를 주지 않고(Immutable), 함수를 통해 새로운 결과를 생성하기 때문에 프로그램의 안정성을 향상 시킬 수 있다는 장점이있습니다.타입 하지만 현실적으로는 이러한 함수형 스타일과 명령형, 객체지향적 스타일이 뒤섞여 쓰이는 것이 일반적입니다.

함수의 축약 문법

```
var func1 = function(x, y){
  console.log(x, y);
};

var func2 = (x, y) => {
  console.log(x, y);
};

var func3 = x => { // 인자가 하나면 주머니를 생략 가능
  console.log(x);
};

var func4 = x => console.log(x);
// 함수내 코드가 한줄이면 블록을 생략 가능
// 이때는 그 한줄의 코드를 계산한 값이 리턴됩니다.

var func5 = x => x + 1;

var func6 = function(x) {
  return x + 1;
};
// 즉 func5와 func6의 기능은 동일합니다.
```

JavaScript의 최신 버전인 ES6에서는 Arrow Function이라고 하는 함수를 정의하는 새로운 방법이 도입되었습니다. Arrow Function 객체는 Function 객체와 기능적으로 거의 동일합니다. 그 약간의 차이에 대해서는 추후 알아보겠습니다. 또한 Node.js 파트에서는 Arrow Function을 사용해도 문제가 없습니다만, 추후 JavaScript 파트에서는 웹 브라우저 호환성을 위해서 Arrow Function을 지양합니다.

말이 되는 코드

```
class Obj{
  constructor(name){
    this.name = name;
  }
}
var objs = [new Obj("a"), new Obj("b"), new Obj("c")];

// 사람보다는 기계에게 친숙한 표현
for(var i=0; i < objs.length; i++) {
  var obj = objs[i];
  console.log(obj.name);
}

// 조금 더 사람에게 말이되는 코드?
objs.forEach(obj => {
  console.log(obj.name);
});
```

2. Array의 메소드

우리가 앞으로 프로그램을 작성하는 데 Collection을 다루는 경우는 비일비재합니다. 아래에서 Array의 forEach 같은 주요한 메소드에 대해 알아보면서 functional 프로그래밍에 대한 감을 익혀보도록 하겠습니다.

```
var arr = [
  {x:1, y:'hello1'},
  {x:2, y:'hello2'},
  {x:5, y:'hello5'},
  {x:3, y:'hello3'},
  {x:4, y:'hello4'},
];

// forEach: arr의 원소들에 반복적으로 같은 작업하기
for(var i=0; i<arr.length; i++) console.log(arr[i].x);
arr.forEach(obj => console.log(obj.x));


// map: arr의 원소들에 반복적으로 같은 변화를 준 새로운 arr
var arr2 = [];
for(var i=0; i<arr.length; i++) {
  var obj = arr[i];
  var str = obj.x + obj.y;
  arr2.push(str);
}
var arr2 = arr.map(obj => obj.x + obj.y);


// filter: arr의 원소들 중에 특정 로직을 만족하는 원소만 걸러낸 새로운 arr
var arr2 = [];
for(var i=0; i<arr.length; i++) {
  var obj = arr[i];
  if (obj.x > 3) arr2.push(obj);
}
var arr2 = arr.filter(obj => obj.x > 3);


// some: arr의 원소들 중에 특정 로직을 만족하는 원소가 하나라도 있는지?
var bool = false;
for(var i=0; i<arr.length; i++) {
  var obj = arr[i];
  if (obj.x > 3) {
    bool = true;
    break;
```

```
  }
}
var bool = arr.some(obj => obj.x > 3);

// every: arr의 원소들이 모두 특정 로직을 만족하는지?
var bool = true;
for(var i=0; i<arr.length; i++) {
   var obj = arr[i];
   if (!(obj.x > 3)) {
    bool = false;
    break;
   }
}
var bool = arr.every(obj => obj.x > 3);

// find: arr의 원소들 중에 특정 로직을 만족하는 첫번째 원소
var obj = null;
for(var i=0; i<arr.length; i++) {
   var o = arr[i];
   if (o.x > 3) {
    obj = o;
    break;
   }
}
var obj = arr.find(o => o.x > 3);

// findIndex: arr의 원소들 중에 특정 로직을 만족하는 첫번째 원소의 인덱스
var objIndex = -1;
for(var i=0; i<arr.length; i++) {
   var o = arr[i];
   if (o.x > 3) {
    objIndex = i;
    break;
   }
}
var objIndex = arr.findIndex(o => o.x > 30);
console.log(objIndex);

// sort: arr의 원소들을 특정 로직에 따라 정렬한 새로운 arr
var arr2 = [];
for(var i=0; i<arr.length; i++)
  arr2.push(arr[i]);

for(var i=0; i<arr2.length; i++) {
  for(var j=i+1; j<arr2.length; j++) {
    if (arr2[i].x < arr2[j].x) {
      var temp = arr2[i];
      arr2[i] = arr2[j];
      arr2[j] = temp;
    }
  }
}

var arr2 = arr.sort((a,b) => a.x < b.x ? 1 : (a.x == b.x ? 0 : -1));
console.log(arr2);
```

```
// indexOf: 단순한 비교로 Array의 인덱스를 찾는 메소드
var arr = [1,2,3,4,5];
var index = arr.findIndex(o => o == 3); // 2
var index = arr.indexOf(3); // 2

// concat: Array를 이어 붙히기
var arr1 = [1,2,3];
var arr2 = arr1.concat(arr1, [1,2,3], [4,5,6]);
console.log(arr2);

// slice: Array를 특정 구간만 잘라내기
var arr1 = [1,2,3,4,5];
var arr2 = arr1.slice(0,-2);
console.log(arr2);

// splice: Array의 특정 구간을 바꿔치우기
var arr1 = [1,2,3,4,5];
var arr2 = arr1.splice(0, 4, 1,2,3,4,5,6,7,8,8);
console.log(arr1, arr2);

// join: Array를 String으로 합성하기
var arr1 = ['abc', 'gmail.com', 'xxx'];
console.log(arr1.join('_'));

// split: String을 분리해 Array로 만들기
var arr2 = '010-1234-1234'.split('-').join('.');
console.log(arr2);



var numbers = [1,2,3,4,5,6,7,8,9];

// reduce: arr의 원소들을 특정 로직에 따라 누적하며 계산해 나간 결과 값

var result = 0;
for(var i=0; i<numbers.length; i++)
  result += numbers[i];

console.log(result);
// 0 1 => 1
// 1 2 => 3
// 3 3 => 6
// ...

var result = numbers.reduce((result, number) => {
  return result + number;
}, 0);
console.log(result);


var result = numbers
  .map(num => num*num)
  .reduce((result, num) => result * num, 1);

console.log(result);
```

```


var numbers2 = [
  [1,2,3],
  [6,5,4],
  [7,8,9],
];

// [9,8,7,6,5,4,3,2,1]
// "9-8-7-6-5-4-3-2-1"
// reduce, concat, sort, join

var numbers3 = numbers2
  .reduce((result, arr) => result.concat(arr), [])
  .sort()
  .reverse()
  .join('-');
console.log(numbers3);



var orders = [
    {
      id:1,
      paid: true,
      amount: 3000,
    },
    {
      id:4,
      paid: false,
      amount: 100,
    },
    {
      id:3,
      paid: true,
      amount: 800,
    },
    {
      id:2,
      paid: false,
      amount: 3000,
    },
    {
      id:5,
      paid: true,
      amount: 2500,
    },
  ];


// 주문을 id 순서로 정렬하여 그 금액만을 나열하기
var amounts = orders
  .sort((a,b) => {
    if (a.id > b.id) return 1;
    return -1;
  })
  .map(order => order.amount);
```

```
console.log(amounts);

// 금액이 1000 이상이고 paid가 참인 주문의 금액의 합계
var total = orders
  .filter(order => order.paid && order.amount > 1000)
  .reduce((result, order) => result + order.amount, 0);
console.log(total);

```

Array의 주요한 메소드와 속성
(https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/A

3. 콜백

콜백(Callback)이란 인자로 전달되는 실행 가능한 코드, 즉 함수를 뜻하는 별명입니다. 콜백을 넘겨 받은 함수는 콜백을 바로 실행할 수도 있고, 아니면 필요에 따라 나중에 실행할 수도, 실행하지 않을 수도 있습니다.

콜백 f를 x번 실행하는 함수

```
function repeatFunc(f, x) {
  for(var i=0; i < x; i++)
    f();
}

// 사용 예시1
function printHelloWorld() {
   console.log("Hello world!");
}
repeatFunc(printHelloWorld, 3);

/*
Hello world!
Hello world!
Hello world!
*/

// 사용 예시2
repeatFunc(function(){
   console.log("Hey Hey!");
}, 2);

/*
Hey Hey!
Hey Hey!
*/
```

특히 본질적인 의미의 콜백은 인자로 넘어간 함수 중에서도 콜백을 받은 함수가 **복잡한, 오래 걸리는 로직을 처리한 뒤에 계산 결과를 콜백에 넘기면서 호출하는 경우**를 일컫습니다. 콜백 함수와 **오래 걸리는 로직?**에 대해서는 나중에 '동기와 비동기'에서 다시 다루도록 하겠습니다.

```
function downloadImage(url, callback){
  var image = ...;

  // 다운로드 로직...

  callback(image);
}

downloadImage('http://..', function(image) {
  // 이미지를 가지고 처리하는 로직...
});
```

> 콜백이 또 콜백을 받고, 그 콜백이 또 콜백을 받고.. 함수가 여러번 중첩되면 가독성이 떨어지는 코드가 됩니다. JavaScript에서는 콜백의 간결한 표현과 제어를 위해서 Promise (https://www.npmjs.com/package/promise)라는 객체를 제공합니다. Promise는 추후 4장 HTTP와 웹 서버에서 다룹니다.

4. 클로저

위의 콜백 f를 x번 실행하는 함수는 repeatFunc(printHelloWorld, 3)와 같이 실행시 콜백을 3번 반복해서 실행하며 리턴 값은 없습니다. 이때 콜백 f를 x번 실행하는 것이 아니라, 콜백 f를 x번 실행하는 함수 자체를 리턴하는 함수를 만들 수 있을까요?

콜백 f를 x번 실행하는 함수를 생성하는 함수

```
function makeRepeatFunc(f, x) {
  return function() {
    for(var i=0; i < x; i++)
      f();
  };
}

// 사용 예시1
function printHelloWorld() {
   console.log("Hello world!");
}
var printHelloWorld3 = makeRepeatFunc(printHelloWorld, 3);
printHelloWorld3();

/*
Hello world!
Hello world!
Hello world!
*/

var printHelloWorld5 = makeRepeatFunc(printHelloWorld, 5);
printHelloWorld5();

/*
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
*/
```

위처럼 합성 함수의 개념으로 함수 자체를 만들어내는 함수를 만들 수 있습니다. 이런 함수를 팩토리 함수(Factory Function)라는 별명으로 부르기도 합니다. 또한 이 때 생성된 함수를 클로저(Closure)라고 부릅니다.

<클로저의 스코프>

생성된 함수 printHelloWorld3를 자세히 보면 함수 내에서 makeRepeatFunc의 스코프 내에 있는 인자 f와 x에 접근하고 있습니다. 이렇게 내부에서 생성된 함수(printHelloWorld3)가 외부 스코프의 변수를 참조하는 경우에는, 외부 함수(makeRepeatFunc)가 종료되어도 외부 스코프의 변수가 해제되지 않습니다. 이렇게 외부의 닫힌 스코프(makeRepeatFunc)의 변수를 참조하는 함수(printHelloWorld3)를 클로저라고 합니다.

여기서 핵심은 클로저의 개념보다는, 런타임에서 코드(함수)를 생산해 낼 수 있는 팩토리 함수라는 개념입니다. JavaScript같은 고수준 스크립트 언어에서는 클로저 같은 방식으로 런타임에 동적으로 코드를 생성 할 수 있습니다. 이 개념은 앞으로 반복적으로 다루도록 하겠습니다.

[1] Immutable
[2] Arrow Function
[3] Callback

3 웹 프론트엔드

1. 웹 브라우저 ... 116
2. HTML ... 120
3. CSS ... 129
4. JavaScript ... 145
5. 모델링 ... 158
6. 이벤트 시스템 ... 172
7. jQuery ... 190
8. 확장성있는 코드짜기 ... 195

웹 브라우저라는 플랫폼에 대해 알아보고, 웹 페이지를 구성하는 HTML, CSS, JavaScript에 대해 배웁니다. 또한 이벤트 시스템, 위임(Delegation)과 같은 소프트웨어의 디자인 패턴에 대해 알아봅니다. 또 Bootstrap, jQuery 같은 웹 프론트엔드 라이브러리에 대해서 알아봅니다.

3 웹 프론트엔드

3.1 웹 브라우저

1. 웹 브라우저
2. 개발자 도구

> 클라이언트 프로그램인 웹 브라우저의 역할에 대해서 알아봅니다.

1. 웹 브라우저

<브라우저>

인터넷의 대표적인 서비스인 웹은 서버-클라이언트 기반의 서비스입니다. 이때의 서버는 인터넷 상의 수 많은 웹 서버들에서 실행중인 서버 프로그램이며, 클라이언트는 사용자들이 이용하는 웹 브라우저입니다. 운영체제에 따라서 다양한 웹 브라우저가 있고, 또 같은 운영체제에서도 다양한 웹 브라우저들이 있습니다. 이 때문에 웹 서비스를 개발할 때는 웹 브라우저 호환성에 대해서 고민해 볼 필요가 있습니다. 수업에서는 구글의 크롬(Chrome) (https://www.google.co.kr/chrome/browser/desktop/)을 사용합니다.

> ... 월드 와이드 웹(World Wide Web, WWW, W3)은 인터넷에 연결된 컴퓨터들을 통해 사람들이 정보를 공유할 수 있는 전 세계적인 정보 공간을 말한다. 간단히 웹(Web)이라 부르는 경우가 많다. 이 용어는 인터넷과 동의어로 쓰이는 경우가 많으나 엄격히 말해 서로 다른 개념이다. 웹은 전자메일과 같이 인터넷 상에서 동작하는 하나의 서비스일 뿐이다. 그러나 1993년 이래로 웹은 인터넷 구조의 절대적 위치를 차지하고 있다... (위키백과)

<웹 브라우저와 웹 서버>

웹 브라우저의 역할

이번 3장에서는 서버쪽은 잠시 내려두고, 웹 브라우저에 대해서만 공부합니다. 웹 브라우저는 HTTP나 HTTPS 프로토콜을 이용해서 특정 웹 서버에 사용자의 요청 데이터를 보내고, 서버의 응답 데이터를 받아 처리해주는 프로그램입니다. 웹 브라우저의 역할을 간단히 나타내면 다음과 같습니다.

1. 요청(Request) 보내기

   사용자가 주소창에 입력한 URI(Uniform Resource Identifier)을 해석하고 해당 웹 서버로 HTTP 프로토콜을 따라 요청 데이터를 보냅니다.

2. 응답(Response) 처리

   서버에서 받은 응답 데이터의 컨텐츠 타입(Content-Type)에 따라서 데이터를 처리합니다. 즉, 그림을 그리거나, 영상을 재생하거나, 파일을 다운로드 하거나, 텍스트나 XML, HTML 문서를 렌더링합니다.

> 통합 자원 식별자(Uniform Resource Identifier, URI)는 인터넷에 있는 자원을 나타내는 유일한 주소이다.
> URI의 존재는 인터넷에서 요구되는 기본조건으로서 인터넷 프로토콜에 항상 붙어 다닌다.
> 프로토콜 (HTTP, HTTPS, FTP 등) + : + // + 호스트이름 + 주소
> 예: http://ko.wikipedia.org
> 위의 프로토콜 형태는 URI의 한 종류인 URL 형식의 표현 방법따른 예 입니다.
> URI의 하위개념으로 URL, URN 이 있다... (위키백과)

예시

1. 브라우저 주소창에 https://www.google.com 이라는 주소를 입력
2. 브라우저는 해당 주소의 자원을 요청하는 데이터를 HTTP 프로토콜을 따라 작성하고 구글의 웹 서버에 전송
3. 구글의 웹 서버는 요청을 받아 브라우저에게 HTML 문서를 전송
4. 브라우저는 HTML 문서를 받아 해석하고 렌더링

2. 개발자 도구

<크롬 개발자 도구>

웹 브라우저와 웹 서버간에 실제로 어떤 HTTP 요청이 오가며, 실제 데이터는 어떤 내용인지 확인해 볼 수 있습니다. 크롬의 경우 (윈도우에서) F12를 누르면 개발자 도구가 나타나고 네트워크(Network) 탭에서 브라우저가 주고받는 데이터를 확인 할 수 있습니다. 또한 Elements나 Sources 탭에서는 현재 보고 있는 페이지가 렌더링되기 전의 소스코드를 확인할 수 있습니다.

[1] WWW, World Wide Web
[2] URI, Uniform Resource Identifier
[3] Content-Type

3 웹 프론트엔드

3.2 HTML

1. HTML 기초
2. 주요 태그
3. 주요 특성
4. HTML 문서 쓰고 읽기

웹 문서를 구성하는 HTML을 이해하고, 주요 태그와 특성들에 대해 알아봅니다.

1. HTML 기초

웹 브라우저는 다양한 응답 데이터를 처리 할 수 있지만, HTML 문서를 해석하고 렌더링하는 것이 기본적인 기능입니다. HTML의 개념과 구조에 대해서 친숙해질 필요가 있습니다. HTML(Hypertext Markup Language)은 프로그래밍 언어라기보다는 웹 문서를 표현하기 위해 고안된 문서의 포맷입니다.

HTML 문서

```html
<!-- HTML 문서의 시작을 알리는 태그 -->
<html>
    <!-- 렌더링 되지 않는 부분 -->
    <head>
    <title>문서의 제목</title>
    </head>

    <!-- 실제 렌더링 되는 부분 -->
    <body>
        <h1>큰 제목</h1>
        <p>문단1</p>
        <p>문단2</p>
        <img src="http://workshop.benzen.io/assets/img/logo-pink.svg">
    </body>

</html>
<!-- HTML 문서의 끝을 알리는 태그-->
```

PDF에서는 미리보기가 지원되지 않습니다.

1. HTML 문서는 HTML 요소(Element)들로 구성

   HTML 문서는 요소들이 레고 블록과 같이 조립된 형태로 이루어져있습니다.
   요소는 다른 요소를 포함하여 상하 관계를 이루기도 합니다.

2. HTML 요소(Element)는 태그(Tag)로 표현

   태그들은 Paragraph(문단), Table(표), Heading(제목), Image(이미지)와 같은 특정 요소들을 표현합니다.
   태그는 부가적인 특성(Attribute)을 가질 수 있습니다.

3. 웹 브라우저는 HTML을 렌더링

   웹 브라우저는 HTML 태그를 그대로(텍스트로) 보여주는 것이 아니라, 태그들의 구조와 특성을 해석(Parsing)하여 그래픽으로 보여줍니다.

짝 태그와 홀 태그

1. 짝 태그(Container Tag)

   시작 태그와 끝 태그로 이루어진 태그를 뜻합니다. 내부에 다른 태그를 포함 할 수 있습니다.
   <tagName attribute1="특성1" attribute2="특2">...</tagName>처럼 적습니다.

```
<div>
    <span>
        <p>Container Tags</p>
    </span>
</div>
```

2. 홀 태그(Empty Tag)

   단독으로 사용하는 태그를 뜻 합니다. 내부에 다른 태그를 포함 할 수 없습니다.
   <tagName attribute1="특성1" attribute2="특2">처럼 적습니다.

```
<img src="../images/photo.png">
<br>
<br />
<!-- 홑태그 끝에 / 가 올 수도 있습니다 -->
```

Display 속성

HTML 태그들은 렌더링될 때 문서의 공간을 차지하는 방식에 따라 Display 속성을 갖습니다. 태그에 따라 기본적인 Display 속성이 정해져 있습니다. 하지만 이 속성을 명시적으로 변경 할 수도 있습니다.

1. 인라인 태그(Inline Tag)

   인라인 태그는 줄을 바꾸지 않고 다른 요소와 같은 줄에 위치하려는 성향의 태그입니다.

```
<a href="http://some/url">어떤 링크</a>
<button>버튼</button>
<input type="text">
<input type="password">
```

PDF에서는 미리보기가 지원되지 않습니다.

2. 블록 태그(Block Tag)

   블록 태그는 한 줄의 공간을 차지하려는 성향의 태그입니다.

```html
<h1>Block Tags</h1>
<div>
    <form>
        <input type="text">
    </form>
</div>
<p>New Line</p>
<table border="1">
    <tr>
        <td>ROW1-COL1</td>
        <td>ROW1-COL2</td>
    </tr>
    <tr>
        <td>ROW2-COL1</td>
        <td>ROW2-COL2</td>
    </tr>
</table>
```

PDF에서는 미리보기가 지원되지 않습니다.

2. 주요 태그

태그명	설명	형태	display
<html>	HTML 문서의 시작을 나타냄	짝	block
<head>	문서 자체에 대한 정보를 나타냄 <link> <title> <meta> 태그 등을 포함	짝	none
<body>	화면에 보여질 태그의 시작을 나타냄	짝	block
<table>	표를 나타냄 행을 <tr> 태그로, 그 안에 열을 <td> 태그로 나타냄	짝	block
	Division; 공간을 분할 함	짝	block
<h1>	Heading; 큰 글씨를 표현 (h1 ~ h6)	짝	block
<a>	Anchor; 링크를 거는 태그	짝	inline
<p>	Paragraph; 문단을 나타냄	짝	block

<ul>	Unordered-list; 리스트 항목을 나타내는 <li> 태그를 포함	짝	block
<ol>	Ordered-list; 번호가 매겨지는 리스트 항목을 나타내는 <li> 태그를 포함	짝	block
<img>	이미지를 그리는 태그	홀	inline
<embed>	다른 응용프로그램이나 플러그인을 불러오는 태그 (영상, 오디오, 플래시 등)	홀	inline
<form>	입력을 받을 수 있는 양식을 나타냄 <input> <textarea> <select> <button> 등이 포함	짝	block

<주요 태그>

전체 태그들의 목록 (http://www.w3schools.com/tags/)

3. 주요 특성

태그의 특성(Attribute)

```
<form action="/write" method="post">
    <input type="text" name="message">
</form>
<img src="sample.jpg">
```

태그를 구체적으로 선언하기 위해서 특성(Attribute)을 지정 할 수 있습니다. 특성은 키-값 형식으로 이루어져있으며, 여러 특성을 가질 경우 그 순서는 중요하지 않습니다. 특정 태그는 필수적인 특성을 가지기도 합니다.

특성	설명	관련 태그
id	태그를 고유하게 식별하는 문자열	모든 태그
name	태그를 식별하기 위한 문자열, 중복 가능	<button> <form> <iframe> <input> <meta> <select> <textarea>
style	렌더링시 스타일의 지정	모든 태그

width	요소의 너비	block 및 inline-block 태그
height	요소의 높이	block 및 inline-block 태그
type	태그가 여러가지 형태를 가질 경우, 그 형태를 지정	<input> <link> <script> <style>
href	Hypertext Reference; 관련 문서/파일의 주소	<a> <link> <base>
value	값	<input> <option>
src	Source; 대상 문서/파일의 주소	<img> <audio> <video> <embed> <iframe> <script>

<주요 특성>

전체 특성들의 목록 (http://www.w3schools.com/tags/ref_attributes.asp)

4. HTML 문서 쓰고 읽기

```html
<html>
<head>
    <title>TEST</title>
</head>
<body>
    <div>
        <h1 style="text-align:center; border:1px dashed black">안녕하세요</h1>
        <p><b>어딘가로</b> <i>가는</i> <a href="https://some/url">링크</a></p>
    </div>
    <div>
        <form name="myForm" method="post" action="http://some/url/">
            <div>
                <input name="id" type="text" placeholder="아이디" value="test">
            </div>
            <div>
                <input name="on" type="checkbox" checked>
            </div>
            <div>
                <select name="select" multiple>
                    <option value="1" selected>One</option>
                    <option value="2">Two</option>
                    <option value="3">Three</option>
                </select>
            </div>
            <div>
                <input name="password" type="password" placeholder="비밀번호">
            </div>
            <div>
                <button type="submit">보내기</button>
            </div>
        </form>
    </div>

    <table style="border-width:1px; border-color:red; border-style: solid">
        <tr>
            <th>이름</th>
            <th>나이</th>
        </tr>
        <tr>
            <td>김영철</td>
            <td>40</td>
        </tr>
        <tr>
            <td>이상민</td>
            <td>41</td>
        </tr>
    </table>
</body>

</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

HTML 태그와 특성에 익숙해지면, 문서를 읽으면서 렌더링된 화면을 상상 할 수 있습니다.

[1] HTML, Hypertext Markup Language

3 웹 프론트엔드

3.3 CSS

1. 태그의 스타일 속성
2. 주요 스타일 속성들
3. CSS 기초
4. CSS 문법
5. CSS 라이브러리
6. 개발자 도구

> HTML 문서의 스타일을 정의하는 CSS를 이해하고, 그 문법에 대해서 알아봅니다.

1. 태그의 스타일 속성

HTML 태그에 <tagname style="속성1:값1; 속성2:값2;">... 처럼 style 속성을 명시해 다양한 스타일 관련 효과를 설정 할 수 있습니다.

```
<p style="
    color:pink; font-weight:bold; font-size:4em;
    text-align: center;
    margin: 30px 50px; padding: 25px;
    background-color: rgba(255,0,0,0.5);
    border: 1px solid black; border-radius:30px;
    box-shadow:5px 5px 5px gray">
WHAT ABOUT MY STYLE?
</p>

<div style="
    width: 100px; height: 100px;
    border:5px solid red; border-radius: 50%;
    background:skyblue; opacity: 0.6;
    position: absolute;
    bottom: 30px;
    right: 50px;
    ">Ball</div>
```

PDF에서는 미리보기가 지원되지 않습니다.

> W3C CSS 레퍼런스 (http://www.w3schools.com/cssref/)에서 전체 스타일 속성과 그 설명을 볼 수 있습니다.

위 링크의 레퍼런스에 모든 스타일 속성이 기술되어 있습니다. 하지만 이런 속성들은 단번에 공부하기보다는 조금씩 써보면서 익혀가는게 배우기에 적합하다고 생각합니다. 참고하실 만한 책을 한권 소개합니다. 최신 HTML 태그 및 스타일 속성들을 사전처럼 정리해 둔 얇은 책입니다.

<세르게이의 HTML5 & CSS3>

2. 주요 스타일 속성들

속성	설명	값

color	텍스트의 색상	color:red;
		color:rgb(255,100,0);
		color:rgba(255,100,200,0.5);

		color:#FFCC00;
opacity	태그의 불투명도	opacity:1; opacity:0.1;

<Color Properties>

속성	설명	값
background	태그의 배경 색상 및 이미지	background:#EEE; background:url('./image/bg1.jpg') no-repeat center center;
border	태그의 외곽선	border:none; border:1px solid black; border-bottom:2px dashed black; border-right-color:red;
border	태그의 외곽선	border:none; border:1px solid black; border-bottom:2px dashed black; border-right-color:red; border-radius:10px; border-radius:50%; border-radius:10px 20px 30px 40px;
box-shadow	태그의 외곽 그림자	box-shadow:none; box-shadow: 30px 10px 30px 5px #ccc;

<Background and Border Properties>

<태그의 박스 속성>

속성	설명	값
display	태그가 어떻게 보여질 지	display:none; display:block;

		display:inline; display:inline-block; display:flex;
position	태그가 어떻게 위치 할 지	position:static; position:relative; position:absolute; position:fixed; top:30px; left:30px; bottom:20px; right: 20px;
margin	태그의 바깥 여백	margin: 10px 5px 20px 30px; margin: 0 auto; margin-top: 15px;
padding	태그의 안쪽 여백	padding: 10px 5px 20px 30px; padding: 10px 15px; padding-top: 15px;
width/height min-width/min-height max-width/max-height	태그의 너비와 높이	width: 100%; width: 50vw; height: 200px; height: 100vh;
visibility	태그가 보일지	visibility: visible; visibility: hidden; visibility: collapse;
overflow	태그 내용이 넘칠 때	overflow: hidden; overflow: scroll; overflow-x: hidden; overflow-y: visible;
vertical-align	태그 내용의 세로 정렬 방식	vertical-align: 15px; vertical-align: top;
z-index	(position이나 opacity가 지정된) 태그의 쌓기 우선 순위	vertical-align: 15px; vertical-align: top;

<Basic Box Properties>

속성	설명	값
text-align	텍스트 정렬	text-align: center; text-align: right;
word-spacing	단어 간격	word-spacing: 0.5em; word-spacing: 5px;
letter-spacing	글자 간격	letter-spacing: -3px; letter-spacing: 5px;
line-height	줄 간격	line-height: 200%; line-height: 1.5;
text-indent	문단 들여쓰기	text-indent: 15px;
text-transform	대문자/소문자	text-transform: capitalize; text-transform: uppercase;

		text-transform: lowercase; text-transform: none;
white-space	공백과 개행 방식	white-space: normal; white-space: pre; white-space: nowrap;
word-break	개행의 기준	word-break: normal; word-break: break-all;
text-overflow	텍스트가 넘칠 때	overflow: hidden; white-space: nowrap; text-overflow: clip; text-overflow: ellipsis;
text-decoration	텍스트 꾸미기	text-decoration: none; text-decoration: underline; text-decoration: line-through;
font	폰트	font: italic bold 12px/30px Georgia, serif; font: normal 15px "맑은 고딕"; font-family: "Arial Black"; font-style: italic; font-size: 200%; font-weight: bold; font-weight: 900;

<Text Properties>

단위	설명	예시
px/pt/in/cm/mm	고정된 값으로 표현할 때의 단위	width: 300px; font-size:32pt;
em	태그의 font-size 속성을 1로 둔 상대적인 단위	width: 10em; line-height: 1.8em;
rem	루트 태그(HTML)의 font-size 속성을 1로 둔 상대적인 단위	font-size: 2rem;
vh	뷰포트(브라우저 창 크기) 높이의 1/100에 해당하는 상대적인 단위	height: 30vh; height: 100vh; font-size: 5vh;
vw	뷰포트(브라우저 창 크기) 너비의 1/100에 해당하는 상대적인 단위	width: 30vw; width: 100vw;
%	기본값에 대한 비율, 또는 최대치에 대한 비율	line-height: 200%; width: 100%; margin: auto 30%;

<크기를 표현하는 단위>

3. CSS 기초

위에서 처럼 태그에 직접 스타일 속성을 줄 수도 있지만, 이런 인라인 방식은 소스코드를 복잡하게 만들고 재사용성을 떨어뜨립니다.
CSS(Cascading Style Sheets)를 이용해 HTML 문서의 스타일 속성에 대한 정보만을 따로 작성 할 수 있습니다.

> 종속형 시트 또는 캐스케이딩 스타일 시트(Cascading Style Sheets, CSS)는 마크업 언어가 실제 표시되는 방법을 기술하는 언어로, HTML과 XHTML에 주로 쓰이며, XML에서도 사용할 수 있다. W3C의 표준이며, 레이아웃과 스타일을 정의할 때의 자유도가 높다...(위키백과)

HTML에 CSS를 적용하는 방법은 두가지가 있습니다.

1. style 태그 안에 CSS 작성하기

```
<style>
h1, h2, h3 {
    color: red;
    font-size: 2em;
}
.hello {
    color: rgba(100,50,50,0.5);
    line-height: 200%;
}
</style>
```

2. link 태그로 외부 CSS 파일을 참조하기

```
<link rel="stylesheet" type="text/css" href="./some/path/style.css">
```

4. CSS 문법

CSS는 스타일의 대상 { 스타일의 속성 및 값들 }의 정의가 반복되는 형식의 텍스트 파일입니다. 이 때 스타일의 대상을 선택자(셀렉터)라고합니다. 선택자는 어떤 요소들에게 스타일을 부여 할 지를 나타냅니다.

```
/* 아이디 선택자 */
#btn {
    margin: auto 50%;
    background-color: red;
}

/* 태그 선택자 */
div {
    border: 2px solid black;
}

/* 클래스 선택자 */
.ball {
    width: 50px;
    height: 50px;
    border-radius: 50%;
    background-color: skyblue;
}

/* CSS에서 주석은 이렇게 표시합니다. */
```

```
<div>
    <input id="btn" type="button" value="#btn">
</div>

<div class="ball"></div>
<div class="ball"></div>
<div class="ball"></div>
```

PDF에서는 미리보기가 지원되지 않습니다.

CSS는 마치 폭포수가 떨어지 듯이 적용됩니다. 태그들은 기본적으로 상위 태그의 스타일 속성을 상속받습니다. 하위 태그에 설정된 중복되는 스타일 속성이 있다면 상위 태그의 내용을 덮어씁니다. 이때 구체적인 선택자(태그명 < 클래스 < 아이디) 일수록 우선 순위가 높게 적용됩니다.

CSS Overriding

```html
<!DOCTYPE html>
<html>
<body>
<style>
body  {background: #333; color: white;}
div   {background: #666;}
h1    {background: #999; color: red;}
p     {background: #BBB; line-height: 2; font-weight:bold; color: blue;}
.some {background: #DDD;}
#some_h1 {background: #FFF;}
#some_p  {background: #FFF;} /* ID는 전체 태그 중에 고유합니다. */
</style>

<div>
    <h1>body > div > h1</h1>
    <h1 class="some">body > div > h1.some</h1>
    <h1 class="some" id="some_h1">body > div > h1.some#some_div</h1>
    <p>body > div > p</p>
    <p class="some">body > div > p.some</p>
    <p class="some" id="some_p">body > div > p.some#some_p</p>
</div>

</body>
</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

위의 선택자들을 조합해서 조금 더 구체적인 태그를 명시 할 수 있습니다.

CSS Combinators

```
.pad-30 {
    padding: 30px;
}
.block {
    border: 1px solid #333;
    width: 200px;
    height: 200px;
}
.pad-30.block { /** selector1 AND selector2, .. **/
    background: darkblue;
}
body, div, .pad-30 { /** selector1 OR selector2, .. **/
    line-height: 2em;
}
.pad-30 .block { /** 모든 후손 **/
    background: skyblue;
}
.pad-30 > .block { /** 직계 후손 **/
    background: gray;
}
.pad-30 ~ div { /** 모든 뒤 따르는 형제 **/
    background: blue;
}
.pad-30 + div { /** 바로 뒤 따르는 형제 **/
    background: red;
}
```

```
<div class="pad-30 block">
.pad-30.block (AND)
</div>

<div class="pad-30">
    .pad-30 이기도 하면서<br>
    .pad-30 + div (바로 뒤 따르는 형제)

    <div class="block">
    .pad-30 > .block (직계 자식)

        <div class="block">
        .pad-30 .block (후손)
        </div>
    </div>
</div>

<div>
.pad-30 + div (바로 뒤 따르는 형제)
</div>

<div>
.pad-30 ~ div (모든 뒤 따르는 형제)
</div>
```

PDF에서는 미리보기가 지원되지 않습니다.

태그의 상태(state)와 속성(attribute) 또한 선택자에 반영 할 수 있습니다.

CSS Attribute, Pseudo

```
a[href] { /* a 태그중에 href 속성이 있는 태그 */
 color: blue;
}

input[type="text"] { /* input 태그중에 type 속성이 text인 태그 */
 color: red;
}

input[type="text"]:focus {
    font-weight: bold;
}

input[type="checkbox"]:checked {
    font-weight: bold;
}

a:visited { /* a 태그중에 상태가 visited인 태그 */
 color: purple;
}

a {
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}

a:active {
    font-weight: bold;
}

/* 자식 및 형제 관련 의사 선택자 */
ul > li:nth-of-type(2n) {
    color: red;
}

ul > li:last-of-type {
    color: red;
}

ul:empty {
    background: black;
}

ul:nth-child(2n-1) {
    color: red;
```

```
    color: red;
}
ul:nth-child(2n):not(.red) {
    color: blue;
}

/* 요소에 텍스트 컨텐츠 삽입 */
ul > li::before {
    content: "START...";
}
ul > li::after {
    content: "...END";
}
```

```
<input type="checkbox" name="checkme" checked>
<input type="text" name="text" placeholder="text...">
<input type="password" name="password">
<a>No href</a>
<a href="http://google.com">Google.com</a>
```

PDF에서는 미리보기가 지원되지 않습니다.

특수 문법

```
/* 최상단에 작성하며 해당 CSS 파일의 인코딩을 선언한다. */
@charset "utf-8";

/* 다른 CSS 파일을 로드하는 구문으로 다른 규칙들 보다 먼저 작성해야 한다. */
@import url(../other-style-sheet.css);
@import url(http://some-site.com/other-style-sheet2.css);

/* 폰트 파일을 로드하며 해당 문서에서 사용 할 수 있는 웹폰트를 정의한다. */
@font-face {
    font-family: FontName;
    src: url(./fonts/font-file.ttf) format('opentype');
}

body {
    font-family: FontName, Arial, '돋움';
}

/* (미디어 쿼리) 웹 문서를 읽는 장치의 스크린 환경에 따라서 동적으로 스타일을 정의한다. */
@media (max-width: 500px) {
    background: green;
    ...
}

@media (min-width: 500px) and (max-width: 900px) {
    background: red;
    ...
}

@media (min-width: 900px) {
    background: blue;
}
```

> W3C CSS 레퍼런스 (http://www.w3schools.com/cssref/css_units.asp)에서 크기 단위에 대한 설명을 볼 수 있습니다.

5. CSS 라이브러리

시각적인 요소를 바닥부터 직접 설계하는 것은 많은 비용을 필요로 합니다. CSS 라이브러리들을 활용하면 탄탄한 디자인을 손쉽게 구현 할 수 있습니다. 수십 수백의 수 많은 오픈소스 CSS 프레임워크들이 있습니다. 그 중 몇 가지를 아래에 소개합니다.

이름	특징	링크
Bootstrap	트위터에서 만든 프레임워크	링크 (http://getbootstrap.com/)
Flat UI	Bootstrap을 기반으로 작성된 라이브러리	링크 (http://designmodo.github.io/Flat-UI/)
Material Design Lite	구글의 Material Design의 웹 버전	링크 (https://getmdl.io/)

28 Powerful Open Source CSS Frameworks & Libraries (https://webdesignledger.com/open-source-css-libs/)

Bootstrap 이용하기

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/cs
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js">
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.j
</head>
<body>
    <div class="container">
        <div class="row">
            <div class="col-md-8 col-md-offset-2">
                <h1>로그인</h1>
                <form>
                    <div class="form-group">
                        <input type="text" placeholder="아이디" class="form-control">
                    </div>
                    <div class="form-group">
                        <input type="password" placeholder="비밀번호" class="form-cont
                    </div>
                    <div>
                        <button type="submit" class="btn btn-primary">버튼</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
</body>
</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

6. 개발자 도구

<크롬 개발자 도구>

CSS파일이나 style 속성을 변경할 때마다 웹 페이지를 새로고침하면서 스타일을 잡는 것은 매우 번거로운 작업입니다. 웹 브라우저들은 이러한 작업을 돕는 개발자 도구를 내장하고 있습니다. 크롬 **개발자 도구에서는 요소(Element)를 선택하여 실시간으로 스타일 속성을 변경하고, 또 스타일 속성의 상속 관계를 파악 할 수 있는 기능**을 제공합니다.

> CSS Preprocessor (CSS 전처리기)
>
> HTML 태그들의 style 속성을 관리하기 위한 문서로 CSS를 도입했으나, CSS가 가진 불편함 때문에 또 다시 LESS, SASS, Stylus와 같은 문서 포맷이 등장합니다. 이러한 문서 포맷은 CSS 작성에 함수, 변수, 계산식과 같은 기능을 도입하여 코드량을 줄여주고, 또 스타일의 상속 관계를 구조화하기 쉽게 해줍니다. 물론 이러한 문서들도 결국엔 CSS 문서로 변환해야 HTML에서 사용 가능합니다. 이후에 6장 개발과 배포 파트에서 다시 다룹니다.

[1] CSS, Cascading Style Sheets

[2] CSS Preprocessor

3 웹 프론트엔드

3.4 JavaScript

1. JavaScript 기초
2. 브라우저 객체 모델
3. 문서 객체 모델
4. 색이 바뀌는 공
5. ECMAScript

> HTML 문서를 동적으로 조작하는 JavaScript를 이해하고, 그 문법에 대해서 알아봅니다. 브라우저 객체 모델과 문서 객체 모델을 학습합니다.

1. JavaScript 기초

<웹 프론트엔드 3형제>

HTML, CSS와 더불어 JavaScript는 웹 프론트엔드를 구성하는 중요한 역할을 하고 있습니다. 2장에서 먼저 Node.js를 통해서 JavaScript 문법을 익혔는데, 애초에 JavaScript는 브라우저 위에서 동작하는 스크립트 언어입니다. 이제 웹 페이지에서 HTML 요소(Element) 등을 동적으로 제어하기 위해서 진짜(?) JavaScript에 대해서 공부해보도록 하겠습니다.

<HTML은 구조를, CSS는 외관을, JavaScript는 동작을 제어함>

HTML에 JavaScript를 적용하는 방법에는 두가지가 있습니다.

1. script 태그 안에 JavaScript 작성하기

```
<!doctype html>
<html>
<head>
    <meta charset="UTF-8">
</head>

<body>
<h1>
<script>
document.write("현재 시간은 " + new Date());
</script>
</h1>

<style>
h1 { color:red; }
</style>
</body>

</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

2. script 태그로 외부 JavaScript 파일을 참조하기

```
<!doctype html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="./some/path/style.css">
    <script src="/some/js_file.js"></script> <!-- script 태그는 꼭 닫아줘야 합니다. -->
</head>
<body>
<script src="/some/js_file2.js"></script>
</body>
</html>
```

2. 브라우저 객체 모델

브라우저 객체 모델(BOM; Browser Object Model)은 웹 브라우저의 창(탭)에 관련된 기능들을 JavaScript로 제어할 수 있도록 추상화한 객체의 집합입니다. JavaScript에서 접근 할 수 있는 모든 전역 객체나 값들은 window라는 최상위 브라우저 객체 (머리 객체)에 연결되어 있습니다.

> 개발에 있어서는 단순히 수 많은 API를 공부하면서 메소드와 속성들에 대해 암기하는 것이 능사가 아닙니다. 큰 그림을 이해하고 분명히 이런 기능이 있을 것이다라고 확신 할 수 있는 배경 지식이 필요합니다. 기능에 대해서 생각해보고 추측해보면서, 또 인터넷에 검색해보면서, 조금씩 능숙해지면 됩니다. 모든 것을 공부 할 수는 없습니다. 배경 지식을 키웁시다.

```
// 모든 전역 객체나 값은 머리 객체 window에 연결됩니다.
var x = 10; // window.x == 10
var y = "abc"; // window.y == "abc"


/** window 객체 **/
window.open('http://google.com'); // 창 열기
window.close(); // 창 닫기
window.print(); // 인쇄

window.alert('알림 문구');
window.confirm('확인 문구');
var answer = window.prompt('질문');

window.innerHeight; // 실제 페이지가 보이는 부분의 해상도
window.innerWidth;
window.pageXOffset; // 스크롤 위치 (IE에서 호환성 문제)
window.pageYOffset;

window.resizeTo(500, 600); // 창 크기를 변경
window.resizeBy(-100, 100); // 창 크기에 주어진 값을 더해서 변경
window.scrollTo(0, 600); // 스크롤을 주어진 좌표로 이동
window.scrollBy(0, 50); // 스크롤을 현재 위치에 주어진 좌표를 더해서 이동

var intervalId = window.setInterval(function(){
    // 1초마다 반복 실행
}, 1000);
window.clearInterval(intervalId);

var timeoutId = window.setTimeout(function(){
    // 1초후에 한번 실행
}, 1000);
window.clearTimeout(timeoutId);


/** window.screen 객체 **/
screen.width; // 디스플레이 해상도
screen.height;


```

```
/** window.history 객체 **/
history.length; // 히스토리 갯수
history.back(); // 뒤로가기
history.forward(); // 앞으로
history.go(2); // 앞으로 2번
history.go(-2); // 뒤로 2번


/** window.navigator 객체 **/
navigator.platform; // MacIntel
navigator.userAgent; // Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKi..
navigator.onLine; // true
navigator.language; // en-US
navigator.geolocation.getCurrentPosition(...); // 위치 정보
navigator.getBattery(...) // 배터리
navigator.getUserMedia(...) // 웹캠 및 마이크


/** window.location 객체 **/
// 주소가 http://test.com/some/path?page=3&keyword=test#anchor1 일 때
location.href; // 위의 URL 전체
location.protocol; // http:
location.host; // test.com
location.pathname; // /some/path
location.search; // ?page=3&keyword=test
location.hash; // #anchor1

location.href = 'http://google.com'; // 페이지 이동
location.assign('http://google.com');
location.replace('http://google.com'); // 히스토리에 기록 없이 페이지 이동
location.reload(); // 페이지 새로고침
```

3. 문서 객체 모델

문서 객체 모델(DOM; Document Object Model)은 웹 브라우저가 HTML이나 XML 같은 구조화된 문서를 JavaScript로 제어 할 수 있도록 추상화한 객체의 집합입니다. 문서 객체 모델은 window.document (Document Node)를 최상위 노드로 하는 트리 구조입니다.

<HTML DOM Tree>

1. 태그

   HTML 문서를 구조화하는 <tagname..>과 같은 문자열

2. DOM

   JavaScript로 제어 할 수 있는 해석된 HTML 문서 객체

3. Node

   - Document Node: 문서 자체를 가리키는 최상위 노드
   - Element Node: 태그 요소 자체를 가리키는 노드
   - Attribute Node: 요소 노드의 특성들
   - Text Node: 요소 노드의 안에 있는 텍스트

Document 노드

```
/** Document 노드의 주요 값 **/
document.title; // 문서 제목 (<title> Element의 값)
document.lastModified; // 문서의 최종 수정 일자
document.cookie; // 쿠키
document.referrer; // 레퍼러 URL (어느 페이지에서 왔는지)
document.write("...");


/** Document 노드에서 자식 Element에 접근하기 **/
document.documentElement; // HTML Element
document.head; // HEAD Element
document.body; // BODY Element
document.script; // HTMLCollection of SCRIPT Elements
document.images; // HTMLCollection of IMG Elements
document.forms; // HTMLCollection of FORM Elements
document['FORM_NAME']; // FORM[name='FORM_NAME'] Element
document.FORM_NAME;

document.getElementsByTagName("div"); // HTMLCollection of DIV Elements
document.getElementsByClassName("someClass"); // HTMLCollection of .someClass Elemer
document.getElementById("someId"); // #someId Element
document.querySelectorAll(".some-class > a.hello:hover"); // HTMLCollection of match
document.querySelector(".some-class > a.hello:hover"); // First matched Element


/** Element의 부모 및 형제 Element 찾기 **/
element.parentElement; // 부모 Element
element.childElementCount; // 자식 Element 갯수
element.children; // HTMLCollection of 자식 Elements
element.firstElementChild; // 첫째 자식 Element
element.lastElementChild; // 마지막 자식 Element
element.nextElementSibling; // 다음 형제 Element
element.previousElementSibling; // 이전 형제 Element

element.getElementsByTagName("div"); // HTMLCollection of DIV Elements in children
element.getElementsByClassName("someClass"); // HTMLCollection of .someClass Element
element.querySelectorAll(".some-class > a.hello:hover"); // HTMLCollection of matche
element.querySelector(".some-class > a.hello:hover"); // First matched Element in ch
```

JavaScript로 Element 제어하기1

```
<html>
<body>

<div id="some-id">div#some-id</div>
<div class="some-class">
    div.some-class
    <div class="some2-class">
    div.some-class.some2-class
    </div>
</div>
<form name="form1">
    <p>form[name=form1] > p</p>
    <input type="text" name="text1">
    <input type="password" name="pass1">
</form>

<script>
var el1 = document.getElementById("some-id");
el1.textContent = "Text 1";

var el2 = el1.nextElementSibling;
el2.childNodes[0].nodeValue = "Text 2";
// el2.textContent = "Text 2"; ???

el2.children[0].textContent = "Text 3";

document.form1.firstElementChild.textContent = "Text 4";
document.forms[0].text1.value = "text1 value...";

document.querySelector("form[name=form1] > input[type=password]").value = "pass1 val
</script>

</body>
</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

Element 노드

```
element.tagName; // Element의 태그명(대문자)

/** 내용 제어 **/
element.textContent = "Some Text"; // 짝 태그인 경우
element.innerHTML = "<b>Some Text <div>And other children<p>...</p></div></b>"; // 
console.log(element.textContent, element.innerHTML); // 물론 내용을 그냥 읽을 수도 있음

/** attribute 제어 **/
element.getAttribute('name');
element.setAttribute('name', 'some-name');
element.removeAttribute('name');

/** Element가 해석한 주요 attribute는 (property) 바로 접근 가능 **/
element.id;
element.type;
element.value;
element.placeholder;
element.checked;
element.disabled;
element.title;
element.src;
element.width/height;

/** 스타일 속성 **/
element.className; // "class1 class2 class3"
element.classList.add("class4");
element.classList.remove("class4");

element.style; // 스타일 속성들에 대해서 HTML/CSS와 다르게 camelCase를 씀
element.style.backgroundColor = '#ff00ff';
element.style.fontSize = '1.2em';
```

JavaScript로 Element 제어하기2

```
<div id="some-id">div#some-id</div>

<script>
var el = document.getElementById("some-id");
el.textContent = "Text 1";
el.innerHTML = "<b>T</b><i>e</i><u>x</u><s>t</s> 1";
el.style.fontSize = '5em';
el.style.fontWeight = '900';
el.style.color = 'rgba(0,100,100,0.5)';
el.style.position = 'relative';

var x = 0, y = 0;
window.setInterval(function(){
    x = (x+2) % 600;
    y = (y+1) % 300;
    el.style.left = x+'px';
    el.style.top = y+'px';
}, 10);
</script>
```

PDF에서는 미리보기가 지원되지 않습니다.

Element 동적으로 생성하기

```
/** Element 생성은 Document 노드의 메소드를 이용 **/
var h1 = document.createElement("h1");
h1.textContent = "Some Headings...";
h1.style.color = "red";

/** Element를 HTML 문서에 삽입하려면, 이미 존재하는 어떤 Element의 자식으로 삽입 **/
document.body.appendChild(h1);

/** Element를 삭제 할 때는, 부모 Element에서 자식을 삭제 **/
document.body.removeChild(h1);

/** 또는 innerHTML을 이용 할 수도 **/
document.body.innerHTML = "<h1 style=\"color:red\">Some Headings..</h1>";
document.body.innerHTML = "";
```

JavaScript로 Element 제어하기3

```
<div id="some-id">div#some-id</div>

<script>
var parent = document.getElementById("some-id");
var img = document.createElement("img");
img.src = "https://workshop.benzen.io/assets/img/benzen-logo.png";
img.width = 300;
parent.appendChild(img);

parent.appendChild(img); // 메모리상에 img Element는 하나만 존재하고, 이미 parent의 자식으로 삽
parent.appendChild(img); // appendChild 를 반복해봐야 실제 img Element는 하나만 붙어 있습니다.

var h2 = document.createElement("h2");
h2.textContent = "Hello!";
parent.appendChild(h2);
</script>
```

PDF에서는 미리보기가 지원되지 않습니다.

4. 색이 바뀌는 공

짜임새 없이 명령형 스타일로, JavaScript만을 이용해서, 색이 랜덤하게 바뀌는 공들을 만들어보겠습니다.

정의	설명
setInterval(callback, time)	일정한 간격(time)으로 callback을 실행합니다. time은 0.001초 단위입니다.
setTimeout(callback, time)	일정한 시간(time) 이후에 callback을 한번 실행합니다. time은 0.001초 단위입니다.
parseInt(value)	value를 정수로 리턴합니다. 문자열이나 숫자를 해석합니다. 해석이 불가능하면 NaN이 리턴됩니다.
Math.floor(number)	number를 내림해서 리턴합니다. 숫자가 아니면 NaN이 리턴됩니다.
Math.random()	0~1 범위에서 임의의 실수를 리턴합니다.

<실습에 이용 할 함수>

색이 바뀌는 공들

```
<html>
<body>
<script>
var container = document.createElement("div");
document.body.appendChild(container);

for (var i=0; i < 20; i++){
    var ball = document.createElement("div");

    ball.style.height = ball.style.width = "20px";
    ball.style.borderRadius = "50%";
    ball.style.backgroundColor = "rgb("+random(255) +","+random(255) +","+random(255

    ball.style.position = "absolute"
    ball.style.left = random(400)+"px";
    ball.style.top = random(400)+"px";

    container.appendChild(ball);
}

setInterval(function(){
    var balls = container.children; // balls가 Array 타입이 아니라서 forEach 메소드는 쓸 수
    for(var i=0; i < balls.length; i++){
        balls[i].style.backgroundColor = "rgb("+random(255) +","+random(255) +","+rc
    }
}, 150);

function random(num){
    return Math.floor(Math.random() * num);
}
</script>
</body>
</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

5. ECMAScript

ECMAScript 또는 ES라고 부르는 JavaScript 언어의 표준이 있습니다. 역사적인 이야기가 있지만 JavaScript의 문법 표준이라고 생각하시면 좋겠습니다. Node.js 같이 로컬에서 컴파일하는 경우나, Chrome 같이 항상 표준을 준수하며, 최신버전으로 자동 업데이트되는 Evergreen 웹 브라우저를 이용하는 경우에는, ES의 최신 문법을 이용해도 호환성에 큰 문제가 없습니다. 다만 Internet Explorer로 대표되는 타 웹 브라우저들에서 호환성 문제를 방지하기 위해서는, HTML 문서에서 최신 JavaScript 문법(ES6, ES2016+)을 사용하는 데 고민이 필요합니다.

> ECMA스크립트(ECMAScript)는 Ecma 인터내셔널의 ECMA-262 기술 규격에 정의된 표준화된 스크립트 프로그래밍 언어이다. 이 언어는 웹 상에서 널리 쓰이며, 흔히 자바스크립트 또는 J스크립트로도 생각할 수 있지만 두 용어는 특별한 의미 차이가 있다. ECMA스크립트와 자바스크립트, J스크립트의 관계를 이해하기 위해서는 ECMA스크립트의 역사를 알아야 한다... (위키백과)

> ES2016+ 브라우저 호환성 테이블 (http://kangax.github.io/compat-table/es2016plus/)
> 브라우저 환경에서 호환성에 관련 없이 최신 문법을 이용하고 싶은 경우 Babel 등과 같은 Transpiler를 이용 할 수 있습니다. 이 프로그램은 최신 JavaScript 문법으로 작성된 소스코드를 낮은 버전의 JavaScript 문법의 소스코드로 번역하여 웹 브라우저의 호환성 문제를 해결해줍니다. 또는 Shim/Polyfill이라고 불리는 호환성을 위한 script 파일을 HTML 문서에 같이 삽입해서, ES 버전간의 간극을 메우는 방식도 있습니다.

[1] BOM, Browser Object Model
[2] DOM, Document Object Model
[3] Node
[4] ECMAScript
[5] Babel
[6] Transpiler
[7] Shim
[8] Polyfill

3 웹 프론트엔드

3.5 모델링

1. 프로토타입 체이닝
2. 모델링
3. 의존성 설계
4. MVC 구조

> JavaScript의 프로토타입 체이닝에 대해서 알아보고, 게시판 흉내내기, 상자 안에서 튕기는 공들을 모델링합니다.

1. 프로토타입 체이닝

OOP 파트에서 다뤘던 JavaScript의 class 키워드는 ES6에서 부터 등장합니다. 웹 브라우저간 호환성을 위해서.. 또는 흥미를 위해서.. ES6 이전 버전의 JavaScript의 class 구현에 대해서 알아보겠습니다. class의 핵심은 생성자 함수, 인스턴스 속성/메소드, 클래스 속성/메소드로 볼 수 있습니다. ES5와 그 이전의 JavaScript에서는 OOP를 구현하기 위해서 프로토타입 체이닝(Prototype Chaining)이라는 간단한 방식을 제공합니다. 이 외에 상속(extends 키워드)과 같은 문법은 제공되지 않습니다.

생성자 함수와 인스턴스

```
var Person = function(name) {
   this.name = name; // 인스턴스 속성
   this.speakName = function(number){ // 인스턴스간 공유되지 않는 메소드
      for(var i=0; i < number; i++) {
        console.log("my name is " + this.name);
      }
   }
};

var kim = new Person("kim");
kim.speakName(10);
```

기본적으로 인스턴스의 모든 속성은 생성자 함수로 정의가 가능합니다. 하지만 위와 같은 코드는 인스턴스들이 공유하는 속성이 없이 제 각각 속성, 함수를 가지게 됩니다. 이때 생성자 함수의 prototype 객체를 통해 인스턴스들이 공유하는 속성을 정의 할 수 있습니다.

prototype 객체와 인스턴스 메소드

```
var Person = function(name) {
    this.name = name; // 인스턴스 속성
};

Person.prototype; // (생성자) 함수에는 prototype Object 객체가 자동으로 붙어 있습니다.

Person.prototype.speakName = function(number){ // 인스턴스간 공유되는 메소드
  for(var i=0; i < number; i++) {
    console.log("my name is " + this.name);
  }
}

var kim = new Person("kim");
kim.speakName(10);

/**
kim.speakName을 찾으면 우선 kim.speakName을 찾습니다.
없으면 kim.__proto__ (= kim.constructor.prototype = Person.prototype)에서 speakName을 찾
없으면 kim.__proto__.__proto__ 에서 speakName을 찾습니다.

이 때 kim.__proto__ 는 Object 객체이기 때문에 kim.__proto__.__proto__ 는 Object.prototype
때문에 Object.prototype의 속성들은 JavaScript의 모든 객체가 공유하는 속성이됩니다.

없으면 kim.speakName은 undefined가 됩니다.
**/
```

<프로토타입 체이닝>

이런 방식의 OOP 구현을 Prototype Chaining이라고 합니다. class로 구현하는 OOP와 비슷하면서도 좀 더 확장성이 있는 듯하고, 캡슐화가 덜된 것 같기도 합니다.
A 클래스에서 B 클래스를 상속하려면 B.prototype의 값들을 하나 하나 A.prototype에 복사하면 되겠죠? 만약 생성자 함수를 같이 쓰거나 메소드를 오버라이딩 하자면 또 다른 길을 찾아봐야 하겠습니다.

prototype 객체와 인스턴스 속성

```
var Person = function(name) {
    if (name)
        this.name = name; // 인스턴스 속성
};

Person.prototype.name = "WHOIS"; // 물론 메소드가 아닌 값도 공유 할 수 있습니다.

var noname = new Person();
console.log(noname.name); // WHOIS

var kim = new Person("KIM");
console.log(kim.name); // KIM
```

객체의 속성을 참조 할 때의 우선 순위

1. 인스턴스에 직접 붙은 속성
2. 생성자 함수의 prototype에 붙은 속성
3. Object 생성자 함수의 prototype에 붙은 속성

클래스 속성/메소드

```
var Person = function(name) {
    this.name = name;

    Person.list.push(this); // 또는 this.constructor.list.push(this);
};

Person.list = []; // 클래스 속성
Person.printAllNames = function(){ // 클래스 메소드
    Person.list.forEach(function(person){
        console.log(person.name);
    });
}

var son = new Person("SON");
var kim = new Person("KIM");
Person.printAllNames();
```

인스턴스 없이 참조 할 속성이나 메소드가 필요하다면, 위와 같이 생성자 함수에 속성을 붙혀서 클래스 속성/메소드를 구현 할 수 있습니다. 클래스 메소드니 static 메소드니 문법 자체에 메이기 보다는, 주어진 환경을 이용해서 필요한 기능을 작성하시면 됩니다.

생성자 함수와 Prototype을 이용해 Dog 클래스를 정의

인스턴스 속성/메소드

1. name (String)
2. age (Number)
3. color (String)
4. speak (Function): console에 인스턴스의 name, age 및 color 속성을 출력

클래스 속성/메소드

1. totalDogs (Number): 지금까지 생성된 Dog 인스턴스의 총 수
2. printTotalDogs (Function): console에 totalDogs를 출력

```
var Dog = function(name, age, color) {
    this.name = name;
    this.age = age;
    this.color = color;
    this.constructor.totalDogs++;
};

Dog.prototype.speak = function(){
    console.log("안녕 내 이름은", this.name, "이고 내 나이는", this.age, "이고 내 색깔은", thi
};

Dog.totalDogs = 0;
Dog.printTotalDogs = function(){
    console.log("모두 ", Dog.totalDogs, "마리의 멍멍이");
};

var mong = new Dog("mong", 6, "black and white");
var mung = new Dog("mung", 3, "white");
mung.speak();
Dog.printTotalDogs();
```

2. 모델링

일시적으로 원하는 결과를 만들어 내는 것 이상으로 코드를 설계 할 수 있는 능력이 필요합니다. 모델링과 뷰와 데이터의 분리라는 개념에 대해서 알아보겠습니다.

색이 바뀌는 공

```
<html>
<body>
<script>
var container = document.createElement("div");
document.body.appendChild(container);

for (var i=0; i < 20; i++){
    var ball = document.createElement("div");

    ball.style.height = ball.style.width = "20px";
    ball.style.borderRadius = "50%";
    ball.style.backgroundColor = "rgb("+random(255) +","+random(255) +","+random(255

    ball.style.position = "absolute"
    ball.style.left = random(400)+"px";
    ball.style.top = random(400)+"px";

    container.appendChild(ball);
}

setInterval(function(){
    var balls = container.children; // balls가 Array 타입이 아니라서 forEach 메소드는 쓸 수
    for(var i=0; i < balls.length; i++){
        balls[i].style.backgroundColor = "rgb("+random(255) +","+random(255) +","+ra
    }
}, 150);

function random(num){
    return Math.floor(Math.random() * num);
}
</script>
</body>
</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

공을 생성하고 색상을 바꾸는 잘 작동하는 코드입니다. 다만 추상화가 되어있지 않아서 코드에 의미를 부여하기 힘듭니다.
OOP를 도입해서 공과 컨테이너를 모델링하고, 색이 바뀌는 공에서 나아가서 컨테이너 안에서 공이 움직이면서 벽을 만나면 튕기도록 구현하겠습니다. 객체를 모델링(의인화)해서 생각하면 다른 객체들과의 관계를 유기적으로 구성할 수 있고 이해하기 쉬운 코드를 만들 수 있습니다.

1. 공이 컨테이너의 오른쪽, 왼쪽 벽에 부딪치면 x축 방향의 속도가 반대가 됩니다.
2. 공이 컨테이너의 위, 아래 벽에 부딪치면 y축 방향의 속도가 반대가 됩니다.

모델링, 튕기는 공

```html
<html>
<body>
<script>
/** 공 클래스 **/
var Ball = function(x, y, radius){
    // Data
    this.x = x || 0;
    this.y = y || 0;
    this.radius = radius || 10;
    this.vx = Math.floor(Math.random()*10)-5;
    this.vy = Math.floor(Math.random()*10)-5;

    // View
    this.view = document.createElement("div");
    this.view.style.backgroundColor = randomRGB();
    this.view.style.width = this.view.style.height = this.radius*2 +"px";
    this.view.style.borderRadius = "50%";
    this.view.style.position = "absolute";
    this.view.style.left = this.x + "px";
    this.view.style.top = this.y + "px";
};

Ball.prototype.move = function(){
    // Data
    this.x += this.vx;
    this.y += this.vy;

    // View
    this.view.style.left = this.x - this.radius + "px";
    this.view.style.top = this.y - this.radius + "px";
    return this;
};

Ball.prototype.collideX = function(){
    // Data
    this.vx *= -1;
    return this;
};
```

```
};

Ball.prototype.collideY = function(){
    // Data
    this.vy *= -1;
    return this;
};

Ball.prototype.getView = function(){
    return this.view;
};

Ball.prototype.getX = function(){
    return this.x;
};

Ball.prototype.getY = function(){
    return this.y;
};


/** 컨테이너 클래스 **/
var Container = function(width, height){
    // Data
    this.objects = [];
    this.width = width || 500;
    this.height = height || 500;

    // View
    this.view = document.createElement("div");
    this.view.style.border = "1px dashed red";
    this.view.style.position = "relative";
    this.view.style.width = this.width + "px";
    this.view.style.height = this.height + "px";
}

Container.prototype.addObject = function(obj){
    // Data
    this.objects.push(obj);

    // View
    this.view.appendChild(obj.getView());
    return this;
}

Container.prototype.moveObjects = function(){
    for(var i = 0; i < this.objects.length; i++) {
        var obj = this.objects[i];
        obj.move();

        var x = obj.getX(),
            y = obj.getY(),
            collision = false;

        if (x < 0 || x > this.width) {
            obj.collideX();
            collision = true;
        }
```

```
        }
        if (y < 0 || y > this.height) {
            obj.collideY();
            collision = true;
        }

        if (collision) {
            obj.move();
        }
    }

    return this;
}

Container.prototype.getView = function(){
    return this.view;
}


/** 공용 함수 **/
function randomRGB(){
    var r = parseInt(Math.random()*255),
        g = parseInt(Math.random()*255),
        b = parseInt(Math.random()*255),
        rgb = "rgb("+r+","+g+","+b+")";
    return rgb;
}


/** 설계 후 이용하기 **/
var container = new Container(400, 400);

for (var i=0; i < 20; i++) {
    var ball = new Ball(200, 200);
    container.addObject(ball);
}

document.body.appendChild(container.getView());

setInterval(function(){
    container.moveObjects();
}, 20);
</script>
</body>
</html>
```

PDF에서는 미리보기가 지원되지 않습니다.

3. 의존성 설계

의존성 이야기...

모델링, 튕기는 공, 다시

```
<body style="margin:0; margin:0">
  <div id="con1" style="background:pink; width: 100%; height: 300px; display:inline-
  </div>
  <div id="con2" style="background:skyblue; width: 500px; height: 300px; display:inl
  </div>
</body>
<script>
class Renderable {
  constructor(state = {}) {
    // data
    this.width = state.width || 10;
    this.height = state.height || 10;

    // view
    this.$view = this.createView(state);
    this.$view.style.position = 'absolute';
    this.$view.style.width = this.width + 'px';
    this.$view.style.height = this.height + 'px';

    this.setState({
      x: state.x || 0,
      y: state.y || 0,
      vx: state.vx || Math.random()*10-5,
      vy: state.vy || Math.random()*10-5
    });
  }

  createView(state) {
    var $view = document.createElement('div');
    $view.style.backgroundColor = 'black';
    return $view;
  }

  getView() {
    return this.$view;
  }

  getSize() {
    return {
      width: this.$view.clientWidth,
      height: this.$view.clientHeight,
    };
  }
```

```
  getState() {
    return {
      x: this.x,
      y: this.y,
      vx: this.vx,
      vy: this.vy,
    };
  }

  setState(state = {x:0, y:0, vx:0, vy:0}) {
    // data
    this.x = state.x;
    this.y = state.y;
    this.vx = state.vx;
    this.vy = state.vy;

    // view
    this.$view.style.left = this.x + 'px';
    this.$view.style.top = this.y + 'px';
  }
}

class Circle extends Renderable {
  createView(state) {
    var $view = document.createElement('div');
    $view.style.borderRadius = '50%';
    $view.style.backgroundColor = state.color || 'black';
    return $view;
  }
}

class Container {
  constructor($view) {
    this.$view = $view;
    this.$view.style.position = 'relative';
    this.objects = [];
  }

  getSize() {
    return {
      width: this.$view.clientWidth,
      height: this.$view.clientHeight,
    };
  }

  appendObject(object) {
    // data
    this.objects.push(object);

    // view
    this.$view.appendChild(object.getView());
  }

  render() {
    var containerSize = this.getSize();

```

```
    this.objects.forEach(object => {
      var state = object.getState();
      var size = object.getSize();

      var newState = {
        x: state.x + state.vx,
        y: state.y + state.vy,
        vx: state.vx,
        vy: state.vy,
      };

      // collision
      var halfWidth = size.width/2;
      var halfHeight = size.height/2;

      if (newState.x - halfWidth < 0) {
        newState.x = halfWidth;
        newState.vx *= -1;
      } else if (newState.x + halfWidth > containerSize.width) {
        newState.x = containerSize.width - halfWidth;
        newState.vx *= -1;
      }

      if (newState.y - halfHeight < 0) {
        newState.y = halfHeight;
        newState.vy *= -1;
      } else if (newState.y + halfHeight > containerSize.height) {
        newState.y = containerSize.height - halfHeight;
        newState.vy *= -1;
      }

      object.setState(newState);
    });
  }

  start() {
    if (!this.timer) {
      this.timer = setInterval(this.render.bind(this), 10);
    }
  }

  stop() {
    if (this.timer) {
      clearInterval(this.timer);
    }
  }
}

var con1 = new Container(document.getElementById('con1'));
for(var i=0; i<100; i++) con1.appendObject(new Circle());
con1.start();

var con2 = new Container(document.getElementById('con2'));
var size = con2.getSize(), centerX = size.width/2, centerY = size.height/2;
for(var i=0; i<50; i++) {
  con2.appendObject(new Circle({x: centerX, y: centerY, color: 'red'}));
}
```

```
}
con2.start();
</script>
```

PDF에서는 미리보기가 지원되지 않습니다.

4. MVC 구조

<결합(Coupling)의 개념>

1. 공과 컨테이너 객체의 의존성 분리

   공 객체는 독립적으로 쓰일 수 있습니다. 컨테이너 객체 역시 독립적으로 쓰일 수 있으며, 공 객체 뿐만 아니라 move, collideX, collideY, getX, getY, getView 메소드를 (올바르게) 지닌 어떤 객체라도 담고 움직이게 할 수 있습니다. 컨테이너의 의존 사항처럼 특정 객체에 대한 직접적인 의존이 아닌, 두리뭉실한 형태 대한 의존을 명시하는 것을 OOP에서는 Interface라고 합니다.

2. 뷰와 데이터의 분리

   뷰가 보여지기 이전에 데이터가 먼저 존재하고 여러 계산이 이루어집니다. 가변적인 값들은 뷰에 직접 속하기 보다는, 객체에서 따로 관리하며 뷰에서 그 데이터를 이용하는 형태가 좋습니다. 뷰와 데이터를 분리해서 관리하면 코드를 구조화하기에 용이하고 추가적인 기능 구현이나 유지 보수 과

정이 깔끔해집니다. 이런 코드는 확장성이 좋다고 합니다.

뷰와 데이터가 긴밀하게 결합되어있으면(Coupled) 새로운 속성이나 기능을 만들 때마다 뷰에 접근하여 뷰의 속성에서 데이터를 추출하고, 다시 어떤 처리를 한 후 또다시 뷰에 반영해야 합니다. 즉, 가능은 하지만.. 코드를 읽어 보았을 때 말이 안되는 코드입니다.

뷰와 데이터를 잘 분리하면 프로그램은 다음과 같은 구조를 갖습니다.
데이터를 처리하는 코드 + 데이터를 뷰에 반영(render)하는 코드

<MVC 패턴>

플랫폼에 상관 없이, 응용프로그램의 대표적인 구조로 MVC라고 불리는 디자인 패턴이 있습니다. 각 글자는 Model, View, Controller를 의미합니다. 지금 튕기는 공을 만들면서 Model을 설계했고, View가 모델에 종속되긴 했지만 View의 로직도 어느 정도 분리를 해보았습니다. 사용자와의 상호작용은 없었기 때문에 Controller라고 부를 코드는 없었습니다. 앞으로 계속 될 실습과 추후 웹서비스 파트에서는 항상 모델과 뷰, 컨트롤러를 분리하려고 애쓰도록 하겠습니다. 먼 길로 돌아가는 것 같지만 이런 원리를 지킴으로써 더욱 빠르고 탄탄한 개발이 가능 할 것입니다.

[1] Prototype Chaining
[2] Interface
[3] MVC, Model, View, Controller

3 웹 프론트엔드

3.6 이벤트 시스템

1. 이벤트 시스템
2. 의존성 분리
3. 커스텀 이벤트
4. 이벤트 전파
5. 위임
6. 크로스 브라우징

JavaScript의 이벤트 시스템과 위임(Delegation)에 대해서 알아봅니다. 코드간의 의존성을 줄이는 소프트웨어 개발 원리에 대해서 배웁니다. 또 크로스 브라우징 문제에 대해 알아봅니다.

1. 이벤트 시스템

<웹 브라우저의 이벤트>

사건 기반 프로그래밍(Event-driven programming)은 프로그램의 흐름을 사건의 발생에 따라서 제어하는 프로그래밍 패턴을 말합니다. GUI를 제공하는 대부분의 플랫폼은 이벤트 관련 API를 제공하여 개발자가 사용자와 프로그램의 상호작용 제어 할 수 있도록 합니다. 또한 GUI 외에도 모듈간의 결합성을 분리(Decoupling)하기 위해서 이벤트 기반의 인터페이스를 도입하는 경우도 많습니다.

<이벤트와 이벤트 핸들러>

웹 브라우저는 사용자의 입력에 따라서 입력 정보가 담긴 이벤트 객체를 생성하고, 이를 이벤트 핸들러라고 하는 함수에 전달 합니다. 이벤트 핸들러는 이벤트 객체의 데이터에 따라서 사용자가 기대 할 만한 작업을 수행합니다. 웹 페이지가 로드되거나, 웹 브라우저의 창 크기를 조절하거나, DOM을 클릭하거나, 마우스를 특정 DOM에 올려 놓거나, 키보드를 입력하거나 하는 행위는 특정 타입의 이벤트를 발생(Trigger)시킵니다. 이때 해당 타입의 이벤트에 대한 핸들러가 바인딩(Binding)된 DOM 객체는 이벤트 객체를 전달 받고 이벤트 핸들러를 호출하게 됩니다. 여기서 이벤트 핸들러(Event Handler)라고 하는 함수는 일반적인 콜백 함수이며 다른 말로 이벤트 리스너(Event Listener)라고도 합니다.

<발행-구독 패턴>

이러한 이벤트 시스템...같은 디자인 패턴을 옵저버 패턴(Observer Pattern) 또는 발행-구독 패턴(Publish-Subscribe Pattern)이라고도 합니다. 웹 브라우저에서는 웹 브라우저 자체가 옵저버/발행자(Observer/Publisher)이고, 이벤트 핸들러를 등록한 개별 DOM 객체들이 구독자(Subscriber)가 됩니다. 프로그램에서 이벤트가 발생하고 처리되는 과정을 축약하면 다음과 같습니다.

1. 사용자나 코드가 특정 타입의 이벤트를 발생시킴
2. Observer/Publisher는 특정 타입의 이벤트를 구독하고 있는 Subscriber들의 이벤트 핸들러를 실행

DOM에 이벤트 핸들러를 바인딩하기

```
var box1 = document.getElementById("box1");
var box2 = document.getElementById("box2");

/** Element는 addEventListener 라는 메소드를 갖습니다. **/
box1.addEventListener('mouseover', function(e){
    this.style.background = '#FFF';

    /**
        이때 핸들러 안에서 this는 이벤트 핸들러를 호출하는 box1 Element를 가리킵니다.
        꼭 기억하세요...!
        마치 Element의 메소드처럼 이벤트 핸들러가 실행됩니다.
    **/
});

box1.onmouseout = function(e){
    this.style.background = 'pink';
};

/** Element는 이벤트 핸들러를 직접 할당 할 수 있는 속성을 갖습니다. **/
box2.onclick = function(event){
    alert("Click event triggered on #box2");
};

/** Element는 removeEventListener 라는 메소드를 갖습니다. **/
var handler = function(e){ // 한번만 실행 될 함수
    this.innerHTML += '<br>First Clicked';
    this.removeEventListener('click', handler);
};

box2.addEventListener('click', handler); // 동일한 이벤트에 대해서 여러 핸들러를 가질 수도 있습니
```

```
<div id="box1" >
Hover BOX1
</div>
<div id="box2">
Click BOX2
</div>

<style>
div {
    margin: 10px;
    padding: 10px;
    width: 100%;
    font-size: 5em;
    background: pink;
}
</style>
```

PDF에서는 미리보기가 지원되지 않습니다.

분류	타입	설명
Mouse	click	클릭
	contextmenu	우클릭
	dblclick	더블클릭
	mousedown	누른 상태
	mouseup	누르고 뗐을 때
	mouseover	Element나 그 자식들 위에 마우스를 위치 할 때
	mousemove	그 안에서 이동 할 때
	mouseout	그 안에서 나왔을 때
Keyboard	keydown	키를 누른 상태 (계속 발생)
	keypress	문자에 해당되는 키를 누른 상태 (계속 발생)
	keyup	누르고 뗐을 때
Form	focus	Input에 커서가 맞춰졌을 때
	blur	focus가 풀렸을 때
	change	Input value가 변경 될 때
	reset	form을 reset시켰을 때
	submit	form을 submit시켰을 때
Form	focus	Input에 커서가 맞춰졌을 때
	blur	focus가 풀렸을 때
	change	Input value가 변경 될 때
	reset	form을 reset시켰을 때
	submit	form을 submit시켰을 때
Frame/Object	load	페이지나 데이터가 로드되었을 때
	beforeunload	페이지가 unload(닫기)되기 직전에
	unload	페이지가 unload(닫기)될 때
	Element 뿐만 아니라 window 객체나 iframe, object 태그가 생성하는 Frame, Object 객체나 또한 이벤트 시스템의 일원입니다.	
Drag	생략	마우스 드래그 관련
Touch	생략	터치(스마트폰 등) 관련
Clipboard	생략	클립보드 관련
Print	생략	인쇄 관련

<주요 DOM 이벤트 목록>

> DOM 이벤트에 대한 상세 목록은 W3C HTML DOM 이벤트 (http://www.w3schools.com/jsref/dom_obj_event.asp)에서 확인 할 수 있습니다.

핸들러가 실행될 때 이벤트 객체가 첫번째 인자로 넘어오게 됩니다. 콘솔에 이벤트 객체를 출력해보면 많은 속성이 있는 것을 확인할 수 있습니다. keyup 이벤트의 어떤 키가 눌렸는지나 click 이벤트의 어느 좌표를 클릭했는지 같이 이벤트 타입별로 가질 수 있는 독특한 속성도 있고, 이벤트 타입을 나타내는 .type이나 이벤트 발생한 근원 DOM을 가리키는 .target같은 공통적인 속성도 있습니다. 이러한 정보를 기반으로 목적에 맞게 핸들러를 작성하면 됩니다.

2. 의존성 분리

이벤트 시스템, 옵저버 패턴은 객체/모듈간의 결합을 분리(Decoupling)하는 데 도입하기도 합니다. 이를 통해 프로그램을 구조적으로 유연하게 구현 할 수 있습니다.

1. 객체간의 의존성 분리

   예를 들어서 움직이는 공들이 있고, 공들은 서로 충돌 할 수 있다고 합시다.

```
var balls = [...];

var Ball = function(...){
    //...
};
Ball.prototype.checkCollision = function(){
    var collision = false;

    balls.forEach(ball => {
        // ...
    });

    return collision;
};
```

이런 구조는 모든 ball 인스턴스가 배열 balls에 의존성을 갖게 하며 반복적인 계산으로 비용을 키웁니다. 이런 상태를 결합도가 높다고(coupled)합니다. 공은 자신의 위치와 충돌이 발생했을 때 어떻게 반응 할지(이벤트 핸들러)에 대해서만 알고 있는게 자연스럽습니다.

```
var CollisionObserver = function(...){
    this.objects = [];
    // ...
}
CollisionObserver.prototype.addObject = function(...){ ...  };
CollisionObserver.prototype.checkCollision = function(...){
    // ...
    this.objects.forEach(function(obj){
        // ...
        if (....) {
            var eventData = {...};
            obj.onCollision(eventData);
        }
    });
};

var Ball = function(...){...};
Ball.prototype.onCollision = function(e){...};

var Square = function(...){...};
Square.prototype.onCollision = function(e){...};

var Triangle = function(...){...};
Triangle.prototype.onCollision = function(e){...};
```

이후 공들을 관리하는 하나의 옵저버(Observer)가 주기적으로 공들의 충돌을 계산하고, 충돌 시 해당 공들에 이벤트를 발생시키면 되겠습니다. 이러한 구조로 확장성을 키우고, 계산 비용을 절감하며, 로직을 자연스럽게 할 수 있습니다.

2. 모듈간의 의존성 분리

공지사항을 등록하고 관리하는 모듈이 있습니다. 이 모듈에서 공지사항이 등록될 때마다 고객들에게 이메일을 보내고 싶습니다.

```
// 모듈 자체가 SOME_EMAIL_MODULE에 의존하고 있음
Announcement.prototype.create = function(...){
    // ... 공지사항 등록 후

    SOME_EMAIL_MODULE.sendEmail(...);
};
```

위 처럼 구현 할 수 있겠습니다. 이때 Announcement 모듈은 SOME_EMAIL_MODULE에 의존성을 갖게 됩니다. 추후에 ANOHTER_EMAIL_MODULE로 이메일 모듈을 교체하고, 이메일 뿐만 아니라 SOME_PUSH_MODULE 로 스마트폰 알림을 주고 싶습니다.

```
// 모듈 자체가 ANOTHER_EMAIL_MODULE, SOME_PUSH_MODULE에 의존하고 있음
Announcement.prototype.create = function(...){
    // ... 공지사항 등록 후

    ANOTHER_EMAIL_MODULE.sendEmail(...);
    SOME_PUSH_MODULE.sendPush(...);
};
```

그렇게 나쁘지 않습니다만, 이럴 때 이벤트 시스템을 도입하면

```
// 모듈 자체는 의존하는 타 모듈이 없음
Announcement.prototype.create = function(...){
    // ... 공지사항 등록 후

    var eventData = { ... };
    this.trigger('afterCreate', eventData); // 이벤트 핸들러들을 실행
};

Announcement.prototype.addHandler = function(eventType, handler){
    this.handlers[eventType].push(handler);
};

Announcement.prototype.trigger = function(eventType, eventData){
    this.handlers[eventType].forEach(handler => {
        handler.call(this, eventData); // Function의 call, apply 메
    });
};

// ...

var announce1 = new Announcement(...);

// 모듈 자체가 아니라 인스턴스만 ANOTHER_EMAIL_MODULE, SOME_PUSH_MODULE에
// 이렇게 하면 Announcement는 좀 더 유연하게 사용 할 수 있는 모듈이 되겠습니다.
announce1.addHandler('afterCreate', function(e){
    ANOTHER_EMAIL_MODULE.sendEmail(...);
    SOME_PUSH_MODULE.sendPush(...);
});
```

더 결합도가 낮고 확장성 있는 코드를 유지 할 수 있습니다. 이러한 개념은 backend를 다룰 때 다시 다뤄보겠습니다.

3. 커스텀 이벤트

JavaScript 코드로 위처럼 직접 이벤트 시스템을 구현해도 좋습니다. 하지만 웹 브라우저 DOM 이벤트 시스템을 그대로 이용하면서 이벤트 타입을 추가하고 싶다면 커스텀 이벤트 객체를 만들어서 이용 할 수 있습니다.

```
var element = document.querySelector('div');

// hello 이벤트 타입에 대한 핸들러 등록
element.addEventListener('hello', function(e) {
    console.log(arguments);
    this.innerHTML = JSON.stringify({type: e.type, detail: e.detail});
});

// 1초에 한번씩 element에 hello 이벤트를 트리거
var i = 32;
setInterval(function(){
    var event = new CustomEvent('hello', {
            detail: { // detail 속성을 전달하면 커스텀 이벤트에 데이터를 첨부 할 수 있습니다.
                code: i++,
                character: String.fromCharCode(i)
            }
        });
    element.dispatchEvent(event);
}, 1000);
```

```
<div style="font-size:2em"></div>
```

PDF에서는 미리보기가 지원되지 않습니다.

4. 이벤트 전파

이벤트는 단순히 target에게 바로 전달 될 수도 있지만, GUI 이벤트 시스템에서는 보통 구조적인 이벤트의 제어를 위해서 이벤트가 전파되는 흐름이 있습니다.

1. 캡처링 (Capturing)

   최상위 객체(window)에서부터 DOM 트리를 타고 내려오며 캡처링 속성의 이벤트 핸들러를 호출

2. 타겟 (Target)

   이벤트가 dispatch된 요소의 이벤트 핸들러를 호출

3. 버블링 (Bubbling)

타겟에서 다시 반대로 최상위 객체(window)까지 DOM 트리를 타고 올라가며 버블링 속성의 이벤트 핸들러를 호출

<이벤트 전파>

이벤트 전파

```
var one = document.getElementById('one'),
    two = document.getElementById('two'),
    three = document.getElementById('three'),
    four = document.getElementById('four'),
    btn = document.querySelector('button');

// window 객체로부터 캡쳐링이 시작되고, 버블링이 끝납니다.
window.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] window clicked');
    else
        log('[capturing] window clicked');
}, true);

// 버블링 속성의 핸들러를 등록하려면 뒤에 false 인자를 추가하고,
window.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] window clicked\n');
    else
        log('[bubbling] window clicked\n');
}, false);

// 캡쳐링 속성의 핸들러를 등록하려면 뒤에 true 인자를 추가합니다.
one.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] one clicked');
    else
```

```
    else
        log('[capturing] one clicked');
}, true);

// 세번째 인자를 주지 않으면 기본적으로 버블링 속성으로 등록됩니다.
two.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] two clicked');
    else
        log('[bubbling] two clicked');
});

three.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] three clicked');
    else
        log('[capturing] three clicked');
}, true);

// 속성에 직접 핸들러를 할당해도 버블링입니다.
four.onclick = function(e){
    if (e.target == this)
        log('[target] four clicked');
    else
        log('[bubbling] four clicked');
};

btn.onclick = function(e){
    if (e.target == this)
        log('[target] button clicked');

    // 사실 <button>이 DOM 트리의 최하위 Element라서 capturing이나 bubbling phase에서 이 핸들러
    else
        log('[bubbling] button clicked');
};


function log(newtext) {
    var logger = document.querySelector('textarea');
    logger.value += newtext + "\n";
    logger.scrollTop = logger.scrollHeight;
}
```

```
[capturing/bubbling] (window)
<div id="one">
    [capturing] one
    <div id="two">
        [bubbling] two
        <div id="three">
            [capturing] three
            <div id="four">
                [bubbling] four

                <button>[bubbling] button</button>
            </div>
        </div>
    </div>
</div>

<textarea readonly></textarea>
```

```
* {
    font-size: 1rem;
}
div {
  margin: 10px;
  padding: 10px;
  background-color: #90CAF9;
}
#two, #four {
    background-color: #1E88E5;
}
button {
    display: block;
}
textarea {
    width: 100%;
    height: 150px;
}
```

PDF에서는 미리보기가 지원되지 않습니다.

웹 브라우저의 이벤트 시스템을 이용 할 땐, 캡처링과 버블링 중, 버블링 단계의 이벤트 전파를 기본으로 생각하는 것이 좋습니다. 이벤트 핸들러를 등록할 때 디폴트가 버블링 속성이며 Internet Explorer 처럼 캡처링 단계의 전파를 지원하지 않는 브라우저가 있기 때문입니다.

event.stopPropagation()

```
var one = document.getElementById('one'),
    two = document.getElementById('two'),
```

```
    three = document.getElementById('three'),
    four = document.getElementById('four'),
    btn = document.querySelector('button');

window.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] window clicked');
    else
        log('[capturing] window clicked');
}, true);

window.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] window clicked\n');
    else
        log('[bubbling] window clicked\n');
}, false);

one.addEventListener('click', function(e){
    /** 이벤트의 전파를 종료합니다. **/
    e.stopPropagation();

    if (e.target == this)
        log('[target] one clicked');
    else
        log('[capturing] one clicked');
}, true);

two.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] two clicked');
    else
        log('[bubbling] two clicked');
});

three.addEventListener('click', function(e){
    if (e.target == this)
        log('[target] three clicked');
    else
        log('[capturing] three clicked');
}, true);

four.onclick = function(e){
    if (e.target == this)
        log('[target] four clicked');
    else
        log('[bubbling] four clicked');
};

btn.onclick = function(e){
    if (e.target == this)
        log('[target] button clicked');
    else
        log('[bubbling] button clicked');
};
```

```

function log(newtext) {
    var logger = document.querySelector('textarea');
    logger.value += newtext + "\n";
    logger.scrollTop = logger.scrollHeight;
}
```

```
[capturing/bubbling] (window)
<div id="one">
    [capturing] one: stopPropagation()
    <div id="two">
        [bubbling] two
        <div id="three">
            [capturing] three
            <div id="four">
                [bubbling] four

                <button>[bubbling] button</button>
            </div>
        </div>
    </div>
</div>

<textarea readonly></textarea>
```

```
* {
    font-size: 1rem;
}
div {
  margin: 10px;
  padding: 10px;
  background-color: #90CAF9;
}
#two, #four {
    background-color: #1E88E5;
}
button {
    display: block;
}
textarea {
    width: 100%;
    height: 150px;
}
```

PDF에서는 미리보기가 지원되지 않습니다.

이벤트 핸들러에서 이벤트 객체의 .stopPropagation() 메소드를 호출하면 이벤트의 전파를 멈출 수 있습니다.

event.preventDefault()

```
<div>
    Anchor Tag: <a href="http://google.com">google.com</a>
</div>
<div>
    Input Tag: <input type="text" placeholder="text...">
</div>
<div>
    Form Tag:
    <form>
        <input type="submit" value="Submit">
    </form>
</div>
```

```
document.querySelector('a').onclick = function(e){
    e.preventDefault();
};

document.querySelector('input[type=text]').onkeydown = function(e){
    e.preventDefault();
};

document.querySelector('form').onsubmit = function(e){
    e.preventDefault();
};
```

PDF에서는 미리보기가 지원되지 않습니다.

(커스텀 이벤트 외에) 이벤트 핸들러에서 이벤트 객체의 .preventDefault() 메소드를 호출하면 이벤트에 따른 브라우저의 기본 작동을 방지 할 수 있습니다.

5. 위임

위임(Delegation)이라 불리는 디자인 패턴이 있습니다. 위임은 말 그대로 객체의 일부 작업을 다른 객체에게 맡긴다는 것입니다. GUI의 이벤트 처리에 위임을 적용하면 계산 비용을 절감 할 뿐만 아니라, 확장성 있는 구조를 만들 수 있습니다.

구체적으로, 부모 요소에 여러 자식 요소들이 있다고 할 때, 모든 자식 요소들이 스스로 이벤트 처리를 하는 것은 큰 비용이 되거나, 프로그램의 구조를 해칠 수 있습니다. 이 때 **자식 요소들의 이벤트 처리를 부모에게 위임** 할 수 있습니다. 이는 자식 요소들에서 발생한 이벤트가 부모 요소에까지 전파되기 때문에 가능한 패턴입니다.

위임을 적용한 이벤트 처리

```
var h4 = document.querySelector('h4'),
    ul = document.querySelector('ul');

// 부모 요소에서 자식들의 이벤트 처리를 위임 받음
ul.addEventListener('click', function(e){
    if(e.target.tagName == "LI") {
        var index = parseInt(e.target.getAttribute('index')),
            value = e.target.getAttribute('some-value');

        h4.textContent = 'Item '+index+' has value '+value;

        if (index % 2 == 0) {
            h4.style.backgroundColor = 'yellow';
        } else {
            h4.style.backgroundColor = 'white';
        }
    }
});
```

```
<h4>Click below items</h4>
<ul>
    <li index="1" some-value="a">list item 1</li>
    <li index="2" some-value="b">list item 2</li>
    <li index="3" some-value="c">list item 3</li>
    <li index="4" some-value="d">list item 4</li>
    <li index="5" some-value="e">list item 5</li>
</ul>

<style>
* {font-size: 1.5rem;}
</style>
```

PDF에서는 미리보기가 지원되지 않습니다.

6. 크로스 브라우징

크로스 브라우징(Cross Browsing)은 웹 브라우저간 호환성에 문제가 없음을 의미하는 단어입니다. 웹 브라우저간 지원하는 JavaScript의 버전의 차이, 웹 표준을 따르지 않는 API(간단한 예로 IE의 JavaScript API에는 DOM 요소에 .addEventListner 라는 메소드가 없고 .attachEvent 라는 메소드가 있습니다.) 등으로, 다양한 웹 브라우저를 사용하는 유저들을 모두 만족시키려면 개발자의 애로사항이 큽니다.

이를 해결하기 위해서는 Shim/Polyfill으로 불리는 native API의 간극을 메꾸어주는 라이브러리를 쓸 수도 있고, jQuery 같이 native API의 크로스 브라우징 문제를 해결하면서, DOM 조작 등 스크립팅의 편의를 위한 라이브러리를 이용 할 수도 있습니다.

[1] Event-driven programming
[2] Event Listener
[3] Observer Pattern
[4] Publish-Subscribe Pattern
[5] Capturing
[6] Target
[7] Bubbling
[8] Delegation
[9] Cross Browsing

3 웹 프론트엔드

3.7 jQuery

1. jQuery API

> JavaScript 라이브러리인 jQuery에 대해서 알아봅니다.

1. jQuery API

jQuery는 빠르고 많은 기능을 지원하는 JavaScript 라이브러리입니다. jQuery는 DOM 조작을 쉽게 해줍니다. 또한 이벤트, 애니메이션, Ajax(추후에 다룸) 등을 다루는 간편한 API를 제공합니다. 또한 크로스 브라우징을 염두에 두고 개발되었습니다.

API	설명
$ 또는 jQuery	jQuery의 최상위 객체
$(DOM 객체)	DOM 객체를 받아 jQuery 인스턴스를 생성
$('태그')	태그를 받아 jQuery 인스턴스를 생성
$(selector)	문서에서 selector에 매치되는 Element들을 찾아 jQuery 인스턴스를 생성
$(callback)	문서가 해석되고 DOM이 로드된 후 (DOMContentLoaded 이벤트 후) callback을 실행

<jQuery 객체 및 주요 함수>

```
jQuery(window);
$(window);
var elements = $([document.head, document.body]);
var div = $('<div style="font-size:2em"><h1>hello</h1>...</div>'); // document.creat
var inputs = $("#some-id input[type=text].valid, input[type=password]"); // document

$(function(){
    // ...
});
```

jQuery 인스턴스는 DOM 객체들을 원소로하는 배열과 비슷합니다.

API	설명

.html(string)	element.innerHTML=string 처럼 작동, 인자가 없으면 현재 html을 리턴
.text(string)	element.textContent=string 처럼 작동, 인자가 없으면 현재 text를 리턴
.append(element)	element.appendChild(element) 처럼 작동
.appendTo(parentElement)	element를 parentElement에 삽입
.prepend()	element.appendChild(element) 처럼 작동하나 첫째 자식으로 삽입
.prependTo(parentElement)	element를 parentElement에 첫째 자식으로 삽입
.remove()	element를 부모에서 제거
.remove(selector)	element의 자식 중에서 Selector에 매치되는 자식을 제거
.css(property, value)	element.style[property] = value 처럼 작동
.css({property1: value1, ..})	
.addClass('class1 class2')	element에 클래스 추가
.removeClass('class1 class2')	element에 클래스 제거
.toggleClass('class-x')	element에 클래스 토글
.hide()	element.style.display = 'none' 처럼 작동
.show()	element.style.display = 'block' 처럼 작동
.toggle()	element.style.display 속성을 토글
.attr(name)	element.getAttribute(name) 처럼 작동
.attr(name, value)	element.setAttribute(name, value) 처럼 작동
.removeAttr(name)	element.removeAttribute(name) 처럼 작동
.val()	element.value나 textarea 태그의 내용, select 태그의 선택된 value들의 배열 등 적절한 값을 리턴
.val(newValue)	element.value=newValue 처럼 적절하게 작동
.on(eventType, handler)	element.addEventListener(eventType, handler) 처럼 동작
.one(eventType, handler)	1회만 실행되는 이벤트 핸들러를 등록
.click(handler)	element.addEventListener('click', handler) 처럼 동작
.click()	element.click() 처럼 동작, element에 click 이벤트를 트리거
.trigger(eventType)	element에 eventType의 이벤트를 트리거
.each(callback)	배열의 .forEach 메소드처럼 인스턴스에 담긴 DOM 객체들에 대해서 callback을 차례대로 실행
.data(key, value)	element와 대응되는 임시 object의 key 속성에 value를 저장

.data(key)	element와 대응되는 임시 object의 key 속성에 해당하는 value를 리턴 또는 element의 data-key attribute를 리턴
.data()	element와 대응되는 임시 object를 리턴 또는 element의 data-* attribute를 모두 리턴
.removeData(key)	element와 대응되는 임시 object의 key 속성을 삭제

<jQuery 주요 인스턴스 메소드>

> jQuery 공식 API (http://api.jquery.com/) 및 다운로드 페이지 (http://jquery.com/download/)와 CDN 페이지 (https://code.jquery.com/)
> CDN(Content Delivery Network)이란 여러 서버에 동일한 자원을 올려놓고, 접속자가 가까운 곳에서 컨텐츠를 빠르게 불러들일 수 있도록 지원하는 네트워크입니다... 쉽게는 가져다 쓰기 좋은 자원들이 올려진 서버라고 보시면 좋겠습니다.

jQuery 예제

```
<script src="https://code.jquery.com/jquery-3.1.1.js"></script>
<!-- jQuery를 사용하려면 jQuery script를 먼저 로드해 줘야함 -->
<script>
$(function(){ // script가 아래에 있는 태그들을 참조하더라도 DOM이 로드된 후에 실행되므로 문제 없음
    $('h1').css({
        color: '#fff',
        fontSize: '2em'
    }).each(function(i, item) {

        $(item)
            .css('backgroundColor', $(item).data('bg'))
            .text($(item).data('xyz'));

    }).addClass('someClass'); // 대부분의 인스턴스 메소드가 체이닝을 지원
});
</script>
<style>
.someClass {
    padding: 10px;
}
</style>

<h1 data-xyz="hello" data-bg="red"></h1>
<h1 data-xyz="bye" data-bg="green"></h1>
<h1 data-xyz="see you" data-bg="blue"></h1>
```

PDF에서는 미리보기가 지원되지 않습니다.

[1] CDN, Content Delivery Network

3 웹 프론트엔드

3.8 확장성있는 코드짜기

1. TODO List 만들기

> 할일 리스트를 관리하는 프로그램을 만들어봅니다. 확장성을 높히는 소프트웨어 개발 원리에 대해서 배웁니다.

1. TODO List 만들기

3장에서 배운 내용으로 모델링, 뷰와 데이터의 분리, 확장성에 집중해서 작은 프로그램을 만들어 보겠습니다. 확장성이 좋은 코드는 구조의 큰 변경 없이도 기능을 추가하고 변경 할 수 있습니다. 또한 코드가 기능별로 잘 분리되면 유지보수와 협업에 큰 장점을 갖습니다. 뷰와 데이터가 분리되어 있지 않은 코드, OOP와 같은 기법으로 좋은 모델링이 되지 않은 코드는 확장성이 좋지 않습니다.

만들어볼 TODO List의 명세는 다음과 같습니다.

1. 항목을 입력하면 리스트에 등록한 시간과 항목이 등록됨
2. 완료한 항목을 누르면 완료되었다는 표시
3. 완료된 항목을 누르면 다시 원래 상태로 복원

코드 작성의 순서는 다음과 같습니다.

1. 명세를 바탕으로 모델링
2. 모델 작성

3. 모델과 뷰 엮기
4. 이벤트 처리
5. 뷰 꾸미기

TODO list 모델

```
/** Item class **/
var TodoListItem = function(text){
    this.text = text;
    this.done = false;
    this.date = new Date();
};

TodoListItem.prototype.toggle = function(){
    this.done = !this.done;
    return this;
};

/** List class **/
var TodoList = function(){
    this.items = [];
};

TodoList.prototype.addItem = function(text){
    var item = new TodoListItem(text);
    this.items.push(item);
    return this;
};
```

임의의 뷰(DOM Element)를 TodoItem이나 TodoListItem의 속성으로 동적으로 생성해도 문제는 없습니다. 다만 뷰와 두 모델의 의존성을 끊고, 뷰를 유연하게 하기 위해서 다른 방식을 사용해보겠습니다.

TODO list 모델과 뷰 엮기

```
<!-- 원하는 태그대로 작성한 뷰, 그러나 특정한 className을 통해서 뷰의 부품들을 명시해주고 있습니다. -->
<ul id="myTodoList">
    <li class="tl-item">
        <span class="tl-text"></span>
        <span class="tl-date"></span>
    </li>
</ul>

<script>
/** Item class **/
var TodoListItem = function(text){
```

```
12      this.text = text;
13      this.done = false;
14      this.date = new Date();
15  };
16
17  TodoListItem.prototype.toggle = function(){
18      this.done = !this.done;
19      return this;
20  };
21
22  /** List class **/
23  var TodoList = function(view){ // 뷰로 사용 할 Element를 인자로 받습니다
24      // data
25      this.items = [];
26
27      // view
28      this.view = view;
29      this.itemView = document.importNode(view.getElementsByClassName(this.className.i
30      this.render(); // 데이터를 뷰에 반영해라!
31  };
32
33  TodoList.prototype.className = { // 유지보수의 편의, 재사용성을 위해서 굳이 따로 빼서 관리 해줍니다
34      item: 'tl-item',
35      itemText: 'tl-text',
36      itemDate: 'tl-date',
37      itemDone: 'tl-done'
38  };
39
40  TodoList.prototype.addItem = function(text){
41      var item = new TodoListItem(text);
42      this.items.push(item);
43      return this;
44  };
45
46   // Data를 View에 반영하는 함수
47  TodoList.prototype.render = function(){
48
49      // 먼저 자식 item들을 전부 삭제 줍니다.
50      var itemViews = this.view.getElementsByClassName(this.className.item);
51      while(itemViews.length > 0) // Element의 자식들을 지울 때는, 자료 구조상의 문제로, 뒷자식부터
52          this.view.removeChild(itemViews[itemViews.length-1]);
53
54      // 데이터를 순회하며 item별 view를 만들고 자식으로 삽입합니다.
55      for(var i=0; i < this.items.length; i++) {
56          var item = this.items[i];
57          var itemView = document.importNode(this.itemView, true); // 아이템 뷰를 복제
58
59          // 데이터를 뷰에 반영
60          itemView.getElementsByClassName(this.className.itemText)[0].innerHTML = item
61          itemView.getElementsByClassName(this.className.itemDate)[0].innerHTML = item
62
63          if (item.done)
64              itemView.classList.add(this.className.itemDone);
65
66          this.view.appendChild(itemView);
67      };
68
```

```
    return this;
};


/** 사용하기 **/
var myView = document.getElementById("myTodoList");
var list = new TodoList(myView);
list
    .addItem("테스트1").addItem("테스트2").addItem("테스트3")
    .render();
</script>
```

PDF에서는 미리보기가 지원되지 않습니다.

이제 항목을 클릭했을 때 toggle 처리를 해보겠습니다. GUI의 이벤트는 뷰(DOM 객체)에서 발생하기 때문에, 이벤트 처리는 뷰에 의존 할 수 밖에 없습니다. 뷰와 객체간에 최소한의 참조만 유지하도록 하면서, 이벤트 처리에 위임을 적용해 보겠습니다.

TODO list 이벤트 처리 및 뷰 꾸미기

```
<!-- 원하는 태그대로 작성한 뷰, 그러나 특정한 className을 통해서 뷰의 부품들을 명시해주고 있습니다. -->
<ul id="myTodoList" class="tl-list">
    <li class="tl-item">
        <span class="tl-text"></span>
        <span class="tl-date"></span>
    </li>
</ul>

<script>
/** Item class **/
var TodoListItem = function(text){
    this.text = text;
    this.done = false;
    this.date = new Date();
};

TodoListItem.prototype.toggle = function(){
    this.done = !this.done;
    return this;
};

/** List class **/
var TodoList = function(view){ // 뷰로 사용 할 Element를 인자로 받습니다
    // data
    this.items = [];

```

```
    // view
    this.view = view;
    this.itemView = document.importNode(view.getElementsByClassName(this.className.i
    this.render(); // 데이터를 뷰에 반영해라!

    // data-view
    var self = this;
    this.view.addEventListener('click', function(e){
        /**
        이벤트 핸들러 안에서 this가 이벤트 핸들러를 호출하는 element로 바뀐 것 기억하시나요?
        핸들러 안에서 TodoList의 인스턴스를 참조하려면 밖에서 정의해둔 self라는 변수를 씁니다.

        이벤트의 근원이 .tl-item이거나 .tl-item의 자식인지 확인하기 위해서
        e.target에서 가장 가까운 .tl-item을 가진 (자신 포함) 부모을 찾습니다.
        **/
        var itemView = e.target.closest('.'+self.className.item);
        if (itemView) {
            itemView.itemObject.toggle();
            self.render(); // 다시 그려라!
        }
    });
};

TodoList.prototype.className = { // 유지보수의 편의, 재사용성을 위해서 굳이 따로 빼서 관리 해줍니다
    item: 'tl-item',
    itemText: 'tl-text',
    itemDate: 'tl-date',
    itemDone: 'tl-done'
};

TodoList.prototype.addItem = function(text){
    var item = new TodoListItem(text);
    this.items.push(item);
    return this;
};

 // Data를 View에 반영하는 함수
TodoList.prototype.render = function(){

    // 먼저 자식 item들을 전부 삭제 줍니다.
    var itemViews = this.view.getElementsByClassName(this.className.item);
    while(itemViews.length > 0) // Element의 자식들을 지울 때는, 자료 구조상의 문제로, 뒷자식부터
        this.view.removeChild(itemViews[itemViews.length-1]);

    // 데이터를 순회하며 item별 view를 만들고 자식으로 삽입합니다.
    for(var i=0; i < this.items.length; i++) {
        var item = this.items[i];
        var itemView = document.importNode(this.itemView, true); // 아이템 뷰를 복제

        // 데이터를 뷰에 반영
        itemView.getElementsByClassName(this.className.itemText)[0].innerHTML = item
        itemView.getElementsByClassName(this.className.itemDate)[0].innerHTML = item

        if (item.done)
            itemView.classList.add(this.className.itemDone);

        // [illegible]
```

```
        // data-view
        itemView.itemObject = item;

        this.view.appendChild(itemView);
    };

    return this;
};


/** 사용하기 **/
var myView = document.getElementById("myTodoList");
var list = new TodoList(myView);
list
    .addItem("테스트1").addItem("테스트2").addItem("테스트3")
    .render();
</script>
```

```
.tl-list {
    margin: 15px;
    padding: 30px;
    list-style: none;
    font-size: 1.5em;
    background: #fff;
    border: 1px solid #eee;
    box-shadow: 1px 0 5px 0 rgba(100,100,100,0.5);
}
.tl-item {
    border-bottom: 1px solid #eee;
    padding:15px 0;
}
.tl-item.tl-done {
    text-decoration: line-through;
    opacity: 0.4;
}
.tl-list .tl-text {
    font-weight: 600;
    color: #333;
}
.tl-list .tl-date {
    font-size: 0.5em;
    color: #888;
}
```

PDF에서는 미리보기가 지원되지 않습니다.

현재 TodoList 모듈의 의존성은 TodoListItem과, 생성자에 넘기는 View가 .tl-item, .tl-text, .. 같은 css 클래스를 지녀야 한다는 점입니다. 상당히 느슨한 프로그램이 됐습니다. 프로그램을 조금 확장해 보겠습니다.

TODO list 확장하기

```
<!-- 원하는 태그대로 작성한 뷰, 그러나 특정한 className을 통해서 뷰의 부품들을 명시해주고 있습니다. -->
<ul id="myTodoList" class="tl-list">
    <h4>TodoList Final</h4>

    <li class="tl-item">
        <span class="tl-text"></span>
        at <span class="tl-date"></span>
    </li>
</ul>

<!-- 뷰를 조금 수정하고 폼을 추가했습니다. -->
<form id="myTodoListForm">
    <input type="text" name="item" placeholder="Enter Item!">
    <input type="submit" value="Add">
    <input type="button" name="do" value="Do All">
    <input type="button" name="undo" value="Undo All">
    <input type="button" name="clearDone" value="Clear Done">
    <input type="button" name="clearAll" value="Clear All">
</form>

<script>
/** Item class **/
var TodoListItem = function(text){
    this.text = text;
    this.done = false;
    this.date = new Date();
};

TodoListItem.prototype.toggle = function(){
    this.done = !this.done;
    return this;
};

TodoListItem.prototype.setDone = function(isDone){
    this.done = !!isDone;
    return this;
};

/** List class **/
var TodoList = function(view){ // 뷰로 사용 할 Element를 인자로 받습니다
    // data
    this.items = [];

    // view
    this.view = view;
    this.itemView = document.importNode(view.getElementsByClassName(this.className.i
    this.render(); // 데이터를 뷰에 반영해라!
```

```

    // data-view
    var self = this;
    this.view.addEventListener('click', function(e){
        /**
        이벤트 핸들러 안에서 this가 이벤트 핸들러를 호출하는 element로 바뀐 것 기억하시나요?
        핸들러 안에서 TodoList의 인스턴스를 참조하려면 self라는 변수를 씁니다.

        이벤트의 근원이 .tl-item이거나 .tl-item의 자식인지 확인하기 위해서
        e.target에서 가장 가까운 .tl-item을 가진 (자신 포함) 부모을 찾습니다.
        **/
        var itemView = e.target.closest('.'+self.className.item);
        if (itemView) {
            itemView.itemObject.toggle();
            self.render(); // 다시 그려라!
        }
    });
};

TodoList.prototype.className = { // 유지보수의 편의, 재사용성을 위해서 굳이 따로 빼서 관리 해줍니다
    item: 'tl-item',
    itemText: 'tl-text',
    itemDate: 'tl-date',
    itemDone: 'tl-done'
};

TodoList.prototype.addItem = function(text){
    var item = new TodoListItem(text);
    this.items.push(item);
    return this;
};

/** 기능 추가 **/

TodoList.prototype.clearAll = function(){
    this.items = [];
    return this;
};

TodoList.prototype.clearDone = function(){
    this.items = this.items.filter(function(item){
        return !item.done;
    });
    return this;
};

TodoList.prototype.doAll = function(){
    this.items.forEach(function(item){
        item.setDone(true);
    });
    return this;
};

TodoList.prototype.undoAll = function(){
    this.items.forEach(function(item){
        item.setDone(false);
    });
```

```
    });
    return this;
};

 // Data를 View에 반영하는 함수
TodoList.prototype.render = function(){

    // 먼저 자식 item들을 전부 삭제 줍니다.
    var itemViews = this.view.getElementsByClassName(this.className.item);
    while(itemViews.length > 0) // Element의 자식들을 지울 때는, 자료 구조상의 문제로, 뒷자식부터
        this.view.removeChild(itemViews[itemViews.length-1]);

    // 데이터를 순회하며 item별 view를 만들고 자식으로 삽입합니다.
    for(var i=0; i < this.items.length; i++) {
        var item = this.items[i];
        var itemView = document.importNode(this.itemView, true); // 아이템 뷰를 복제

        // 데이터를 뷰에 반영
        itemView.getElementsByClassName(this.className.itemText)[0].innerHTML = item
        itemView.getElementsByClassName(this.className.itemDate)[0].innerHTML = item

        if (item.done)
            itemView.classList.add(this.className.itemDone);

        // data-view
        itemView.itemObject = item;

        this.view.appendChild(itemView);
    };

    return this;
};


/** 사용하기 **/
var myView = document.getElementById("myTodoList");
var list = new TodoList(myView);
list
    .addItem("테스트1").addItem("테스트2").addItem("테스트3")
    .render();

/** form과 list 인스턴스 연결하기 **/
myTodoListForm.onsubmit = function(e){
    e.preventDefault();

    if (this.item.value != '') {
        list.addItem(this.item.value).render();
        this.item.value = '';
    }
};


myTodoListForm.clearDone.onclick = function(){
    list.clearDone().render();
};

myTodoListForm.clearAll.onclick = function(){
```

```
    list.clearAll().render();
};

myTodoListForm.do.onclick = function(){
    list.doAll().render();
};

myTodoListForm.undo.onclick = function(){
    list.undoAll().render();
};
</script>
```

```
.tl-list {
    margin: 15px;
    padding: 30px;
    list-style: none;
    font-size: 1.5em;
    background: #fff;
    border: 1px solid #eee;
    box-shadow: 1px 0 5px 0 rgba(100,100,100,0.5);
}
.tl-item {
    border-bottom: 1px solid #eee;
    padding:15px 0;
}
.tl-item.tl-done {
    text-decoration: line-through;
    opacity: 0.4;
}
.tl-list .tl-text {
    font-weight: 600;
    color: #333;
}
.tl-list .tl-date {
    font-size: 0.5em;
    color: #888;
}

form {
    margin-top:30px;
}
input {
    margin:10px;
    font-size:2em
}
```

PDF에서는 미리보기가 지원되지 않습니다.

4 웹 백엔드

1. 모듈, NPM ... 209
2. 스트림, 표준입출력, 소켓 ... 218
3. HTTP 프로토콜 ... 224
4. 웹 브라우저의 Request ... 231
5. 정적 웹 서버의 Response ... 239
6. 동적 웹 서버 ... 245
7. Express.js ... 254
8. 쿠키와 세션, 인증 ... 262
9. 동기와 비동기, Promise ... 272
10. Ajax, WebSocket ... 285
11. 보안, Same Origin Policy ... 298
12. REST API, OAuth, SPA ... 306

Node.js의 코어 모듈을 이용해 TCP 서버를 만들어 서버-클라이언트 구조에 대해 이해합니다. 이후 HTTP 프로토콜에 대해 공부하고 웹 서버를 만들어 봅니다. 나아가 Express.js라는 웹 서버 프레임워크에 대해 알아봅니다. 나아가서 쿠키, 세션, Ajax, 웹 소켓, REST API, OAuth, JWT, SPA 등 웹의 다양한 요소와 웹 보안에 대해 공부합니다.

4 웹 백엔드

4.1 모듈, NPM

1. 모듈
2. 내장 모듈
3. 전역 객체
4. NPM

> 모듈의 개념과 Node.js의 모듈에 대해서 알아보고, NPM에 대해서 알아봅니다.

1. 모듈

<모듈>

> ... 모듈은 여러 가지로 정의될 수 있지만, 일반적으로 큰 체계의 구성요소이고, **다른 구성요소와 독립적**으로 운영된다... 프로그램에서 **임의의 두 부분이 직접적인 상호관계가 많아지면, ... 모듈성이 떨어진다...** (위키백과)

특정 기능을 하는 재사용 가능한, 독립적인 코드를 모듈(Module)이라고 합니다. 모듈을 작성 할 때는 다른 모듈과의 **직접적인 상호관계를 최소화**하며, 상호간에 **느슨한 인터페이스를 노출**하는 것이 스파게티 코드를 방지하고, 짜임새 있는 프로그램을 작성하는 지름길 입니다. Node.js에서는 **파일 하나를 하나의 모듈로** 인식합니다. 코드 상에서 **모듈의 실체는 일반적인 값, 또는 객체**입니다.

./gugu.js

```
function notExported(){
    // ...
}

function printGuguAll(){
    printGugu(1,9);
}

function printGugu(x,y){
    for(var i=x; i <= y; i++)
        for(var j=1; j <= 9; j++)
            console.log(i+'*'+j+'='+i*j);
}

module.exports = { // 이 모듈은 이 객체를 노출합니다.
    print: printGugu,
    printAll: printGuguAll
};
```

./some-module.js

```
var fs = require('fs'); // 내장 모듈은 경로를 지정 할 필요가 없습니다.
var gu = require('./gugu'); // 상대 경로나 절대 경로로 모듈의 경로를 정확히 지정해야합니다. .js 확장

gu.print(3,4);

// 이 모듈은 {}를 노출합니다.

/*
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
4*1=4
4*2=8
4*3=12
4*4=16
4*5=20
4*6=24
4*7=28
4*8=32
4*9=36
*/
```

위처럼 module.exports라는 객체에 값을 덮어 씌우거나 속성을 할당함으로써 모듈의 내부는 숨기고, 모듈 밖으로 제공하고 싶은 인터페이스를 노출 할 수 있습니다. 타 모듈을 사용 할 때는 require라는 내장 함수를 이용합니다.

require('some-module')

1. some-module이 내장 모듈(Core Module)이면 그 바이너리 파일을 로드한다.
2. 그렇지 않다면 ./node_modules 폴더가 있는지 확인하고 ./node_modules/some-module 폴더를 찾는다.
3. 폴더를 찾으면 some-module 폴더를 로드하고 없으면 상위 디렉토리로 이동하여 다시 위 작업을 반복한다.
4. /node_modules에서도 some-module 폴더를 찾지 못하면 오류

require('../path/some-module')

1. 절대 경로나 상대 경로가 주어질 때 그 경로가 폴더라면 폴더를 로드한다.
2. 그렇지 않다면 경로 뒤에 .js .json .node 를 붙혀가면서 찾아보고 파일을 로드한다.
3. 그래도 찾지 못하면 오류

파일을 로드

1. module.exports 값을 리턴

폴더를 로드

1. 폴더에 package.json 파일이 있다면 main 파일의 경로를 찾아 그 main 파일을 로드한다.
2. package.json이 없다면 폴더의 index.js 파일을 로드한다.
3. 파일의 module.exports 값을 리턴

이미 로드된 모듈을 로드하는 경우

1. 처음 로드하는 모듈의 경우, 모듈의 코드를 실행하고 module.exports 값을 리턴하는 동시에 메모리에 module.exports 값을 기억해둔다.
2. 다시 모듈을 로드하는 경우, 모듈의 코드를 실행하지 않고 메모리에 있는 해당 모듈의 module.exports 값을 참조한다.

package.json

```
{
"name": "workshop",
"version": "1.0.0",
"main": "server/server.js", // main 파일의 경로
"scripts": {
  "lint": "eslint .",
  "start": "node .",
  "dev": "nodemon .",
  "posttest": "npm run lint && nsp check"
},
"dependencies": {
  "async": "^2.0.1",
  "compression": "^1.0.3",
  "cors": "^2.5.2",
  "escape-html": "^1.0.3",
...
```

> JSON(JavaScript Object Notation)은 속성-값 쌍으로 이루어진 ... 포맷이다... 인터넷에서 자료를 주고 받을 때 그 자료를 표현하는 방법으로 알려져 있다. ... 본래는 자바스크립트 언어로부터 파생되어 자바스크립트의 구문 형식을 따르지만 언어 독립형 데이터 포맷이다. 즉, 프로그래밍 언어나 플랫폼에 독립적이므로, 구문 분석 및 JSON 데이터 생성을 위한 코드는 C, C++, C#, 자바, 자바스크립트, 펄, 파이썬 등 수많은 프로그래밍 언어에서 쉽게 이용할 수 있다. ... (위키백과)

2. 내장 모듈

모듈	이름	기능
os	OS	운영체제 및 하드웨어 정보 관련
fs	**File System**	**파일 시스템 관련**
path	**Path**	**운영체제별 경로 관련**
child_process	Child Processes	(자식) 프로세스 실행 관련
events	Events	이벤트 시스템 관련

cluster	Cluster	멀티 코어 CPU를 위한 병렬처리 관련
crypto	Crypto	데이터 암호화 관련
http	**HTTP**	**HTTP 프로토콜 관련**
https	**HTTPS**	**HTTPS 프로토콜 관련**
net	**Net**	**TCP 관련**
dgram	UDP/Datagram	UDP 관련
tls	TLS/SSL	TCP/IP 암호화 프로토콜 관련
dns	DNS	도메인과 IP 관련
url	URL	URL 관련
querystring	Query Strings	URL 중 query string 관련
readline	Readline	CLI에서 사용자 입력 관련
repl	REPL	대화형 CLI를 만드는 관련
buffer	Buffer	바이너리 관련
string_decoder	String Decoder	바이너리를 인코딩에 따라 해석
util	Utilities	잡다한 도우미 함수 관련

<Node.js 주요 내장 모듈>

Node.js의 공식 API 문서 (https://nodejs.org/api/)에는 전체 내장 모듈들의 인터페이스에 대한 설명과 예제 코드들이 잘 정리 되어있습니다. 위 표에서 어둡게 강조된 모듈은 수업 과정에서 주요하게 사용하게 될 내장 모듈들입니다.

3. 전역 객체

객체	설명	예시
global	최상위 전역 객체	global.someThing = 10; console.log(someThing); // 10
console	**CLI에 텍스트 출력 관련**	**console.log();**
process	현재 프로세스 관련	process.exit();

		process.env // 환경 변수
module	**현재 모듈 관련**	**module.exports = 20;**
require	**모듈 로드 관련**	**require('express');**
__dirname	**현재 모듈이 위치한 디렉토리의 절대 경로**	**/Users/dehypnosis/benzene/workshop**
__filename	현재 모듈의 절대 경로	/Users/dehypnosis/benzene/workshop/test.js

<Node.js 주요 전역 객체>

전역 객체 (https://nodejs.org/api/globals.html)는 console.log의 console이나 require 또는 표준 내장 객체(Array, String 등) 처럼 따로 require하지 않고서도 바로 이용할 수 있는 객체, 값들을 의미합니다.

4. NPM

어떤 프로그램을 만들 때 수 많은 부품이 필요 할 수 있습니다. 모든 부품을 장인 정신으로 만들 수도 있지만, 생산성을 위해 타인이 만들어 둔 부품들을 이용 할 수도 있습니다. Node.js에서는 인터넷에 공개된 패키지(모듈)를 설치하고 관리하는 도구로 NPM(Node Package Manager)을 제공합니다. Node.js를 설치할 때 같이 CLI 프로그램 npm이 설치됩니다.

웹 서버를 만들 때는 Express.js, 웹 소켓을 구현할 때는 Socket.io 등 개발에 도움이 되는 오픈소스 패키지들이 많습니다. 유명한 패키지에 대해서 모르더라도 필요한 기능에 대한 공개 패키지를 구글링 해보면 많은 시간을 줄일 수도 있습니다. 또는 패키지를 만들고 인터넷에 공개하여 오픈소스 생태계에 도움을 줄 수도 있습니다.

> https://www.npmjs.com/ (https://www.npmjs.com/)
> Top 30 NPM Packages for Node.js Developers 2016
> (https://colorlib.com/wp/npm-packages-node-js/)

package.json

```
{
"name": "workshop", // 패키지에 대한 설명
"version": "1.0.0",
"main": "server/server.js", // main 파일의 경로
"scripts": {
  "lint": "eslint .",
  "start": "node .",
  "dev": "nodemon .",
  "posttest": "npm run lint && nsp check"
},
"dependencies": {
  "async": "^2.0.1", // 의존 모듈들
  "compression": "^1.0.3",
  "cors": "^2.5.2",
  "escape-html": "^1.0.3",
...
```

이렇게 프로그램에서 다른 독립적인 모듈을 이용하는 것을 의존성(Dependency)을 갖는다고 합니다. npm은 이러한 의존성을 관리하기 위해서 모듈의 ./package.json 파일에 모듈이 의존하고 있는 모든 모듈의 이름과 버전을 기록해둡니다. 이후에 모듈을 배포 할 때는 타 모듈들이 설치된 ./node_modules 폴더를 제외하고 package.json을 포함한 모듈의 코드를 배포하면 충분합니다.

npm 프로그램 사용법

```
mkdir package
cd package

# 현재 폴더에 package.json 파일을 생성
npm init

# nodemon 모듈을 전역으로 설치
npm install nodemon -g

# express 모듈을 ./node_modules에 설치하고 package.json에 기록
npm install express --save

# express 모듈을 제거
npm remove express

# 현재 패키지의 의존성 트리를 출력
npm list

# package.json에 기록된 의존 모듈들을 모두 ./node_modules에 설치
npm install

# express 모듈을 제거하고 package.json에서 지움
npm remove express --save
npm list
```

[1] Module

[2] JSON, JavaScript Object Notation

[3] NPM, Node Package Manager

[4] Dependency

4 웹 백엔드

4.2 스트림, 표준입출력, 소켓

1. 스트림
2. 프로세스와 소켓

> 스트림(Stream), 표준입출력(Standard I/O), 소켓(Socket)에 대해 이해하고, TCP 통신을 이용해서 CLI 채팅 서비스와 Non-SSH를 만들어봅니다.

1. 스트림

OS는 기억장치, 디스플레이, 키보드 등의 주변 장치 및 네트워크와 프로세스 간의 소통을 담당하고 있습니다. 이 때 OS는 주변 장치나 네트워크를 모두 파일로 추상화한 API를 프로세스에 제공합니다.
(UNIX 기반 OS뿐만 아니라 Windows 또한 부분적으로)

<스트림이라는 개념 자체는 프로그래밍 언어와 무관합니다>

이렇게 운영체제가 제공하는 프로세스와 장치 사이에 데이터를 주고 받을 수 있는 인터페이스를 스트림(Stream) 또는 파일 포인터(File Pointer)라고 일컫습니다. 스트림은 입력 스트림(Input Stream)이나 출력 스트림(Output Stream), 또는 양방향 스트림(Duplex Stream)이 될 수 있습니다.

<프로세스와 스트림>

프로세스는 키보드나 마우스, 트랙패드, 마이크, 파일 등의 입력 스트림을 통해 데이터를 입력을 받을 수 있고, 모니터나 스피커, 프린터, 파일 등의 출력 스트림을 통해 데이터를 출력 할 수 있습니다.

주요한 스트림

1. 표준 입출력 스트림 (STDIN, STDOUT, STDERR)

   프로세스가 생성되면 기본적으로 3개의 스트림이 부착됩니다. 이는 차례대로 기본적인 입력, 출력, 오류에 관련된 정보를 주고 받기 위한 스트림입니다. CLI 프로그램에서 기본적으로 STDIN은 키보드 입력에 연결 되어 있으며, 셸로 실행한 CLI 프로그램의 경우에는 STDOUT, STDERR는 모니터를 통해 출력 될 수 있습니다.

2. 파일 또는 주변 장치 스트림

   프로세스는 실행 중에 기억장치의 파일이나, 주변 장치에 입출력 스트림을 생성 할 수 있습니다. 이를 통해서 파일을 읽고, 쓰고, 생성하고 또 주변 장치에 데이터를 읽고 쓸 수 있습니다.

3. 네트워크 장치 (소켓)

   프로세스는 실행 중에 다른 프로세스와 네트워크 연결(TCP 또는 UDP)을 맺을 수 있습니다. 이 때 데이터 입출력의 매개가 되는 스트림을 생성하는 데 이를 소켓 (Socket)이라고 부릅니다.
   물론 통신은 두 프로세스 사이에 이루어지기 때문에 통신하는 두 호스트의 프로세스마다 소켓이 생성됩니다. 소켓 스트림은 양방향(Duplex Stream)일 수도 또는 단순히 쓰기 전용(Output Stream), 읽기 전용(Input Stream) 일 수 있습니다.

<소켓 스트림은 IP와 Port로 구분되는 특정 프로세스에 부착됩니다>

> 네트워크 소켓(Network Socket)은 컴퓨터 네트워크를 경유하는 프로세스 간 통신의 종착점이다. 오늘날 컴퓨터 간 통신의 대부분은 인터넷 프로토콜을 기반으로 하고 있으므로, 대부분의 네트워크 소켓은 인터넷 소켓이다. 네트워크 통신을 위한 프로그램들은 소켓을 생성하고, 이 소켓을 통

해서 서로 데이터를 교환한다.... 인터넷 소켓은 다음과 같은 요소들로 구성되어 있다. 인터넷 프로토콜 (TCP, UDP, raw IP), 로컬 IP 주소 및 포트, 원격 IP 주소 및 포트. 인터넷 소켓은 크게 UDP, TCP 프로토콜을 사용하는 두 개의 타입으로 분류할 수 있다... (위키백과)

2. 프로세스와 소켓

이제는 웹 브라우저 위에서 작동하는 단순한 스크립트가 아니라, 단독 프로세스로 실행될 프로그램을 작성하기 때문에, OS가 프로세스에 제공하는 정보나 기능들에 대해서 먼저 살펴 보도록 하겠습니다. Node.js의 Process API (https://nodejs.org/api/process.html)에 대해서 알아봅니다.

속성	내용
process.pid :Number	프로세스 아이디(운영체제가 할당해주는 번호)
process.arch :String	프로세스가 실행되는 CPU의 아키텍쳐
process.env :Object<String>	환경 변수를 담고 있는 객체
process.argv :Array<String>	프로그램 실행시 전달된 매개 변수들의 배열
process.exit()	프로세스 종료하기
process.abort()	프로세스 비정상 종료하기
process.cpuUsage()	프로세스의 CPU 사용량 반환
process.memoryUsage()	프로세스의 메모리 사용량 반환
process.stdin :Stream	프로세스의 표준입력 스트림 객체
process.stdout :Stream	프로세스의 표준출력 스트림 객체
process.stderr :Stream	프로세스의 표준오류 스트림 객체
process.on('Event Type', eventHandler)	프로세스에 이벤트(exit, SIGINT, 등) 핸들러 부착하기

<Node.js process 객체의 주요 속성 및 메소드>

프로세스를 실행 중에 강제로 종료시키려면 Crtl+C 를 누르세요. 프로세스의 표준 입력 스트림에 Crtl+C가 입력되면 OS는 프로세스에 SIGINT라는 신호(Signal)를 보냅니다. 모든 프로세스는 기

> 본적으로 SIGINT 신호를 받으면 프로세스를 종료합니다.

TCP 프로토콜, Node.js의 net 모듈 (https://nodejs.org/api/net.html)을 이용해서 서버-클라이언트 구조의 CLI 채팅 서비스를 만들어 보겠습니다. 상시 작동하는 서버 프로그램, 최종 사용자들이 이용할 클라이언트 프로그램이 필요합니다.

TCP 채팅 서버

```
const net = require('net');

let sockets = [];
function broadcast(message) {
  sockets.forEach(socket => socket.write(message));
}

let server = net.createServer(socket => {
  sockets.push(socket);
  let ip = socket.address().address;
  broadcast(`${ip}님이 입장하셨습니다.`);

  socket.on('data', data => {
    let packet = data.toString().split('|');
    let name = packet[0] || '아무개';
    let message = packet[1] && packet[1].trim() || '';
    if (message != '') {
      broadcast(`${name}> ${message}`);
    }
  });

  socket.on('close', () => {
    let index = sockets.indexOf(socket);
    sockets.splice(index, 1);
    broadcast(`${ip}님이 퇴장하셨습니다.`);
  });

  socket.on('error', e => {
    console.log(e);
  });
});

const port = process.env.PORT || 5000;
server.listen(port, () => {
  console.log(`Server started at port ${port}`);
});
```

TCP 채팅 클라이언트

```javascript
const net = require('net');

// prompt name
console.log('이름을 입력하세요!');
process.stdin.once('data', data => {
  let name = data.toString().replace('\n', '');

  // connect to server
  let socket = net.connect(5000, '192.168.1.150', () => {
    socket.on('data', data => {
      console.log(data.toString());
    });

    process.stdin.on('data', data => {
      let message = data.toString().replace('\n', '');
      let packet = `${name}|${message}`;
      socket.write(packet);
    });
  });
});
```

[1] Stream

[2] File Pointer

[3] STDIN

[4] STDOUT

[5] STDERR

[6] Socket

4 웹 백엔드

4.3 HTTP 프로토콜

1. HTTP
2. HTTP Request
3. HTTP Response
4. HTTP는 꼭 웹 브라우저와?
5. MIME 타입

> HTTP 프로토콜을 이해하고, HTTP 요청과 응답의 구조를 배웁니다. 또 MIME 타입에 대해서 알아봅니다.

1. HTTP

<HTTP 프로토콜을 이용하는 웹 서비스>

HTTP(Hyper Text Transfer Protocol)는 이전 강의에서 살펴본 TCP/IP 프로토콜 위에서 정의된 웹 서비스를 위한 프로토콜입니다. 쉽게 생각해서 TCP/IP로 전송하는 데이터를, 웹 브라우저와 웹 서버가 HTML 문서를 요청하고 응답하기 좋은 형태로 구조화한 데이터 구조 등에 대한 약속입니다. 웹 브라우저가 웹 서버와 통신하는 과정을 통해서 HTTP의 포맷에 대해서 알아보고, 이어지는 챕터에서 브라우저에서 요청을 보내는 방식, 웹 서버에서 요청을 처리해 응답하는 방식에 대해서 알아보겠습니다.

웹 브라우저의 통신 과정

1. URL 분석 및 접속

   웹 브라우저는 URL을 분석해 서버의 IP 주소와 포트(기본은 80)를 이용해 서버와 TCP/IP 연결을 요청합니다.

2. Request 헤더 전송

   브라우저에서 요청 파일명 등이 기술된 헤더를 전송합니다.

3. Request 바디 전송

   필요한 경우에, 로그인 폼에 입력한 데이터나 첨부 파일 등의 추가적인 데이터를 전송합니다.

4. Response 헤더 수신

서버에서 헤더를 수신하고 응답 상태(404 등)를 확인하며, 바디의 Content-Type 등을 확인합니다.

5. Response 바디 수신

   바디가 있는 경우에, 서버에서 수신한 바디를 헤더에 기술된 Content-Type에 따라서, text/html인 경우에 HTML을 렌더링하고, image/jpeg인 경우에는 그림을 띄우는 등 적절히 해석합니다.

> HTTPS(Hypertext Transfer Protocol over Secure Socket Layer)는 웹 서버와 주고 받는 HTTP 데이터를 SSL/TLS 프로토콜을 통해 **암호화하여 보안성**을 높힌 프로토콜입니다. **443번 포트**를 사용합니다.

2. HTTP Request

<HTTP Request>

Request Header

종류	설명	예시
URL	요청을 보내는 URL	GET /some/path?page=1&anything=abc HTTP/1.1 Host: www.google.co.kr
Method	요청의 종류를 의미	GET/POST/PUT/DELETE...
Content-Type	요청에 바디가 있는 경우 그 포맷	Content-Type: application/x-www-form-urlencoded
Cookie	웹 브라우저에 저장된 쿠키들	Cookie: someID=1234512; anything=abcdefg;
User-Agent	클라이언트의 정보	User-Agent: Mozilla/ ... Chrome/54.0.2840.71 Safari/537.36

<Request Header의 주요 정보>

HTTP Request 헤더의 필드 목록
(https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Request_fields)

Request Method

종류	설명
GET	주어진 URL에서 자원을 요청
POST	주어진 URL로 자원의 생성을 요청
PUT	주어진 URL로 자원의 대체를 요청
DELETE	주어진 URL로 자원의 삭제를 요청
HEAD	주어진 URL에서 자원의 헤더만을 요청
OPTIONS	주어진 URL에서 처리 가능한 메소드의 목록을 요청

<HTTP Request 메소드>

이외에도 PATCH, CONNECT, TRACE 등의 메소드가 있습니다. HTTP 규약에 메소드라는 것이 존재하는 이유는 동일한 URL에 다양한 요청과 응답을 구성하기 위해서 입니다. 실제로 메소드의 정의대로 요청을 처리 할 지는 웹 서버를 개발하는 사람에게 달려있습니다.

Request Body

요청을 보낼 때 바디는 요청의 목적에 따라서 있을 수도 없을 수도 있습니다. HTTP는 단순한 약속이기 때문에 GET이나, HEAD 메소드의 요청에 바디를 포함에도 무관합니다. 하지만 메소드의 정의를 잘 따른다면 보통 POST나, PUT 메소드의 요청에 바디를 포함하기 마련입니다. 이때 웹 브라우저에서는 주로 Content-Type이 application/x-www-form-urlencoded 이거나 multipart/form-data인 바디를 보냅니다. 폼 데이터(로그인 폼, 게시판 글쓰기 폼 등)를 전송할 때는 urlencoded로, 동영상, 이미지, 파일 등 바이너리 파일을 전송할 때 multipart/form-data를 이용합니다.

3. HTTP Response

<HTTP Response>

Response Header

종류	설명	예시
Status	서버의 응답 상태 (1xx~5xx)	HTTP/1.1 200 OK
Set-Cookie	웹 브라우저에게 쿠키 생성을 요청	Set-Cookie: UserID=SonSon; Max-Age=3600; Version=1
Content-Type	응답에 바디가 있는 경우 그 포맷	Content-Type: text/html; charset=utf-8
Location	웹 브라우저에게 다른 URL로 요청을 보내길 요청 (Status 3xx와 같이 쓰임)	Location: http://www.google.co.kr

<Request Header의 주요 정보>

HTTP 응답 상태 코드들은 번호에 따라서 각각 1xx (조건부 응답), 2xx (성공), 3xx (리다이렉션), 4xx (요청 오류), 5xx (서버 오류)의 의미가 있습니다.

> HTTP Response 헤더의 필드 목록
> (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Response_fields)
> HTTP Response Status Code 목록
> (https://ko.wikipedia.org/wiki/HTTP_%EC%83%81%ED%83%9C_%EC%BD%94%EB%

Response body

바디는 응답의 목적에 따라서 있을 수도 없을 수도 있습니다. 응답 역시도 헤더에 Content-Type을 명시하고, 바디에 그 포맷에 맞는 데이터를 담습니다. HTML 문서는 html/text, JPG 이미지는 image/jpeg의 Content-Type을 가집니다. JSON 문서를 전송한다면 application/json이 Content-Type될 것 입니다.

4. HTTP는 꼭 웹 브라우저와?

본래 HTTP가 웹 브라우저를 위해 개발된 프로토콜이지만, HTTP 프로토콜과 웹 서비스의 생태계는 인터넷의 아주 큰 부분을 차지하고 있습니다. 그렇다 보니 HTTP 프로토콜을 기반으로 어떠한 용도의 서버를 개발하더라도, (꼭 웹 브라우저가 아닌) 클라이언트 플랫폼(Windows, Android, iOS 등)에서도 생산성 높은 개발을 할 수가 있습니다. 실제로도 모바일 게임 등의 경우에는 HTTP를 이용해 게임 서버를 구축하는 경우가 많습니다. HTTP는 웹 브라우저만을 위한 프로토콜이 아닙니다!

5. MIME 타입

HTTP 헤더의 Content-Type 필드는 MIME 타입에 해당하는 값을 가질 수 있습니다.

> MIME이란 Multipurpose Internet Mail Extensions의 약어로 **메시지 컨텐트 형식을 정의하기 위한 인터넷 표준**을 말합니다. MIME 메시지는 텍스트, 이미지, 오디오, 비디오, 및 기타 애플리케이션의 특정 데이터를 포함할 수 있으며 일반적으로 홈페이지 상에 표현되는 멀티미디어 데이타에 대한 형식을 말합니다...
> 공식 MIME 타입의 전체 목록 (http://www.iana.org/assignments/media-types/media-types.xhtml)

타입	설명

text/plain	텍스트 파일
text/css	CSS 텍스트
text/html	HTML 텍스트
image/jpeg	JPG 이미지
image/gif	GIF 이미지
image/png	PNG 이미지
image/x-icon	ICO 이미지
video/mpeg	MPG 비디오
audio/ogg	OGG 오디오
application/javascript	JavaScript 텍스트
application/json	JSON 텍스트
application/pdf	PDF 파일
application/octet-stream	알려지지 않은 파일, 기타 바이너리

<주요 MIME 타입>

[1] HTTP, Hyper Text Transfer Protocol

[2] HTTPS, Hypertext Transfer Protocol over Secure Socket Layer

[3] MIME, Multipurpose Internet Mail Extensions

4 웹 백엔드

4.4 웹 브라우저의 Request

1. 주소창을 통한 요청
2. 링크를 통한 요청
3. 폼을 통한 요청
4. 다양한 태그를 통한 요청
5. JavaScript를 통한 요청

> 웹 브라우저에서 서버에 HTTP 요청을 보내는 방법들에 대해서 알아봅니다.

1. 주소창을 통한 요청

<주소창을 통한 요청>

주소창에 URL을 입력하거나, 웹 페이지에서 새로고침을 누르면 웹 브라우저는 해당 URL로 GET Request를 보냅니다.

2. 링크를 통한 요청

<링크를 통한 요청>

웹 페이지에서 <a> 태그로 구현되는 링크를 클릭하면 href 속성에 명시한 URL로 이동하게 됩니다. 이때 href 속성에 명시된 URL로 브라우저가 GET Request를 보냅니다.

3. 폼을 통한 요청

val1

val2

Choose File DSC07697.jpg Submit

<폼을 통한 요청>

<form action="Server URL" method="HTTP method" enctype="Body Content-Type">

웹 페이지의 <form> 태그로 구현되는 양식에 데이터를 입력하고 전송하면, 웹 브라우저는 폼 태그의 method 속성에 명시된 HTTP 메소드로, action 속성에 명시된 URL로 Request를 보냅니다. 이 때 폼 태그 내부에 <input>, <select>, <textarea> 와 같은 태그에 있는 값들이 enctype 속성을 기반으로 인코딩되어 전송됩니다.

1. method='GET' (기본값)

```
<form action="http://benzen.io" method="get">
<input type="text" placeholder="key1" name="key1" value="val1">
<input type="text" placeholder="key2" name="key2" value="val2">
<input type="file" name="key3">
<input type="submit">
</form>
```

▼ Request Headers view parsed

```
GET /?key1=val1&key2=val2&key3=DSC07697.jpg HTTP/1.1
Host: benzen.io
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1
Accept: text/html,application/xhtml+xml,application/xml;q=
Referer: http://localhost:3000/learn/400/430
Accept-Encoding: gzip, deflate, sdch
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: _ga=GA1.2.959228138.1475234300
```

▼ Query String Parameters view parsed

```
key1=val1&key2=val2&key3=DSC07697.jpg
```

<form method='GET'>

폼 데이터를 GET 메소드로 보내면, 폼 input들의 name 속성들을 기반으로 ?name1=value2&name2=value2 꼴의 QueryString이 요청 URL의 뒷 부분에 붙게 됩니다. 또한 파일을 첨부한 경우에도 오직 파일명만 전송되는 것을 알 수 있습니다. 이때는 Request의 바디가 존재하지 않으며, 폼 태그의 enctype 속성은 무시됩니다.

2. method='POST', enctype='application/x-www-form-urlencoded' (기본값)

```
<form action="http://benzen.io" method="post" enctype="application/x-www-form-urlenc
<input type="text" placeholder="key1" name="key1" value="val1">
<input type="text" placeholder="key2" name="key2" value="val2">
<input type="file" name="key3">
<input type="submit">
</form>
```

```
Request Headers        view parsed
POST / HTTP/1.1
Host: benzen.io
Connection: keep-alive
Content-Length: 37
Pragma: no-cache
Cache-Control: no-cache
Origin: http://localhost:3000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/x
Referer: http://localhost:3000/learn/400/430
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: _ga=GA1.2.959228138.1475234300

Form Data        view parsed
key1=val1&key2=val2&key3=DSC07697.jpg
```

<form method='POST' enctype='application/x-www-form-urlencoded'>

폼 데이터를 POST 메소드로 보내면, 폼 데이터는 바디에 첨부됩니다. 이 때 폼 태그에 enctype을 명시하지 않을 경우 폼 데이터의 인코딩 방식(Content-Type)은 기본적으로 application/x-www-form-urlencoded를 따릅니다. urlencoded 방식은 querystring의 형태와 비슷합니다. 이 때도 파일을 첨부한 경우에 오직 파일명만 전송되는 것을 알 수 있습니다.

3. method='POST', enctype='multipart/form-data'

```
<form action="http://benzen.io" method="post" enctype="multipart/form-data">
<input type="text" placeholder="key1" name="key1" value="val1">
<input type="text" placeholder="key2" name="key2" value="val2">
<input type="file" name="key3">
<input type="submit">
</form>
```

<form method='POST' enctype='multipart/form-data'>

폼 데이터를 POST 메소드로 보내면서, 폼 태그에 enctype을 multipart/form-data로 명시 할 경우 첨부된 바이너리 파일의 내용이 모두 전송됩니다.

4. method='POST', enctype='text/plain'

```
<form action="http://benzen.io" method="post" enctype="text/plain">
<input type="text" placeholder="key1" name="key1" value="val1">
<input type="text" placeholder="key2" name="key2" value="val2">
<input type="file" name="key3">
<input type="submit">
</form>
```

```
▼ Request Headers        view parsed
    POST / HTTP/1.1
    Host: benzen.io
    Connection: keep-alive
    Content-Length: 39
    Pragma: no-cache
    Cache-Control: no-cache
    Origin: http://localhost:3000
    Upgrade-Insecure-Requests: 1
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
    Content-Type: text/plain
    Accept: text/html,application/xhtml+xml,applicatio
    Referer: http://localhost:3000/learn/400/430
    Accept-Encoding: gzip, deflate
    Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0
    Cookie: _ga=GA1.2.959228138.1475234300
▼ Request Payload
    key1=val1
    key2=val2
    key3=DSC07697.jpg
```

<form method='POST' enctype='text/plain'>

폼 데이터를 POST 메소드로 보내면서, 폼 태그에 enctype을 text/plain으로 명시 할 수 있습니다. 이때도 파일이 첨부된 경우 파일명만 전송되며, 이 인코딩 타입은 디버깅 용도로 쓰입니다.

폼 전송 방식의 선택 기준

방식	기준	예시
method="GET"	폼 데이터에 파일이 첨부되지 않았으며, 폼 데이터가 URL에 쿼리스트링으로 노출되어도 상관 없거나, 노출되는 편이 좋을 때	게시판 검색 폼 /board/bbs.php?keyword=computer&page=1
method="POST"	폼 데이터에 파일이 첨부되지 않았으며, 폼 데이터가 URL에 노출되지 않았으면 할 때	쇼핑 몰 주문 폼, 로그인 폼 등
method="POST" enctype="multipart/form-data"	폼 데이터에 파일이 첨부될 때	프로필 사진 업로드

<폼 전송 방식의 선택 기준>

참고로 POST 메소드로 폼 데이터를 보내는 경우에도 폼 데이터가 정말로 노출되지 않는 것은 아닙니다. 예를 들어 로그인 폼에서 POST로 보낸다고 해도 사용자의 아이디나 패스워드는 패킷 감청시 노출될 수 있습니다. 이런 일을 방지하려면 HTTPS 프로토콜을 이용하거나, 나름의 암호화 기법을 사용해야 합니다.

4. 다양한 태그를 통한 요청

<한 웹 페이지를 렌더링하는데 총 62개의 Request를 보냄>

```
<link href="./somepath/something.css" rel="stylesheet" type="text/css">
<link href="../somepath/something.png" rel="icon" type="image/png">
<script src="https://anotherhost/somepath/something.js"></script>
<iframe src="./somepath/something.html"></iframe>
<object data="/somepath/something.pdf"></object>
<img src="http://anotherhost/somepath/something.jpg">
<video src="../somesomepath/somepath/something.mp4">
...
```

브라우저가 최초의 HTML 문서를 전송받고 이를 해석하는 도중에 위와 같이 다른 URL의 자원을 참조하는 태그를 만날 경우에 새로운 HTTP Request를 보내게 됩니다.

5. JavaScript를 통한 요청

<AJAX>

AJAX(Asynchronous JavaScript and XML)는 페이지 이동 없이, JavaScript를 이용해 동적으로 HTTP Request를 보내는 기술입니다. 링크나 폼을 통해 Request를 보낼 경우에는 Response를 받으며 페이지를 전환하게 되지만 JavaScript를 이용하면 페이지 전환 없이 데이터를 요청하고 응답 받을 수 있습니다. 상세한 내용은 추후에 다룹니다.

[1] QueryString

4 웹 백엔드

4.5 정적 웹 서버의 Response

1. 웹 서버
2. 라우팅

> Node.js의 HTTP 모듈을 사용해서 웹 서버를 구현합니다. 라우팅이라는 개념과 웹 서버의 HTTP 응답에 대해서 알아봅니다.

1. 웹 서버

<웹 서버의 구조>

클라이언트(웹 브라우저, 스마트폰 앱 등)가 웹 서버와 TCP/IP 연결을 맺고 HTTP 프로토콜을 따라 Request를 보내면, 웹 서버는 적절한 Reponse를 보내주고 연결을 끊습니다. Node.js의 http 모듈을 이용해서 기본적인 웹 서버 프로그램을 작성해 보겠습니다. http 모듈에 어떤 클래스와 메소드, 속성이 있는지 속속들이 알 필요는 없습니다. HTTP의 이론적인 배경을 생각해보면서 그 내용이 어떻게 추상화 되었을지를 생각하면 처음 접하는 모듈에 금방 익숙해질 수 있습니다. Node.js의 http 모듈은 결국 HTTP 프로토콜와 관련된 기능을 구현하기 위해 만들어진 것이기 때문입니다. 추후 다른 플랫폼에서 다른 라이브러리를 사용하더라도 이러한 배경을 생각하면 새로운 API 쉽게 익숙해질 수 있습니다.

> Node.js http 모듈 API (https://nodejs.org/api/http.html)
> Node.js https 모듈 API (https://nodejs.org/api/https.html)

server-hello.js

```
const http = require('http'); // HTTP 관련 내장 모듈

// 새 요청이 생성될 경우의 핸들러를 넘기며 서버 객체 생성
http.createServer((req,res) => {
    console.log(123);

  res.end('Benzen Workshop');

}).listen(5555); // 서버 소켓을 5555번 포트에 바인딩

/**
PC에 바인딩된 공인/사설 아이피나 아래와 같은 특수 아이피로 접속 가능
http://localhost:5555
http://0.0.0.0:5555
http://127.0.0.1:5555
**/

console.log("server-hello HTTP server running in 5555");
```

서버 프로그램을 강제로 종료시키려면 셸에서 Crtl+C 를 누르세요.

2. 라우팅

<라우팅>

웹 서버에서 Request의 처리는 라우팅에서 시작됩니다. 라우팅은 Request Method와 URL을 기반으로 코드를 분기하는 것을 의미합니다. 예를 들어 웹 브라우저에서 메인 페이지를 요청하면, 서버에서는 메인 페이지를 응답하는 코드가 실행되어야 하고, 게시판에 글쓰기를 요청하면, 게시판 글을 저장하고 또

저장 완료를 응답하는 코드가 실행되어야합니다.

위의 server-hello.js 코드에 라우팅이 없는 것처럼, 서버를 어떻게 설계 할 지는 개발자 마음에 달렸습니다. 하지만 일반적으로 Request URL에서 host와 querystring을 제외한 URL path와 Method를 기반으로 라우팅합니다.

server-static.js

```
const http = require('http');
const fs = require('fs');
const path = require('path');

// 서버 실행 전에 정적인 파일들을 메모리에 읽어 둠
var htmls = {
  index: fs.readFileSync('./index.html'),
  photo: fs.readFileSync('./photo.html')
};

http.createServer((req, res) => {

    var route = req.method + " " + req.url;

    // / 라우팅
    if(route == "GET /"){
      res.statusCode = 200;
      res.setHeader("Content-type", "text/html; charset=utf-8");
      res.write(htmls.index);
      res.end();

    // /photo 라우팅
    } else if(route == "GET /photo"){
      res.statusCode = 200;
      res.setHeader("Content-type", "text/html; charset=utf-8");
      res.write(htmls.photo);
      res.end();

    // URL로 라우팅, 404 처리
    } else {
      // __dirname 폴더에서 파일을 찾아봄
      var fileName = path.join(__dirname, req.url);

      fs.readFile(fileName, (err, data) => {
        if(err){
          res.statusCode = 404;
          res.end("Not Found");

        } else {
          res.write(data);
          res.end();
        }
      });
    }

}).listen(5555);

console.log("server-static HTTP server running in 5555");
```

위의 웹 서버들은 정해진 정적인 데이터만을 응답해주고 있습니다. 또한 뷰(html)의 재사용성이 부족하기에 유지보수 및 확장에 불리합니다. 실상 위에서 사용한 http 모듈은 저수준 모듈이기 때문에 생산성 있게 웹 서버를 작성하는 데 적합하지는 않습니다. 다음 챕터에서는 http 모듈로 동적인 데이터를 응답하

는 웹 서버를 만들어보면서, 고수준으로 추상화된 웹 서버 라이브러리에 대한 필요성을 느껴보도록 하겠습니다.

4 웹 백엔드

4.6 동적 웹 서버

1. 동적 웹 서버
2. Flickr API 모듈
3. Flickr 갤러리
4. 템플릿

> Flickr에 올라온 사진들을 보여주는 동적인 웹 서버를 만듭니다. 동적인 데이터와 뷰를 분리하기 위해 템플릿이라는 개념에 대해 알아봅니다.

1. 동적 웹 서버

<정적/동적 웹 서버의 구조>

이전 챕터에서 Request에 따라서 정적인 페이지를 Response로 주는 웹 서버를 만들었습니다. 이제 구글, 페이스북, 쇼핑몰 등의 웹 서비스처럼 서버의 데이터에 따라 Response가 동적으로 변하는 웹 서버를 만들어 보겠습니다.

2. Flickr API 모듈

<날씨 API>

온라인 갤러리를 만듭니다. 이때 사진 데이터는 플리커(Flickr)라는 웹 서비스에서 제공하는 API를 이용해서 실시간으로 가져옵니다. 여기서 API란 자사의 서비스를 타 서비스의 개발에 이용 할 수 있도록 제공하는 기능을 말합니다. Google API, Facebook API, Twitter API, Naver API 등 큰 규모의 서비스 회사들은 생태계를 확장하기 위해서 자사의 API를 부분적으로 공개해서 제공하는 경우가 많습니다. 또 정부기관 및 비영리 단체 등에서 공개 API를 제공하기도 합니다.

<Google API의 API Key>

HTTP 프르토콜을 사용하는 API 서버는 당연히 일반적인 웹 서버와 동일한 구조를 지닙니다. 주요한 차이점은 HTML 문서를 주고 받기보다는 클라이언트와 가벼운 포맷으로, 즉 JSON, XML 등의 Content-Type으로 응답하는 경우가 많습니다. 또한 로그인 등의 인증 절차를 대신하면서 API 이용 권한을 관리하고, 이용 현황 등을 모니터링하기 위해서, API 키를 생성해주고 API 이용시 Request에 API 키를 첨부하도록하는 경우가 많습니다.

flickr-api 모듈의 .getRecentPhotos(callback) 함수 스펙

1. flickr API의 flickr.photos.getRecent API

(https://www.flickr.com/services/api/explore/flickr.photos.getRecent)를 이용합니다.

2. API의 주소로 https 프로토콜로 GET Request (https://nodejs.org/api/https.html#https_https_get_options_callback)를 보내서 사진 정보를 받습니다.
3. 사진 정보들을 가공해서 {title:'각 사진의 제목', small:'작은 사진 URL', medium:'중간 사진 URL'} 꼴의 객체의 배열로 만듭니다.
4. 생성된 배열을 callback의 인자로 넘기면서 callback을 실행합니다.

flickr-api.js

```
const https = require('https');
const API_URL = "https://api.flickr.com/services/rest/?method=flickr.photos.getRecer

module.exports = {
    getRecentPhotos: getRecentPhotos
};

function getRecentPhotos(callback){
    https.get(API_URL, res => {

        let data = "";
        res.on('data', d => {
          data += d;
        });

      res.on('end', () => {
            let rawPhotos = JSON.parse(data).photos.photo;
            var photos = [];

        rawPhotos.forEach(photo =>
            photos.push({
                title: photo.title,
                small : `https://farm${photo.farm}.staticflickr.com/${photo.server}/
                medium: `https://farm${photo.farm}.staticflickr.com/${photo.server}/
            })
        );

        callback(photos);
      });
    });
}
```

3. Flickr 갤러리

<Flickr 갤러리>

만들어둔 flickr-api 모듈을 이용해 실시간으로 Flickr 사진들을 보여주는 웹 서버를 만듭니다.
데모: http://photos-flickr.herokuapp.com (http://photos-flickr.herokuapp.com)

Flickr 갤러리 스펙

1. 일단은 라우팅 없이 모든 Request에 대해서 같은 Response를 줍니다.
2. Flickr에서 받아온 사진 100개를 포함한 HTML로 문서를 Reponse로 줍니다.
3. Bootstrap Flat-UI (http://designmodo.github.io/Flat-UI/)로 문서를 꾸미고, Bootstrap 그리드 시스템 (http://bootstrapk.com/css/#grid)을 이용해 한 행에 4개의 사진이 있도록 합니다.
4. JavaScript를 이용해서 사진을 클릭하면 사진이 확대되고, 큰 사진을 다시 클릭하면 사라지도록 합니다.

flickr-app.js

```
const flickr = require('./flickr-api');
const http = require('http');
const fs = require('fs');

const htmlHeader = fs.readFileSync('./html/header.html'),
      htmlFooter = fs.readFileSync('./html/footer.html');

http.createServer((req, res) => {

  flickr.getRecentPhotos(photos => {

    let html = htmlHeader + `<div>`;

    for (let i in photos) {
      let photo = photos[i];

      if (i % 4 == 0) html += `</div><div class='row'>`;

      html += `
        <div class='col-xs-3' style="margin-top:25px">
          <img class="img-rounded img-responsive" src="${photo.small}" medium-src="$
          <img src="${photo.medium}" style="display:none" />
          <p style="text-overflow:ellipsis;overflow:hidden;white-space:nowrap;"><smc
        </div>`;
    }

    html += `</div>` + htmlFooter;

    res.writeHeader(200, {'Content-Type': 'text/html'});
    res.write(html);
    res.end();

  });

}).listen(8888);
```

JavaScript for(var i in object|array) { ... } (https://msdn.microsoft.com/ko-kr/library/55wb2d34(v=vs.94).aspx)

ES6 In Depth: let and const (http://hacks.mozilla.or.kr/2016/03/es6-in-depth-let-and-const/)

ES6 Template literals (https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals

JavaScript에서 문자열(String)을 표현하는 방법은 "string", 'string', `string` 세가지 방식이 있습니다. ES6에서 도입된 백틱(`)을 이용한 표기법을 템플릿 스트링이라고 합니다. 템플릿 스트링(Template String)을 이용하면 문자열을 여러줄로 쓰거나, 문자열 사이에 변수나 표현식을 간편하게 삽입 할 수 있습니다. `Hello my name is ${expression}...` 이라고 쓰면 expression의 표현식이 계산되어 문자열 안에 삽입이 됩니다.

4. 템플릿

<템플릿>

위 웹 서버는 데이터에 따라서 HTML 조각을 만들어 하나의 HTML 문서로 조합 한 후 Response를 주고 있습니다. 즉, 작동은 하지만 뷰와 데이터가 뒤섞여 있습니다. 이런 방식은 데이터의 구조나 웹 페이지의 디자인이 바뀔 때마다 많은 수정 사항을 동반합니다. 뷰와 데이터를 분리해 확장성과 유지보수의 편리함을 꾀하기 위해서 템플릿(Template)이라는 개념을 도입합니다.

1. HTML 파일(template.html)을 읽습니다. 이때 데이터가 들어갈 부분에 특수한 표시자({photos})를 만들어둡니다.
2. 표시자가 있는 모든 부분을 데이터와 관련된 태그들로 치환합니다.

이때 {photos}란 표식은 임의로 정한 것이고 어떤 표식이라도 html 태그와 구분만 된다면 좋습니다.

flickr-template.js

```
const flickr = require('./flickr-api');
const http = require('http');
const fs = require('fs');

const htmlTemplate = fs.readFileSync('./html/template.html').toString();

http.createServer((req, res) => {

  flickr.getRecentPhotos(photos => {

    let photosHtml = renderPhotosHTML(photos);
    let response = htmlTemplate.replace("{photos}", photosHtml);

    res.writeHeader(200, {'Content-Type': 'text/html'});
    res.write(response);
    res.end();
  });

}).listen(8888);

function renderPhotosHTML(photos){
    var html = `<div>`;
    for (let i in photos) {
      let photo = photos[i];

      if (i % 4 == 0) html += `</div><div class='row'>`;

      html += `
        <div class='col-xs-3' style="margin-top:25px">
          <img class="img-rounded img-responsive" src="${photo.small}" medium-src="$
          <img src="${photo.medium}" style="display:none" />
          <p style="text-overflow:ellipsis;overflow:hidden;white-space:nowrap;"><smc
        </div>`;
    }

    html += `</div>`;
    return html;
}
```

뷰와 데이터가 분리되면 웹 디자이너는 HTML 파일만으로 작업을 하면서 데이터와 관련된 부분에 특수한 표시자를 작성하면되고, 개발자는 데이터와 관련된 로직에만 신경을 쓰면되기 때문에 분업에도 유리합니다. 하지만 실상 위 코드도 renderPhotosHTML라는 함수로 템플릿이란 개념을 흉내만 냈을 뿐, 데이터와 뷰가 전혀 분리되지 않았습니다. 실제 템플릿 엔진들은 for문과 같은 제어 로직을 지원하여 데이터와 뷰를 완전히 분리 할 수 있도록 돕습니다. 추후에 상용 템플릿 엔진에 대해서 알아보도록 하겠습니다.

[1] Template String

[2] Template

4 웹 백엔드

4.7 Express.js

1. Express.js
2. EJS 템플릿
3. Flickr 리팩토링

> Node.js의 웹 서버 프레임워크인 Express에 대해서 공부하고, Express를 기반으로 Flickr 갤러리를 리팩토링합니다.

1. Express.js

Express는 Node.js 기반의 가벼운 웹 서버 프레임워크입니다. Node.js의 http 내장 모듈을 기반으로 웹 서버 개발에 적합하도록 추상화된 Application (서버), Router, Request, Response 객체들을 제공합니다. 라우팅, 미들웨어, 템플릿 등의 기능을 제공하며 다른 패키지들을 결합해 확장성있게 이용 할 수 있습니다. 앞으로 4,5장 파트의 실습에서 Express를 이용합니다. 챕터를 진행하시기 전에 Express 안내서 및 API 문서 (http://expressjs.com/ko/)를 어느 정도 공부하시길 바랍니다.

express 및 ejs 템플릿 엔진, nodemon 설치

```
npm install nodemon -g

mkdir flickr-express
cd flickr-express
npm init
npm install express ejs --save
```

Application 객체

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

app.locals.forTemplate = "Hello World!";
app.locals.forTemplate2 = [1,2,3,4,5];
app.set('views', 'templates');
app.set('view engine', 'ejs');
console.log(app.locals);

app.listen(8000);
```

라우팅과 미들웨어, Router 객체

```
// 라우팅 .post, .put, .delete, .all, ..
app.get('/', (req,res) => {
    res.send("Hello!");
});

app.get('/hello/:name', (req,res) => {
    res.send("Hello "+req.params.name);
});

// 미들웨어 (어플리케이션)
app.use((req, res, next) => {
        console.log('Time: %d', Date.now());
    res.locals.forTemplate3 = "Hello Again!";

    next();
});

app.use('/bye', (req, res, next) => {
    res.send('Bye');
});

// 미들웨어 (서드파티)
app.use(express.static('assets'));

// 미들웨어 (라우터 객체)
var router = express.Router();
router.use((req,res,next) => { ... });
router.get('/some/path', (req,res) => { ... });
app.use('/basePath', router);

// 미들웨어 (라우트 핸들러)
app.get('/profile/:user_id', (req,res,next) => {
    if (req.params.user_id == 0)
        next('Not Logined'); // 오류 처리 미들웨어로
    else
        next(); // 다음 핸들러로
}, (req,res) => {
    res.send("Hello "+req.params.user_id);
});

// 미들웨어 (오류 처리)
app.use((err,req,res,next) => {
    console.error(err);
    res.status(500).end();
});
```

Request, Response 객체

```
req.cookies
req.headers['Content-Type']
req.params
req.query
req.ip
req.method
req.get('Content-Type');

res.status(404);
res.redirect('/...');
res.set('Content-Type', '...');
res.set({
    'Content-Type': '...',
    '...': '...'
});

res.end();
res.send('....');
res.render('viewFileName', {
  title: 'aaa',
  list: ['aa','bb'],
  obj: {a:1,b:2}
});
res.json({a: 1,b: 2});
```

2. EJS 템플릿

EJS(Embedded JavaScript)는 템플릿 엔진의 일종입니다. .ejs 파일에 HTML과 JavaScript (Server-side)를 섞어 쓰고, EJS 엔진을 통해 HTML 문서를 생성합니다.

> EJS 공식 페이지 (https://github.com/tj/ejs)

Express에서 EJS 이용하기

```
app.set('view engine', 'ejs');
app.set('views', 'templates');

res.render('index', { // ./templates/index.ejs를 렌더링
            title: "Hello World!",
            users: ["user1","user2"]
        });
```

./templates/index.ejs

```
<html>
<body>
<%
// Server-side JavaScript (Node.js)
if ( .. ) {
    // window... 존재하지 않습니다.
    // document... 도 존재하지 않습니다.
}
console.log("Hi!"); // 웹 브라우저가 아니라 셸에 출력됩니다.
%>

<!-- title의 값을 문서에 삽입 -->
<h1><%=title%></h1>

<%
// 제어 로직을 통해 문서에 태그를 삽입
for (var i=0; i < 10; i++) {
%>
<span><%=i%></span>
<% } %>

<% users.forEach(user => { %>
<span><%=user%></span>
<% }); %>

<!-- 아래 script 태그의 javascript는 렌더링시에 단순한 텍스트에 불과합니다. -->
<script>
doc<%='u'%>ment.title = "this is client-side javascript";
</script>
</body>
</html>
```

반복적인 HTML을 재사용함으로서 유지보수의 효율을 높힐 수 있습니다. 일반적인 웹페이지의 HTML의 구조를 (1) html-head-body-네비게이션/사이드바-(2) 페이지마다 달라지는 내용-(3) 하단부-/body-/html로 본다면 (1)과 (3)은 공통적인 부분이므로 여러 페이지에서 재사용 할 수 있습니다.

./template/header.ejs

```
<html>
<body>
<h1>My Homepage</h1>
<ul class="nav navbar-nav">
    <li><a href="/home">Home</a></li>
    <li><a href="/about">About</a></li>
</ul>

<div class="contents">
```

./template/footer.ejs

```
</div>
<div>© 2016 benzen</div>
</body>
</html>
```

./template/home.ejs

```
<% include ./header %>

home contents..

<% include ./footer %>
```

./template/about.ejs

```
<% include ./header %>

about contents..

<% include ./footer %>
```

3. Flickr 리팩토링

Express를 이용하여 지난 챕터의 Flickr 갤러리를 리팩토링합니다.

스펙

1. 작은 사진을 보여주는 라우팅, 큰 사진을 보여주는 라우팅, 사진 데이터를 JSON으로 주는 라우팅 만들기
2. EJS를 이용해 데이터와 뷰의 완전한 분리
3. header와 footer를 쪼개 재사용하며, header에는 네비게이션을 포함하고, 현재 위치한 페이지를 인식할 수 있게 하이라이팅
4. 404 Not Found를 처리하는 미들웨어 만들기

5. 500 Error를 나타내는 미들웨어 만들기

app.js

```
const express = require('express');
const app = express();

// 환경 설정
app.set('view engine', 'ejs');
app.set('views', 'templates');

// app.locals와 res.locals
app.locals.title = "Flickr Photos";
app.use((req,res,next)=>{
    res.locals.url = req.url;
    next();
});

// 라우터 적용
app.use(require('./router'));

// 404 처리
app.use((req, res, next) => {
  res.status(404).send('Not Found');
});

// 에러 처리 미들웨어
app.use((err, req, res, next) => {
    console.error(err);
  res.status(500).send('Internal Server Error');
});

// 서버 시작
app.listen(8000);
console.log('Server started at port 8000', new Date);
```

router.js

```
const flickr = require('./flickr-api');
const express = require('express');
const router = express.Router();

// 리다이렉션
router.get('/',(req,res)=>{
    res.redirect('/small');
});

// 작은 사진 보여주기
router.get('/small', (req, res) => {
    flickr.getRecentPhotos(photos => {
      res.render('small', {
        photos: photos
      });
    });
});

// 큰 사진 보여주기
router.get('/large', (req, res) => {
    flickr.getRecentPhotos(photos => {
      res.render('large', {
        photos: photos
      });
    });
});

// JSON으로 응답
router.get('/json', (req, res) => {
    flickr.getRecentPhotos(photos => {
      res.json({
        title: req.app.locals.title, // req에 Application 객체를 참조 할 수 있는 app 속성이
        photos: photos
      });
    });
});

module.exports = router;
```

[1] EJS, Embedded JavaScript

4 웹 백엔드

4.8 쿠키와 세션, 인증

1. 쿠키
2. 세션과 인증
3. TodoList 서비스

> 웹 브라우저의 쿠키에 대해서 배우고, 쿠키를 기반으로 사용자를 인증하는 데 쓰이는 세션을 공부합니다. 로그인 기능이 있는 할일 리스트 서비스를 만듭니다.

1. 쿠키

<session-id라는 쿠키의 생성 과정>

쿠키(Cookie)는 웹 브라우저에 저장되어있는 키-값 형식의 데이터입니다. 크롬의 경우 개발자도구(F12->Application->Cookie)에서 현재 저장되어있는 쿠키들을 확인 할 수 있습니다. 쿠키는 웹 서버가 Response을 보낼때 Set-Cookie 헤더를 통해서 생성됩니다. Cookie 헤더를 받은 웹 브라우저는 저장장치에 쿠키 데이터를 생성하고 저장합니다.

이후 웹 브라우저가 Request를 보낼 때, 웹 서버의 도메인에 해당되는 모든 쿠키값이 Cookie 헤더를 통해 함께 전달됩니다. 이를 통해 서버는 Request를 보낸 클라이언트의 고유한 상태를 구분 할 수 있습니다. 쿠키는 팝업창을 몇 일간 띄우지 않기, 회원 로그인 등과 같은 기능을 구현 할 때 사용 할 수 있습니다.

> 쿠키(Cookie)란 하이퍼 텍스트의 기록서(HTTP)의 일종으로서 인터넷 사용자가 어떠한 웹사이트를 방문할 경우 그 사이트가 사용하고 있는 **서버에서 인터넷 사용자의 컴퓨터에 설치하는 작은 기록 정보 파일**을 일컫는다. '쿠키'라는 이름은 그림 동화 '헨젤과 그레텔'에서 가져온 것이다. 헨젤과 그레텔이 지나온 길을 표시하기 위해 쿠키 조각을 떨어뜨리며 표시했다는 이야기에서 따온 것이다. HTTP 쿠키, 웹 쿠키, 브라우저 쿠키라고도 한다. 이 기록 파일에 담긴 정보는 인터넷 사용자가 같은 웹사이트를 방문할 때마다 읽히고 수시로 새로운 정보로 바뀐다... 누군가의 **쿠키를 훔쳐서 해당 사용자의 웹 계정 접근권한을 획득 할 수도** 있다... (위키백과)

Express, Set-Cookie 헤더와 Cookie 헤더

```
// 쿠키 생성 요청
app.get('/set',(req,res)=>{
    res.setHeader('Set-Cookie','key1=val1');
    res.setHeader('Set-Cookie',['key2=val2', 'key3=val3']);
    res.cookie('key4','val4');

    /**
    Response에 아래와 같은 Set-Cookie 헤더들이 추가됩니다.
    Set-Cookie: key1=val1
    Set-Cookie: key2=val2
    Set-Cookie: key3=val4
    Set-Cookie: key4=val5
    **/

    res.send("웹 브라우저님 이 쿠키들 좀 만들어 주세요.");
});

// 클라이언트의 쿠키 이용하기
app.get('/get', (req,res)=>{
    res.send("네 웹 브라우저가 보낸 쿠키들: "+req.headers.cookie);
});

// cookie-parser 미들웨어
const cookieParser = require('cookie-parser');
app.use(cookieParser());

app.get('/get2', (req,res)=>{
    res.send("네 웹 브라우저가 보낸 쿠키 key1: "+req.cookies.key1);
});
```

HTTP Response의 Set-Cookie 헤더를 주는 방법 외에, 웹 브라우저에서 JavaScript로 쿠키를 제어할 수도 있습니다.

JavaScript로 쿠키 제어

```
<textarea rows="10" style="width:100%"></textarea>
<div>
    <button onclick="setCookie('x','123')">set-cookie x=123</button>
    <button onclick="deleteCookie('x')">delete-cookie x</button>
    <button onclick="setCookie('y','456')">set-cookie y=456</button>
    <button onclick="deleteCookie('y')">delete-cookie y</button>
</div>

<script>
/**
    document.cookie를 통해 도메인에 해당하는 모든 쿠키를 얻어 올 수 있습니다.
    이 때 이 값은 단순히 쿠키들이 key=value; 로 연결된 문자열입니다.
**/
function showCookie(){
    document.querySelector('textarea').textContent = document.cookie;
}
showCookie();

// document.cookie = "x=1"; 를 통해 쿠키를 하나씩 생성 할 수 있습니다.
function setCookie(key, val){
    document.cookie = key+'='+val;
    showCookie();
}
/**
    document.cookie = "x=1; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
    쿠키에는 만료일자와 쿠키를 보낼 base path를 설정 할 수도 있습니다.
    이는 HTTP Set-Cookie 헤더에서도 마찬가지입니다.
    쿠키를 삭제하려면 삭제하려는 쿠키를 expiresd의 지나간 일자로 설정해서 덮어씁니다.
**/
function deleteCookie(key){
    document.cookie = key+'=; expires=Thu, 18 Dec 2013 12:00:00 UTC';
    showCookie();
}
</script>
```

PDF에서는 미리보기가 지원되지 않습니다.

2. 세션과 인증

웹 서버는 수 많은 요청들이 들어올 때 어떤 요청이 어떤 사용자로부터 오는지 구분해야 할 필요가 있습니다. 하지만 HTTP 프로토콜의 연결에는 지속성이 없기 때문에 웹 서버에서 클라이언트를 구분하고, 클라이언트의 상태를 관리하기 위해서는 쿠키를 이용해야 합니다.

요청이 올 때마다 쿠키를 통해서 클라이언트를 구분하고 그 상태를 확인 할 수 있습니다. Cookie: name=dongwook을 가진 Request와 Cookie: name=seunghyun을 가진 Request에 따라 다른 Respone를 줄 수 있습니다. 이를 위해서 form 태그를 통해 유저의 아이디와 패스워드를 받고, 올바른 경우에 Set-Cookie: name=dongwook과 같이 쿠키를 생성해주고, 이후엔 name 쿠키를 확인해서 클라이언트를 구분 할 수 있겠습니다.

쿠키는 쉽게 조작이 가능하기 때문에 인증 방식을 위처럼 구현해서는 위험합니다.

<쿠키-세션을 이용한 인증 방식>

세션(Session)이란 개념이 도입됩니다. 웹 브라우저에게 클라이언트를 고유하게 나타내는 특수한 값을 쿠키로 갖게 합니다. 이 쿠키를 세션 쿠키라고 합니다. 이 세션 쿠키의 키를 임의로 session_id로 합니다. 클라이언트의 Request을 받으면 웹 서버는 session_id 쿠키가 없는 경우 Set-Cookie: session_id=aksdjabskd21424aslxnadsasdi192ru와 같이 고유하고, 예측이 힘든 값을 주며 세션 쿠키 생성을 요청 합니다. 동시에 메모리 등에 aksdjabskd21424aslxnadsasdi192ru와 일대일로 대응되는 데이터를 생성합니다. 이렇게 서버에 보관되는 데이터를 세션이라고 합니다.

이후 클라이언트가 Cookie: session_id=aksdjabskd21424aslxnadsasdi192ru를 통해 본인을 밝히며 Request를 보내는 경우, 웹 서버에서는 해당 세션 쿠키와 대응되는 세션을 통해 클라이언트의 고유한 상태(인증 정보 등)를 이용하거나 갱신 할 수 있습니다. 세션을 이용하면 민감한 데이터를 클라이언트에게 노출시키지 않고 클라이언트의 상태를 제어 할 수 있습니다.

Express, express-session 미들웨어

```
const express = require('express');
const session = require('express-session');
```

```

const app = express();
app.use(session({
  secret: 'benzen',
  resave: true,
  saveUninitialized: true
}));

app.use((req,res,next) => {
    /*
        express-session 미들웨어를 사용하면 req 객체에 .session 속성이 생깁니다.
        이 값은 요청을 보내는 클라이언트 별로 고유하며, 여기에 클라이언트의 고유한 데이터를 관리 할 수 있습
    */
    req.session.count = (typeof req.session.count == 'undefined') ? 1 : req.session.
    console.log(`${req.sessionID}: request ${req.session.count}`);
    next();
})

// 세션에 따라 다른 Response를 주기
app.get('/', (req,res)=>{
    if (req.session.admin) {
        res.send(`
<html>
<body>
<h1>Hello Admin!</h1>
<a href="/logout">Logout!</a>
</body>
</html>
`);
    } else {
        res.send(`
<html>
<body>
<form action="/login" method="get">
<input type="text" name="id" placeholder="User ID">
<input type="text" name="pw" placeholder="Password">
<input type="submit" value="Login!">
</form>
</body>
</html>
`);
    }
});

// 로그인
app.get('/login',(req,res)=>{
    if (req.query.id=='admin' && req.query.pw=='1234')
        req.session.admin = true;

    res.redirect('/');
});

// 로그아웃
app.get('/logout',(req,res)=>{
    delete req.session.admin;

    res.redirect('/');
```

```
    res.redirect('/');
});

app.listen(8000);
```

Express의 express-session 미들웨어 API (https://github.com/expressjs/session)

3. TodoList 서비스

<TodoList 서비스 데모>

로그인 기능이 있는 TodoList 서비스를 만들어보겠습니다.

TodoList 서비스 스펙

1. 서버의 메모리에 데이터를 유지하기
2. id, password를 통한 유저 로그인 기능
3. 유저당 하나의 리스트에 할 일을 추가하고 삭제 할 수 있음
4. 404 Not Found를 처리하는 미들웨어 만들기
5. 에러 미들웨어 만들기

이때 POST로 Request를 받을 때 바디에 실려오는 폼 데이터를 해석하기 위해서는 body-parser 미들웨어를 사용합니다.

Express, body-parser 미들웨어

```
const bodyParser = require('body-parser');

// application/x-www-form-urlencoded을 파싱해주는 미들웨어
app.post('/write', bodyParser.urlencoded(), (req,res)=>{
    /**
    req.body에 폼 데이터가 파싱되어 있습니다.
    req.body.inputName1
    req.body.inputName2
    **/

    // ...
});
```

Express의 body-parser 미들웨어 API (https://github.com/expressjs/body-parser)

todolist/data.js

```
module.exports=[
{
    id: 1,
    uid: 'admin',
    password: '1234',
    list: [
        '워크샵 하기',
        '책 읽기'
    ]
},
{
    id: 2,
    uid: 'kim',
    password: '2222',
    list: [
        '양파사기',
        '달걀 사오기'
    ]
},
{
    id: 3,
    uid: 'son',
    password: '3333',
    list: [
        '화분 물주기',
        '강아지 밥주기'
    ]
}];
```

todolist/index.js

```
const express = require('express');
const session = require('express-session');
const app = express();

app.set('views', 'views');
app.set('view engine', 'ejs');

// 세션 미들웨어
app.use(session({
  secret: 'todolist',
  resave: true,
  saveUninitialized: true
}));

// / => /list로 리다이렉션
app.get('/',(req,res)=>{
    res.redirect('/list');
});

// res.locals를 사용하여 뷰에서 세션 데이터를 바로 사용할 수 있게
app.use((req, res, next) => {
    res.locals.user = req.session.user || null;
    next();
});

// 인증 확인 미들웨어
app.use((req,res,next)=>{
    if(req.session.user || req.url=='/user/login'){
        next();
    } else {
        res.redirect('/user/login');
    }
});

// 라우터(컨트롤러)를 다른 파일로 분리하기
app.use('/user', require('./controllers/login-router'));
app.use('/list', require('./controllers/list-router'));
app.use(require('./controllers/not-found'));
app.use(require('./controllers/error'));

app.listen(4000);
```

[1] Cookie

[2] Session

4 웹 백엔드

4.9 동기와 비동기, Promise

1. Thread
2. 동기와 비동기
3. Promise
4. 비동기에 대한 오해

> 쓰레드(Thread)에 대해서 알아보고, 동기와 비동기의 개념을 익힙니다. 또한 Javascript의 콜백 지옥을 해결하는 데 쓰이는 Promise에 대해서 공부합니다.

1. Thread

대부분의 OS는 멀티태스킹(Multi-tasking), 또는 멀티프로세싱(Multi-processing)을 지원합니다. 즉 아주 짧은 시간 단위로 각 프로세스의 코드를 조금씩 조금씩 번갈아가며 실행(Context Switch)하면서 여러 프로세스가 동시에 실행되는 것처럼 흉내를 낼 수 있습니다.

<Multi-threading>

나아가서 OS는 한 프로세스가 다수의 쓰레드(Thread)를 가질 수 있게합니다. 쓰레드는 프로세스의 안의 작은 프로세스, 쉽게는 함수와 같은 코드 블럭으로 생각 할 수 있습니다. 기본적으로 프로세스는 하나의 쓰레드를 기반으로 작동하지만, OS 위에서 프로그램을 작성하면 OS API를 통해 쓰레드를 직접 제어할 수도 있습니다.

예를 들어 웹 브라우저가 싱글 쓰레드 기반이라면 HTML을 렌더링하다가 <img> 태그를 만나 그 이미지를 다운로드 받게 된다면 다운로드가 완료될 동안 HTML의 렌더링이 멈춰있을 것입니다. 이러한 이유로 사용자에 대한 응답성이 중요한 GUI 프로그램들은 대부분 다수의 쓰레드를 기반으로 작성됩니다. 멀티쓰레딩(Multi-threading)의 원리 역시 각 쓰레드의 코드를 조금씩 조금씩 번갈아가며 실행(Thread Switch)하는 것입니다.

2. 동기와 비동기

<동기와 비동기>

많은 플랫폼(Android, iOS, Windows, ..)의 GUI 프로그램들은 보통 하나의 Main Thread (UI Thread)를 가지며 UI 작업 외의 처리를 위한 다수의 Background Thread를 가집니다. 이러한 쓰레드 모델을 통해 UI가 항상 갱신되기에, 사용자는 프로그램이(UI가) 멈췄다라는 느낌을 받지 않을 수 있습니다.

Blocking Code

```
for(var i=0; i < 100000; i++) Math.random();
```

안타깝게도 웹 브라우저의 JavaScript는 Main Thread에서 작동하며 쓰레드를 직접 제어 할 수가 없습니다. 때문에 처리가 오래걸리는 JavaScript 코드를 실행하게 되면 웹 페이지의 UI가 한동안 반응하지 않을 수 있습니다. 위와 같은 코드는 UI를 버벅거리게 합니다. 이렇게 Main Thread에서 작동하는 코드를 동기(Synchronous) 또는 블록킹(Blocking) 코드라고 합니다.

Non-blocking Code

```
// native
var req = new XMLHttpRequest();
req.open('GET', 'URL', true);
req.onreadystatechange = function(){
  // ...
};
req.send();

// jQuery
$.get('URL', function(response){
  // ...
});
```

다행히 웹 브라우저도 Background Thread에서 작동하는 코드를 제공하긴 합니다. 세세한 사항들이 있지만, 대표적으로 XHR (Ajax Request)는 비동기(Asynchronous) 또는 논블록킹(Non-blocking)으로 작동하는 코드입니다. 이러한 이유로 Ajax를 이용 할 땐, Background Thread에서 Request와 Response를 주고 받은 후, Main Thread에서 실행 할 코드를 콜백이나 핸들러 형태로 제공하게 됩니다.

<Node.js Processing Model>

또한 우리가 지금 백엔드에서 사용하고 있는 Node.js 역시 그 구조상 콜백을 이용한 비동기 처리가 빈번합니다. 이는 Node.js의 구조가 Single-Thread-Event-Loop에 기반하기 때문입니다. 참고로 Node.js는 I/O 처리를 담당하는 메소드를 Sync/Async 방식 두 가지로 제공하여, 개발자가 필요에 따라서 적절히 선택 할 수 있도록 합니다.

Node.js I/O Methods

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

// 동기 방식의 코드
try {
  let data = fs.readFileSync('/some/file/path');
  console.log(data);

} catch (err) {
  console.error(err);
}

// 비동기 방식의 코드
fs.readFile('/some/file/path', (err, data)=>{
  if (err) {
    console.error(err);
  } else {
    console.log(data);
  }
});
```

3. Promise

```
app.get('/some resources', function (req, res) {
  db.query('SELECT A ...', function (err, a) {
    if (err) return res.end(err);

    db.query('SELECT B ... WHERE a=' + a, function (err, b) {
      if (err) return res.end(err);

      db.query('SELECT C ... WHERE b=' + b, function (err, c) {
        if (err) return res.end(err);

        db.query('SELECT D ... WHERE c=' + c, function (err, d) {
          if (err) return res.end(err);

          res.end(d);
        });
      });
    });
  });
});
```

<Callback Hell>

비동기 처리의 난관은 에러 처리를 위한 (1) try/catch가 작동하지 않으며, 비동기 로직이 조금만 복잡해지면 (2) 콜백 지옥(Callback Hell)이라고 불리는 가독성이 떨어지는 코드가 된다는 점입니다.

에러 처리를 위한 구조

```
// Background Thread에서 실행될 함수가 이런식으로 제공된다면
function asyncTask(successCallback, errorCallback){
  try {
    // ...작업 후
    successCallback(data);
  } catch(err) {
    // ...에러
    errorCallback(err);
  }
}

// 성공시, 오류시 각각의 콜백을 제공 할 수 있음
asyncTask(function(data){
  // .. 성공시
}, function(err){
  // .. 오류시
});
```

위 방법으로 에러 처리는 어느 정도 우아하게 해결 할 수 있으나, 콜백 지옥의 문제는 원칙적으로 해결하기가 힘듭니다. 때문에 JavaScript의 다양한 라이브러리에서 Promise라고 불리는 개념을 도입하였으며, ES6에서는 Promise 객체가 표준으로 자리잡았습니다. Promise 객체는 추후에 어떤 데이터를 가지고 해결될 것이라는 약속을 나타냅니다. 그리고 그 약속에 대해서 성공 할 경우의 콜백, 실패 할 경우의 콜백 등을 등록 할 수 있습니다.

Promise 이용하기

```
// Promise 객체를 리턴하는 asyncTask() 함수
asyncTask()
  .then(data => { // 성공 콜백
    console.log(data);
  })
  .catch(err => { // 에러 콜백
    console.error(err);
  });

// 여러 Promise 객체를 연쇄적으로 연결 할 수 있습니다.
asyncTask()
  .then(data => { return asyncTask2(data); }) // 역시 Promise 객체를 리턴
  .catch(err2 => { console.error(err2, "from ayncTask2"); }) // 중간 중간에 에러 제어 콜백
  .then(data2 => { return asyncTask3(data2); }) // 역시 Promise 객체를 리턴
  .then(data3 => { console.log(data3) })
  .catch(err => { console.error(err); });

// asyncTask, asyncTask2, asyncTas3가 모두 종료된 후에 실행 될 콜백을 연결 할 수 있습니다.
Promise
  .all([asyncTask(), asyncTask2(), asyncTask3()]);
  .then(results => { ... })
  .catch(err => { ... });
```

직접 Promise 객체를 리턴하는 코드를 작성 할 수도 있습니다.

Promise 정의하기

```
 var promise = new Promise(function(resolve, reject){
  // ... 작업

  // 성공시
  resolve(data);

  // 실패시
  reject(err);
});

promise.then(...)
```

Promise를 이용한 비동기 함수

```
// 비동기 작업을 흉내내며 Promise 객체를 반환하는 함수
function incrementAsync(num){
  return new Promise(function(resolve, reject){
    setTimeout(function(){
      resolve(num + 1);
    }, 1000);
  });
};

// Promise에 콜백 연결하기
incrementAsync(0)
  .then(incrementAsync)
  .then(incrementAsync)
  .then(incrementAsync)
  .then(incrementAsync)
  .then(function(result){ // 5초 후 실행
    document.write("<h1>First promise says " + result +"</h1>");
  });

// Promise.all로 콜백 병렬로 연결하기
Promise
  .all([
    incrementAsync(0),
    incrementAsync(1),
    incrementAsync(2),
    incrementAsync(3),
  ])
  .then(function(results){ // 1초 후 실행
    return Promise.all([
      incrementAsync(results[0]+results[1]),
      incrementAsync(results[1]+results[2])
    ]);
  })
  .then(function(results){ // 2초 후 실행
    document.write("<h1>Second promise says " + (results[0]+results[1]) +"</h1>");
  });

document.write("<h1>Promises are pending...</h1>");
```

PDF에서는 미리보기가 지원되지 않습니다.

JavaScript 뿐만 아니라, Node.js에서도 역시 Promise를 이용해 복잡한 비동기 처리를 우아하게 할 수 있습니다.

ES6 Promise API
(https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/P

4. 비동기에 대한 오해

콜백을 받는 함수나, Promise를 이용한 함수는 항상 비동기로 처리된다는 오해를 할 수 있습니다. 하지만 코드가 동기/비동기로 실행되는지, 혹은 코드가 메인 쓰레드를 Blocking하는 지의 여부는 콜백이나 Promise와 같은 형태에 달려있는 것이 아니라, 실제 그 코드가 어느 쓰레드에서 실행되는 지에 달려있습니다.

콜백을 받는 동기 코드

```
<body>
<h1 id="text"></h1>
<script>
// 콜백을 받지만 비동기로 실행되지 않습니다.
function notAsyncTask(callback){
  for(var i=0; i < 200000000; i++) Math.random(); // 이 코드가 메인 쓰레드에서 실행되기 때문일
  callback();
}

var h1 = document.getElementById('text');
h1.innerHTML += 'One<br>';
notAsyncTask(function(){ // 실행되는 순간 브라우저 UI가 잠시 먹통이 될 수 있습니다.
  h1.innerHTML += 'Two<br>';
});
h1.innerHTML += 'Three<br>';

// One-Three-Two 일까요?
</script>
</body>
```

PDF에서는 미리보기가 지원되지 않습니다.

콜백을 받는 비동기 코드

```
<body>
<h1 id="text"></h1>
<script>
function asyncTask(callback){
  var req = new XMLHttpRequest();
  req.open('GET', 'https://api.flickr.com/services/rest/?method=flickr.photos.getRec
  req.onreadystatechange = function(){
    if (this.readyState == 4) {
      callback(); // 전송이 완료되면 콜백 실행
    }
  }

  // req의 send() { [native code] } 함수는 백그라운드 쓰레드에서 네트워킹 작업을 합니다.
  // 즉, 이 코드는 메인 쓰레드를 블록킹 하지 않습니다.
  req.send();
}

var h1 = document.getElementById('text');
h1.innerHTML += 'One<br>';
asyncTask(function(){ // 이 콜백은 비동기 작업이 끝난 이후 다시 메인 쓰레드에서 호출됩니다.
  h1.innerHTML += 'Two<br>';
});
h1.innerHTML += 'Three<br>';
</script>
</body>
```

PDF에서는 미리보기가 지원되지 않습니다.

Promise를 이용한 동기 코드

```
<body>
<h1 id="text"></h1>
<script>
function notAsyncTask(){
  return new Promise(function(resolve, reject){
    for(var i=0; i < 200000000; i++) Math.random();
    resolve();
  });
}

var h1 = document.getElementById('text');
h1.innerHTML += 'One<br>';
notAsyncTask().then(function(){ // 실행되는 순간 브라우저 UI가 잠시 먹통이 될 수 있습니다.
  h1.innerHTML += 'Two<br>';
  // Promise를 이용하면 약간의 시간차를 두고 Promise 생성자에 전달된 함수가 메인 쓰레드에서 실행됩니다.
  // 물론 Promise 내부에서 비동기(Ajax) 작업을 한다면 그 작은 백그라운드 쓰레드에서 실행됩니다.
});
h1.innerHTML += 'Three<br>';
setTimeout(function(){
  h1.innerHTML += 'Four<br>';
}, 1);

// One-Two-Three-Four 일까요?
// 혹은 One-Three-Four-Two 일까요?
</script>
</body>
```

PDF에서는 미리보기가 지원되지 않습니다.

Promise를 이용한 비동기 코드

```
<body>
<h1 id="text"></h1>
<script>
function asyncTask(){
  return new Promise(function(resolve, reject){
    var req = new XMLHttpRequest();
    req.open('GET', 'https://api.flickr.com/services/rest/?method=flickr.photos.getF
    req.onreadystatechange = function(){
      if (this.readyState == 4) {
        resolve(); // 전송이 완료되면 Promise를 resolve
      }
    }
    req.send();
  });
}

var h1 = document.getElementById('text');
h1.innerHTML += 'One<br>';
asyncTask().then(function(){ // 이 콜백은 비동기 작업이 끝난 이후 다시 메인 쓰레드에서 호출됩니다.
  h1.innerHTML += 'Two<br>';
});
h1.innerHTML += 'Three<br>';
</script>
</body>
```

PDF에서는 미리보기가 지원되지 않습니다.

Thread를 제어 할 수 있는 API를 제공하는 Low Level의 플랫폼인 경우에는, 개발자 스스로가 Thread의 구획을 설계 할 수 있습니다. 하지만 JavaScript나 Node.js처럼 High Level의 플랫폼인 경우 native API가 Threading에 대한 계획을 내부적으로 처리하는 경우가 많습니다. 이런 경우엔 특정 native API를 사용할 때, 그 API가 어느 쓰레드에서 실행되는 지 확인이 필요하겠습니다.

JavaScript에서 Background Thread를 직접 제어 할 수 있는 Worker API (https://developer.mozilla.org/ko/docs/Web/API/Web_Workers_API/basic_usage)
Node.js에서 Multi-threading을 구현 할 수 있는 threads 모듈 (https://www.npmjs.com/package/threads)

[1] Multi-tasking

[2] Multi-processing

[3] Context Switch

[4] Thread

[5] Multi-threading

[6] Thread Switch

[7] Synchronous

[8] Blocking

[9] Asynchronous

[10] Non-blocking

[11] Promise

4 웹 백엔드

4.10 Ajax, WebSocket

1. Ajax
2. WebSocket
3. Socket.io

> JavaScript로 비동기 HTTP 요청을 보내는 Ajax에 대해서 공부합니다. 또한 웹 브라우저 상에서 게임, 채팅 등 양방향 통신에 사용되는 웹 소켓 기술과 Socket.io에 대해서 알아봅니다.

1. Ajax

<Ajax>

동기 Request	비동기 Request (Ajax)
통신이 완료 될 때까지 웹 브라우저가 대기	통신 중 웹 브라우저는 다른 작업을 수행
Reponse가 도착하면 페이지가 갱신	Response가 도착하면 콜백을 실행
페이지가 갱신되기에 일반적으로 Response는 HTML 문서	Response가 오직 데이터(JSON 등)라도 동적으로 DOM을 생성 할 수 있음

<동기, 비동기 Request의 비교>

Ajax(Asynchronous JavaScript And XML)는 웹 브라우저 상에서 JavaScript로 비동기 HTTP Request를 보내는 기술을 의미합니다. 이름과는 관계 없이 비동기 HTTP 통신으로 XML 뿐만 아니라, HTML, JavaScript, 바이너리 등 어떠한 Content-Type의 데이터도 주고 받을 수 있습니다.

Ajax를 이용한 Flickr 갤러리

```
<html>
<body>

<h1>getRecentPhotos by Ajax</h1>
<h5 id="loading">now Loading...</h5>
<div class="photos">
```

```
</div>

<script>
var loadingElement = document.getElementById('loading');
loadingElement.style.display = 'block';

getRecentPhotos(function(photos){
    loadingElement.style.display = 'none';

    var containerDiv = document.querySelector('.photos');
    photos.forEach(function(photo){
        var div = document.createElement('div');
        div.classList.add('photo');
        div.innerHTML = '<img src="'+photo.small+'"><p>'+photo.title+'</p>';

        containerDiv.appendChild(div);
    });

});

function getRecentPhotos(callback) {
    var API_URL = "https://api.flickr.com/services/rest/?method=flickr.photos.getRec

    // XMLHttpRequest 인스턴스를 생성
    var req = new XMLHttpRequest();
    req.open('GET', API_URL, true);
    req.send();

    /** POST의 경우 바디를 첨부하는 예
    req.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
    req.send('this_is=nothing&just=a_example_for_post');
    **/

    // Request의 상태 변화에 따른 핸들러
    req.onreadystatechange = function(){

        // readyState의 종류: 0 (uninitialized), 1 (loading), 2 (loaded), 3 (interact
        if (this.readyState == 4) {

            if (this.status == 200) { // Response status가 200 OK인 경우

                var responseBody = this.responseText;
                var rawPhotos = JSON.parse(responseBody).photos.photo;
                var photos = [];

                rawPhotos.forEach(function(photo){
                    photos.push({
                        title: photo.title,
                        small : 'https://farm'+photo.farm+'.staticflickr.com/'+photo
                        medium: 'https://farm'+photo.farm+'.staticflickr.com/'+photo
                    });
                });

                callback(photos);

            } else {
```

```
                throw new Error("Connection Error");
            }
        }
    };
}
</script>

</body>
</html>
```

```
.photos {
    margin: 30px;
}
.photos .photo {
    border: 1px solid #ccc;
    box-shadow: 5px 0 5px 0 rgba(0,0,0,0.1);
    border-radius: 10px;
    padding: 10px;
    margin: 10px;
    display: inline-block;
    width: 120px;
    height: 120px;
    overflow: hidden;
}
.photos .photo > img {
    width: 100%;
    max-height: 85px;
    display: block;
}
.photos .photo > p {
    color: gray;
  white-space: nowrap;
  text-overflow: ellipsis;
  overflow: hidden;
}
#loading {
    color:gray;
    display: none;
}
```

PDF에서는 미리보기가 지원되지 않습니다.

XMLHttpRequest나 fetch 함수 같은 브라우저의 내장 객체를 이용해도 좋지만, 좀 더 추상화된, 또 크로스 브라우징 문제를 해결해주는 jQuery와 같은 라이브러리의 Ajax 관련 API를 이용 할 수 있습니다.

XMLHttpRequest 객체 (https://developer.mozilla.org/ko/docs/XMLHttpRequest)
ES6의 fetch 함수 (http://hacks.mozilla.or.kr/2015/05/this-api-is-so-fetching/)
jQuery Ajax API (http://api.jquery.com/category/ajax/)
jQuery Ajax Shorthand API (http://api.jquery.com/category/ajax/shorthand-methods/)

jQuery Ajax Shorthand API

```
$.get("URL_TO_GET_REQUEST")
    .done(function() {
      // 성공적 응답
    })
    .fail(function() {
      // 오류 응답 또는 바디의 해석 오류
    })
    .always(function() {
      // 응답 후 항상
    });

$.post("URL_TO_POST_REQUEST")
    .done(function() {
      // 성공적 응답
    });
```

jQuery Ajax API를 이용한 Flickr 갤러리

```
<html>
<body>

<h1>getRecentPhotos by Ajax (with jQuery)</h1>
<h5 id="loading">now Loading...</h5>
<div class="photos">
</div>

<script src="https://code.jquery.com/jquery-3.1.1.js"></script>
<script>
var loadingElement = document.getElementById('loading');
loadingElement.style.display = 'block';

getRecentPhotos(function(photos){
    loadingElement.style.display = 'none';

    var containerDiv = document.querySelector('.photos');
    photos.forEach(function(photo){
        var div = document.createElement('div');
        div.classList.add('photo');
        div.innerHTML = '<img src="'+photo.small+'"><p>'+photo.title+'</p>';

        containerDiv.appendChild(div);
    });

});

function getRecentPhotos(callback) {
    var API_URL = "https://api.flickr.com/services/rest/?method=flickr.photos.getRec

    $.get(API_URL).done(function(data){
        var rawPhotos = data.photos.photo; // jQuery API가 알아서 JSON을 객체로 변환해 둠
        var photos = [];

        rawPhotos.forEach(function(photo){
            photos.push({
                title: photo.title,
                small : 'https://farm'+photo.farm+'.staticflickr.com/'+photo.server+
                medium: 'https://farm'+photo.farm+'.staticflickr.com/'+photo.server+
            });
        });

        callback(photos);
    });
}
</script>

</body>
</html>
```

```
.photos {
    margin: 30px;
}
.photos .photo {
    border: 1px solid #ccc;
    box-shadow: 5px 0 5px 0 rgba(0,0,0,0.1);
    border-radius: 10px;
    padding: 10px;
    margin: 10px;
    display: inline-block;
    width: 120px;
    height: 120px;
    overflow: hidden;
}
.photos .photo > img {
    width: 100%;
    max-height: 85px;
    display: block;
}
.photos .photo > p {
    color: gray;
  white-space: nowrap;
  text-overflow: ellipsis;
  overflow: hidden;
}
#loading {
    color:gray;
    display: none;
}
```

PDF에서는 미리보기가 지원되지 않습니다.

2. WebSocket

<Google Analytics Realtime>

HTTP 프로토콜은 기본적으로 TCP 연결을 지속적으로 열어두지 않습니다. 태생적으로 문서 전달을 위한 프로토콜이기에 Request와 Response가 교환되면 TCP 연결을 종료합니다. 또한 HTTP 프로토콜은 단방향 통신이기에 클라이언트의 Request 없이는 서버에서 일방적으로 데이터를 전송(Server Push) 할 수 없습니다. 즉 HTTP 프로토콜로는 실시간 상호작용이 힘들고, 반복적인 통신으로 실시간을 흉내내더라도 계속해서 첨부되는 헤더 때문에 효율성이 떨어집니다.
웹 서비스가 발달하면서 웹 브라우저에서 실시간 연결에 대한 수요가 생겼습니다. 채팅 서비스, 모니터링 서비스 등은 정말 데이터를 실시간으로 반영 할까요? 실시간이라는 기준 자체가 명확하지 않지만, 웹 소켓의 등장 이전에는 Server Push를 구현하기 위해서 Comet으로 불리는 다양한 편법들을 쓰거나, 서버에 몇 초, 몇 십초에 한번씩 Ajax 등을 이용해 페이지 전환 없이 Request를 보냄으로써 실시간을 흉내 낼 수 있었습니다.

<WebSocket 연결 과정>

그러던 2011년 웹 브라우저에서 실시간, 양방향 통신을 지원하기 위한 WebSocket 프로토콜(ws://)이 표준화되었으며, 대부분의 최신 웹 브라우저가 이를 지원하고 있습니다. (IE 10버전 이하는 지원하지 않음)
웹 소켓은 최초 연결시 HTTP 프로토콜을 이용합니다. 클라이언트가 Upgrade: WebSocket 등의 헤더를 웹 서버에 보내고, 웹 서버가 웹 소켓을 지원하는 경우 TCP 연결을 유지하면서 WebSocket 프로토콜로 전환합니다. 즉 웹 서버측에서도 WebSocket 프로토콜을 지원 할 준비가 되어있어야 합니다. 이후 WebSocket 프로토콜을 통해 TCP 연결처럼 양방향으로 바이너리, 또는 UTF-8로 인코딩된 텍스트를 전송 할 수 있습니다.
추가로 HTTPS 프로토콜에 대응되는 암호화된 Secure WebSocket 프로토콜(wss://)도 존재합니다.

> 웹 브라우저의 WebSocket 객체
> (https://developer.mozilla.org/ko/docs/WebSockets/Writing_WebSocket_client_app

3. Socket.io

<Socket.io>

WebSocket은 TCP for Web의 표준으로 채택되었지만 크로스 브라우징 문제에서 자유롭지는 않습니다. Socket.io 라이브러리는 TCP for Web 서버 작성을 위한 추상화된 Node.js 모듈과, 웹 브라우저 환경에 맞게 적절한 Comet 방식 또는 WebSocket을 이용해 연결을 맺어주는 JavaScript 모듈을 제공

하고 있습니다. 즉 디바이스나 웹 브라우저에 관계 없이 WebSocket 연결이나 실시간의 흉내를 위한 서버, 클라이언트 양측을 위한 모듈입니다.

<Realtime Graph with WebSocket>

실시간 통신을 이용하면 웹에서 게임, 채팅, 스트리밍 등 다양한 형태의 서비스를 제공 할 수 있습니다. 예시로 Socket.io를 이용해서 간단한 실시간 그래프를 구현해보도록 하겠습니다.

> Socket.io 홈페이지 및 API 문서 (http://socket.io/)

ws/index.js (Server)

```
const http = require('http');
const fs = require('fs');
const path = require('path');
const io = require('socket.io');

// 클라이언트에게 줄 index.html, socket.io.js
let clientHTML = fs.readFileSync(path.join(__dirname, 'index.html'));
let clientJS = fs.readFileSync(path.join(__dirname, 'node_modules/socket.io-client/s

// 일반적인 웹 서버 (HTTP)
let webServer = http.createServer((req,res) => {
    if (req.url == '/') {
        res.statusCode = 200;
        res.write(clientHTML);
        res.end();

    } else if (req.url == '/socket.io.js') {
        res.statusCode = 200;
        res.write(clientJS);
        res.end();

    } else {
        res.statusCode = 404;
        res.end();
    }

}).listen(7000);

// 웹소켓, 혹은 Comet 방식을 지원하는 확장된 webServer (WS / HTTP)
let wsServer = io(webServer);
wsServer.on('connect', socket => { // 새 소켓이 연결 될 때

    // hello_my_name_is 타입의 메세지를 받았을 때
    socket.on('hello_my_name_is', data => {
        console.log(data, 'is connected');

        // 0.05초에 한번씩 graph_data 타입의 메세지를 보내줌
        setInterval(() => {
            let data = socket.oldData || {x:0, y: Math.floor(Math.random()*300)};
            data.x += 1;
            data.y += Math.floor(Math.random()*30 - 15) // -15~15
            data.y = data.y < 0 ? 0 : (data.y > 300 ? 300 : data.y);

            socket.emit('graph_data', data);
            socket.oldData = data;
        }, 50);
    });
});
```

ws/index.html (Client)

```html
<html>
<body>
<style>
#map {
    border: 1px solid #eee;
    box-shadow: 5px 0 5px 0 rgba(0,0,0,0.1);
    width: 100%;
    height: 300px;
    position:relative;
    overflow-x: scroll;
}
#map div {
    position:absolute;
    display:inline-block;
    width: 4px;
    height: 4px;
    background-color: red;
    border-radius: 50%;
}
</style>

<h1>Realtime Graph <small>(<span id="x">0</span>)</small></h1>
<div id="map">
</div>

<script src="/socket.io.js"></script>
<script>
var name = "Client-"+Math.floor(Math.random()*1000);
var socket = io('http://localhost:7000');

socket.on('connect', function(){ // 서버와 연결 시
  socket.emit('hello_my_name_is', name); // hello_my_name_is 타입의 메세지 보냄
});

socket.on('ok_nice_to_meet_you', function(){ // ok_nice_to_meet_you 타입의 메세지 받음
    console.log('server said nice to meet you');
});

socket.on('graph_data', function(data){ // graph_data 타입의 메세지 받음
    var mapDiv = document.getElementById('map');
    var pointDiv = document.createElement('div');
    var xSpan = document.getElementById('x');
    xSpan.textContent = data.x;

    pointDiv.style.left = data.x + 'px';
    pointDiv.style.top = data.y + 'px';

    mapDiv.appendChild(pointDiv);
    mapDiv.scrollLeft = mapDiv.scrollWidth;
});
</script>
</body>
</html>
```

[1] Ajax, Asynchronous JavaScript And XML

[2] Comet

[3] WebSocket

[4] Socket.io

4 웹 백엔드

4.11 보안, Same Origin Policy

1. SOP
2. CSRF
3. JSONP
4. CORS

> 브라우저의 보안 정책인 SOP에 대해서 알아보고, CSRF라는 공격 방법에 대해 알아봅니다. 또한 다른 출처로 Ajax 요청을 보내기 위한 JSONP와 SOP을 제어할 수 있는 CORS에 대해서 알아봅니다.

1. SOP

<Same Origin Policy>

동일 출처 정책(Same Origin Policy)은 웹 브라우저의 핵심적인 보안 모델입니다.
http://workshop.benzen.io/api/exit 라는 URL로 Request를 보내면 워크샵 계정을 탈퇴시킨다고 해봅시다. 물론 이때 워크샵 웹 서버는 세션을 통해서 인증 절차를 거칩니다. 한 유저가 웹 브라우저에서 http://workshop.benzen.io에 로그인된 상태로, 다시 http://www.hacking.com로 접속했습니다.
이 때 hacking.com에서 응답해준 HTML 문서에서 아래처럼 Ajax를 통해 Request를 보낸다면, 유저는 생각도 못한 새에 계정을 탈퇴당하고 맙니다.

```
<script>$.get("http://worshop.benzen.io/api/exit");</script>
```

이와 같은 문제를 막고자 웹 브라우저들은 동일 출처 정책을 기본적으로 따르고 있습니다. 동일 출처 정책에 따라 A 출처에서 받은 문서는 A 출처에서 받은 다른 문서에 접근하고 제어 할 수 있으나, B 출처에서 받은 문서에 접근하거나 제어 할 수 없습니다.
좀 더 구체적으로 생각해보면

1. A 출처에서 받은 HTML에는 B 출처의 JavaScript나 CSS 파일이 적용되지 않는다?
2. A 출처에서 받은 HTML에서 B 출처의 이미지 파일을 불러 올 수 없다?

위처럼 된다면 상당히 불편하겠죠. 실제로는 자원의 종류에 따라서 SOP가 다르게 적용됩니다.

1. HTML 문서에서 다른 출처로 폼 제출(GET/POST Request), 링크 클릭(GET Request) 등은 허용됩니다.
2. HTML 문서에서 다른 출처의 CSS, JavaScript, Iframe, Image, Media 등을 포함(GET Request)하는 것은 허용됩니다.
3. HTML 문서에서 다른 출처로 XHR (Ajax Request)을 보내는 것은 거부됩니다.

이외에도 정책에 미묘한 차이점들이 있지만 가장 주요한 이슈는 다른 출처의 대부분의 자원에 대해서 GET Request는 허용되며, Ajax Request는 거부된다는 점입니다.

출처 (Origin)

출처는 프로토콜, 도메인(호스트), 포트를 모두 포함합니다. 즉 동일 출처(Same Origin)이려면 셋 모두가 동일해야합니다. 그렇지 않을 때는 교차 출처(Cross Origin)로 판단되어 SOP가 적용됩니다.

2. CSRF

<Cross-site Request Forgery>

위에서 예로 든 워크샵 계정 탈퇴 같은 공격을 사이트 간 요청 위조(Cross-site Request Forgery)라고 합니다. 다른 출처로의 Ajax 요청은 SOP에 의해 거부되더라도, 이미지 태그 등을 통해서 GET Request를 위조 할 수 있습니다.

```
<img src="http://workshop.benzen.io/api/exit">
```

이렇게 위조된 GET 요청들은 어떻게 막아야 할까요? HTTP 프로토콜 스펙에서는 GET 메소드를 통해서는 서버의 데이터를 변경하는 부가 작용(Side-Effect)에 대해 라우팅하지 않기를 권고하고 있습니다. 애초에 웹 서버에서 탈퇴를 처리하는 라우팅을 POST 메소드 등으로 설정하는 것이 바람직하겠습니다.

```
<h1>You Idiot!</h1>
<form name="hack" method="post" action="http://workshop.benzen.io/api/exit"></form>
<script>document.forms.hack.submit();</script>
```

그래도 아직 한가지 문제가 있습니다. 폼 제출의 경우는 다른 출처로의 POST 메소드 요청이 허용됩니다. 간단하게는 POST 메소드를 받는 라우팅에서 Referer 헤더를 확인하여 출처를 비교하면 타 출처에서 위조된 요청을 막을 수 있습니다. (Referer 헤더는 요청을 생성된 웹페이지의 URL을 명시합니다.) 다만 Referer 헤더 역시 조작이 가능하고, Referer 헤더가 없는 경우도 있을 수 있기 때문에 완벽하지는 않습니다.
좀 더 확실한 보안을 위해서는 Anti-CSRF Token이라는 보안 요소를 도입합니다. 토큰이라함은 무작위로 생성된 문자열을 의미합니다. Anti-CSRF를 구현하는 간단한 방식으로는 POST를 받을 때,

1. 서버가 폼을 가진 HTML을 응답해줄 때, 매번 새로운 토큰을 생성하고 세션에 기록해둡니다.
2. 이때 HTML의 폼에 토큰을 hidden 타입으로 추가 합니다.
3. 서버는 POST 요청을 받은 경우 폼 데이터에 첨부된 토큰을 마지막에 생성해준 토큰과 비교합니다.

<Anti-CSRF Token: Double Submit Cookie Method>

위에서 다룬 중요한 보안요소들을 정리하자면 다음과 같습니다.

1. GET 메소드의 라우팅에는 사이드 이펙트가 없도록 한다.
2. POST 메소드의 라우팅에 헤더의 레퍼러를 확인 할 수 있다.
3. POST 메소드의 라우팅에 Anti-CSRF Token을 사용 할 수 있다.

3. JSONP

<JSON with Padding>

Ajax를 통해 다른 출처의 API를 이용하고 싶어도 SOP 때문에 불가능한 경우가 있습니다. 이 때는 JSONP(JSON with Padding)라는 편법을 쓸 수 있습니다. 원리는 <script src="http://other.site/some/path"></script>는 SOP의 제한을 받지 않는데 있습니다.

JSONP의 원리

```
<script>
function otherSiteCallback(data){
    // ... 다른 사이트에서 데이터를 받으면 하고 싶은 처리를 명시
    console.log(data);
}
</script>
<script src="http://othersite.com/some/path?callback=otherSiteCallback"></script>
<!--
othersite.com에서는 /some/path로 GET Request를 받으면
동적으로

otherSiteCallback([
    // server data...,
    // server data...,
    // server data...,
])

의 내용을 가진 스크립트를 생성해서 Response로 줍니다.
물론 서버에서 otherSiteCallback 같은 콜백의 이름을 요청에 따라 생성하지 않고, jsonCallback 등으로 고
-->
```

위와 같은 소통 방식을 JSONP라고 합니다.

JSONP의 추상화

```
function callJSONP(url, callbackName, callback) {
    // 웹 브라우저에 미리 함수 등록
    window[callbackName] = callback;

    // script 태그 생성
    var script = document.createElement("script");
    script.src = url+callbackName;
    document.body.appendChild(script);
}

// 이후 아래처럼 사용 할 수 있습니다.
callJSONP('http://othersite.com/some/path?callback=', 'otherSiteCallback', function(
    // ... 다른 사이트에서 데이터를 받으면 하고 싶은 처리를 명시
    console.log(data);
});
```

4. CORS

<Cross Origin Resource Sharing>

SOP는 웹 브라우저에서 많은 보안 문제를 해결해주지만, 그와 함께 JSONP와 같은 희한한 소통 방식을 만들어 내기도 했습니다. 교차 출처 자원 공유, CORS(Cross Origin Resource Sharing)는 SOP를 원칙적으로 제어 할 수 있는 HTTP 프로토콜의 메커니즘입니다. 교차 출처 간의 XHR처럼 SOP를 위배하는 Request가 발생 할 경우 실제 웹 브라우저의 Request는 아래 그림과 같이 발생합니다.

<Preflighted Request, Main Request>

Request가 일어나기 전에 CORS의 여부를 물어보는 Preflighed Request가 선행되고, 웹 서버가 CORS를 지원하고 있다고 응답해주면, 이후 정상적인 Regeust, Response가 교환됩니다. 자사의 API를 공개하고자 하는 웹 서비스에서는 CORS를 지원하여 API 사용자들이 Ajax를 이용해 개발을 할 수 있도록 지원 할 수 있겠습니다.

> HTTP Access-Control 헤더 (CORS 헤더) (https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)
> CORS는 XHR만을 위한 메커니즘은 아닙니다. (https://remysharp.com/2013/01/14/cors-isnt-just-for-xhr)

모든 출처에 대해 CORS를 지원하는 서버

```
const app = require('express')();

app.use((req, res, next) => {
    if (req.headers.origin) {
        res.set({
            'Access-Control-Allow-Origin': req.headers.origin,
            'Access-Control-Allow-Methods': 'POST, GET, PUT, DELETE OPTIONS'
        });

        if (req.method == 'options')
            res.end();

    }

    next();
});

app.get('/', (req,res)=>{
    res.end('hey');
});

app.listen(8787);
```

[1] Same Origin Policy
[2] Cross Origin
[3] Cross-site Request Forgery
[4] Anti-CSRF Token
[5] JSONP, JSON with Padding
[6] CORS, Cross Origin Resource Sharing

4 웹 백엔드

4.12 REST API, OAuth, SPA

1. REST API
2. API Key를 이용한 인증
3. OAuth 프로토콜을 이용한 인증
4. JWT (JSON Web Token)
5. SPA (Single Page Application)

> 일반적인 웹 서버 외에 범용 API 서버를 구축 할 때 쓰이는 REST 아키텍쳐에 대해 공부합니다. 또한 API 서버의 사용자 인증에 쓰이는 OAuth 프로토콜에 대해 알아봅니다. 이후 간단한 REST API 서버와 SPA를 작성해봅니다.

1. REST API

REST(Representational State Transfer)는 클라이언트-서버 구조의 서비스에서, (일반적으로 HTTP 프로토콜을 기반으로하는) 서버를 구현 할 수 있는 아키텍쳐의 한 종류입니다. REST 아키텍쳐의 핵심적인 특징으로는 범용성, 리소스(데이터) 중심의 API 명세, 상태 없음(stateless)라고 볼 수 있겠습니다.

<REST API의 범용성>

먼저 범용성에 대해서 생각해보면, 지금까지 작성했던 웹 서비스의 백엔드 서버는 단순히 HTML 문서만을 응답해주기 때문에 웹 브라우저 클라이언트 전용의 서버라고 생각 할 수 있습니다. 하지만 REST API는 웹 브라우저를 포함해서 HTTP 통신이 가능한 모든 클라이언트 플랫폼을 타겟으로 합니다.
이런 범용성을 갖추기 위해서 REST API의 HTTP Response Body는 HTML 보다는 JSON, XML 등 여러 플랫폼에서 사용하기 적절한 단순한 텍스트 포맷을 사용합니다.

<REST API의 자기서술성>

또한 REST API는 HTTP Request Header 자체로 메시지의 목적이 뚜렷히 드러나도록 설계됩니다. 일반적인 웹 페이지의 Request는 GET /board/index.html 처럼 요청 자체로 그 메세지가 서버에 미치는 작용에 대해서 추측하기 힘듭니다.
RESTful한 설계라면 GET /articles와 같이 동사 + 명사의 결합으로 리소스에 대해 서술적인 방식으로 API의 각 기능(End-Point)을 설계하게 됩니다.
이 때 서버에서 유용하는 자원들에 대해서 일반적으로 CRUD(Create, Read, Update, Delete) 기능을 갖출 수 있도록 API를 설계합니다.
다중 플랫폼을 타겟으로, 데이터 제어 및 조회를 제공하는 서버에 REST 아키텍쳐를 도입하면 적절하겠습니다.

URL/Method	Create	Read	Update	Delete
	POST	GET	PUT	DELETE
/users	User를 생성	전체 User를 조회	전체 User를 일괄 수정	전체 User를 일괄 삭제
/users/10	-	10번 User를 조회	10번 User를 수정	10번 User를 삭제
/users/10/posts	10번 User의 Post를 생성	10번 User의 모든 Post를 조회	10번 User의 모든 Post를 일괄 수정	10번 User의 모든 Post를 일괄 삭제

<REST API 요청 예시>

위처럼 REST API의 각 엔드포인트는 자원의 명사형(보통 복수형으로)에 자원의 고유 번호(PK)를 결합하고, HTTP Method를 동사로 결합하여 API 엔드포인트 자체가 자기서술적인 성격을 띄고 있습니다.
users/10/posts 처럼 종속적인 자원에 대한 엔드포인트를 설계 할 수도 있겠습니다.

Response Status Code	의미 및 용례
200	일반적인 성공, Reponse Body에 조회, 생성 또는 수정한 데이터를 첨부
304	자원을 조회 할 시, 이전과 변동 사항 없음 (클라이언트가 캐싱을 제공하는 경우)

400	잘못된 요청, Response Body에 오류의 구체적인 정보를 첨부 할 수 있음
401	인증이 필요함
403	(인증이 되었으나) 권한 없음
404	자원이 존재하지 않음
500	서버 오류

<REST API 응답 예시>

REST API의 응답은 HTTP의 Response Status Code를 활용해서 HTTP Response 헤더 자체가 그 결과를 서술 할 수 있도록 합니다. 일반적인 웹 서버는 Status Code를 잘 활용하지 않지만, REST에서는 Status Code를 적절히 활용하여, 클라이언트 프로그램들이 응답에 적절히 반응 할 수 있도록 해줍니다. 물론 Response Body에 데이터가 첨부 될 때는, HTML 보다는 JSON이나 XML과 같은 단순한 텍스트 포맷을 이용합니다.

REST API 서버 예시 (Express.js)

```
app.get('/users', (req,res)=>{
    res.json(db.users);
});

app.get('/users/:id', (req,res)=>{
    let user = db.users.find(user => user.id == req.params.id);
    if (user) {
        res.json(user);
    } else {
        res.status(404).end();
    }
});

app.post('/users', ...);

app.delete('/users/:id', ...);

...
```

2. API Key를 이용한 인증

REST (State Transfer)의 마지막 특성으로는 Stateless를 들 수 있습니다. 상태가 없다함은 하나의 요청이 그 자체로 고립되고 완전하여, 로그인 등과 같은 이전의 요청에 영향을 받지 않으며, 항상 같은 결과를 낸다는 의미입니다.

즉, 서버에서 클라이언트의 상태를 관리하지 않아야합니다. State Transfer는 요청/응답 간에 상태 관리가 필요한 경우에, 메세지 자체에 그 상태를 포함하여 통신한다는 의미로 볼 수 있겠습니다.

<REST API의 Stateless>

전통적인 웹에서는 클라이언트의 상태를 관리하기 위해서 쿠키-세션을 기반으로 클라이언트의 상태를 추적, 관리하는 것이 일반적입니다. 하지만 범용성, 나아가 확장성(Scalability)을 기조로 설계된 REST 아키텍쳐는 서버의 Stateless 구조를 강조합니다.

왜냐하면 세션을 기반으로 클라이언트를 추적하는 것은 서버 쪽에 지속적인 비용을 초래하기 때문에 대규모 서비스로의 확장에 걸림돌이 될 수 있기 때문입니다. 또한 범용성을 위해서 (웹 브라우저를 위한) Cookie 기반의 세션을 이용하지 않도록 유도한다고 볼 수도 있겠습니다.

자, 그렇다면 상태에 대한 유지 및 관리는 클라이언트 측에 맡긴다고 쳐도, 서버 측에서는 어떻게 클라이언트의 신원을 확인 할 수가 있을까요?

<REST API KEY>

가장 기초적인 방법으로는, 서버측에 미리 추측하기 힘든 무작위 문자열(API KEY)을 생성해두고, 클라이언트가 API를 호출 할 때마다 HTTP 헤더의 특정 필드 또는 쿼리스트링을 통해 전달하는 방법이 있습니다. 이를 통해 서버측은 메세지에 포함된 KEY를 통해서 클라이언트의 신원과 권한을 확인 할 수 있습니다.

물론 이 방식에선 메세지가 감청되거나 API KEY가 노출되었을 때, 메시지 위조 등의 공격 받기 쉽습니다. 따라서 민감하거나 보안이 필요한 데이터를 다룰 때는 메세지 전문을 암호화(HTTPS 등으로) 할 필요가 있습니다.

API KEY 방식의 예시

```
function getRecentPhotos(callback) {
    var API_URL = "https://api.flickr.com/services/rest/?method=flickr.photos.getRec
    // cf. RESTful End-point로는 url 구조가 적합하지 않습니다.

    $.get(API_URL).done(function(data){
        var rawPhotos = data.photos.photo;
        // ... 중략
        callback(photos);
    });
}
```

3. OAuth 프로토콜을 이용한 인증

<OAuth>

위 API KEY 방식은 간단하고 합리적이지만, 서비스 제공자(Service Provider), 즉 API 제공자가 인터넷에서 서비스를 공개적으로 확장하기에는 충분하지 않습니다. 예를 들어 페이스북이 당사의 인지도 및 서비스 이용을 증대시키기 위해서 일부 API를 공개한다고 해도, 페이스북 친구 목록을 활용해 특정 기능을 제공하려는 서비스 소비자(Service Consumer)에게 아무런 제한 없이 API KEY만으로 고객의 개인 정보 등에 대한 API에 대한 접근 권한을 허가 할 수는 없습니다.

이렇게 고객, 서비스 제공자, 서비스 소비자, 3자가 얽힌 관계에서 API를 제공하기 위해서 OAuth (Open Authentication)이라는 프로토콜이 자리잡았습니다.

<Facebook OAuth>

1. 서비스 소비자(Consumer)는 서비스 제공자(Provider)가 제공하는 고객(User)의 특정 정보에 대한 API를 이용하기 위해서 Provider에게 User의 위임장(Request Token)을 생성해주길 요청합니다.
2. Provider가 Consumer에게 위임장을 생성해주면, Consumer는 다시 User에게 제공 받은 위임장을 확인해주길 요청 (Provider에 로그인하고, Consumer가 요구하는 권한을 인가하는 등) 합니다.
3. User가 위임장을 확인 및 수락하면, Provider는 Consumer에게 위임에 대한 확인서(Access Token)를 발급해 줍니다.
4. 이후 Consumer는 그 확인서가 유효한 동안, 위임 받은 권한을 이용해 Provider의 API를 호출합니다.

<OAuth 프로토콜>

또한 OAuth 프로토콜은 자사의 API를 공개하는 것 외에도, 같은 방식으로 자사의 인증 로직을 어플리케이션 서버에서 분리하여, 별도의 인증 서버를 구축하거나, 또 나아가서 SSO(Single Sign On)과 같은 통합 인증 서비스를 구축하는 데 이용되기도 합니다.

4. JWT (JSON Web Token)

마지막으로 위에서 다룬 API KEY 방식과 Oauth의 Access Token 의 구현방식에 대해서 생각해보겠습니다. 서버는 API 호출 요청에 대해서 API KEY나 Token(다를바 없이 암호화된 무작위 문자열)이 유효한지를 확인 할 필요가 있습니다. 이는 서버에서 클라이언트의 상태(토큰의 유효성)를 관리하게끔 하며, 또 API를 호출 할 때마다 그 토큰에 해당하는 유저의 상태를 열람(DB 등에서)하는 비용을 초래 합니다.

<JWT 흐름도>

이 상황의 근본적인 이유는 토큰 자체가 무의미한 문자열로 구성되어있기 때문입니다. 여기서 무의미한 토큰을 의미(유저의 상태를 포함한)가 있는 토큰으로 구성한다면, API 서버 쪽의 비용을 절감하면서 Stateless한 아키텍쳐를 구성 할 수 있습니다. JWT (JSON Web Token)는 유저의 상태(고유 번호, 권한, 토큰 만료 일자 등을 포함)를 JSON 포맷으로 구성하고, 이 텍스트를 다시 특정 알고리즘(Base 64 (https://ko.wikipedia.org/wiki/%EB%B2%A0%EC%9D%B4%EC%8A%A464))에 따라 일련의 문자열로 인코딩한 토큰을 의미합니다.

<JWT 구성>

JWT는 클라이언트가 본인의 상태를 서버에게 전달해주는 방식이기에 토큰 위변조 문제가 있습니다. 서버는 토큰의 위변조를 방지하기 위해서, 토큰의 뒷 부분에 토큰의 내용과 특정 암호(secret key)를 기반으로 Signature 문자열을 붙혀서 토큰을 생성하게 됩니다. 이를 통해서 서버는 클라이언트에게 상태 관리를 위임하면서도, 토큰의 위변조를 방지 할 수 있습니다.

5. SPA (Single Page Application)

고전적인 프로그램 아키텍쳐는 모놀리식 아키텍쳐(Monolithic Architecture), 즉 일체형 아키텍쳐를 따르는 경우가 많습니다. 이는 UI와 데이터 계층, 통신 계층이 모두 일체형으로 작성된 구조를 말합니다. 우리가 이전에 작성했던 웹 서버들이 대표적입니다.

일체형 아키텍쳐에서 나아가서, REST 아키텍쳐는 UI 계층을 분리하여 클라이언트 프로그램의 범용성을 높힙니다.

<Monolithic vs Micro Service Architecture>

REST에서 더 나아가서, 대형 시스템의 개발에서는 하나의 응용프로그램을 계층별로 세분화하고 여러 프로그램으로 분리하여, 협업 및 배포의 효율성을 높히고, 확장성 있는 구조를 띄는 마이크로 서비스 아키텍쳐(Micro Service Architecture)를 지향하기도 합니다.

아키텍쳐에 대한 자세한 내용은 더 이상 다루지 않고, REST API 서버와 대응되는 웹 응용프로그램을 하나 작성해보도록 하겠습니다.

<SPA 흐름도>

SPA(Single Page Application)는 REST API와 대응되는 JavaScript로 작성된 웹 프로그램을 말합니다. 고전적인 웹과는 아래의 차이점을 보입니다.

- 페이지의 이동이 없음, 모든 로직은 최초의 html 문서에서 시작
- URL이 페이지의 상태를 표현하지 않음, 이 때문에 앱의 상태에 따라 URL을 가상으로 변경하기도 함
- 유저 이벤트 등에 따라서 DOM이 동적으로 생성, 삭제, 수정되는 일이 빈번
- 데이터를 제어하기 위해서 REST API 서버에 ajax를 통해서 소통

<SPA (연락처 앱)>

SPA의 설계를 도와주는 다양한 JavaScript 프레임워크(Angular.js, Ember.js, Redux, Vue.js, 등)가 있습니다. 이런 프레임워크의 선택에 대해서는 추후 고민해보도록 하고 당장은 jQuery만을 이용해서 아래 스펙을 만족하는 연락처 SPA를 작성해보겠습니다.

1. {name, phone, email}을 갖는 연락처 정보를 모델링
2. 연락처 모델의 CRUD를 수행하는 REST API를 구현
3. REST 서버와 Model은 다른 모듈(파일)로 분리
4. REST API와 통신하는(인증은 생략) SPA을 하나의 html문서로 작성
5. SPA에는 연락처 생성, 조회(검색창에 값이 바뀌는 대로 실시간 검색), 수정, 삭제 기능

데모 프로그램 (https://benzen-spa-demo.herokuapp.com/) 및 소스코드 (/course/490/src)를 확인 하실 수 있습니다. 백엔드는 인증 과정을 생략하고 Data 부분만 모델링하였고, 프론트는 코드량의 부담을 줄이기 위해서 모델링 과정을 생략하고 절차지향적으로 단순히 설계하였습니다. 다음 챕터에서 본격적으로 완성된 웹 서비스를 개발해보기 전에, 개별적으로 코딩 연습 및 웹에 대한 이해도를 높히기 위해서 본 프로젝트를 완수하는 것이 큰 도움이 되겠습니다.

[1] REST, Representational State Transfer

[2] CRUD(Create, Read, Update, Delete), Create, Read, Update, Delete

[3] API KEY

[4] Service Provider

[5] Service Consumer

[6] OAuth, Open Authentication

[7] SSO, Single Sign On

[8] JWT, JSON Web Token

[9] Base 64

[10] Monolithic Architecture

[11] Micro Service Architecture

[12] SPA, Single Page Application

5 데이터베이스

1. 메모리와 파일 ... 320

2. DB와 DBMS ... 327

3. MySQL과 SQL ... 338

4. Connector, SQL Injection, ORM ... 380

DB와 DBMS, 그리고 RDBMS와 NoSQL에 대해서 알아봅니다. RDBMS의 일종인 MySQL 서버를 설치하고 CLI/GUI MySQL 클라이언트를 이용해 기본적인 SQL을 공부해봅니다. 이후 DB를 모델링하는 원칙과 타 소프트웨어에서 DB를 이용하는 방법을 공부합니다. 나아가서 MVC 패턴과 ORM에 대해 알아봅니다.

5 데이터베이스

5.1 메모리와 파일

1. 파일에 데이터를 보관하기
2. 데이터 포맷
3. TodoList 리팩토링

> 메모리의 데이터를 영구적으로 보관하기 위해서 데이터 포맷을 정의하고 파일로 저장해봅니다.

1. 파일에 데이터를 보관하기

<Memory vs. FileSystem>

CPU는 메모리에 있는 데이터만을 이용해 작업을 할 수 있습니다. 지난 4장에서 작성한 TodoList 서비스는 모든 데이터를 메모리 상에 유지하였습니다. 때문에 서버 프로그램을 종료하게 되면 데이터의 변경 사항을 잃어버리게 됩니다. 또한 메모리는 그 용량이 보조기억장치에 비해 작기 때문에 대규모의 데이터를 유지하지 못합니다. 이러한 한계를 극복하기 위해서 메모리의 데이터를 OS의 파일 시스템을 통해 보조기억장치에 파일로 저장 할 수 있습니다. 또한 데이터를 파일의 형태로 저장함으로써, 프로그램과 데이터를 분리 할 수 있고, 이를 통해 데이터의 보존과 이동이 가능합니다.

<Loading?>

특히 게임 같이 이미지, 미디어 등의 용량이 큰 리소스를 많이 다루는 프로그램엔 로딩(Loading)이라는 작업이 빈번합니다. 메모리를 하나의 프로세스가 독식 할 수 없고, 또 메모리 용량보다 프로그램에 필요한 전체 리소스의 용량이 큰 경우엔 리소스 전체를 메모리에 유지 할 수가 없습니다. 이 때문에 리소스를 파일의 형태로 저장해둔 채로, 실행 중에 실시간으로 필요한 리소스를 조금씩 메모리 상으로 로드하고, 불필요한 리소스는 메모리에서 해제하는 과정이 필요합니다. 이 과정을 로딩이라고 표현합니다.

2. 데이터 포맷

<바이너리 포맷>

데이터를 파일 시스템을 통해 저장하려면 데이터를 어떻게 구조화해서 저장 할 지 계획해야 합니다. 이때 **독자적인 파일 포맷을 고안해서 바이너리로 저장** 할 수 있습니다. (이미지, 비디오나 오디오, 포토샵, PDF 등)

<텍스트 포맷>

또 쉽게는 데이터를 특정 인코딩을 따라 텍스트로 변환해서 파일로 저장 할 수 있습니다. 이때 데이터를 구조화하기 위해서 CSV, XML, JSON과 같은 널리 쓰이는 텍스트 포맷을 이용 할 수 있겠습니다. 텍스트로 저장하는 경우엔 사람이나, 다른 프로그램이 데이터를 이해하기에 좋습니다. 다만 인코딩 과정에서 파일의 크기가 증가 할 수 있습니다.

3. TodoList 리팩토링

웹 서비스에서 유저들이 바이너리 데이터를 생산 할 일이 많지는 않습니다. 4장에서 작성했던 TodoList의 데이터를 JSON 포맷의 텍스트 파일로 저장하고 관리 할 수 있도록 리팩토링해보겠습니다.

todolist_file/models/model.js

```
const fs = require('fs-promise'); // fs 코어 모듈의 콜백 기반 API를 promise로 대체한 모듈
const path = require('path');
const dataFilePath = path.join(__dirname, 'data');

module.exports = {
    getAll: getAll,
    getList: getList,
    addToList: addToList,
    deleteFromList: deleteFromList
};

function getAll(){
    return fs.readFile(dataFilePath).then(JSON.parse);
}

// user_id의 유저의 리스트를 가져온다.
function getList(user_id){
    return fs.readFile(dataFilePath).then(data => {

        data = JSON.parse(data);
        let user = data.find(user => user.id == user_id);

        return user.list;
    });
}

// user_id의 유저의 리스트에 아이템을 추가한다.
function addToList(user_id, item){
    return fs.readFile(dataFilePath).then(data => {

        if (item) { // 일종의 데이터 검증...
            data = JSON.parse(data);
            data.find(user => user.id == user_id).list.push(item);
            data = JSON.stringify(data);

            return fs.writeFile(dataFilePath, data);
        }
    });
}

// user_id의 유저의 리스트에서 item_index번째 아이템을 삭제한다.
function deleteFromList(user_id, item_index){
    return fs.readFile(dataFilePath).then(data =>{

        data = JSON.parse(data);
        data.find(user => user.id == user_id).list.splice(item_index, 1);
        data = JSON.stringify(data);

        return fs.writeFile(dataFilePath, data);
    });
}
```

todolist_file/controllers/list-router.js

```
const express = require('express');
const bodyParser = require('body-parser');
const model = require('../models/model');
const router = express.Router();

// 할 일 리스트 보여주기
router.get('/', (req,res)=>{
    model.getList(req.session.user.id).then(list => {
        res.render('list', {
            list: list
        });
    });
});

// 할 일 추가
router.post('/', bodyParser.urlencoded(), (req,res)=>{
    model.addToList(req.session.user.id, req.body.item).then(() => {
        res.redirect('/list');
    });
});

// 할일 데이터 지우기
router.get('/delete', (req,res)=>{
    model.deleteFromList(req.session.user.id, req.query.index).then(() => {
        res.redirect('/list');
    });
});

module.exports = router;
```

파일에 데이터를 읽고 쓰는 것을 model이라는 모듈로 추상화하여 데이터를 관리하고 있습니다. 잘 작동합니다만 몇가지 고려해볼 문제가 있습니다.

1. 데이터의 구조적 변경이 쉽지 않음: ?
2. 모든 유저의 데이터를 한개의 파일이 갖고 있어서 여러명이 동시에 읽고 쓸 수가 없음: 유저 별로 파일을 분리?
3. 아이템을 추가하고 삭제하는 작업에도 파일 전체를 읽었다가 해석했다가 다시 저장하는 비용이 큰 작업을 반복함: 버퍼링?

> 버퍼링(Buffering)은 다양한 의미로 쓰이는 단어입니다. 파일 시스템에서 버퍼링이란, 파일에 변경이 있을 때마다 보조기억장치에 데이터를 저장하는 작업을 수행하지 않고 (메모리의 IO와 HDD의 IO 속도는 10만배까지도 차이가 납니다), 덮어 쓸 파일의 내용이나 변경사항들을 메모리에 기억해두었다가, 여러번의 파일 변경 작업을 한꺼번에 보조기억장치에 반영하여 성능을 향상시키는 방법입니다.

[1] Loading

[2] Buffering

5 데이터베이스

5.2 DB와 DBMS

1. DB와 DBMS
2. RDBMS
3. RDB의 관계들
4. NoSQL
5. DBMS의 선택과 모델링

> DB와 DBMS에 대해서 알아봅니다. RDBMS의 주요 개념과 NoSQL, DB를 모델링을 하는 원칙을 알아봅니다.

1. DB와 DBMS

DB(Database)는 데이터의 집합을 의미하는 용어입니다. 즉 지난 장에서 TodoList의 데이터를 저장해 둔 파일은 일종의 DB로 볼 수 있습니다. 그리고 이 때 DB의 접근과 관리를 좀 더 추상화한 소프트웨어들을 DBMS(Database Management System)라고 합니다. 지난 장에서 작성한 model 모듈도 일종의 저수준 DBMS라고 할 수는 있겠습니다. 상용 DMBS들이 일반적으로 제공하는 기능은 아래와 같습니다.

속도	파일을 읽고 쓰는 것보다 빠르게 데이터를 읽고 씀
확장성	단순히 단일 파일로 관리되지 않으며, 그 규모를 쉽게 확장 가능
동시성	서버로 작동하여 다수의 요청을 안정적으로 처리
관계	데이터간의 관계를 정의 가능
무결성	데이터의 구조를 엄밀하게 정의하고, 쉽게 변경 가능
	정의된 구조에 적합하지 않은 데이터를 방지
	데이터의 중복을 방지
	데이터 간에 성립할 수 없는 관계를 방지
	'A통장에서 100만원 감소, B통장에서 100만원 증가'처럼 이체와 같이 중도에 중지되면 안되는 작업에 대한 보장
질의	필요한 데이터를 질의(Query)하는 방식으로 쉽게 추출
보안	로그인을 통한 인증 수단과 DB 제어에 대한 권한을 설정 가능

<DBMS의 기능>

> 위에서 예로 든 통장 이체와 같이 일련의 작업을 트랜잭션(Transaction)(https://ko.wikipedia.org/wiki/%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0
> 이라고 합니다. DBMS는 트랜잭션이 상호 독립적이며, 트랜잭션의 요소들이 모두 성공하거나(commit), 모두 실패하도록(rollback) 보장 할 수 있습니다.

<DBMS as server>

DBMS는 프로그램에 종속되어 로컬(local)에서 작동 할 수도 있지만 **많은 DBMS는 TCP 기반의 서버-클라이언트 구조**를 취하고 있습니다. 즉, 일반적인 DBMS는 단일한 서버 프로그램으로 운영이되며, 마치 웹 서버처럼 다수의 클라이언트 프로그램의 요청에 응답하게 됩니다. 이를 통해 **데이터를 프로그램에서 분리 할 수 있고, 또 물리적인 확장성**을 가질 수 있습니다.

<Mysql CLI Client>

이 때 DBMS는 서버 프로그램과 함께 **CLI 클라이언트 프로그램**을 제공합니다.

<Mysql GUI Client>

또 GUI 클라이언트 프로그램을 제공하기도 합니다.

<Mysql Client Module for Node.js>

또한 DBMS의 프로토콜을 따라 TCP 수준에서 직접 클라이언트 프로그램을 개발 할 수도 있습니다만, 보통의 응용 프로그램에서는 플랫폼 별로 제공되는 클라이언트 모듈을 이용해서 DBMS 서버와 통신합니다.

2. RDBMS

<Relational Model>

DBMS의 대표적인 종류로 RDBMS(Relational DBMS)가 있습니다. RDBMS는 데이터의 독립적인 모델들을 각각의 테이블로 표현하고, 테이블간의 관계를 정의 함으로써 데이터를 모델링합니다. 예를 들어 자동차라는 모델은 cars라는 테이블로, 색상이라는 모델을 colors라는 테이블로 추상화하고, shops와 products 사이의 관계를 정의 할 수 있습니다.

항목	설명
데이터베이스 Database	DBMS 서버에는 여러개의 DB가 존재 할 수 있으며, DB별로 인증과 권한을 설정할 수 있음
테이블 Table	하나의 DB는 모델을 추상화한 테이블/모델/스키마를 여러개 갖음
열 Column	테이블은 여러개의 열/컬럼/속성/필드를 갖고, 각 열은 이름과 데이터 타입 등 여러 옵션을 갖음
주 키 Primary Key	테이블의 각 행을 고유하게 대표하는 열, 값이 고유하지 않으면 거부됨
외래 키 Foreign Key	다른 테이블의 PK를 가리키는 열, 다른 테이블의 존재하지 않는 행을 참조하면 거부됨
색인 Index	특정 열에 대한 색인을 생성하면 데이터 조회시 성능을 향상 시킬 수 있음, PK는 기본적으로 색인을 갖음 (색인과 테이블은 독립적)
행 Row	테이블은 여러개의 행/로우/레코드/튜플/데이터를 갖고, 각 행은 속성에 맞는 값들의 그룹으로 이루어짐

<RDBMS의 구조>

RDMBS에서는 DB를 생성하고, 모델들을 각각 독립적인 테이블로 정의 한 뒤, 테이블 간의 관계를 설정하고, 이를 기반으로 테이블에 데이터를 저장하며, 또 테이블 간의 데이터를 조합하여 조회(join) 할 수 있습니다. 이 때 데이터의 조회, 삽입, 수정, 삭제 및 테이블의 정의 및 수정, DB의 생성 등, 모든 작업은 SQL(Structured Query Language)이라는 언어를 통해 DBMS 서버에 요청됩니다. SQL의 문법을 바탕으로 작성된 문장을 질의(Query)라고 합니다.

예시 Query

```
select students.name, departments.name
    from students, departments
    where students.dept_id=departments.id;
```

SQL 문법에 대해서는 다음 챕터에서 알아보도록 하겠습니다.

3. RDB의 관계들

<PK와 FK>

PK는 테이블 내의 특정 데이터를 고유하게 식별해주는 역할을 합니다. 따라서 일반적으로 모든 테이블은 하나의 PK를 갖습니다. 또한 서로 다른 테이블의 데이터 간의 관계는 일반적으로 PK와 FK의 값을 일치시켜 표현합니다. 테이블간에 구성 할 수 있는 관계의 종류에 대해서 알아보겠습니다.

1. 일 대 일 관계 (1:1, One-to-One, HasOne/BelongsTo)

<한 부서에는 한 매니저 직원이 있다>

- 일대일 관계에선 A 테이블의 데이터가 B 테이블의 데이터 하나와만 연관이 있습니다.
- 이 때 FK는 두 테이블 중 어디에 있어도 상관 없습니다.

2. 일 대 대 관계 (1:N, One-to-Many, HasMany/BelongsTo)

<한 선생님은 여러 과목을 가르친다>

- 다대일 또는 일대다 관계에선 A 테이블의 데이터가 B 테이블의 데이터 여러개와 연관이 있습니다.
- A hasMany B일 때 FK는 B에 생성됩니다.

3. 다 대 다 관계 (M:N, Many-to-Many, belongsToMany)

<한 학생은 여러 과목을 수강하고, 한 과목은 여러 학생에게 수강된다>

- 다대다 관계에서는 A 테이블의 데이터와 B 테이블의 데이터가 서로 여러개와 연관 될 수 있습니다.
- 다대다 관계는 A/B 테이블의 관계를 나타내기 위한 추가적인 테이블이 필요합니다. 이를 피벗 테이블(Pivot Table)이라하며 이 피벗 테이블에 두개의 FK가 필요합니다.

4. NoSQL

<스케일 업, 스케일 아웃>

인터넷이 발전하면서 빅데이터라는 용어가 생길만큼 데이터의 규모가 기하급수적으로 커지고 있습니다. 이에 따라 DBMS 서버의 머신은 많은 사용자의 요청을 처리하고, 대규모의 데이터를 저장하기 위해서 그 성능과 규모를 확장해야 할 필요가 있습니다. 이 때 단일 머신의 성능과 규모를 확장하는 스케일 업을 통한 확장은 그 한계가 있으며 기하급수적으로 비용이 증가합니다. 따라서 일반적인 사양의 머신을 여러대로 늘려서, 사용자의 요청을 분산해서 처리하며, 데이터를 분산 저장하는 스케일 아웃을 통한 확장을 통해 비용을 절감 할 수 있습니다. 하지만 이 때 RDBMS는 데이터 처리에 무결성과 관계성을 갖는 특성상, 스케일 아웃을 통한 규모 확장을 꾀하기가 힘듭니다. 이러한 배경에서 RDBMS의 한계를 극복하고 분산 DB를 구축하기 위한 DBMS들이 등장했습니다.

<RDBMS vs. NoSQL>

NoSQL(Not-Only-SQL)은 RDBMS의 강력한 무결성과 관계성을 버린 DBMS들을 포괄하는 용어입니다. NoSQL은 무결성과 관계성을 적당히 포기하면서 스케일 아웃을 통해 대규모의 분산 DB를 구축 할 수 있는 배경을 마련했습니다. 또한 정해진 테이블의 정의에 딱 맞는 평면적인 데이터만을 저장 할 수 있는 RDBMS와 다르게, NoSQL에서는 비정형화된 데이터를 저장 할 수 있는 특성을 갖습니다. NoSQL에는 단순한 Key-Value 형식의 DB, JSON과 같은 Document 형식의 DB, 그래프 형식의 DB 등 다양한 종류가 있습니다.

5. DBMS의 선택과 모델링

종류	기준
RDBMS	데이터를 독립적인 모델들과 그 관계로 표현 할 수 있다. 데이터 처리에 신뢰성과 무결성이 필요하다.
NoSQL	데이터를 정형화 하기 힘들다. 신뢰성과 무결성보다는 확장성(Scale-Out)과 빠른 속도가 필요하다.
파일 시스템	주로 그래픽, 영상 같은 바이너리 데이터들을 다룬다. 확장성이나 신뢰성, 무결성, 데이터의 구조화가 필요하지 않다. DBMS 서버와 통신하는 비용(속도)을 감당 할 수 없다.

<DBMS의 선택 기준>

적절한 DBMS를 선택했다면, 다음 단계로는 데이터를 모델링해야 합니다. 서비스에서 이용 될 데이터를 추상화하고 그 관계를 설정하는 일을 데이터 모델링(Data Modeling)이라고 합니다. 서비스 개발의 전체적인 흐름이 데이터 모델에 크게 의존적이기 때문에, 시작부터 모델을 잘 정의 할 필요가 있습니다. 모델링 방식은 그 프로젝트의 성격에 따라 천차만별일 수 있지만, 일반적으로 지켜야 할 기본적인 원칙들이 있습니다.

<개념적 데이터 모델링>

항목	설명
추상화	현실 세계의 객체를 추상화를 통해, 단순하고 명확하게 표현합니다. 불필요한 속성은 최대한 제거하고, 속성들은 최대한 단순한 데이터로써 표현합니다.
데이터 분리	(RDB의 경우) 중복적으로 사용 될 수 있는 데이터들은 새로운 모델로 분리합니다. 직원 모델에 부서명이라는 속성이 있을 수 있지만, 부서라는 모델을 새로 분리하면 중복 데이터의 제거, 모델의 확장 및 변경에 유리합니다.
확장성	서비스에서 다루는 데이터가 변경 또는 확장 될 경우를 따져 유연하게 모델링합니다.
비즈니스	서비스 기획을 바탕으로, 사용자가 서비스와 상호작용하는 과정을 하나씩 고려해보며 모델을 점검 합니다.

로직	

<데이터 모델링의 기본 원칙>

데이터 모델의 완성도를 판단하는 절대적인 척도는 없습니다. 또한 서비스의 변화에 따라서 모델은 끊임없이 변할 수 있습니다. 특히 개발 초기에는 **무결성, 확장성, 효율성을 점검하며 모델을 빈번하게 개선**해야합니다.

> **비즈니스 로직(Business Logic)**은 컴퓨터 프로그램에서 실세계의 규칙에 따라 데이터를 생성·표시·저장·변경하는 부분을 일컫는다... (위키백과)

[1] DB, Database
[2] DBMS, Database Management System
[3] Transaction
[4] RDBMS, Relational DBMS
[5] Primary Key
[6] Foreign Key
[7] Index
[8] Pivot Table
[9] NoSQL, Not-Only-SQL
[10] Data Modeling
[11] Business Logic

5 데이터베이스

5.3 MySQL과 SQL

1. MySQL 설치(CLI/GUI)
2. MySQL의 DB
3. 쿼리의 종류
4. DB와 Table
5. Field의 데이터 타입
6. CRUD
7. PK, FK, 무결성 제약
8. not null, default, auto_increment, unique
9. where, group by, having, order by, limit
10. join, union
11. 인덱스, 뷰, 프로시저, 트리거

MySQL 서버를 설치하고 CLI/GUI 클라이언트 사용법을 익힙니다. MySQL의 주요 개념들에 대해 공부하고 실제 모델링을 해봅니다.

1. MySQL 설치(CLI/GUI)

<MySQL Community Server>

오픈소스 RDBMS 중 MySQL을 이용해 DBMS에 대해서 배워보겠습니다. MySQL 다운로드 링크 (http://dev.mysql.com/downloads/mysql/)에서 OS에 맞는 MySQL Community Server, MySQL Shell (CLI Client), MySQL Workbench (GUI Client)를 설치합니다.

MySQL Community Server는 DBMS 서버 프로그램이고, MySQL Shell은 CLI 클라이언트 프로그램입니다. 또 MySQL Workbench는 GUI 클라이언트 프로그램으로, GUI를 통해 용이하게 DB를 설계하고 관리 할 수 있게 해줍니다.

로컬에서 mysql 서버가 실행중이라면, Shell에서 아래와 같이 mysql CLI 클라이언트 프로그램으로 로컬 mysql 서버에 접속 할 수 있습니다. 셸에서 직접 mysql CLI 클라이언트를 실행 할 수 없다면 (Windows의 경우), 위에서 설치한 MySQL Shell을 실행합니다.

Mysql CLI 프로그램을 실행

```
mysql -uroot -p
Enter password:
```

2. MySQL의 DB

<DB Hierarchy>

DBMS는 프로젝트 단위로 구분되는 다수의 DB를 갖습니다. 또한 하나의 DB는 모델 단위로 구분되는 다수의 Table을 갖습니다. 그리고 하나의 Table은 데이터 단위로 구분되는 다수의 Record를 갖습니다. 마지막으로 하나의 Record는 키와 값의 집합으로 이루어지며, 이 때 이 키를 Field라고 합니다.

서버의 모든 DB를 아래와 같이 출력 수 있습니다.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| blog               |
| iad                |
| mysql              |
| performance_schema |
| study              |
| sys                |
| todo               |
| workshop           |
| workshop2          |
+--------------------+
10 rows in set (0.01 sec)
```

DB를 설계하기 위해서 프로젝트에서 사용 할 DB를 생성합니다.

```
mysql> create database name_of_db;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| blog               |
| iad                |
| mysql              |
| name_of_db         |
| performance_schema |
| study              |
| sys                |
| todo               |
| workshop           |
| workshop2          |
+--------------------+
11 rows in set (0.00 sec)
```

아래처럼 특정 DB를 삭제 할 수 있습니다. 백업되지 않은 데이터는 복원 할 수 없으니, 사용에 주의해야 할 쿼리입니다.

```
mysql> drop database name_of_db;
Query OK, 0 rows affected (0.00 sec)
```

3. 쿼리의 종류

쿼리의 문법 체계, SQL에 빠르게 익숙해지는 것은 쉽지 않습니다. 하지만 대부분의 DBMS들이 비슷한 체계의 SQL을 제공하기에 한번 SQL 문법에 익숙해지면 다른 DBMS를 사용하더라도 큰 어려움이 없습니다. DB를 설계하는 과정을 계속하기 전에, 쿼리의 종류에 대해서 먼저 정리해보도록 하겠습니다.

DDL (Data Definition Language)	
create	DB나 테이블을 생성
drop	DB나 테이블을 삭제
alter	테이블을 수정

truncate	테이블을 초기화
DML (Data Manipulation Language)	
select	테이블의 레코드를 조회
insert	테이블에 레코드를 추가
delete	테이블의 레코드를 삭제
update	테이블의 레코드를 수정
DCL (Data Control Language)	
grant	(관리자가) 사용자에게 DB 제어의 권한을 부여
revoke	(관리자가) 사용자에게 DB 제어의 권한을 박탈
start transaction	트랙잭션의 시작
commit	트랜잭션 승인
rollback	트랜잭션 복원

<SQL 종류>

4. DB와 Table

	Field	Type	Collation	Attributes	Null	Default	Extra
☐	id	int(11)			No		auto_increment
☐	title	text	utf8_general_ci		No		
☐	content	text	utf8_general_ci		No		
☐	ordering	int(11)			No	0	
☐	position	varchar(150)	utf8_general_ci		Yes	*NULL*	
☐	checked_out	int(11)		UNSIGNED	No	0	
☐	checked_out_time	datetime			No	0000-00-00 00:00:00	
☐	published	tinyint(1)			No	0	

<Table 정보>

프로젝트의 모델을 표현하고 그 데이터를 보관하기 위한 테이블을 생성합니다. 이 때 테이블의 이름을 설정하고, 필드들의 이름과 데이터 타입, 기본값, 인덱스 여부, PK, FK등을 설정합니다.

먼저 사용 할 DB를 선택합니다.

```
mysql> create database name_of_db;
Query OK, 1 row affected (0.00 sec)

mysql> use name_of_db;
Database changed

mysql> show tables;
Empty set (0.00 sec)
```

테이블을 생성, 수정, 변경, 삭제해봅니다.

```
mysql> create table test(
    -> field1 int,
    -> field2 char
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> describe test;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| field1 | int(11) | YES  |     | NULL    |       |
| field2 | char(1) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> alter table test add column(
    -> field3 int
    -> );

mysql> describe test;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| field1 | int(11) | YES  |     | NULL    |       |
| field2 | char(1) | YES  |     | NULL    |       |
| field3 | int(11) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> alter table test add column( field4 int, field5 int );
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table test drop field3, drop field4;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table test add column(fieldx int), drop field5;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe test;
+--------+---------+------+-----+---------+-------+
| Field  | Type    | Null | Key | Default | Extra |
+--------+---------+------+-----+---------+-------+
| field1 | int(11) | YES  |     | NULL    |       |
| field2 | char(1) | YES  |     | NULL    |       |
| fieldx | int(11) | YES  |     | NULL    |       |
+--------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

테이블을 생성하고 삭제하거나, 필드들을 추가, 삭제 할 수 있습니다. 이 때 테이블에 레코드가 존재하는 경우엔 필드를 삭제하는 과정에서 데이터의 손실이 발생 할 수 있습니다.

5. Field의 데이터 타입

위에서 빈 테이블을 생성하고 필드를 변경하는 중에 field1 int, field2 char와 같이 필드의 이름과 데이터 타입을 명시했었습니다. int는 Integer, char는 Character로 각각 정수와 문자열 타입을 의미합니다. DBMS에서는 다양한 데이터를 저장하고 또 적합하지 않은 데이터를 방지 할 수 있도록 다양한 타입을 지원합니다.

데이터 타입	설명
int	4 Byte로 저장되는 정수
tinyint	1 Byte로 저장되는 정수
bigint	8 Byte로 저장되는 정수
float(M,D)	4 Byte로 저장되는 최대 M자리, 소수점 아래 최대 D자리의 실수 (부정확)
double(M,D)	8 Byte로 저장되는 최대 M자리, 소수점 아래 최대 D자리의 실수 (부정확)
decimal(M,D)	가변 크기로 저장되는 최대 M자리, 소수점 아래 최대 D자리의 실수 (정확)
char(N)	정해진 인코딩을 따르는 N자리의 문자열
varchar(N)	정해진 인코딩을 따르는 최대 N자리의 가변 길이 문자열
text	정해진 인코딩을 따르는 최대 64KB의 가변 길이 문자열
mediumtext	정해진 인코딩을 따르는 최대 16MB의 가변 길이 문자열
longtext	정해진 인코딩을 따르는 최대 4GB의 가변 길이 문자열
blob	최대 64KB의 가변 길이 바이너리
enum('abc', '123', ...)	지정된 값들 중에서만 입력 할 수 있음
date	년-월-일의 날짜
datetime	날짜에 시:분:초까지
timestamp	1970년 0초 부터 특정 시각까지의 기간을 초로 환산

<MySQL의 주요 데이터 타입>

MySQL 필드의 전체 데이터 타입 (http://dev.mysql.com/doc/refman/5.7/en/data-

types.html)

6. CRUD

CRUD (Create, Read, Update, Delete)는 응용프로그램에서 데이터를 조작하는 방식을 표현하는 키워드입니다. 응용프로그램은 사용자의 요청에 따라 데이터를 CRUD하고 그 결과를 화면에 반영해주는 방식으로 구현됩니다. DBMS에서는 위에서 봤던 DML(Data Manipulation Language) Query를 통해 DB의 데이터를 CRUD합니다.

데이터 Create

```
mysql> create table users( id int, name varchar(32), created_at datetime );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into users values(1, 'admin', '2017-11-11');
Query OK, 1 row affected (0.00 sec)

mysql> insert into users values(2, 'guest', now());
Query OK, 1 row affected (0.00 sec)
```

마지막 쿼리에서 현재의 시간을 반환하는 now() 처럼 SQL들은 특정한 기능을 수행하는 함수를 제공합니다.

데이터 Read

```
mysql> select name, created_at from users;
+-------+---------------------+
| name  | created_at          |
+-------+---------------------+
| admin | 2017-11-11 00:00:00 |
| guest | 2016-12-08 18:57:40 |
+-------+---------------------+
2 rows in set (0.00 sec)

# 모든 필드는 as를 통해 조회시의 필드명을 다르게 나타 낼 수 있습니다.
mysql> select id as 번호, name as 이름, created_at as 가입일 from users;
+--------+--------+---------------------+
```

```
| 번호   | 이름   | 가입일               |
+--------+--------+---------------------+
|      1 | admin  | 2017-11-11 00:00:00 |
|      2 | guest  | 2016-12-08 18:57:40 |
+--------+--------+---------------------+
2 rows in set (0.00 sec)

# as 키워드는 생략 가능합니다.
mysql> select id 번호, name 이름, created_at 가입일 from users;
+--------+--------+---------------------+
| 번호   | 이름   | 가입일               |
+--------+--------+---------------------+
|      1 | admin  | 2017-11-11 00:00:00 |
|      2 | guest  | 2016-12-08 18:57:40 |
+--------+--------+---------------------+
2 rows in set (0.00 sec)

# * 은 모든 필드를 나타내는 특수 문자입니다.
mysql> select * from users;
+------+-------+---------------------+
| id   | name  | created_at          |
+------+-------+---------------------+
|    1 | admin | 2017-11-11 00:00:00 |
|    2 | guest | 2016-12-08 18:57:40 |
+------+-------+---------------------+
2 rows in set (0.00 sec)

mysql> select users.id from users;
+------+
| id   |
+------+
|    1 |
|    2 |
+------+
2 rows in set (0.00 sec)

# 테이블명에도 as를 사용 할 수 있습니다.
mysql> select users.id from users as u;
ERROR 1054 (42S22): Unknown column 'users.id' in 'field list'

mysql> select u.id, u.name 이름 from users u;
+------+--------+
| id   | 이름   |
+------+--------+
|    1 | admin  |
|    2 | guest  |
+------+--------+
2 rows in set (0.00 sec)
```

데이터를 Read하는 쿼리는 상당히 복잡한 로직을 동반할 수 있습니다. 이에 대해서는 뒤에서 더 다룹니다.

데이터 Update

```
mysql> select * from users;
+------+-------+---------------------+
| id   | name  | created_at          |
+------+-------+---------------------+
|    1 | admin | 2017-11-11 00:00:00 |
|    2 | guest | 2016-12-08 18:57:40 |
+------+-------+---------------------+
2 rows in set (0.00 sec)

mysql> update users set name='dumb';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select * from users;
+------+------+---------------------+
| id   | name | created_at          |
+------+------+---------------------+
|    1 | dumb | 2017-11-11 00:00:00 |
|    2 | dumb | 2016-12-08 18:57:40 |
+------+------+---------------------+
2 rows in set (0.00 sec)


# update 쿼리는 기본적으로 테이블의 모든 레코드를 대상으로 적용됩니다.
# 이때 update 쿼리의 뒷 부분에 where절을 추가해서 특정한 레코드를 대상으로 쿼리를 실행할 수 있습니다.
# where절은 boolean으로 평가되는 표현식입니다.

mysql> update users set name='smart' where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from users;
+------+-------+---------------------+
| id   | name  | created_at          |
+------+-------+---------------------+
|    1 | smart | 2017-11-11 00:00:00 |
|    2 | dumb  | 2016-12-08 18:57:40 |
+------+-------+---------------------+
2 rows in set (0.00 sec)

# update 쿼리의 set 구문에 기존 필드명을 참조할 수 있습니다.
# 또 where의 표현식을 다양하게 조합 할 수 있습니다.

mysql> update users set name=concat(name, '_hi') where name like 'sm%';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from users;
+------+----------+---------------------+
| id   | name     | created_at          |
+------+----------+---------------------+
|    1 | smart_hi | 2017-11-11 00:00:00 |
|    2 | dumb     | 2016-12-08 18:57:40 |
+------+----------+---------------------+
```

```
+------+----------+---------------------+
2 rows in set (0.00 sec)

# set 구문의 콤마(,)를 통해 한 쿼리로 여러 필드를 수정 할 수 있습니다.

mysql> update users set id=id+10, name='smart_11' where id != 2 and created_at > '20
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from users;
+------+----------+---------------------+
| id   | name     | created_at          |
+------+----------+---------------------+
|   11 | smart_11 | 2017-11-11 00:00:00 |
|    2 | dumb     | 2016-12-08 18:57:40 |
+------+----------+---------------------+
2 rows in set (0.00 sec)
```

눈치 채셨겠지만 where 구문은 insert 쿼리를 제외한 select, update, delete 쿼리에 모두 적용 할 수 있습니다.

데이터 Delete

```
mysql> delete from users;
Query OK, 2 rows affected (0.00 sec)

mysql> select * from users;
Empty set (0.00 sec)
```

delete 쿼리는 where 조건이 없으면 테이블의 모든 레코드를 삭제하므로 사용에 주의가 필요합니다.

7. PK, FK, 무결성 제약

필드에 타입을 지정하는 이유는 적합하지 않은 데이터를 방지하기 위해서입니다. 이렇게 데이터 타입을 통한 제약 조건을 도메인 무결성 제약 (Domain Integrity Constraint)이라고 합니다.

도메인 무결성

```
mysql> insert into users values('id', 'name', now());
ERROR 1366 (HY000): Incorrect integer value: 'id' for column 'id' at row 1
```

뿐만 아니라 모든 테이블은 PK라고 하는 고유한 레코드를 대표하는 필드(들)을 지정 할 수 있습니다. 이를 통해서 PK 필드는 테이블 내에서 빈 값을 갖지 않으며, 고유한 값으로 유지될 수 있습니다. 이러한 제약 조건을 개체 무결성 제약 (Entity Integrity Constraint)이라고 합니다.

개체 무결성

```
mysql> drop table users;
Query OK, 0 rows affected (0.00 sec)

mysql> create table users(
    -> id int primary key,
    -> name varchar(32)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into users values(1, 'user1');
Query OK, 1 row affected (0.00 sec)

mysql> insert into users values(1, 'user2');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

mysql> insert into users values(null, 'user2');
ERROR 1048 (23000): Column 'id' cannot be null
```

일반적으로 테이블에 PK를 설정하고 이 PK를 기반으로 read, update, delete 작업을 수행합니다.

이전 챕터에서 배웠던 테이블간의 관계를 적용해보겠습니다. users 테이블과 1:M 관계인 orders 테이블을 생성합니다.

```
mysql> create table orders(
    -> id int primary key,
    -> user_id int,
    -> product varchar(32)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into orders values(1, 1, '어떤 상품');
Query OK, 1 row affected (0.00 sec)

mysql> insert into orders values(2, 10, '어떤 상품2');
Query OK, 1 row affected (0.00 sec)

mysql> select * from orders;
+----+---------+---------------+
| id | user_id | product       |
+----+---------+---------------+
|  1 |       1 | 어떤 상품     |
|  2 |      10 | 어떤 상품2     |
+----+---------+---------------+
2 rows in set (0.00 sec)
```

orders.user_id 가 users.id를 나타내도록 설계하였습니다. 이렇게 관계된 레코드의 PK 값을 통해서 레코드와 레코드간의 관계를 설정합니다. orders.id=1인 첫번째 레코드는 users.id=1인 user가 '어떤 상품'을 주문했다는 데이터를 표현합니다. 적절합니다만, orders.id=2인 둘째 레코드는 존재하지 않는 users.id=10인 user와의 관계를 표현하고 있습니다. 또 orders.user_id=1인 주문 데이터가 존재하는 상태에서, users.id=1인 사용자 데이터가 삭제 될 경우도 전체 데이터의 무결성이 무너집니다.

참조 무결성 제약

```
mysql> create table orders(
    -> id int,
    -> user_id int,
    -> product varchar(32),
    -> primary key(id),
    -> foreign key(user_id) references users(id)
    -> );
Query OK, 0 rows affected (0.01 sec)

# PK를 primary key(필드 [, ...])로 정의 할 수 있습니다.
# FK는 foreign key(필드) references 참조테이블(참조필드 [, ...])로 정의 할 수 있습니다.

mysql> insert into orders values(1, 1, '상품1');
Query OK, 1 row affected (0.00 sec)

mysql> insert into orders values(2, 10, '상품2');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails

mysql> delete from users where id=1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint f
```

FK를 설정하면 존재하지 않는 레코드의 PK를 참조 할 수 없습니다. 또한 1:1이나 1:M 관계에서 자식 레코드가 존재할 때 부모 레코드를 삭제하거나, 부모 레코드의 PK를 업데이트 할 수 없도록 할 수 있습니다. 이러한 제약 조건을 참조 무결성 제약 (Referential Integrity Constraint)이라고 합니다.

참조 무결성은 기본적으로 제약을 위반하는 경우가 위의 예시에서 알 수 있듯 restrict되어있습니다. 하지만 update, delete의 두 경우에 대해서 restrict, cascade, set null, no action을 설정 할 수도 있습니다. 예를 들어 user가 삭제되는 경우 모든 자식 order를 삭제하려면 on delete cascade, user의 PK가 업데이트 되는 경우 모든 자식 order의 연결된 FK를 업데이트하려면 on update cascade, user가 삭제되더라도 order는 남겨두되, 연결된 FK를 빈 값으로 지정하려면 on delete set null을 지정 하면 됩니다.

참조 무결성 제약, cascade

```
mysql> create table orders(
    -> id int,
    -> user_id int,
    -> product varchar(32),
    -> primary key(id),
    -> foreign key(user_id) references users(id) on delete cascade on update cascade
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into orders values(1, 1, "상품1");
Query OK, 1 row affected (0.00 sec)

mysql> delete from users where id = 1;
Query OK, 1 row affected (0.00 sec)

mysql> select * from orders;
Empty set (0.00 sec)
```

8. not null, default, auto_increment, unique

nullable 필드

```
mysql> describe users;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(32) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> insert into users values(1, null);
Query OK, 1 row affected (0.00 sec)

mysql> insert into users(id) values(2);
Query OK, 1 row affected (0.00 sec)

mysql> select * from users;
+----+------+
| id | name |
+----+------+
|  1 | NULL |
|  2 | NULL |
+----+------+
2 rows in set (0.00 sec)
```

users 테이블의 name 필드에 빈값을 허용하지 않으려면 not null 속성을 추가합니다. 참고로 PK 필드는 개체 무결성을 위해서 기본적으로 not null 속성을 갖습니다.

not null, default

```
mysql> drop table orders;
Query OK, 0 rows affected (0.00 sec)

mysql> drop table users;
Query OK, 0 rows affected (0.00 sec)

mysql> create table users(
    -> id int primary key,
    -> name varchar(32) not null default '이름없음'
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into users values(1, null);
ERROR 1048 (23000): Column 'name' cannot be null

mysql> insert into users values(1, '일번');
Query OK, 1 row affected (0.00 sec)

# 테이블명 뒤에 필드명을 나열해서 values 구문과 매치될 필드를 명시 할 수 있습니다.
mysql> insert users(id, name) values(2, '이번');
Query OK, 1 row affected (0.00 sec)

mysql> insert users(name, id) values('삼번', 3);
Query OK, 1 row affected (0.00 sec)

# 이때 기본값이 지정된 필드는 생략 할 수 있습니다.
mysql> insert users(id) values(4);
Query OK, 1 row affected (0.00 sec)

mysql> select * from users;
+----+--------------+
| id | name         |
+----+--------------+
|  1 | 일번         |
|  2 | 이번         |
|  3 | 삼번         |
|  4 | 이름없음     |
+----+--------------+
4 rows in set (0.00 sec)
```

이 때 PK 필드를 생략하고 싶다면 어떻게 해야 할까요? 프로그램에서 테이블의 id 최대값을 기억해두면서 매번 그 id 값을 통해 insert 시킬 수도 있겠습니다. 또는 insert 쿼리에 select 쿼리를 연결 할 수 도 있습니다. 하지만 이러한 방식보단 MySQL에서는 AI (auto_increment) 속성을 이용 할 수 있습니다.

auto_increment

```
mysql> insert into users (select max(id)+1, '삼식이' from users);
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> select * from users;
+----+--------------+
| id | name         |
+----+--------------+
|  1 | 일번         |
|  2 | 이번         |
|  3 | 삼번         |
|  4 | 이름없음     |
|  5 | 삼식이       |
+----+--------------+
5 rows in set (0.00 sec)

mysql> drop table users;
Query OK, 0 rows affected (0.00 sec)

mysql> create table users(
    -> id int primary key auto_increment,
    -> name varchar(32) not null default '이름 없음'
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> insert into users(name) values('이름1');
Query OK, 1 row affected (0.00 sec)

mysql> insert into users() values();
Query OK, 1 row affected (0.00 sec)

mysql> select * from users;
+----+---------------+
| id | name          |
+----+---------------+
|  1 | 이름1         |
|  2 | 이름 없음     |
+----+---------------+
2 rows in set (0.00 sec)
```

마지막으로 1:1 관계와 1:M 관계에 대해서 생각해 보겠습니다. user가 profile과 1:M 관계라면 profile에 user_id 필드를 두고, FK를 통해 제약을 줄 수 있습니다. 하지만 1:1 관계의 경우엔 user에 profile_id 필드를 두거나, profile에 user_id 필드를 두고 FK로 제약을 주는 것만으로는 부족합니다.

unique

```
mysql> create table orders(
    -> id int primary key auto_increment,
    -> user_id int not null,
    -> foreign key(user_id) references users(id) on delete cascade on update cascade
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> insert into profiles(user_id) values(1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into profiles(user_id) values(1);
Query OK, 1 row affected (0.00 sec)

mysql> select * from profiles;
+----+---------+
| id | user_id |
+----+---------+
|  1 |       1 |
|  2 |       1 |
+----+---------+
2 rows in set (0.00 sec)

# truncate는 테이블을 초기화하는 쿼리입니다.
mysql> truncate profiles;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from profiles;
Empty set (0.00 sec)

# profiles.user_id 컬럼에 unique 속성을 추가합니다.
mysql> alter table profiles modify user_id int not null unique;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> insert into profiles(user_id) values(1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into profiles(user_id) values(1);
ERROR 1062 (23000): Duplicate entry '1' for key 'user_id'
```

unique 속성은 위처럼 1:1 관계를 설정하거나, 회원의 로그인 아이디와 같이 테이블 내에서 고유해야 하는 필드에 설정 할 수 있는 속성입니다.

9. where, group by, having, order by,

limit

<subquery in select>

위에서 다뤘던 select 쿼리에 대해서 좀 더 자세히 알아보겠습니다. 응용프로그램에서는 일반적으로 복잡한 insert, update, delete 쿼리를 필요로하지 않습니다. 하지만 데이터를 조회하는 select 쿼리의 경우에는 응용프로그램의 요구에 따라 많게는 십여줄에 이를 수도 있습니다.

<todo DB>

select 쿼리를 공부하기 전에 다음 챕터에서 계속 이용 할 todo 데이터베이스를 생성합니다. todo DB는 회원(user)이 여러 목록(list)을 가질 수 있으며, 목록은 여러 항목(item)을 가질 수 있습니다. 이번에는 설치한 Workbench GUI 클라이언트 프로그램으로 DB를 모델링하고 더미 데이터를 생성해봅니다. 이 todo DB는 다음 챕터에서 웹 서버를 작성하면서 다시 이용하게 됩니다.

todo DB의 테이블 및 더미 데이터

```
mysql> describe users;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
```

```
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| uid      | varchar(45) | NO   | UNI | NULL    |                |
| password | varchar(45) | NO   |     | NULL    |                |
| name     | varchar(45) | NO   |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> select * from users;
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  1 | admin | 1234     | 관리자    |
|  2 | test  | 1234     | 손님      |
+----+-------+----------+-----------+
2 rows in set (0.00 sec)

mysql> describe lists;
+---------+-----------------------------------+------+-----+---------+------------
| Field   | Type                              | Null | Key | Default | Extra
+---------+-----------------------------------+------+-----+---------+------------
| id      | int(11)                           | NO   | PRI | NULL    | auto_increm
| name    | varchar(45)                       | NO   |     | NULL    |
| icon    | enum('work','other','shop','study') | NO   |     | NULL    |
| user_id | int(11)                           | NO   | MUL | NULL    |
+---------+-----------------------------------+------+-----+---------+------------
4 rows in set (0.00 sec)

mysql> select * from lists;
+----+--------------+-------+---------+
| id | name         | icon  | user_id |
+----+--------------+-------+---------+
|  1 | 공부하기     | study |       1 |
|  2 | 일하기       | work  |       1 |
|  3 | 빈리스트     | other |       1 |
|  4 | aa           | other |       1 |
+----+--------------+-------+---------+
4 rows in set (0.00 sec)

mysql> describe items;
+---------+-------------+------+-----+-------------------+----------------+
| Field   | Type        | Null | Key | Default           | Extra          |
+---------+-------------+------+-----+-------------------+----------------+
| id      | int(11)     | NO   | PRI | NULL              | auto_increment |
| name    | varchar(45) | NO   |     | NULL              |                |
| checked | tinyint(1)  | NO   |     | 0                 |                |
| date    | datetime    | NO   |     | CURRENT_TIMESTAMP |                |
| user_id | int(11)     | NO   | MUL | NULL              |                |
| list_id | int(11)     | NO   | MUL | NULL              |                |
+---------+-------------+------+-----+-------------------+----------------+
6 rows in set (0.00 sec)

mysql> select * from items;
+----+-----------------------------+---------+---------------------+---------+-----
| id | name                        | checked | date                | user_id | list
+----+-----------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기         |       0 | 2016-11-07 22:28:36 |       1 |
```

```
|  6 | Rx 공부하기                    |       1 | 2016-11-07 22:28:36 |       1 |
|  7 | Angular 2.0 공부하기           |       1 | 2016-11-07 22:28:36 |       1 |
| 10 | aa                             |       1 | 2016-11-09 23:20:24 |       1 |
| 11 | asdsd                          |       1 | 2016-11-09 23:20:45 |       1 |
| 12 | asdasd                         |       1 | 2016-11-09 23:22:19 |       1 |
| 13 | asd                            |       1 | 2016-11-09 23:22:19 |       1 |
| 38 | asda                           |       0 | 2016-11-14 22:38:48 |       1 |
| 42 | asasdasdasd                    |       1 | 2016-11-14 22:40:49 |       1 |
| 43 | fs                             |       0 | 2016-11-14 22:40:51 |       1 |
+----+--------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)
```

우선 select 뿐만 아니라 update, delete 쿼리에서도 이용 할 수 있는 where 구문에 대해 자세히 알아보겠습니다. where 구문은 boolean으로 평가되는 표현식을 받아 쿼리가 적용될 레코드를 필터링하는데 이용됩니다.

X = Y	같다
X != Y 또는 X <> Y	다르다
X > Y, <, >="," <="</th">	대소를 비교하는 연산자는 숫자 뿐만 아니라 문자열 및 날짜 타입에도 적용 가능
X is null	빈 값이다
X is not null	빈 값이 아니다
X between Y and Z	X >= Y and X <= Z
X not between Y and Z	X < Y or X > Z
X like Y	X(문자열)이 Y와 같다, 이 때 Y에서 %를 와일드카드로 이용 가능
X in (subquery or values)	X가 목록 내에 존재한다
X not in (subquery or values)	X가 목록 내에 존재하지 않는다
exists (subquery)	subquery의 결과 레코드가 존재한다
not exists (subquery)	subquery의 결과 레코드가 존재하지 않는다
not (expr) 또는 !(expr)	expr가 거짓이다
((expr1) and (expr2)) or not (expr3)	and (또는 &&), or (또는 ||), not (또는 !)을 조합해 복잡한 조건을 표현 가능

<MySQL 비교 연산자>

위의 연산자들을 통해 where 절에 다양한 조건을 적용 할 수 있습니다.

where

```
mysql> select * from users;
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  1 | admin | 1234     | 관리자      |
|  2 | test  | 1234     | 손님      |
+----+-------+----------+-----------+
2 rows in set (0.00 sec)


mysql> select count(*) as total from users;
+-------+
| total |
+-------+
|     2 |
+-------+
1 row in set (0.00 sec)


mysql> select * from users where id in (1,2) and name like '%리자';
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  1 | admin | 1234     | 관리자    |
+----+-------+----------+-----------+
1 row in set (0.00 sec)


# 리스트가 있는 유저만 불러오기
# subquery를 사용 할 때 필드명이 중첩되는 경우 필드명 앞에 테이블명을 표기해줍니다.

mysql> select * from users where exists (select * from lists where lists.user_id = u
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  1 | admin | 1234     | 관리자    |
+----+-------+----------+-----------+
1 row in set (0.00 sec)
```

데이터에 대한 통계적인 결과를 출력 할 때는 위의 count()와 같은 집계함수를 이용합니다. 집계함수는 레코드들을 그룹으로 묶고서 통계적인 연산을 수행합니다.

count(*)	null인 레코드를 포함한 갯수를 셈
count([distinct] expr)	expr가 null인 레코드를 제외한 갯수를 셈, distinct로 중복값을 하나로 처리할 수 있음
sum([distinct] expr)	expr의 합계

max(expr), min(expr)	expr의 최대값, 최소값

<MySQL 집계함수>

이외에도 다양한 통계함수가 있습니다.

aggregation

```
mysql> select * from users;
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  1 | admin | 1234     | 관리자    |
|  2 | test  | 1234     | 손님      |
+----+-------+----------+-----------+
2 rows in set (0.00 sec)

mysql> select count(*) from users;
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql> select sum(id) from users;
+---------+
| sum(id) |
+---------+
|       3 |
+---------+
1 row in set (0.00 sec)

# 이 때 집계함수와 일반 필드를 같이 select 하려면 오류가 납니다.
mysql> select sum(password), name from users;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #2 of SELECT li
```

위의 마지막 쿼리에서 sum(password), name을 같이 조회하려 했습니다만 오류가 발생했습니다. 집계함수를 이용하면, 기본적으로 전체 테이블을 하나의 그룹으로 취급하기 때문에 어떤 name을 조회해야 할지 명확하지 않기 때문입니다. 집계함수를 적용하기에 앞서서 전체 테이블을 group by를 이용해 여러개의 그룹으로 분리 할 수 있습니다. group by는 항상 select 쿼리, 집계함수와 함께 쓰입니다.

group by, having

```
mysql> select * from items;
```

```
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1 |
|  6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|  7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
| 10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| 11 | asdsd                        |       1 | 2016-11-09 23:20:45 |       1 |
| 12 | asdasd                       |       1 | 2016-11-09 23:22:19 |       1 |
| 13 | asd                          |       1 | 2016-11-09 23:22:19 |       1 |
| 38 | asda                         |       0 | 2016-11-14 22:38:48 |       1 |
| 42 | asasdasdasd                  |       1 | 2016-11-14 22:40:49 |       1 |
| 43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |
+----+------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)


# 항목이 완료되었는지(checked)를 그룹으로 집계
mysql> select count(*), checked from items group by checked;
+----------+---------+
| count(*) | checked |
+----------+---------+
|        3 |       0 |
|        7 |       1 |
+----------+---------+
2 rows in set (0.00 sec)


# where절은 group by 앞에 옵니다.
mysql> select count(*), checked, list_id from items where list_id=1 group by checked
+----------+---------+---------+
| count(*) | checked | list_id |
+----------+---------+---------+
|        1 |       0 |       1 |
|        6 |       1 |       1 |
+----------+---------+---------+
2 rows in set (0.00 sec)


# group by를 여러번 적용 할 수도 있습니다.
mysql> select count(*), checked, list_id from items group by checked, list_id;
+----------+---------+---------+
| count(*) | checked | list_id |
+----------+---------+---------+
|        1 |       0 |       1 |
|        2 |       0 |       2 |
|        6 |       1 |       1 |
|        1 |       1 |       2 |
+----------+---------+---------+
4 rows in set (0.00 sec)


# group by에 표현식을 적용 할 수도 있습니다.
mysql> select count(*), checked, list_id > 1 from items group by checked, list_id >
+----------+---------+-------------+
| count(*) | checked | list_id > 1 |
+----------+---------+-------------+
```

```
|        1 |       0 |           0 |
|        2 |       0 |           1 |
|        6 |       1 |           0 |
|        1 |       1 |           1 |
+----------+---------+-------------+
4 rows in set (0.01 sec)


# having절로 group by로 집계된 결과를 다시 필터링 할 수 있습니다.

mysql> select count(*) as cnt, checked from items group by checked;
+-----+---------+
| cnt | checked |
+-----+---------+
|   3 |       0 |
|   7 |       1 |
+-----+---------+
2 rows in set (0.00 sec)

mysql> select count(*) as cnt, checked from items group by checked having cnt > 5;
+-----+---------+
| cnt | checked |
+-----+---------+
|   7 |       1 |
+-----+---------+
1 row in set (0.00 sec)
```

조회시 조건을 통해 결과를 필터링하는 방법, 또 결과를 그룹으로 나누고 집계함수를 이용하는 방법을 알아봤습니다. 더 나아가 order by절로 결과를 정렬하고, limit절로 출력 갯수를 제한하는 방법을 알아보겠습니다.

order by, limit

```
# desc(descending): 내림차순
mysql> select * from users order by uid desc;
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  2 | test  | 1234     | 손님      |
|  1 | admin | 1234     | 관리자    |
+----+-------+----------+-----------+
2 rows in set (0.00 sec)

# asc(ascending): 오름차순
# order by를 순서대로 여러차례 적용할 수 있습니다.
mysql> select * from items order by checked asc, id desc;
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
| 43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |
```

```
| 38 | asda                          |       0 | 2016-11-14 22:38:48 |       1 |
|  5 | 안드로이드 공부하기           |       0 | 2016-11-07 22:28:36 |       1 |       1 |
| 42 | asasdasdasd                   |       1 | 2016-11-14 22:40:49 |       1 |
| 13 | asd                           |       1 | 2016-11-09 23:22:19 |       1 |
| 12 | asdasd                        |       1 | 2016-11-09 23:22:19 |       1 |
| 11 | asdsd                         |       1 | 2016-11-09 23:20:45 |       1 |
| 10 | aa                            |       1 | 2016-11-09 23:20:24 |       1 |
|  7 | Angular 2.0 공부하기          |       1 | 2016-11-07 22:28:36 |       1 |
|  6 | Rx 공부하기                   |       1 | 2016-11-07 22:28:36 |       1 |
+----+-------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)


# order by절에도 표현식을 적용 할 수 있습니다.
mysql> select id,checked,id*checked from items order by id*checked asc;
+----+---------+------------+
| id | checked | id*checked |
+----+---------+------------+
|  5 |       0 |          0 |
| 38 |       0 |          0 |
| 43 |       0 |          0 |
|  6 |       1 |          6 |
|  7 |       1 |          7 |
| 10 |       1 |         10 |
| 11 |       1 |         11 |
| 12 |       1 |         12 |
| 13 |       1 |         13 |
| 42 |       1 |         42 |
+----+---------+------------+
10 rows in set (0.00 sec)

# 난수를 발생시키는 rand() 함수로 랜덤으로 정렬 할 수 있습니다.
mysql> select * from items order by rand();
+----+-------------------------------+---------+---------------------+---------+-----
| id | name                          | checked | date                | user_id | list
+----+-------------------------------+---------+---------------------+---------+-----
| 38 | asda                          |       0 | 2016-11-14 22:38:48 |       1 |
|  6 | Rx 공부하기                   |       1 | 2016-11-07 22:28:36 |       1 |
| 13 | asd                           |       1 | 2016-11-09 23:22:19 |       1 |
| 42 | asasdasdasd                   |       1 | 2016-11-14 22:40:49 |       1 |
| 10 | aa                            |       1 | 2016-11-09 23:20:24 |       1 |
| 43 | fs                            |       0 | 2016-11-14 22:40:51 |       1 |
| 11 | asdsd                         |       1 | 2016-11-09 23:20:45 |       1 |
|  5 | 안드로이드 공부하기           |       0 | 2016-11-07 22:28:36 |       1 |       1 |
| 12 | asdasd                        |       1 | 2016-11-09 23:22:19 |       1 |
|  7 | Angular 2.0 공부하기          |       1 | 2016-11-07 22:28:36 |       1 |
+----+-------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)


# limit절은 select 쿼리의 가장 마지막에서 결과의 갯수를 제한합니다.
mysql> select * from items where checked order by id desc limit 5;
+----+-------------+---------+---------------------+---------+---------+
| id | name        | checked | date                | user_id | list_id |
+----+-------------+---------+---------------------+---------+---------+
| 42 | asasdasdasd |       1 | 2016-11-14 22:40:49 |       1 |       2 |
```

```
| 13 | asd          |       1 | 2016-11-09 23:22:19 |       1 |       1 |
| 12 | asdasd       |       1 | 2016-11-09 23:22:19 |       1 |       1 |
| 11 | asdsd        |       1 | 2016-11-09 23:20:45 |       1 |       1 |
| 10 | aa           |       1 | 2016-11-09 23:20:24 |       1 |       1 |
+----+--------------+---------+---------------------+---------+---------+
5 rows in set (0.00 sec)

# limit [offset, ] number에서 offset은 출력을 스킵 할 레코드의 갯수를 나타냅니다.
mysql> select * from items where checked order by id desc limit 0, 5;
+----+--------------+---------+---------------------+---------+---------+
| id | name         | checked | date                | user_id | list_id |
+----+--------------+---------+---------------------+---------+---------+
| 42 | asasdasdasd |       1 | 2016-11-14 22:40:49 |       1 |       2 |
| 13 | asd          |       1 | 2016-11-09 23:22:19 |       1 |       1 |
| 12 | asdasd       |       1 | 2016-11-09 23:22:19 |       1 |       1 |
| 11 | asdsd        |       1 | 2016-11-09 23:20:45 |       1 |       1 |
| 10 | aa           |       1 | 2016-11-09 23:20:24 |       1 |       1 |
+----+--------------+---------+---------------------+---------+---------+
5 rows in set (0.00 sec)

mysql> select * from items where checked order by id desc limit 2, 5;
+----+--------------------------+---------+---------------------+---------+---------
| id | name                     | checked | date                | user_id | list_id
+----+--------------------------+---------+---------------------+---------+---------
| 12 | asdasd                   |       1 | 2016-11-09 23:22:19 |       1 |       1
| 11 | asdsd                    |       1 | 2016-11-09 23:20:45 |       1 |       1
| 10 | aa                       |       1 | 2016-11-09 23:20:24 |       1 |       1
|  7 | Angular 2.0 공부하기     |       1 | 2016-11-07 22:28:36 |       1 |       1 |
|  6 | Rx 공부하기              |       1 | 2016-11-07 22:28:36 |       1 |       1 |
+----+--------------------------+---------+---------------------+---------+---------
5 rows in set (0.00 sec)
```

order by와 limit절을 이용해서 게시판 등의 프로그램에서 게시물들을 정렬하고, 페이지 번호 등을 기반으로 데이터의 출력을 제한 할 수 있습니다.

10. join, union

```
(5)  SELECT (6) DISTINCT
(1)  FROM
(2)  WHERE
(3)  GROUP BY
(4)  HAVING
(7)  ORDER BY
```

FIGURE I ANSI SQL logical query-processing order

<쿼리 처리 순서>

1. FROM, JOIN 목표 데이터셋을 정의
2. WHERE 조건에 따라 필터링
3. GROUP BY 결과를 그룹으로 나눔
4. HAVING 결과를 다시 조건에 따라 필터링
5. SELECT 결과에서 필드를 추출
6. ORDER BY 결과를 정렬
7. LIMIT 결과의 스킵, 출력 갯수를 제한

복잡한 쿼리의 구문들은 대략적으로 위의 순서로 실행됩니다. from과 함께 첫째로 처리되는 join절에 대해서 알아보겠습니다.

앞서 배웠듯이 RDB에서는 하나의 모델을 하나의 테이블로 독립적으로 표현하면서, 모델간의 관계는 FK를 통해 설정합니다. 때문에 잘 분리된 모델에서는 item과 그 주인인 user를 한번의 쿼리의 결과로 출력할 수 없습니다.

```
mysql> select * from items where id=5;
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1 |
+----+------------------------------+---------+---------------------+---------+-----
1 row in set (0.00 sec)

mysql> select * from users where id=1;
+----+-------+----------+-----------+
| id | uid   | password | name      |
+----+-------+----------+-----------+
|  1 | admin | 1234     | 관리자    |
+----+-------+----------+-----------+
1 row in set (0.00 sec)
```

join절은 위처럼 분리된 테이블을 조합하여 데이터를 조회하는 데 사용됩니다.

join

```
# ... from 테이블 join 테이블2 on 조건식으로 테이블을 조합합니다.
mysql> select * from users join items on users.id=items.user_id limit 5;
+----+-------+----------+-----------+----+-----------------------------+---------+-
| id | uid   | password | name      | id | name                        | checked |
+----+-------+----------+-----------+----+-----------------------------+---------+-
|  1 | admin | 1234     | 관리자    |  5 | 안드로이드 공부하기         |       0 | 2016-1
|  1 | admin | 1234     | 관리자    |  6 | Rx 공부하기                 |       1 | 2016
|  1 | admin | 1234     | 관리자    |  7 | Angular 2.0 공부하기        |       1 | 2016
|  1 | admin | 1234     | 관리자    | 10 | aa                          |       1 | 2
|  1 | admin | 1234     | 관리자    | 11 | asdsd                       |       1 | 2
+----+-------+----------+-----------+----+-----------------------------+---------+-
5 rows in set (0.00 sec)

# 꼭 FK 관계가 아니여도 join을 사용 할 수 있습니다.
# 이 예시는 유의미한 결과는 아닙니다.
mysql> select * from users join items on length(users.uid) < length(items.name) limi
+----+-------+----------+-----------+----+-----------------------------+---------+-
| id | uid   | password | name      | id | name                        | checked |
+----+-------+----------+-----------+----+-----------------------------+---------+-
|  1 | admin | 1234     | 관리자    |  5 | 안드로이드 공부하기         |       0 | 2016-1
|  2 | test  | 1234     | 손님      |  5 | 안드로이드 공부하기         |       0 | 2016-1
|  1 | admin | 1234     | 관리자    |  6 | Rx 공부하기                 |       1 | 2016
|  2 | test  | 1234     | 손님      |  6 | Rx 공부하기                 |       1 | 201
|  1 | admin | 1234     | 관리자    |  7 | Angular 2.0 공부하기        |       1 | 2016
+----+-------+----------+-----------+----+-----------------------------+---------+-
5 rows in set (0.01 sec)

# join을 여러 테이블에 걸쳐 적용 할 수 있습니다.
# 중첩되는 필드명이 있을 땐 as를 이용해서 구분하여 출력합니다.
mysql> select users.name user_name, lists.name as list_name, items.name as item_name
+-----------+--------------+-----------------------------+---------+
| user_name | list_name    | item_name                   | checked |
+-----------+--------------+-----------------------------+---------+
| 관리자    | 공부하기     | 안드로이드 공부하기         |       0 |
| 관리자    | 공부하기     | Rx 공부하기                 |       1 |
| 관리자    | 공부하기     | Angular 2.0 공부하기        |       1 |
| 관리자    | 공부하기     | aa                          |       1 |
| 관리자    | 공부하기     | asdsd                       |       1 |
+-----------+--------------+-----------------------------+---------+
5 rows in set (0.00 sec)
```

join절이야 말로 NoSQL과 구분되는 RDB의 핵심적인 요소라고 할 수 있습니다. 위처럼 독립적인 테이블들을 조합하여 응용프로그램에서 원하는 데이터를 추출하는 쿼리를 작성할 수 있는 능력이 필요합니다.

위처럼 join절은 쿼리를 처리하기에 앞서서 목표 데이터셋을 정의하는 역할을 합니다. 이때 각각의 테이블을 하나의 집합으로 보면 아래 그림처럼 다양한 join이 가능합니다.

<join의 종류>

cross join

```
CROSS JOIN ...,
cartesian product ...,
on의 의미 필터링,
각종 퀴즈들 영상 및 교재에 추가
```

from A [inner] join B on X는 조건 X를 만족하는 데이터쌍만을 조합하는 inner 조인을 실행합니다.

inner join

```
# items 테이블의 list_id를 nullable하게 수정합니다.
# 그리고 몇개의 더미 데이터의 list_id를 null로 수정했습니다.

mysql> alter table items modify column list_id int null;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> update items set list_id = null where id > 10;
ERROR 1048 (23000): Column 'list_id' cannot be null
mysql> alter table items modify column list_id int null;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from items;
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1 |
|  6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|  7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
| 10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| 11 | asdsd                        |       1 | 2016-11-09 23:20:45 |       1 |    N
| 12 | asdasd                       |       1 | 2016-11-09 23:22:19 |       1 |    N
| 13 | asd                          |       1 | 2016-11-09 23:22:19 |       1 |    N
| 38 | asda                         |       0 | 2016-11-14 22:38:48 |       1 |    N
| 42 | asasdasdasd                  |       1 | 2016-11-14 22:40:49 |       1 |    N
| 43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |    N
+----+------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)


# 이제 join을 실행한 결과 items.list_id가 null인 데이터쌍은 추출되지 않았습니다.
mysql> select * from items join lists on items.list_id=lists.id;
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1 |
|  6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|  7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
| 10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
+----+------------------------------+---------+---------------------+---------+-----
4 rows in set (0.00 sec)
```

다음으로 from A left join B on X는 A의 데이터 전체를 추출하면서 X를 만족시키는 B를 조합합니다.

left join

```
# X를 만족시키는 쌍이 없는 A의 레코드는 B의 필드를 모두 null로 갖는 결과를 냅니다.
mysql> select * from items left join lists on items.list_id=lists.id;
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1 |
|  6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|  7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
| 10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| 11 | asdsd                        |       1 | 2016-11-09 23:20:45 |       1 |    N
| 12 | asdasd                       |       1 | 2016-11-09 23:22:19 |       1 |    N
| 13 | asd                          |       1 | 2016-11-09 23:22:19 |       1 |    N
| 38 | asda                         |       0 | 2016-11-14 22:38:48 |       1 |    N
| 42 | asasdasdasd                  |       1 | 2016-11-14 22:40:49 |       1 |    N
| 43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |    N
+----+------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)
```

inner join과 left join으로 대부분의 일반적인 로직을 수행 할 수 있습니다. right join은 left join을 뒤집은 구문에 불과하며, full join은 left join과 right join의 합에 지나지 않습니다.

right join, full join

```
mysql> select * from items left join lists on items.list_id=lists.id;
+----+------------------------------+---------+---------------------+---------+-----
| id | name                         | checked | date                | user_id | list
+----+------------------------------+---------+---------------------+---------+-----
|  5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1 |
|  6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|  7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
| 10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| 11 | asdsd                        |       1 | 2016-11-09 23:20:45 |       1 |    N
| 12 | asdasd                       |       1 | 2016-11-09 23:22:19 |       1 |    N
| 13 | asd                          |       1 | 2016-11-09 23:22:19 |       1 |    N
| 38 | asda                         |       0 | 2016-11-14 22:38:48 |       1 |    N
| 42 | asasdasdasd                  |       1 | 2016-11-14 22:40:49 |       1 |    N
| 43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |    N
+----+------------------------------+---------+---------------------+---------+-----
10 rows in set (0.00 sec)

mysql> select * from items right join lists on items.list_id=lists.id;
+------+------------------------------+---------+---------------------+---------+---
| id   | name                         | checked | date                | user_id | li
+------+------------------------------+---------+---------------------+---------+---
|    5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1
|    6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|    7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
|   10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| NULL | NULL                         |    NULL | NULL                |    NULL |
| NULL | NULL                         |    NULL | NULL                |    NULL |
| NULL | NULL                         |    NULL | NULL                |    NULL |
+------+------------------------------+---------+---------------------+---------+---
7 rows in set (0.00 sec)

## MySQL은 full join을 지원하지 않습니다.
mysql> select * from items full join lists on items.list_id=lists.id;
ERROR 1054 (42S22): Unknown column 'items.list_id' in 'on clause'
```

마지막으로 MySQL에서 지원하지 않는 full join을 union 연산을 통해 구현해보겠습니다. union 연산은 여러 select 쿼리의 결과를 하나로 합치는 데 이용됩니다. 이 때 union 할 결과들은 모두 똑같은 필드들을 가져야 합니다.

union [distinct], union all

```
mysql> (select * from items right join lists on items.list_id=lists.id) union (selec
+------+------------------------------+---------+---------------------+---------+---
| id   | name                         | checked | date                | user_id | li
+------+------------------------------+---------+---------------------+---------+---
|    5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1
|    6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|    7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
|   10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| NULL | NULL                         |    NULL | NULL                |    NULL |
| NULL | NULL                         |    NULL | NULL                |    NULL |
| NULL | NULL                         |    NULL | NULL                |    NULL |
|   11 | asdsd                        |       1 | 2016-11-09 23:20:45 |       1 |
|   12 | asdasd                       |       1 | 2016-11-09 23:22:19 |       1 |
|   13 | asd                          |       1 | 2016-11-09 23:22:19 |       1 |
|   38 | asda                         |       0 | 2016-11-14 22:38:48 |       1 |
|   42 | asasdasdasd                  |       1 | 2016-11-14 22:40:49 |       1 |
|   43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |
+------+------------------------------+---------+---------------------+---------+---
13 rows in set (0.00 sec)


# union 연산은 union dinstinct 연산으로 처리되며, 이는 결과 중에서 중복되는 레코드를 제거해줍니다.
# 중복되는 레코드를 제거할 필요가 없거나, 비용을 줄이려면 union all 연산을 이용합니다.

mysql> (select * from items right join lists on items.list_id=lists.id) union all (s
+------+------------------------------+---------+---------------------+---------+---
| id   | name                         | checked | date                | user_id | li
+------+------------------------------+---------+---------------------+---------+---
|    5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1
|    6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|    7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
|   10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
| NULL | NULL                         |    NULL | NULL                |    NULL |
| NULL | NULL                         |    NULL | NULL                |    NULL |
| NULL | NULL                         |    NULL | NULL                |    NULL |
|    5 | 안드로이드 공부하기          |       0 | 2016-11-07 22:28:36 |       1 |       1
|    6 | Rx 공부하기                  |       1 | 2016-11-07 22:28:36 |       1 |
|    7 | Angular 2.0 공부하기         |       1 | 2016-11-07 22:28:36 |       1 |
|   10 | aa                           |       1 | 2016-11-09 23:20:24 |       1 |
|   11 | asdsd                        |       1 | 2016-11-09 23:20:45 |       1 |
|   12 | asdasd                       |       1 | 2016-11-09 23:22:19 |       1 |
|   13 | asd                          |       1 | 2016-11-09 23:22:19 |       1 |
|   38 | asda                         |       0 | 2016-11-14 22:38:48 |       1 |
|   42 | asasdasdasd                  |       1 | 2016-11-14 22:40:49 |       1 |
|   43 | fs                           |       0 | 2016-11-14 22:40:51 |       1 |
+------+------------------------------+---------+---------------------+---------+---
17 rows in set (0.00 sec)
```

MySQL 함수 목록 (http://dev.mysql.com/doc/refman/5.7/en/functions.html)

W3School SQL 튜토리얼 (http://www.w3schools.com/sql/default.asp)

11. 인덱스, 뷰, 프로시저, 트리거

DBMS에서 제공하는 추가적인 기능에 대해서 알아봅니다. 인덱스에 대한 개념을 정리하고 뷰, 프로시저, 트리거에 대해서 간단히 소개하겠습니다.

<인덱스>

특정 레코드를 조작하는 query를 실행하면 테이블의 첫째 열에서부터 전체 테이블을 순차적으로 검색하기 때문에 테이블이 클 수록 검색 비용이 높아집니다. 이 때 백과사전의 색인처럼 검색 속도를 높히기 위해서 테이블의 특정 컬럼만으로 구성된 보조 테이블을 인덱스(Index)라고 합니다.

primary key	PK로 설정된 컬럼은 자동으로 primary key 인덱스를 생성
unique	unique로 설정된 컬럼은 자동으로 unqiue 인덱스를 생성
index	일반적인 필드에 대한 인덱스, FK로 설정되면 자동으로 index 인덱스를 생성
fulltext	TEXT, VARCHAR, CHAR 등 대용량 텍스트에 특화된 인덱스

<MySQL의 인덱스 종류>

자주 사용되는 쿼리의 비용을 절감할 수 있도록 적절한 인덱스를 생성하면, 대규모의 데이터를 관리할 때 최적화를 꾀 할 수 있습니다. 참고로 인덱스를 생성하면 select 쿼리의 비용을 줄어들지만, insert, update 쿼리의 비용은 색인을 생성하는 과정 때문에 늘어날 수 있습니다.

Index

```
mysql> create table index_test(
    -> id int primary key auto_increment,
    -> some_value int(128),
    -> some_value2 text,
    -> index(some_value),
    -> fulltext(some_value2)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> show index from index_test;
+------------+------------+-------------+--------------+-------------+-----------+--
| Table      | Non_unique | Key_name    | Seq_in_index | Column_name | Collation | (
+------------+------------+-------------+--------------+-------------+-----------+--
| index_test |          0 | PRIMARY     |            1 | id          | A         |
| index_test |          1 | some_value  |            1 | some_value  | A         |
| index_test |          1 | some_value2 |            1 | some_value2 | NULL      |
+------------+------------+-------------+--------------+-------------+-----------+--
3 rows in set (0.00 sec)
```

뷰(View)는 일종의 가상의 테이블을 생성하는 것입니다. join 등으로 복잡해진 쿼리를 단순화 시키고, 응용프로그램에서 이용 할 데이터셋을 추상화하며, 실제 테이블은 은닉하여 보안을 꾀할 수 있는 장점이 있습니다.

View

```
mysql> select users.id user_id, users.name user_name, lists.id list_id, lists.name l
+---------+-----------+---------+--------------+---------+-------------------------
| user_id | user_name | list_id | list_name    | item_id | item_name
+---------+-----------+---------+--------------+---------+-------------------------
|       1 | 관리자    |       1 | 공부하기     |       5 | 안드로이드 공부하기          |
|       1 | 관리자    |       1 | 공부하기     |       6 | Rx 공부하기                  |
|       1 | 관리자    |       1 | 공부하기     |       7 | Angular 2.0 공부하기         |
|       1 | 관리자    |       1 | 공부하기     |      10 | aa
+---------+-----------+---------+--------------+---------+-------------------------
4 rows in set (0.00 sec)

mysql> create view user_list_items as select users.id user_id, users.name user_name,
Query OK, 0 rows affected (0.01 sec)

mysql> select * from user_list_items where item_id > 5;;
+---------+-----------+---------+--------------+---------+-------------------------
| user_id | user_name | list_id | list_name    | item_id | item_name
+---------+-----------+---------+--------------+---------+-------------------------
|       1 | 관리자    |       1 | 공부하기     |       6 | Rx 공부하기              |
|       1 | 관리자    |       1 | 공부하기     |       7 | Angular 2.0 공부하기     |
|       1 | 관리자    |       1 | 공부하기     |      10 | aa                        |
+---------+-----------+---------+--------------+---------+-------------------------
3 rows in set (0.01 sec)
```

DBMS에 따라서 가상 테이블인 view에 인덱스를 생성하거나, insert, update, delete가 가능하지만 여러 제약사항이 따릅니다. 기본적으로 view는 select 쿼리의 편의를 위한 가상 테이블로 이해 할 수 있겠습니다.

프로시저(Procedure)

프로시저는 DB에 정의하는 일종의 함수입니다. 프로시저는 일반적인 프로그래밍 언어의 함수처럼 인자를 받을 수 있으며, 실행시 일련의 query를 실행합니다.

트리거(Trigger)

트리거는 DB에 정의하는 이벤트 핸들러(함수)입니다. 트리거는 insert, update, delete 쿼리의 실행 전이나 후에 일련의 query를 실행합니다.

[1] DDL, Data Definition Language

[2] DML, Data Manipulation Language

[3] DCL, Data Control Language

[4] CRUD, Create, Read, Update, Delete

[5] Domain Integrity Constraint

[7] View

[8] Procedure

[9] Trigger

5 데이터베이스

5.4 Connector, SQL Injection, ORM

1. Connector
2. SQL Injection
3. ORM

> 응용프로그램에서 DBMS 서버와 통신하는 방법에 대해서 알아보고, SQL Injection이라는 해킹 기법, RDBMS를 객체처럼 다루는 ORM에 대해 알아봅니다.

1. Connector

내장형 DBMS를 제외한 DBMS는 일반적으로 서버-클라이언트 구조로 작성되어있습니다. 따라서 DBMS는 CLI, GUI 클라이언트 프로그램과 더불어 응용프로그램을 개발 할 때 사용 할 수 있는 커넥터 모듈(또는 라이브러리)을 다양한 플랫폼에 맞게 제공합니다. Node.js에서 MySQL을 사용하기 위해 npm으로 Node.js 플랫폼용 MySQL 커넥터 (https://github.com/mysqljs/mysql)를 설치합니다.

```
npm install mysql
```

먼저 DBMS 서버가 작동 중인 호스트, 포트로 TCP 연결을 맺어주고 로그인(익명 DB가 아니라, 권한이 필요한 DB의 경우) 과정을 거치게 됩니다. 이때 DB 작업을 수행 할 때마다 연결을 맺고 작업 후 바로 끊을 수도 있지만, 백엔드 프로세스가 종료되기 전까지 연결을 지속적으로 유지하는 방안도 있습니다. 이때는 OS나 DBMS 서버가 유휴 연결(특정 시간 동안 통신이 없는)을 강제로 종료 할 수도 있습니다.

연결 및 쿼리 실행

```
const mysql = require('mysql');
let connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '1234',
    database: 'db_name'
});

connection.connect();
connection.query('select * from users', (err, rows, fields) => {
  connection.end();

  if (err) throw err;
  console.log(rows);
});
```

2. SQL Injection

DBMS를 이용한 유저 로그인 과정을 생각해봅니다. 아래처럼 요청으로 들어온 유저의 아이디와 패스워드를 DBMS의 users 테이블과 비교하는 로직이 있을 수 있습니다.

```
db.query(`select * from users where uid like '${req.body.uid}' and password like '${
    // ...
});
```

위의 코드도 일반적인 요청의 경우에는 별 문제가 없습니다만, 악의적인 입력 값을 넣으면 아래 같이 관리자 권한을 탈취 할 수 있습니다.

```
// req.body.uid = `admin' or `;
// req.body.password = ` like '`;
select * from users where uid like 'admin' or ' and password like ' like '';
```

혹은 여러줄의 쿼리가 실행된다면, 아래처럼 테이블 자체를 드랍시킬 수도 있고,

```
// req.body.uid = `uid'; drop table users;`;
// req.body.password = `xxxx`;
select * from users where uid like 'uid'; drop table users;' and password like 'xxx
```

단순히 서버에 런타임 오류를 발생시킬 수도 있습니다.

```
// req.body.uid = `''xxxx;`;
// req.body.password = `xxxx`;
select * from users where uid like ''xxxx;' and password like 'xxxx';
```

<정부 DB를 공격?!>

물론 공격자는 백엔드 코드를 볼 수 없기에 반복적인 공격을 통해서 위처럼 SQL Injection을 시도하겠지만, 성공하는 경우 위처럼 서버에 큰 타격을 줄 수 있겠습니다. 특히 내부 코드나 DB 스키마가 유출되면 큰 위협이 되겠습니다.

따라서 이를 원천적으로 방지 할 필요가 있습니다. 이를 방지하는 방법은 쿼리 포맷에 조합될 수 있는 문자열(입력 값)들에서 특수문자를 검증하고 치환하는 것으로 단순합니다. 물론 DBMS는 일반적으로 Connector단에서 이러한 기능을 간편히 제공합니다. Node.js MySQL Connector는 입력 값과 쿼리를 조합 할 때 아래처럼 코드를 작성 할 수 있습니다.

```
db.query(`select * from users where uid like ? and password like ?`, [req.body.uid,
    // ...
});
```

3. ORM

우리가 DBMS를 쓰기전에, 메모리 상에서 구현했던 데이터 모델들을 생각해봅니다. 각 모델들을 객체 지향적으로 설계해서 우아하고 구조적인 코드를 작성 할 수 있었습니다.

<ORM>

ORM(Object Relational Mapping)은 코드상에서 DB의 테이블들을 타입(Class) 내지는 객체로 다룰 수 있도록, DBMS와의 통신 과정을 내포한 데이터 모델을 생성하는 모듈을 의미합니다. 이때 Table은 하나의 타입(Class)에 맵핑되고, Row는 그 타입의 인스턴스로 맵핑된다고 볼 수 있습니다.

ORM을 이용하면 데이터 처리 로직 및 DBMS와의 통신 과정을 은닉, 추상화 할 수 있기 때문에 구조적인 코드를 작성하는데 필수적입니다. 물론 시스템에서 다루는 데이터의 스키마가 복잡하지 않다면 굳이 DB 작업에 부가적인 비용을 초래하는 ORM을 이용 할 필요는 없습니다. 또한 ORM을 쓰더라도 복잡한 쿼리(집계, JOIN 등)를 수행 할 때는 DBMS에 직접 쿼리를 전송하는 코드를 혼용 할 수도 있겠습니다.

ORM은 DBMS 제조사에서 지원하기 보다는 별개의 모듈로 작성되는 것이 일반적입니다. 또한 대부분의 웹 프레임워크들은 ORM 모듈을 내장하고 있습니다. 이번 챕터에서는 특정 ORM을 선택해서 공부하지 않고, ORM을 이용한 코드의 예시와 함께 Node.js의 ORM 모듈을 몇가지 소개하는 것으로 마무리합니다.

node-orm2

```
var orm = require("orm");

orm.connect("mysql://username:password@host/database", function (err, db) {
  if (err) throw err;

  // Table 및 모델을 생성
    var Person = db.define("person", {
        name      : String, // 필드와 타입을 정의, PK는 자동으로 생성
        surname   : String,
        age       : Number, // FLOAT
        male      : Boolean,
        continent : [ "Europe", "America", "Asia", "Africa", "Australia", "Antartica
        photo     : Buffer, // BLOB/BINARY
        data      : Object // JSON encoded
    }, {
        // 인스턴스 메소드
        methods: {
            fullName: function () {
                return this.name + ' ' + this.surname;
            }
        },
        // 필드 값 검증 로직
        validations: {
            age: orm.enforce.ranges.number(18, undefined, "under-age")
        }
    });

  // 메모리상 Model과 DB Table을 동기화
    db.sync(function(err) {
        if (err) throw err;

        // Row를 추가
        Person.create({ id: 1, name: "John", surname: "Doe", age: 27 }, function(err
            if (err) throw err;

            // select 쿼리를 통해서 Row를 검색
            // SQL: "SELECT * FROM person WHERE surname = 'Doe'"
            Person.find({ surname: "Doe" }, function (err, people) {
                if (err) throw err;

                console.log("People found: %d", people.length);
                console.log("First person: %s, age %d", people[0].fullName(), people

                people[0].age = 16;
                people[0].save(function (err) { // UPDATE 쿼리가 수행됨
                    // err.msg = "under-age";
                });
            });
        });
    });
});
```

node-orm2 (https://github.com/dresende/node-orm2/): MySQL & MariaDB, PostgreSQL, Amazon Redshift, SQLite, MongoDB를 지원하며 복잡하지 않은 인터페이스를 지닌 가벼운 ORM

Node.js 생태계가 생긴지도 많은 시간이 지났는데, NPM에서 ORM 패키지를 검색해보면, 아직 지배적이라고 추천드릴 만한 모듈이 없는 것 같습니다. 선택지가 많을 수록 고생을 하기 마련이라 생각하여, 두개의 패키지만 더 소개해봅니다.

Sequelize

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize('database', 'username', 'password');

const User = sequelize.define('user', {
  username: Sequelize.STRING,
  birthday: Sequelize.DATE
});

sequelize.sync()
  .then(() => User.create({
    username: 'janedoe',
    birthday: new Date(1980, 6, 20)
  }))
  .then(jane => {
    console.log(jane.get({
      plain: true
    }));
  });
```

Sequelize.js (http://docs.sequelizejs.com/): PostgreSQL, MySQL, MariaDB, SQLite, MS-SQL를 지원하며 GitHub에서 최고의 활성도를 보이고 있는 오픈소스, 기능 많음

Waterline

```
module.exports = {
  attributes: {
    // e.g., "Polly"
    name: {
      type: 'string'
    },

    // e.g., 3.26
    wingspan: {
      type: 'float',
      required: true
    },

    // e.g., "cm"
    wingspanUnits: {
      type: 'string',
      enum: ['cm', 'in', 'm', 'mm'],
      defaultsTo: 'cm'
    },

    // e.g., [{...}, {...}, ...]
    knownDialects: {
      collection: 'Dialect'
    }
  }
};
```

Waterline (http://waterlinejs.org/): Node.js에서 높은 인기를 가진 Sails.js (http://sailsjs.com/documentation/reference/waterline-orm) 웹 프레임워크에서 채택한 ORM, 문서 완성도 높음, 확장성 높음

다양한 패키지들의 코드의 짜임새나 기능을 보시고, 본인의 목적과 필요, 코드 스타일에 적합한 패키지를 선택하시면 좋겠습니다. 모든 것들이 그렇지만 라이브러리나 프레임워크는 필수가 아니라 선택입니다.

[1] SQL Injection
[2] ORM, Object Relational Mapping

6 개발과 배포

1. 패키지 매니저, 자동화 도구 ... 389

2. 버전 관리, Git, GitHub ... 406

3. 호스팅, SSH, FTP ... 432

4. DNS, 도메인, 메일 서버 (ing) ... 449

5. 암호화, 전자서명, 인증서와 SSL ... 451

6. 비밀번호 해싱 ... 470

개발에 수반되는 반복적인 작업들을 자동화하는 방법을 배웁니다. 버전 관리 시스템으로 코드를 안전하게 관리하며 협업 하는 방법을 배웁니다. 호스팅이나 클라우드를 통해 서버를 임대하고, 원격 서버에 FTP, SSH 프로토콜을 통해 접속해 서비스를 온라인에 배포합니다. 도메인 네임 서버(DNS)와 메일 서버에 대해 알아보고 SSL, 인증서, 암호화에 대해 공부합니다.

6 개발과 배포

6.1 패키지 매니저, 자동화 도구

1. 패키지 매니저
2. 자동화 도구
3. 트랜스파일러
4. 태스크 러너
5. JavaScript 모듈화
6. 모듈 번들러

> 패키지 매니저, 자동화 도구(Transpiler, Task Runner, Module Bundler) 등 생산성을 높히는 도구들에 대해서 배웁니다.

1. 패키지 매니저

<NPM (Node Package Manager)>

NPM을 통해 Node.js의 모듈이나 JavaScript 라이브러리를 설치 할 수 있었습니다. 이처럼 패키지(하나의 완성된 소프트웨어나, 부품으로 쓰이는 라이브러리 및 모듈)들의 설치, 업데이트 및 패키지간의 의존성과 버전 정보를 관리해주는 프로그램을 패키지 매니저(Package Manager)라고 합니다.

분류	예시
엔드유저용 프로그램 배포	Apple App Store, Google Play Store, MS Windows Apps, ...
OS별 프로그램 및 라이브러리 배포	cygwin/Windows, yum,apt/Linux, brew/macOS, ...
플랫폼별 라이브러리 및 모듈 배포 및 패키징	npm/Node.js, pip/Python, composer/PHP, gem/Ruby, nuget/.NET, pod/iOS,macOS, gradle,maven/Java,Android ...

<패키지 매니저>

패키지 매니저에는 엔드유저가 사용 할 실행 가능한 프로그램을 배포하는 프로그램도 있고, yum, apt, brew처럼 셸에서 OS별로 쓰이는 여러 프로그램 및 각종 라이브러리의 설치를 돕는 CLI 프로그램도 있으며, 특정 플랫폼의 라이브러리 및 모듈만을 모아둔 CLI 프로그램들도 있습니다.

<Package Manger>

이중 소프트웨어 개발에 있어서 자주 쓰게될 패키지 매니저는 플랫폼별로 제공되는 패키지 매니저가 되겠습니다. Node.js의 NPM을 생각해보시면 되겠습니다.

패키지 매니저는 개발중인 패키지의 의존성 정보를 기반으로(Node.js의 경우 package.json), 원격 저장소(Repository)로부터 의존 패키지들을 다운로드 및 업데이트 할 수 있도록 합니다. 이 때문에 프로그램을 배포 할 때, 의존하고 있는 패키지들을 모두 소스코드에 포함 할 필요 없이 그 메타 정보만을 포함(ex; package.json에)하는 것으로 충분합니다. 패키지 매니저를 사용하는 가장 큰 목적은, 이처럼 **의존 라이브러리들의 설치 및 관리의 편리함**에 있다고 볼 수 있겠습니다.

낯선 플랫폼에서 개발을 시작 할 때, 우선적으로 그 플랫폼의 공식적인 또는 우세한 패키지 매니저가 있는지 알아보고, 개발에 도움이 되는 라이브러리들을 검색해보면 생산성을 높힐 수 있겠습니다.

2. 자동화 도구

<웹 프론트엔드의 각종 도구들>

JavaScript 생태계와 웹의 UI/UX가 발전하면서, 웹 서비스에서 프론트엔드의 비중이 비약적으로 높아졌습니다. 애초에 JavaScript는 전문적인 IDE 없이 HTML/CSS에 곁다리 느낌으로 쓰이던 스크립트인지라, JS로 복잡한 시스템을 만드는 것을 돕기 위해 많은 도구들이 함께 등장했습니다.

프론트엔드에 큰 관심이 있다면, 위 도구들에 대해서 조금씩 공부해 보는 것도 나쁘지 않겠습니다. 아래에서는 주요한 자동화 도구들의 개념에 대해서만 소개합니다.

3. 트랜스파일러

프로그래밍 언어로 작성된 소스코드를 실행 가능한 바이너리로 번역해주는 프로그램을 컴파일러(Compiler)라고 했습니다. 트랜스파일러(Transpiler; Transcompiler; Source-to-Source Compiler)는 특정 언어로 작성된 소스코드를 다른 언어의 소스코드로 번역해주는, 즉 텍스트를 텍스트로 번역하는 프로그램입니다.

Source Language	Target Language	Transpiler
C++	C	CFRONT
C#	JavaScript	SCRIPTSHARP
PHP	C++	HIPHOP FOR PHP
COBOL	C	Open COBOL

<Transpiler>

위처럼 아예 서로 다른 플랫폼간의 트랜스파일을 구현하는 경우도 있습니다만, 아래에서는 웹 생태계에서 쓰이는 트랜스파일러에 대해서 알아보겠습니다.

CSS Transpiler

프로젝트가 복잡해지면 CSS 역시 복잡해지기 마련입니다. CSS가 변수나 함수(mixin), 또는 nesting 같은 기능을 제공하지 않기 때문에, CSS 작성의 생산성을 높히기 위해서 LESS (http://lesscss.org/), SASS (http://sass-lang.com/) 등의 CSS Transpiler들이 등장했습니다.

style.less

```
/* nesting */
div {
    color: black;

    p {
        font-weight: bold;

        &.danger {
            color: red;
        }
    }
}

/* minxin */
.rounded (@radius : 10px) {
  border-radius: @radius;
  -moz-border-radius: @radius;
  -webkit-border-radius: @radius;
}
#ball { .rounded(50%); }
#footer { .rounded; }
```

전용 트랜스파일러로 LESS로부터 CSS를 생성

```
lessc style.less style.css
```

style.css

```
/* nesting */
div { color: black; }
div p { font-weight: bold; }
div p.danger { color: red; }

/* minxin */
#ball {
  border-radius: 50%;
  -moz-border-radius: 50%;
  -webkit-border-radius: 50%;
}
#footer {
  border-radius: 10px;
  -moz-border-radius: 10px;
  -webkit-border-radius: 10px;
}
```

CSS 트랜스파일러는 위처럼 CSS 작성을 돕기 위해 등장했습니다. 최근에는 PostCSS (http://postcss.org/)라고 불리는 트랜스파일러가 상승세에 있습니다. PostCSS를 이용하면 변수, 함수(mixin), nesting, 테마, grid 및 모듈화, 에러 검증 등 수 많은 기능들 뿐만 아니라 차기 CSS의 문법(

W3C; World Wide Web Consortium에서 준비 중인)을 이용 할 수 있습니다.

PostCSS 예시

```
:root {
  --red: #d33;
}
a {
  &:hover {
    color: color(var(--red) a(54%));
  }
}
```

웹 프론트엔드의 생태계는 워낙 거미줄처럼 서로 엮여 있어서, 배보다 배꼽이 커지는 경우가 많습니다. 위의 PostCSS, LESS, SASS, SCSS, .. 무엇이 되었든 CSS3에 충분히 통달 한 후, 생산성 향상이 절실할 때 시작해보시길 바랍니다.

JavaScript Transpiler

이번엔 JavaScript의 문제점에 대해서 생각해보겠습니다. JS의 가장 큰 골칫거리는 호환성이 되겠습니다. 호환성은 CSS3나 HTML5에도 적용되는 문제지만, 특히 JS의 경우에는 호환성으로 인한 오류가 전체 코드를 종료시키는 문제를 발생시킬 수 있습니다.

<ECMAScript 호환성 표>

이런 와중에 Chrome이나 Firefox, Safari 같은 <u>에버그린 웹 브라우저는 ECMAScript 표준을 바로바로 적용시키면서, 다른 브라우저보다 앞서서 최신 문법들을 지원</u>하기도 합니다. 이런 문제는 시간이 많이 흘러, 구형 웹 브라우저들이 사장되면 자연히 해결되겠지만, <u>개발자 입장에서는 당장 최신 문법을 쓰고 싶어도 호환성 문제가 걱정</u>되기 마련입니다.

<Babel>

이러한 문제를 해결해주는 Babel (https://babeljs.io/)은 최신 문법의 JS를 구형 브라우저에서 실행 가능하도록 번역해주는 트랜스파일러입니다. 물론 웹 브라우저 뿐만 아니라 Node.js 용도로도 이용 할 수 있습니다.

New JS

```
[1, 2, 3].map(n => n ** 2);

let [a,,b] = [1,2,3];

const x = [3, 4, 5];
var y = [1, 2, ...x];

let name = "Guy Fieri";
let place = "Flavortown";
let hi = `Hello ${name}, ready for ${place}?`;
```

Old JS

```
[1, 2, 3].map(function(n) {
    return Math.pow(n, 2);
});

var _ref = [1,2,3];
var a = _ref[0];
var b = _ref[2];

var x = [3, 4, 5];
var y = [1, 2].concat(x);

var name = "Guy Fieri";
var place = "Flavortown";
var hi = 'Hello ' + name + ', ready for ' + place + '?';
```

<TypeScript>

다음으로, JavaScript가 Dynamically Typed Language라는 점은 생산성을 높히는 장점이 될 수도 있지만, 큰 시스템을 작성할 때는 단점으로 작용 할 수 있습니다. JavaScript의 쓰임새가 최초의 단촐한 용도를 넘어서, SPA처럼 거대한 시스템을 작성하는데 확장되면서 동적인 타이핑이 시스템의 안정성이나 개발을 취약하게 하는 경우가 생겼습니다.

TypeScript

```
let obj: string;
obj = 'yo';
// Error: Type 'number' is not assignable to type 'string'.
obj = 10;

// types can be inferred (return type)
function sayIt(what: string) {
    return `Saying: ${what}`;
}
const said: string = sayIt(obj);

class Sayer {
    // mandatory
    what: string;

    constructor(what: string) {
        this.what = what;
    }

    // return type if you want to
    sayIt(): string {
        return `Saying: ${this.what}`;
    }
}
```

이를 해결하기 위해서 등장한 TypeScript (https://www.typescriptlang.org/)는 JS의 최신 문법 및 정적 타입을 지원하는 TypeScript라는 언어를 JS로 번역해주는 트랜스파일러입니다.

MS에서 개발한 TypeScript는 Angular.js, React.js 등 유명 SPA 프레임워크에서 지원하는 등 높은 인기를 보이고 있습니다. 또한 최근에는 Facebook에서 개발한 Flow (https://flow.org/)라는 비슷한 정적 타입 언어가 상승세를 보이기도 합니다.

4. 태스크 러너

태스크 러너(Task Runner)는 프로그램 개발에 수반되는 반복적인 작업(Task)들을 스크립트로 작성해 한번에 실행 할 수 있게끔 도와주는 도구입니다. 예를 들어서 어떤 웹 프론트엔드 개발자는 아래와 같은 작업을 수도 없이 반복 할 수 있습니다.

1. LESS 파일들을 CSS로 트랜스파일
2. TypeScript 파일들을 JS로 트랜스파일
3. 기존에 작성한 JS 코드들의 말단이 모두 잘 작동하는 지 테스트(Unit Test)하고, JS 문법 및 코딩 스타일 검사(Lint)
4. 작성한 JS/CSS 및 라이브러리 JS/CSS 파일들을 한개의 JS/CSS 파일로 합치기(Bundling)
5. JS/CSS 파일의 불필요한 공백 및 주석 등을 제거해(Minification) 응답 속도 높히기
6. JS 파일을 난독화(Uglification)하여 소스코드의 로직 유출 방지하기

개발자가 코드를 조금씩 수정 할 때마다, 손 수 위 작업들을 반복한다면 말그대로 배보다 배꼽이 크겠습니다. Task Runner는 이런 Task들을 순차적으로 배치하여, 마치 컴파일 언어의 빌드 과정처럼 구성 할 수 있도록 돕습니다. 또한 소스코드 파일들의 변경을 감지하여, 그 때마다 자동으로 스크립트를 수행하게 할 수 도 있습니다.

<Gulp.js>

Gulp (http://gulpjs.com/), Grunt (https://gruntjs.com/)는 Node.js로 작성된 인기있는 Task Runner입니다. 제공되는 API 및 플러그인을 이용해서 Task 스크립트를 Node.js로 작성하고 실행 할 수 있습니다.

Gulp 스크립트 예시

```
const gulp = require('gulp');
const rename = require('gulp-rename');
const uglify = require('gulp-uglify');

const DEST = 'build/';

gulp.task('default', function() {
  return gulp.src('foo.js')
    .pipe(uglify())
    .pipe(rename({ extname: '.min.js' }))
    .pipe(gulp.dest(DEST));
});
```

5. JavaScript 모듈화

HTML에서 로드된 모든 JavaScript는 전역 공간에서 순차적으로 실행될 뿐, 애초에 JavaScript에는 모듈이라는 개념이 존재하지 않습니다. JavaScript 생태계가 발달하면서, JS에 인위적으로 모듈을 도입하는 CommonJS, AMD (Asynchronous Module Definition)라는 두 진영이 등장했습니다.

1. CommonJS

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

exports.doSomethingElse = function() {
  return someModule.doSomething() + "bar";
};
```

Node.js의 모듈 패턴은 CommonJS 방식을 따릅니다. CommonJS는 백엔드, 서버사이드를 위한 방식입니다.

2. AMD

```
define(
  // 이 모듈은 jquery와 underscore 두 모듈에 의존합니다.
  ['jquery', 'underscore'],

  // 의존 모듈이 로드되면 인자로 모듈들이 전달되며 이 함수가 실행됩니다.
  function ($, _) {

  // 격리된 스코프
  function a(){};
  function b(){};
  function c(){};

  // 함수의 리턴 값이 모듈이 exports하는 값이 됩니다.
  return {
    b: b,
    c: c
  };
});
```

반면 웹 브라우저 환경에 좀 더 적합하게 구현된 방식을 AMD라 합니다. AMD 방식은 콜백을 이용해서 모듈을 동기적으로 로드 할 수 없는 웹 브라우저 환경에서의 문제점을 해결합니다.

3. UMD

```
// jquery 모듈에 의존하는 모듈을 UMD 방식으로 정의
(function (root, factory) {
  // AMD
  if (typeof define === 'function' && define.amd) {
    define(['jquery'], factory);

  // CommonJS (Node.js)
  } else if (typeof exports === 'object') {
    module.exports = factory(require('jquery'));

  // Web Browser (root is window)
  } else {
    root.returnExports = factory(root.jQuery);
  }
}(this, function ($) {
  // ...
  return ...;
}));
```

CommonJS, AMD 방식에 모두 호환되는 모듈을 작성하고 싶은 경우에는 UMD (Universal Module Definition) 방식으로 모듈을 배포 할 수 있습니다.

6. 모듈 번들러

대형 프로그램을 개발하다보면, 스코프를 분리하고 코드의 재사용성을 높히기 위해서, 소스코드를 여러 파일로 분리하며 모듈화하기 마련입니다. 이 때 Node.js 같은 플랫폼에서는 모듈 패턴을 지원하므로 단순히 엔트리 스크립트를 실행하면 되겠지만, 웹 브라우저에서 동작 할 JavaScript에는 모듈 패턴을 웹 브라우저가 이해 할 수 있도록, 의존 모듈들의 코드를 모두 인라인으로 복사하고, 소스코드에 모듈 로더 코드(ex; require 함수 등)를 추가하고, 모듈 간의 스코프를 분리하고, 결과를 몇개의 파일로 합쳐주는(쓰는 모듈마다 script 태그를 쓰기 싫다면) 등의 부가적인 작업들이 필요합니다.

<Webpack>

모듈 번들러(Module Bundler)는 위처럼 모듈 패턴이 적용된 JavaScript 파일들을 웹 브라우저에서 실행가능한 번들(Bundle)로 생성해주는 프로그램입니다. Webpack (https://webpack.js.org/)은 Node.js로 작성된, 대표적인 JavaScript 모듈 번들러입니다. 번들링 기능 뿐만 아니라, 위에 소개한 트랜스파일 및 태스크 러너의 기능도 포함 수 있어서 프론트엔드 개발자들에게 높은 인기를 얻고 있습니다. 현재 Angular.js, React.js 등 대표적인 SPA 프레임워크에서도 사용되고 있습니다.

webpack.config.js (설정 파일 예시)

```
module.exports = {
  entry: './app.js', // 엔트리 스크립트 파일
  output: {
    filename: 'bundle.js' // 생성될 번들 파일
  }
}
```

app.js

```
import bar from './bar'; // ES6에서 고안된 import/export 문법을 제공 (아직 웹 브라우저 상에서는

bar();
```

bar.js

```
export default function bar() {
  // ...
}
```

bundle.js (웹 브라우저에서 로드 가능)

```
/******/ (function(modules) { // webpackBootstrap
/******/  // The module cache
/******/  var installedModules = {};

/******/  // The require function
/******/  function __webpack_require__(moduleId) {

/******/    // Check if module is in cache
/******/    if(installedModules[moduleId])
/******/      return installedModules[moduleId].exports;

/******/    // Create a new module (and put it into the cache)
/******/    var module = installedModules[moduleId] = {
/******/      i: moduleId,
/******/      l: false,
/******/      exports: {}
/******/    };

/******/    // Execute the module function
/******/    modules[moduleId].call(module.exports, module, module.exports, __webpack

/******/    // Flag the module as loaded
/******/    module.l = true;

/******/    // Return the exports of the module
/******/    return module.exports;
/******/  }


/******/  // expose the modules object (__webpack_modules__)
/******/  __webpack_require__.m = modules;

/******/  // expose the module cache
/******/  __webpack_require__.c = installedModules;

/******/  // identity function for calling harmory imports with the correct context
```

```
/******/  __webpack_require__.i = function(value) { return value; };

/******/  // define getter function for harmory exports
/******/  __webpack_require__.d = function(exports, name, getter) {
/******/    Object.defineProperty(exports, name, {
/******/      configurable: false,
/******/      enumerable: true,
/******/      get: getter
/******/    });
/******/  };

/******/  // getDefaultExport function for compatibility with non-harmony modules
/******/  __webpack_require__.n = function(module) {
/******/    var getter = module && module.__esModule ?
/******/      function getDefault() { return module['default']; } :
/******/      function getModuleExports() { return module; };
/******/    __webpack_require__.d(getter, 'a', getter);
/******/    return getter;
/******/  };

/******/  // Object.prototype.hasOwnProperty.call
/******/  __webpack_require__.o = function(object, property) { return Object.protot

/******/  // __webpack_public_path__
/******/  __webpack_require__.p = "";

/******/  // Load entry module and return exports
/******/  return __webpack_require__(__webpack_require__.s = 1);
/******/ })
/************************************************************************/
/******/ ([
/* 0 */
/***/ function(module, exports, __webpack_require__) {

"use strict";
/* harmony export (immutable) */ exports["a"] = bar;
function bar() {
  // ...
}


/***/ },
/* 1 */
/***/ function(module, exports, __webpack_require__) {

"use strict";
/* harmony import */ var __WEBPACK_IMPORTED_MODULE_0__bar__ = __webpack_require__(0)


__webpack_require__.i(__WEBPACK_IMPORTED_MODULE_0__bar__["a" /* default */])();


/***/ }
/******/ ]);
```

<자바스크립트를 사랑해야하는 이유>

JavaScript 생태계를 공부하다보면 배보다 배꼽이 크다는 생각이 듭니다. 위에 거론한 도구들이 뭔가 거창한 일을 하는 것도 아닌데, 웹 생태계와 오픈소스의 동시다발적인 발전에 의해서 이런 복잡한 모습을 띄게 됐다고 생각합니다. 위에서 다룬 내용들은, **일반적인 컴파일 언어에서는 빌드 한번으로 해결되는 문제들에 불과**합니다.

첫 시간에 언어는 단순한 도구에 불과하는 말씀을 드린 기억이있습니다. 재차 말씀드리지만, 일단 위 도구들에 대한 개념 정도만 인지하시고, **필요 할 때마다 필요한 만큼만 배워나가시길 바랍니다.**

> How it feels to learn JavaScript in 2016 (번역본)
> (http://www.looah.com/article/view/2054)

[1] Package Manager
[3] Transcompiler
[4] Source-to-Source Compiler
[5] LESS
[6] CSS Transpiler
[7] PostCSS
[8] W3C, World Wide Web Consortium

[10] TypeScript

[11] Task Runner

[12] Unit Test

[13] Lint

[14] Bundling

[15] Minification

[16] Uglification

[17] Gulp

[18] Grunt

[19] CommonJS

[20] AMD (Asynchronous Module Definition), Asynchronous Module Definition

[21] UMD, Universal Module Definition

[22] Module Bundler

[23] Webpack

6 개발과 배포

6.2 버전 관리, Git, GitHub

1. 버전 관리
2. Git
3. GitHub
4. 오픈소스

버전 관리 시스템의 개념과 Git, GitHub에 대해서 배웁니다. 또 오픈소스에 대해서 알아봅니다.

1. 버전 관리

버전 관리(Version Control, Revision Control)란 소프트웨어 개발 과정에서 소스코드 및 리소스의 변화와 그 파일들을 기록하는 것을 말합니다. 역시 그 필요성에 대해서 먼저 생각해보겠습니다.

1. 소스코드를 안전하게 백업해두고 싶다.
2. 개발 중 문제점이 발견되어, 과거로 돌이가고 싶다. 혹은 과거의 소스코드를 참고하고 싶다.
3. 두 명 이상이 같은 프로젝트를 진행하고 싶다.

<Version Control>

Backup은 어렵지 않습니다. 로컬 보조기억장치, 공유 파일 시스템이나 원격 서버 등에 소스코드를 주기적으로 백업 할 수 있겠습니다. 이런 작업을 자동화하는 것도 어려운 문제는 아닙니다.

코드를 과거로 되돌리는 것(Revert)도 과거 백업본을 이용하면 불가능하지 않겠습니다. 물론, 과거 백업본과 현재 소스코드의 차이점을 일일이 비교하면서, 코드를 특정 시점으로 복원하는 작업이 쉽지는 않겠습니다.

마지막으로 파일시스템 기반의 프로젝트의 협업은 상당히 어려울 것 같습니다. 내가 할 일, 네가 할 일을 명확히 분담하고, 최대한 프로젝트를 파일별로 분리하여 개발해도, 작업하다보면 서로 간의 소스코드 변경사항에 충돌(Conflict)이 있기 마련입니다.

<협업과 그 기록>

또 소스코드를 취합(Merge)하려면, 이메일, 외부저장장치를 쓸까요? 혹은 공유 파일 시스템이나 원격 서버의 도움을 받을까요? 최신 소스코드의 공유는 해결되었다 쳐도, 소스코드나 리소스를 취합하려면 단순히 파일을 덮어쓰는 것보단 더 많은 고민이 필요합니다. 안정적인 코드인지? 코드 품질은 나쁘지 않은지? 이 코드는 어떤 이슈를 해결하는지?

소규모 프로젝트라면 단순히 소스코드를 서로 공유하면서 주기적으로 취합하는 것으로 충분할지 몰라도, 10명, 100명, 1000명이 동일한 프로젝트에 기여하기 위해선 버전 관리를 위한 견고한 시스템이 필요합니다.

2. Git

버전 관리 시스템(VCS; Version Control System) 또는 소스코드 관리 도구(SCM; Source Code Management)는 위와 같은 버전 관리 문제를 해결해주는 소프트웨어를 말합니다. 과거부터 쓰이던 여러가지 프로그램들(SVN, CVS 등)이 있지만, 강제적으로 SVN 등을 써야하는 이유가 아니라면, Git으로 VCS를 시작해나가는 것이 최선이라고 생각합니다.

<Git>

깃(Git)

- 사용법이 복잡함
- 무료이며 오픈소스, CVS 중 가장 높은 점유율
- 개개 파일(텍스트 외에 바이너리까지)별 생성 및 파일명 변경, 이동, 내용 추가 및 삭제에 대한 모든 이력을 효율적으로 보관
- 생성되는 버전(Commit)별로 작성자 및 변경 사항을 기록 할 수 있음
- 원격 저장소 및 서버를 두면 손쉽게 협업 할 수 있음
- 소스코드 충돌(Conflict)시 가능한 경우 자동으로 취합(Merge) 할 수 있음
- 버전을 분기(Branch)하여 충돌을 줄이면서 짜임새있고 장기적인 소프트웨어 개발 계획을 수립할 수 있음

<Github Desktop>

Git 설치

Windows 사용자는 강의 초반에 Git Bash를 설치했다면, 이미 Windows용 Git을 설치한 셈입니다. Git을 설치하지 않은 분이나 Unix 계열 사용자는 Git 홈페이지 (https://git-scm.com/downloads)에서 CLI 프로그램을 다운 받을 수 있습니다. 또 Git GUI 프로그램에 관심이 있다면 홈페이지를 방문해 보세요.

<Git Repository Structure>

저장소(Repository)

협업을 떠나서 로컬에서 백업, 히스토리를 기록 용도로만 Git을 사용한다고 생각해 보겠습니다. Git을 이용하면 특정 디렉토리를 기준으로 저장소(Repository)를 생성 할 수 있습니다. 저장소는 그 루트 디렉토리(Top Level Directory) 하부에 .git 디렉토리를 생성하고, 루트 디렉토리에 생성되는 모든 파일(.git 디렉토리를 제외하고)에 대한 버전별 스냅샷(Snapshot; 특정 시점의 모든 데이터)을 .git 디렉토리에 저장하게 됩니다.

이때 저장소에서는 프로젝트의 이력 및 데이터를 OS 종속적인 파일시스템(디렉토리 구조 등)을 통해서 보관하지 않고 독자적인 형태의 데이터베이스를 구축합니다.

git init

```
works$ mkdir git-test
git-test$ cd git-test/
git-test$ git init
Initialized empty Git repository in /Users/dehypnosis/works/git-test/.git/
```

작업 디렉토리(Working Directory)

당장 저장소 내부에 대해서는 신경을 끄고, **현재 편집중인 소스코드 및 리소스가 존재하는 작업 디렉토리**에서 프로젝트의 이력을 저장소에 반영하는 메커니즘을 알아보겠습니다. 작업 디렉토리 내의 모든 파일은 4가지 상태를 갖습니다.

<Git File Status>

먼저 저장소에서 이력을 추적하지 않는 파일은 Untracked 상태입니다. 물론 최초의 파일들은 모두 Untracked 입니다. 이후에 저장소에 추가(git add)되면 추적되는 파일들은 세가지 상태를 오가게 됩니다.

<Working Directory - Staging Area (Index) - Repository (History)>

마지막 버전(스냅샷)에서 변경된 파일은 Modified 상태입니다. 이때 사용자가 git add를 통해 Staged 상태로 변경할 수 있습니다. Staged란 새로운 버전에 반영될 준비가 되었다는 의미입니다.

이후 git commit -m '메세지'를 통해서 마지막 버전에 Staged 파일들을 반영해 새로운 버전을 생성하게 됩니다. 이때 새로운 버전이 생성된 이후에 파일들은 다시 Unmodified 상태로 돌아갑니다.

위처럼 Untracked/Modified -> Staged -> Committed의 상태를 거치면서 새로운 버전을 생성할 수 있습니다.

git add/status/commit/log

```
git-test$ touch a b c
git-test$ ls
a b c
git-test$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  a
  b
  c

nothing added to commit but untracked files present (use "git add" to track)
git-test$ git add a b
git-test$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:   a
  new file:   b

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  c

git-test$ git commit -m 'a b files added'
[master (root-commit) 153cab6] a b files added
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a
 create mode 100644 b
git-test$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

  c

nothing added to commit but untracked files present (use "git add" to track)
git-test$ git log
commit 153cab6ac3dbbd6e0a621af3f3025c1d483b34e0
Author: <dehypnoss@gmail.com>
Date:   Tue Jun 6 18:14:02 2017 +0900

    a b files added
```

<git rm>

외부적인 요인으로 파일이 삭제된 경우(rm 등으로)에는 Git은 파일을 Modified 상태로 추적합니다. git rm 명령어도 존재하는데, rm 명령어와 큰 차이는 없고, 파일을 삭제한 뒤 바로 Staged 상태로 변경해줍니다.

Tracked 파일을 Untracked 상태로 바꾸려면 git rm --cached를 이용합니다. 이때 파일은 새로운 버전(스냅샷)에서 제거되지만 작업 디렉토리에서 삭제되지는 않습니다.

git rm

```
git-test$ git rm --cached a
rm 'a'
git-test$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  deleted:    a

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  a
  c

git-test$ rm b
git-test$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  deleted:    a

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  deleted:    b

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  a
  c

git-test$ git rm b
rm 'b'
git-test$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  deleted:    a
  deleted:    b

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  a
  c

git-test$ ls
a c
```

<Working Directory - Staging Area (Index) - Repository (History)>

이외에 git reset을 통해서 Staged 상태를 다시 Modified로 복구(파일 내용은 유지)하거나, git checkout을 통해서 Modified 상태를 다시 Unmodified로 복구(파일 내용도 복원)할 수 있습니다. 또는 git revert를 통해서 특정 버전으로 회귀하는 새로운 버전을 발행(commit) 할 수 있습니다.

이외에도 파일을 세세하게 복원하는 방법이나, 프로젝트를 분기(Branch)하는 법, 버전간 변경 사항을 비교하는 법, 저장소에 대한 설정 등 Git은 방대한 기능을 제공합니다. 추가적인 내용들은 필요에 따라서 이외의 경로로 학습하시길 바랍니다. 이후에 아래에서는 협업에 필요한 기초적인 내용과 GitHub에 대해서 다루도록 하겠습니다.

> Git에서 제공하는 온라인 교재 (https://git-scm.com/book/ko/v2)

3. GitHub

협업을 위해서는 우선 모두가 접근 가능한 원격 서버가 필요하겠습니다. 그리고 그 서버에 저장소(Bare Repository; Working Directory 없이, .git 디렉토리와 동일)를 생성하고, 클라이언트의 요청을 처리할 Git 서버 프로그램을 실행하면 되겠습니다. 물론 Git에서 원격 저장소 서버 구축을 위한 프로그램 및 가이드 (https://git-scm.com/book/ko/v2/Git-%EC%84%9C%EB%B2%84-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C)를 제공합니다.

Github

<GitHub>

당장 원격 저장소 서버를 구축하는 일에 대해서는 신경을 끄고, GitHub에 대해서 알아보겠습니다. Git Hosting Service는 Git 원격 저장소 서버 임대 및 다양한 협업 도구를 제공하는 서비스입니다. 수 많은 업체들이 있지만, 그 중에서도 GitHub는 오픈소스의 성지라고 불립니다.

- Git 원격 저장소 생성 및 권한 관리
- 웹에서 저장소의 소스코드를 리뷰, 생성하거나 편집 가능
- 웹에서 저장소의 소스코드를 취합(Merge) 가능
- 저장소 별로 협업을 위한 도구들(게시판 등)을 제공
- 오픈소스 프로젝트를 검색하고 또 기여 할 수 있음

<GitHub Diff tool>

GitHub의 핵심적인 서비스는 무료로 원격 저장소를 생성하고 웹 상에서 쉽게 관리 할 수 있다는 점입니다. 이를 통해서 원격 저장소를 위한 서버 구축 비용을 절감 할 수 있습니다. 하지만 이때 Private 저장소는 유료이기 때문에 무료로 저장소를 생성하려면 프로젝트를 모두에게 공개해야 합니다.

<GitHub Issues tool>

또한 이슈 게시판, 마일스톤 작성 등 강력한 협업 도구를 제공하기 때문에 전세계의 수 많은 개발자들이 오픈소스 프로젝트에 효율적으로 기여 할 수 있습니다. 물론 Private 저장소, 영리 프로젝트에서도 협업 도구는 큰 도움이 됩니다.

GitHub 가입

https://github.com/join (https://github.com/join)에서 GitHub 계정을 생성합니다.

원격 저장소 생성

새로운 프로젝트를 위해서, 먼저 원격 저장소를 GitHub에서 생성합니다. https://github.com/new (https://github.com/new)에서 저장소를 생성 할 수 있습니다. 저장소 생성시 README.md 파일을 생성하지 않으면, 최초의 저장소에는 아무런 버전이 존재하지 않습니다.

먼저 로컬에서 Git 저장소를 생성하고, 최초의 버전을 만들어 보도록 하겠습니다.

로컬 저장소 및 최초 버전 생성

```
works$ mkdir workshop-git
works$ cd workshop-git/
workshop-git$ git init
Initialized empty Git repository in /Users/dehypnosis/works/workshop-git/.git/
workshop-git$ echo "# Workshop GitHub Practice" > README.md;
workshop-git$ git add .
workshop-git$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:   README.md

workshop-git$ git commit -m 'initial commit'
[master (root-commit) 49738c0] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
workshop-git$ git log
commit 49738c0070c2aaf46eabecd86e4d055b7ce35888
Author: <dehypnoss@gmail.com>
Date:   Wed Jun 7 12:41:08 2017 +0900

    initial commit
```

I/O Redirection: echo "# Workshop GitHub Practice" > README.md는 "..." 문자열을 출력하고(echo 명령어), 그 표준 출력을 README.md 라는 파일로 연결합니다. 파일이 존재하지 않을 경우 생성하기 때문에, "..." 문자열을 내용으로하는 README.md 파일을 생성하는 명령입니다. .md는 마크다운(MarkDown)으로 불리는 텍스트 포맷의 확장자 입니다.

최초의 버전(49738c0070c2aaf46eabecd86e4d055b7ce35888)을 생성했습니다. 이제 이 최초의 버전을 원격 저장소에 업로드하겠습니다. 먼저 원격 저장소의 URL을 등록하고, 버전을 업로드합니다.

로컬 저장소의 버전을 원격 저장소로 push

```
workshop-git$ git remote add origin git@github.com:dehypnosis/benzen-workshop-git.gi
workshop-git$ git remote -v
origin  git@github.com:dehypnosis/benzen-workshop-git.git (fetch)
origin  git@github.com:dehypnosis/benzen-workshop-git.git (push)
workshop-git$ git status
On branch master
nothing to commit, working directory clean
workshop-git$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 238 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:dehypnosis/benzen-workshop-git.git
 * [new branch]      master -> master
```

git push origin master 명령으로 origin이라는 이름으로 등록된 원격 저장소에, master라는 브랜치(뒤에서 설명)로 현재 로컬 저장소의 내용을 업로드 했습니다.

자 이번에는 반대로, 다른 개발자 B의 입장에서 이미 존재하는 원격 저장소의 리소스를 다운로드해봅니다. 뿌리가(버전 히스토리가) 다른 원격 저장소에서 스냅샷을 다운로드하거나, 업로드 할 수 없습니다. 때문에 git clone 명령어를 통해서 원격 저장소의 리소스를 다운로드 함과 동시에 로컬 저장소를 생성합니다.

원격 저장소를 기반으로 로컬 저장소를 clone

```
works$ git clone git@github.com:dehypnosis/benzen-workshop-git.git workshop-git-clor
Cloning into 'workshop-git-clone'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
Checking connectivity... done.
works$ cd workshop-git-clone/
workshop-git-clone$ ls
README.md
workshop-git-clone$ git log
commit 49738c0070c2aaf46eabecd86e4d055b7ce35888
Author: dehypnosis <dehypnoss@gmail.com>
Date:   Wed Jun 7 12:41:08 2017 +0900

    initial commit
workshop-git-clone$ git remote -v
origin  git@github.com:dehypnosis/benzen-workshop-git.git (fetch)
origin  git@github.com:dehypnosis/benzen-workshop-git.git (push)
```

git clone을 통해서 리소스 뿐만 아니라 설정까지 완전히 동일한 로컬 저장소가 생성됐습니다. 새로운 파일을 생성하고, 기존 파일을 수정한 뒤 새로운 버전을 생성하고(add & commit), 원격 저장소에 업로드(push)해봅니다.

새로운 버전을 원격 저장소에 업로드 push

```
workshop-git-clone$ touch b1 b2 b3
workshop-git-clone$ echo "B updated README.md" >> README.md
workshop-git-clone$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  modified:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  b1
  b2
  b3

no changes added to commit (use "git add" and/or "git commit -a")
workshop-git-clone$ git add .
workshop-git-clone$ git commit -m 'b updated'
[master fd3603f] b updated
 4 files changed, 1 insertion(+)
 create mode 100644 b1
 create mode 100644 b2
 create mode 100644 b3
workshop-git-clone$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 329 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:dehypnosis/benzen-workshop-git.git
   49738c0..fd3603f  master -> master
workshop-git-clone$ git log
commit fd3603f8aa0ca72ae111b14f6fd89dbbb948e28a
Author: dehypnosis <dehypnoss@gmail.com>
Date:   Wed Jun 7 13:00:17 2017 +0900

    b updated

commit 49738c0070c2aaf46eabecd86e4d055b7ce35888
Author: dehypnosis <dehypnoss@gmail.com>
Date:   Wed Jun 7 12:41:08 2017 +0900

    initial commit
```

echo "B updated README.md" >> README.md는 "..." 문자열을 출력하고, 그 표준 출력을 README.md 라는 파일로 연결합니다. 파일이 존재하지 않을 경우 생성하고, 파일이 존재하면 파일의 뒷부분에 그 표준 입력을 추가합니다.

이제 다시 개발자 A가 업데이트된 원격저장소의 리소스를 다운로드합니다.

원격 저장소에서 다운로드 pull

```
workshop-git-clone$ cd ../workshop-git
workshop-git$ ls
README.md
workshop-git$ git pull origin master
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), done.
From github.com:dehypnosis/benzen-workshop-git
 * branch            master     -> FETCH_HEAD
   49738c0..fd3603f  master     -> origin/master
Updating 49738c0..fd3603f
Fast-forward
 README.md | 1 +
 b1        | 0
 b2        | 0
 b3        | 0
 4 files changed, 1 insertion(+)
 create mode 100644 b1
 create mode 100644 b2
 create mode 100644 b3
workshop-git$ git log
commit fd3603f8aa0ca72ae111b14f6fd89dbbb948e28a
Author: dehypnosis <dehypnoss@gmail.com>
Date:   Wed Jun 7 13:00:17 2017 +0900

    b updated

commit 49738c0070c2aaf46eabecd86e4d055b7ce35888
Author: dehypnosis <dehypnoss@gmail.com>
Date:   Wed Jun 7 12:41:08 2017 +0900

    initial commit
workshop-git$ ls
README.md b1        b2        b3
```

git pull 명령어로 원격 저장소의 최신 버전을 다운로드 할 수 있습니다. 마지막으로 동료끼리 동시에 같은 파일을 수정해서 충돌이 일어나는 경우에 대해서 알아보겠습니다. A가 먼저 README.md를 수정하고 원격저장소에 업로드 한 뒤, B가 뒤이어 README.md를 수정하고 원격저장소에 업로드 하는 시나리오입니다.

A가 README.md를 수정하고 새로운 버전을 업로드

```
workshop-git$ ls
README.md b1        b2        b3
workshop-git$ echo "A updated readme.." >> README.md
workshop-git$ git add .
workshop-git$ git commit -m 'readme update'
[master efea8e1] readme update
 1 file changed, 1 insertion(+)
workshop-git$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 334 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:dehypnosis/benzen-workshop-git.git
   fd3603f..efea8e1  master -> master
```

B가 README.md를 수정하고 새로운 버전을 업로드

```
workshop-git$ cd ../workshop-git-clone/
workshop-git-clone$ ls
README.md b1        b2        b3
workshop-git-clone$ echo "B updated readme.." >> README.md
workshop-git-clone$ git add .
workshop-git-clone$ git commit -m 'b update'
[master e1c6dae] b update
 1 file changed, 1 insertion(+)
workshop-git-clone$ git push origin master
To git@github.com:dehypnosis/benzen-workshop-git.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:dehypnosis/benzen-workshop-git.gi
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

B가 새로운 버전을 생성하는 데(Commit) 성공했으나, 원격 저장소에 업로드 하려니 에러가 발생했습니다. 버전 히스토리가 맞지 않아 업로드 할 수 없으니, 먼저 원격 저장소의 내용을 다운로드하라고 합니다.

B가 로컬에 새로운 버전을 유지한채, 원격 저장소에서 다운로드

```
workshop-git-clone$ git pull origin master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:dehypnosis/benzen-workshop-git
 * branch            master     -> FETCH_HEAD
   fd3603f..efea8e1  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

다운로드에 성공했으나 Automatic merge failed라며 CONFLICT 에러를 보이고 있습니다. 버전 히스토리는 차곡 차곡 쌓일 수 밖에 없기 때문에, 결국 저장소는 순서대로 최초 버전/.../A가 업데이트한 버전/B가 취합(Merge)한 버전의 구성을 따를 수 밖에 없습니다. 취합된 버전을 생성하기 위해서 텍스트 편집기를 통해서 충돌이 일어난 README.md 파일을 수정합니다.

<수정 전>

<수정 후>

Merged 버전 생성 후 업로드

```
workshop-git-clone$ cat README.md
# Workshop GitHub Practice
B updated README.md
B updated readme.. OKAY B FIRST!
A updated readme..
workshop-git-clone$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

  both modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
workshop-git-clone$ git add .
workshop-git-clone$ git commit -m 'merged readme'
[master cb309ea] merged readme
workshop-git-clone$ git push origin master
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 625 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To git@github.com:dehypnosis/benzen-workshop-git.git
   efea8e1..cb309ea  master -> master
```

협업의 흐름

<Git Remote Workflow>

git remote, git push, git clone, git pull 등의 명령어를 이용한 협업의 흐름에 대해서 알아봤습니다. 이외에도 협업시 이슈별로 브랜치(branch)를 생성하고, 충돌을 최소화하며 독립적으로 작업을 진행한 후, merge를 통해서 결과를 합치거나, 또는 원격 저장소의 메인 브랜치(master)에 직접 push 할 권한이 없는 경우 원격 저장소의 관리자에게 pull-request를 보내서 브랜치간에 작업 내역을 합칠 수 있습니다.

Git은 단순히 사용 할 수 있지만, 한편으로 방대한 기능을 제공합니다. 브랜칭, 서브모듈 등 이외의 다양한 기능들은 실제 프로젝트를 진행하는 데 의미가 있기 때문에, 당장 Git을 처음 쓰는 입장에서 혼자서 공부하는 것은 사상누각이라고 생각합니다. 실제로 Git을 통해 프로젝트를 진행하면서 혹은 GitHub의 오픈소스에 기여하면서 Git이나 GitHub에 익숙해지고, Git의 새로운 기능이 궁금 할 때 쯤, Git에서 제공하는 온라인 교재 (https://git-scm.com/book/ko/v2)를 통해서 궁금한 주제에 대해서 공부하면 충분하겠습니다.

.gitignore

DB 설정 파일이나, node_moudules 같은 의존 패키지들 등 저장소의 스냅샷에 포함하고 싶지 않은 파일이 있을 수 있습니다. 물론 add 할 때마다 걸러서 tracking 하지 않으면 되지만, 애초에 tracking 하지 않도록 설정파일을 작성 할 수 있습니다. 아래처럼 루트 디렉토리(혹은 하위 폴더)에 .gitignore 텍스트 파일을 작성합니다.

.gitignore 예시

```
/**/node_modules/
.DS_Store
assets/dist/
config.json
```

> 자세한 .gitignore 작성법 (https://git-scm.com/docs/gitignore)

Git 원격 저장소 서버의 프로토콜

Git 저장소 서버는 로컬 파일 시스템이나 공유 파일 시스템(NFS, CIFS, NAS 등) 또는 HTTP나 SSH 프로토콜을 이용한 서버 프로세스로 운영 할 수 있습니다. 또한 Git 자체 프로토콜(git://)을 이용한 서버 프로세스를 운영 할 수도 있습니다.

> Git 서버 프로토콜의 자세한 내용 (https://git-scm.com/book/ko/v2/Git-%EC%84%9C%EB%B2%84-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C)

4. 오픈소스

오픈소스(Open Source)란 소스코드가 공개되어 누구나 사용/활용 가능한 소프트웨어(또는 하드웨어)를 의미합니다. 프로젝트의 구성 요소 중, 오픈소스로 공개된 모듈이나 라이브러리를 이용하면 개발 비용을 줄이고 품질을 향상시킬 수 있습니다. 지금도 전세계의 개발자들에 의해 수 많은 오픈소스 프로젝트가 진행되고 있으며, 개인이나 기업에 의해서 영리/비영리적으로 활용되고 있습니다.

<Open Source Projects>

이때 오픈소스 역시 라이센스를 갖습니다. 라이센스란 이용자의 사용 방법 및 조건의 범위를 명시한 계약을 말합니다. 아직 국내 및 외국에서 오픈소스 라이센스의 법적 구속력에는 한계가 있지만, 라이센스 위반시 도의적 책임을 피할 수는 없습니다. 몇가지 대표적인 라이센스는 아래와 같습니다.

라이센스	요약	소프트웨어
BSD License; Berkeley Software Distribution License (https://en.wikipedia.org/wiki/BSD_licenses)	라이센스 명시 필요	FreeBSD OS, Nginx 웹 서버, OpenCV 시각처리 라이브러리 등
MIT License (https://en.wikipedia.org/wiki/MIT_License)	라이센스 명시 필요	X11 유닉스 계열 GUI 프레임워크, jQuery/Angular.js/Backbone.js JS 라이브러리, BootStrap CSS 프레임워크 등
Apache License (https://www.apache.org/licenses/LICENSE-2.0)	소스코드를 수정하여 재배포시 라이센스 명시 필요	Android OS, Apache 웹 서버, Hadoop 분산처리 플랫폼, PhoneGap 하이브리드 앱 프레임워크, Tensorflow 머신러닝 라이브러리 등
GNU GPL; GNU General Public License (https://www.gnu.org/licenses/gpl-3.0.en.html)	GPL 코드를 이용한 소프트웨어는 무조건 GPL 라이센스를 따라야함, 프로그램을 영리/비영리로 배포시 소스코드 공개 필요	Linux OS 커널, GCC 컴파일러, Git 버전 관리 시스템, Firefox 웹 브라우저 등
GNU LGPL; GNU Lesser General Public License (https://www.gnu.org/licenses/lgpl-3.0.en.html)	LGPL 코드를 동적(dynamic) 라이브러리로 링크한 프로그램은 소스코드를 공개 불필요, 이외의 경우는 GPL을 따름	FFmpeg 영상/음성 처리 라이브러리, Open Office 사무 소프트웨어 등
Beerware License (https://en.wikipedia.org/wiki/Beerware)	맥주 사주기	?

<오픈소스 라이센스들>

<카카오톡 앱의 오픈소스 라이센스 및 저작권 명시>

영리적으로(물론 비영리의 경우에도) 오픈소스를 이용하는 경우엔, 항상 라이센스를 확인하고 사용 조건을 준수 할 필요가 있겠습니다.

[1] Version Control

[2] Revision Control

[3] Revert

[4] Conflict

[5] Merge

[6] VCS

[7] SCM, Source Code Management

[8] Git

[9] Repository

[11] Snapshot

[12] Working Directory

[13] Bare Repository

[14] Git Hosting Service

[15] I/O Redirection

[16] MarkDown

[17] Open Source

[18] BSD License, Berkeley Software Distribution License

[19] MIT License

[20] Apache License

[21] GNU GPL, GNU General Public License

[22] GNU LGPL, GNU Lesser General Public License

[23] Beerware License

6 개발과 배포

6.3 호스팅, SSH, FTP

1. 호스팅 (Hosting)
2. SSH (Secure Shell)
3. CLI 텍스트 편집기
4. SCP, RSYNC, FTP
5. 배포 (Deployment)

> IDC의 서버를 임대하는 방법에 대해 알아봅니다. 또 원격 서버에 SSH와 FTP 및 RSYNC, SCP 등 여러 프로토콜로 연결해 필요한 작업을 수행해봅니다.

1. 호스팅 (Hosting)

지금까지 서버 프로그램을 로컬 머신에서 개발하고 실행했습니다. 하지만 실제로 로컬의 서버 프로그램과 클라이언트 사이에 네트워킹이 이루어지려면, 서버에 공인 아이피를 할당 할 필요가 있습니다. 이를 위해서 기업용 공인아이피를 구매하고, 원할한 트래픽을 위해서 네트워크 인프라를 구축 할 수 있습니다.

<IDC>

하지만 아무래도 부대비용이 만만치 않기 때문에 주택을 임대해주듯 IDC(Internet Data Center)의 서버를 임대해주는 호스팅(Hosting) 서비스가 존재합니다. 호스팅의 몇가지 방식에 대해서 알아보겠습니다.

방식	설명	제한	용도
코로케이션	IDC에 자사의 머신을 입주시키고 관리를 맡김	-	기업에서 운영중인 네트워크 인프라의 관리를 아웃소싱 할 때
서버 호스팅	IDC에 위치한 물리적 서버를 단독으로 임대	서버의 root 계정을 갖음	자원이 많이 필요하거나, 최고 관리자 권한이 필요한 경우
클라우드 호스팅	IDC의 가상화된 서버 자원을 단독으로 임대	서버의 root 계정을 갖음, 서버의 물리적 자원 및 OS 등을 손쉽게 조정 가능	탄력적인 자원이 필요하며, 최고 관리자 권한이 필요한 경우
웹 호스팅	IDC에 위치한 물리적 서버를 여러 임차인이 공유	일반 유저 계정을 갖으며, 보조기억장치, 메모리, 포트 등 자원을 제한 받음	소규모 웹 서비스(블로그, 쇼핑몰, 단순 홈페이지 등)를 운영 할 경우

<호스팅 방식>

<AWS 클라우드 서비스의 IDC>

용도에 따라서 적절한 임대 방식을 선택 할 수 있겠습니다. 이때 고려 할 점은 서비스의 규모, 최고 관리자 권한이 필요한지, 또는 주요 고객의 위치(IDC가 국내인지 해외인지), 또 서버 자원을 탄력적으로 조절할 필요가 있는지 등의 요소가 있겠습니다. 검색 엔진에서 서버/클라우드/웹 호스팅 또는 server/cloud/web hosting으로 검색해보면 국내나 해외에 수 많은 호스팅 업체들을 확인 할 수 있습니다.

실제 서버를 임대하는 과정은 생략합니다. 검색 엔진을 통해 적절한 업체를 선택하신 후, 학습용으로 저사양의 가상 서버(클라우드)를 임대하시길 추천드립니다. (단독 서버 호스팅은 보통 비용이 높습니다)

아래에서는 여러분들이 서버 호스팅이나 클라우드 호스팅을 통해 단독 서버(Unix 계열 OS)를 임대하여 최고관리자 권한을 획득한 상태로 가정하고 내용을 진행합니다.

2. SSH (Secure Shell)

우리가 임대한 서버에서 어떤 작업을 하고 싶을 때마다 원격 서버가 존재하는 IDC까지 직접 이동해서 작업을 하긴 힘듭니다. 서버를 호스팅 받고 나면 일반적으로 서버 아이피, root 패스워드 또는 SSH 개인키/공개키(나중에 다룹니다)가 주어집니다. 이 정보를 이용해서 원격으로 서버를 제어하는 방법에 대해 알아보겠습니다.

GUI 원격 제어

<Windows Remote Desktop>

먼저 원격 데스크톱 프로토콜(RDP; Remote Desktop Protocol)을 통해 서버를 GUI로 제어 할 수 있습니다. RDP는 MS사에서 개발한 프로토콜로, Windows에는 원격 데스크탑 프로그램(서버 및 클라이언트)을 포함되어 있습니다.

하지만 Unix 계열 OS는 기본적으로 RDP 프로그램를 포함하지 않습니다. 물론 RDP와 유사한 많은 오픈 프로토콜들이 개발되어 Unix 계열 서버를 GUI를 통해 원격 제어하는 것도 불가능하지는 않습니다.

CLI 원격 제어

<Putty>

반면에 Unix 계열 OS는 SSH(Secure Shell) 프로토콜을 통해 서버를 CLI로 제어 할 수 있습니다. SSH는 서버와 클라이언트간의 인증 및 패킷 암호화를 위한 프로토콜 또는 그 프로그램을 일컫으며, 쉽게는 SSH는 안전한 원격 셸이라고 생각해도 좋겠습니다. Unix 계열 OS는 기본적으로 SSH 프로그램(서버 및 클라이언트)을 포함하고 있습니다.

참고로 Windows 서버 역시 PowerShell이라는 프로토콜을 통해 CLI 원격 제어를 제공 할 수 있습니다.

OS	SSH 서버	SSH 클라이언트
Unix 계열	sshd (SSH Daemon)	ssh
Windows	- (cygwin (https://www.cygwin.com/)을 통해 에뮬레이트; openssh (https://www.openssh.com/))	Putty

<SSH 프로그램>

SSH는 Unix 계열 서버의 원격 제어를 위한 프로토콜이지만, Windows용 클라이언트 또한 여러 종류 개발되어 있습니다. Putty (https://www.ssh.com/ssh/putty/download) 같은 프로그램을 설치해도 좋고, Git Bash에서 바로 ssh나 sshd를 실행 할 수 있습니다.

SSH 서버 및 클라이언트

sshd는 기본적으로 22번 포트를 사용하며, 대부분의 Unix 계열 OS는 sshd를 부팅시 자동으로 실행합니다. 호스팅한 서버에서도 22번 포트에서 sshd 프로세스가 실행 중에 있겠습니다. sshd의 설정을 변경하고 재시작하는 등의 내용은 다루지 않습니다.

<원격 서버에 로그인>

SSH는 연결시 원격 서버와 인증 과정을 거치게 됩니다.

로그인

```
# 아이디 생략시 benzen.io 서버의 dehypnosis로 로그인합니다.
dehypnosis-mac:~ dehypnosis$ ssh benzen.io
```

접속시 유저명을 생략하면 현재 로컬 셸의 로그인한 유저 이름으로 로그인을 요청합니다.

유저 이름을 명시하여 로그인

```
# @ 앞에 유저를 명시 할 수 있습니다. benzen.io 서버의 kim 계정으로 접속합니다.
dehypnosis-mac:~ dehypnosis$ ssh kim@benzen.io
```

이렇게 서버에 로그인 할 때 입력 할 비밀번호는 서버에 해당 계정에 설정된 비밀번호와 동일합니다.

패스워드 기반 인증

```
dehypnosis-mac:~ dehypnosis$ ssh kim@benzen.io
dehypnosis-mac:~ dehypnosis$ kim@benzen.io's password:
dehypnosis-mac:~ dehypnosis$ Permission denied, please try again. # 잘못 입력한 경우
dehypnosis-mac:~ dehypnosis$ kim@benzen.io's password:
Last login: Wed Jun 14 15:02:05 2017 from 121.xx.xx.xx # 로그인 성공

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
[kim@ip-172-xx-xx-xx ~]$ ls
benzen.io
[kim@ip-172-xx-xx-xx ~]$ pwd
/home/kim
[kim@ip-172-xx-xx-xx ~]$ exit
logout
Connection to benzen.io closed.
```

인증에 성공한 경우 ssh는 원격 서버에서 CLI 셸(엄밀히 말하면 ssh는 셸 프로그램이 아닙니다)을 실행합니다. 이제 로컬 셸에서처럼 원격 서버를 제어 할 수 있습니다. 서버 작업 후 원격 셸을 종료하려면 exit를 이용합니다.

비대칭키 기반 인증

```
dehypnosis-mac:~ dehypnosis$ ssh kim@benzen.io
Permission denied (publickey).
```

원격 서버 측 sshd의 설정에 따라서, 위와 같이 로그인이 실패 할 수 있습니다. 위의 공개키(Public Key)에 얽힌 얘기는 상당히 많기 때문에, 뒷 챕터에서 비대칭키 암호화, 공개키/개인키, 전자서명 등 암호화에 다루면서 다시 한번 언급하도록 하겠습니다.

<SSH Key 방식 인증>

서버 sshd 측에서 비대칭키 기반 인증을 요구하는 경우에는 개인키가 로그인 과정에 필요할 수 있습니다. 그리고 개인키와 쌍을 이루는 공개키가 원격 서버의 kim 유저의 특정 파일 (/home/kim/.ssh/authorized_keys)에 등록되어 있어야 합니다.

```
dehypnosis-mac:keys dehypnosis$ ssh kim@benzen.io -i ~/works/keys/benzen.io.pem
Last login: Wed Jun 14 15:15:43 2017 from 121.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
[kim@ip-172-xxx-xxx-xxx ~]$
```

ssh kim@benzen.io -i 개인키 파일 경로를 통해서 비대칭키 기반 인증을 수행하면 되겠습니다. 인증의 원리가 잘 상상이 안가나요? 후에 SSL 챕터에서 암호화를 다루면서 다시 언급하겠습니다.

> SSH는 엄밀히 셸이 아닙니다. 원격 서버에서 실행되는 셸은 실제로 bash, csh 등의 셸 프로그램입니다. SSH는 원격 서버와 암호화 채널을 생성한 후, 원격 프로그램을 실행(기본적으로 bash 등의 셸)하거나, 패킷을 다른 포트로 포워딩하는 용도로 응용될 수 있습니다. 추가적인 설명은 (심화) SSH vs SSL/TLS에 대해서 (/course/650#section-8)에서 더 다룹니다.

3. CLI 텍스트 편집기

이제 SSH를 이용해서 원격 서버를 로컬 셸에서처럼 제어 할 수 있습니다. 그런데 실제 웹 서비스 개발의 프로세스를 생각해보면, CLI 셸이 다루기에 편하지만은 않습니다. 가장 먼저 원격으로 소스코드를 작성한다고 생각해보면 어떨까요?

<vim>

지금까지 언급한적은 없지만 vim, emacs, nano 등의 CLI 텍스트 편집기(Unix 계열)들이 있습니다. 이를 이용하면 원격 셸에서 텍스트 파일을 바로 작성 할 수 있지만, 아무래도 복사, 붙혀넣기, 커서 이동 등 대부분의 작업을 키보드로 조작해야 하니 사용법이 상당히 복잡합니다.

<텍스트 에디터별 러닝커브>

CLI 텍스트 에디터에 대해서는 더 이상 다루지 않습니다. CLI에 충분히 익숙해 지신 후에 하나를 선택해서 천천히 익히시는 편을 추천드립니다. 하지만 **당장은 아니더라도 원격 서버를 다루기 위해서는 분명히 배우실 필요가 있습니다.**

> 나무위키의 vi 문서 (https://namu.wiki/w/vi?from=vim)에 vim에 대한 현실적인 소개 및 입문을 위한 매뉴얼을 볼 수 있습니다.

4. SCP, RSYNC, FTP

서버에서 직접 파일을 편집 할 땐 CLI 텍스트 편집기를 이용하면 됩니다. 아래에서는 **로컬에서 원격 서버로, 원격 서버에서 로컬로 파일을 전송**하는 방법을 알아보겠습니다.

1. SCP

SCP(Secure Copy)는 cp 프로그램에서 나아가서 **ssh 프로그램을 응용해 로컬과 원격 서버 사이에서 파일을 복사**하는 기능을 제공하는 프로그램입니다. **scp 프로그램은 내부적으로 ssh 프로그램을 이용**하기에 인증 방식은 SSH와 동일하며, 전송되는 데이터 역시 암호화됩니다. 또한 서버 프로세스를

로컬 파일을 원격 서버에 복사 (업로드)

```
touch x y

# scp [복사하고 싶은 파일들] [원격 계정]@[원격 호스트 주소]:[원격 호스트에 복사될 경로]
dehypnosis-mac:~ dehypnosis$ scp x y kim@benzen.io:~/
kim@benzen.io.com's password:
x                                             100%    0     0.0KB/s   00:00
y                                             100%    0     0.0KB/s   00:00

# 복사 경로에 ~(원격 계정의 홈 폴더)를 쓸 수도, 절대경로로 명시 할 수도 있습니다.
dehypnosis-mac:~ dehypnosis$ scp x y kim@benzen.io:/some/where
kim@benzen.io.com's password:
scp: /some/where: No such file or directory

# -r 옵션으로 폴더를 복사 할 수 있습니다.
dehypnosis-mac:~ dehypnosis$ scp -r test kim@benzen.io:~/
kim@benzen.io.com's password:
.DS_Store                                     100% 6148   972.6KB/s   00:00
any.h                                         100% 4724   846.2KB/s   00:00
any.pb.h                                      100% 10KB   1.6MB/s     00:00
any.proto                                     100% 5236   651.1KB/s   00:00
...
```

마찬가지로 원격 서버의 파일을 로컬로 복사 (다운로드) 할 수 도 있습니다.

원격 서버의 파일을 로컬에 복사 (다운로드)

```
dehypnosis-mac:~ dehypnosis$ scp kim@benzen.io:~/.bash_profile hello
kim@benzen.io.com's password:
.bash_profile                                 100%  213    53.5KB/s   00:00
dehypnosis-mac:~ dehypnosis$ ls
hello         ...
```

---

2. RSYNC

RSYNC(Remote Sync)는 로컬과 원격 파일을 동기화하는데 사용됩니다. scp와 마찬가지로 rsync 역시 (기본적으로) ssh를 통해 통신하기 때문에 SSH 인증을 필요합니다. rsync는 동기화가 목적이기 때문에 효율적으로 변경된 파일만 복사합니다. 이 때문에 rsync는 백업이나, 미러링 등의 용도로 자주 쓰입니다.

원격 서버를 로컬과 동기화 (업로드)

```
dehypnosis-mac:~ dehypnosis$ rsync -r ~/play kim@benzen.io:~/
kim@benzen.io.com's password:
building file list ... done
play/
play/.DS_Store
play/a
play/b
play/ccccc
play/e
play/f
play/g
play/f2/
play/f2/d
play/f3/
play/pp/
play/pp/c.txt
play/pp2/
play/pp2/c.txt
play/pp2/xxxxx

sent 6954 bytes  received 292 bytes  14492.00 bytes/sec
total size is 6148  speedup is 0.85

# rsync는 변경 사항이 있는 파일만 복사합니다.
dehypnosis-mac:~ dehypnosis$ touch play/abcd
dehypnosis-mac:~ dehypnosis$ rsync -r ~/play kim@benzen.io:~/
kim@benzen.io.com's password:
building file list ... done
play/
play/abcd

sent 380 bytes  received 48 bytes  285.33 bytes/sec
total size is 6148  speedup is 14.36
```

마찬가지로 로컬을 원격 서버와 동기화 (다운로드) 할 수도 있습니다.

로컬을 원격 서버와 동기화 (다운로드)

```
dehypnosis-mac:~ dehypnosis$ rsync -r kim@benzen.io:~/some-folder .
receiving file list ... done
kim/
kim/.bash_history
kim/.bash_logout
kim/.bash_profile
kim/.bashrc
...
```

3. FTP

FTP (File Transfer Protocol)는 Unix 계열 OS 뿐만 아니라 Windows 서버에서도 활발하게 쓰이는 파일 전송 프로토콜입니다. 프로토콜의 구체적인 내용과 원격 서버에서 FTP 서버 프로세스를 구축하는 방법은 생략하고, FTP 클라이언트에 대해서만 소개합니다.

<ForkLift (macOS 용)>

FTP 클라이언트 프로그램들은 일반적으로 위와 같은 GUI를 제공합니다. 양쪽에 로컬과 원격의 디렉토리 및 파일들이 표시되고, 마우스를 이용해서 서로 간에 파일을 복사하거나 파일을 삭제, 수정 하는 등의 작업을 수행 할 수 있습니다. GUI를 이용해서 손쉽게 파일 작업을 할 수 있는 장점이 있습니다.

<SFTP 비대칭키 방식 인증으로 접속>

FTP 서버 프로세스는 기본적으로 21번 포트를 쓰는데, FTP 프로토콜은 암호화를 수행하지 않기 때문에 파일 정보 뿐만 아니라, 유저의 계정 정보까지 노출 될 수가 있습니다.

따라서 서버는 보안을 향상시키기 위해서 SSH 프로토콜과 결합된 SFTP (22번 포트), 또는 SSL/TLS(뒤에서 다룸)에서 작동하는 FTPS (21번 포트) 프로토콜을 이용 할 수도 있습니다.

클라이언트 측에서는 서버에서 제공하는 프로토콜에 맞추어 접속하면 되겠습니다.

이름	지원 OS
탐색기	Windows
FileZilla (http://filezilla-project.org/)	Windows (서버 프로그램도 제공), Linux, macOS
SmartFTP (https://www.smartftp.com/)	Windows
ForkLift (http://www.binarynights.com/forklift/)	macOS (유료)

<몇가지 FTP 클라이언트>

---

4. Git

이전 챕터에서 다뤘던 Git을 통해서 파일을 동기화 할 수도 있습니다. 소프트웨어의 최신 소스코드가 Github 같은 원격 Git 저장소에 저장되어 있다면, git 클라이언트로 간단하게 소스코드를 최신 버전으로 동기화 할 수 있겠습니다.

최초로 원격 서버에 소스코드 저장소를 복제하기 위해서는 ssh로 원격 서버에 접속한 후 git clone을 이용하면 되겠습니다.

git clone

```
dehypnosis-mac:~ dehypnosis$ ssh kim@benzen.io
dehypnosis-mac:~ dehypnosis$ kim@benzen.io's password:
Last login: Wed Jun 14 19:59:30 2017 from 121.xx.xx.xx

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
[kim@ip-172-xx-xx-xx ~]$ ls
benzen.io
[kim@ip-172-xx-xx-xx ~]$ git clone https://github.com/dehypnosis/workshop2.git
'workshop2'에 복제합니다...
remote: Counting objects: 6228, done.
remote: Compressing objects: 100% (99/99), done.
오브젝트를 받는 중:  13% (832/6228), 38.73 MiB | 3.57 MiB/s
...

[kim@ip-172-xx-xx-xx ~]$ ls
benzen.io  workshop2
```

이후에는 ssh로 원격 서버에 접속한 후 git pull로 최신 소스코드로 동기화 할 수 있겠습니다.

git pull

```
dehypnosis-mac:~ dehypnosis$ ssh kim@benzen.io
dehypnosis-mac:~ dehypnosis$ kim@benzen.io's password:
Last login: Wed Jun 14 19:46:08 2017 from 121.xx.xx.xx

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
[kim@ip-172-xx-xx-xx ~]$ cd benzen.io/workshop2/
[kim@ip-172-xx-xx-xx workshop2]$ git pull origin master
Username for 'https://github.com': dehypnosis
Password for 'https://dehypnosis@github.com':
remote: Counting objects: 29, done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 29 (delta 11), reused 29 (delta 11), pack-reused 0
오브젝트 묶음 푸는 중: 100% (29/29), 완료.
https://github.com/dehypnosis/workshop2 URL에서
 * branch            master     -> FETCH_HEAD
   4aae423..64caffc  master     -> origin/master
업데이트 중 4aae423..64caffc
Fast-forward
 sessions/630/web-farm.gif                                          | Bin 0 -> 987
 sessions/650/ext1_sshsessionforwarding.jpg                         | Bin 0 -> 304
...

[kim@ip-172-xx-xx-xx workshop2]$
```

5. 배포 (Deployment)

How to update your website?

- FTP, SCP
- Rsync
- SSH + GIT

@rakodev

Capistrano

<Deployment>

서버 프로그램(웹 서버 등)의 배포(Deployment)는 일반적으로 위에서 소개한 방식처럼, 원격 서버에 소스코드를 동기화하고, 재컴파일 한 후에 프로그램을 재실행하는 방식으로 이루어집니다. 물론 플랫폼에 따라서 그 절차는 상이합니다. 예를 들어 Node.js의 경우에는 코드를 동기화하고 바로 node 프로그램만 재실행하면 되겠습니다.

물론 이러한 번거로운 과정들을 자동화하려는 시도가 존재하며, 이를 배포 자동화(Continuous Deployment)라고 합니다. 간단히는 셸 스크립트(Shell Script)를 작성 할 수도 있고, 또는 별도의 소프트웨어나 시스템을 이용 할 수도 있습니다.

> 쉘 스크립트(shell script)는 쉘이나 명령 줄 인터프리터에서 돌아가도록 작성되었거나 한 운영 체제를 위해 쓰인 스크립트이다. 쉘 스크립트가 수행하는 일반 기능으로는 파일 이용, 프로그램 실행, 문자열 출력 등이 있다... (위키백과)

<이런 경우라면?>

서버 프로세스가 단일 호스트에서만 작동하지 않고, 트래픽을 분산하기 위해서 게임, 웹 등의 대형 서비스처럼 복수의 호스트에서 동일한 프로세스를 실행시키는 경우에는 배포 자동화가 유용하겠습니다.

[1] IDC, Internet Data Center
[2] Hosting
[3] RDP, Remote Desktop Protocol
[4] SSH, Secure Shell

[5] PowerShell

[6] vim

[7] SCP, Secure Copy

[8] RSYNC, Remote Sync

[9] FTP, File Transfer Protocol

[10] SFTP, Secure FTP

[11] FTPS, FTP over SSL

[12] Deployment

[13] Shell Script

6 개발과 배포

6.4 DNS, 도메인, 메일 서버 (ing)

1. 작성중

> 도메인을 IP로 해석해주는 DNS에 대해 공부하고, 도메인을 구입하고 IP를 할당하는 방법을 알아봅니다. 또한 메일 서버와 SMTP, POP3에 대해 알아봅니다.

1. 작성중

...

6 개발과 배포

6.5 암호화, 전자서명, 인증서와 SSL

1. SSL (Secure Sockets Layer)
2. 암호화 (Encryption)
3. 전자서명 (Digital Signature)
4. 인증서 (Certificate)
5. 인증기관 (CA; Certification Authority)
6. SSL HandShake
7. HTTPS (HTTP over SSL)
8. (심화) SSH vs SSL/TLS에 대해서

암호화의 필요성, 전자서명, 인증서와 SSL에 대해서 알아봅니다. 또한 무료 인증서(Let's Encrypt)를 발급 받는 과정을 안내합니다.

1. SSL (Secure Sockets Layer)

<Packet Sniffing>

TCP/IP(또는 TCP 위의 HTTP 같은 응용프로토콜) 패킷이 유선이나 무선 환경을 통해 전송되는 순간을 생각해봅시다. 발신자와 수신자 사이의 네트워크엔 수 많은 라우터나 다른 노드들이 존재하고 있습니다. 이때 이 사이에 위치한 해커는 손쉽게 인터넷을 흐르는 패킷들을 감청(Packet Sniffing) 할 수 있습니다. 즉, 여러분이 로그인을 하기 위해 전송한 아이디나 패스워드가 손쉽게 해커의 손에 들어갈 수 있다는 문제가 있습니다.

<DNS Spoofing>

또 해커는 피해자에 머신에 DNS Spoofing을 통해 자기 머신의 주소가 서버의 주소인 것처럼 속일 수 있습니다. 이땐 피해자와 서버 사이의 패킷을 중계하면서 정보를 가로채거나 변조 할 수 있습니다.

<TCP - SSL/TLS - Application>

이러한 문제는 웹 뿐 아니라, TCP/IP를 이용하는 모든 응용프로그램의 보안에 위협이 됩니다. 패킷을 해커가 읽거나 변조 할 수 없도록 암호화하기 위해서 네트워크의 TCP/IP - Application 계층 사이에 SSL(Secure Sockets Layer)라는 계층을 고안하였습니다. (TLS; Transport Layer Security; 전송 계층 보안이라고 부르기도 함) 물론 이 때 TCP/IP 위에 SSL을 입히는 것은 응용프로토콜 개발자의 선택사항입니다.

SSL의 목적

1. 패킷의 감청 방지
2. 패킷의 변조 방지
3. 서버의 신원 인증

2. 암호화 (Encryption)

SSL의 구체적인 메커니즘을 살펴보기 전에, 암호화에 대해서 알아보겠습니다. 먼저 헷갈릴 수 있는 용어들과 비교해봅니다.

인코딩 (Encoding)

인코딩은 정보의 형태나 형식을 변환하는 처리를 말한다.

Base64는 바이너리 데이터를 6bit 포맷(2^6개의 문자로)의 문자열로 변환한다. (간단한 연산으로 복원 가능)

```
Base64("The quick brown fox jumps over the lazy dog") = "VGhlIHF1aWNrIGJyb3duIGZveCE
Base64("The quick brown fox jumps over the lazy dog.") = "VGhlIHF1aWNrIGJyb3duIGZve(
Base64("") = ""
```

해싱 (Hashing)

해싱은 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 처리를 말한다.

SHA-1 해시 함수는 최대 2^{64}비트의 메시지로부터 160비트의 해시값을 생성한다. (복원 불가)

```
SHA1("The quick brown fox jumps over the lazy dog") = "2fd4e1c67a2d28fced849ee1bb76e
SHA1("The quick brown fox jumps over the lazy dog.") = "408d94384216f890ff7a0c3528e8
SHA1("") = "da39a3ee5e6b4b0d3255bfef95601890afd80709"
```

암호화 (Encryption)

암호화는 임의의 키를 통해 평문을 암호화하여 암호문으로 만드는 처리를 말한다. 반대로 키를 통해 암호문을 평문으로 만드는 처리는 복화화라고 한다.

RSA 공개키 암호화 알고리즘은 큰 수를 소인수분해하기 어렵다는 점을 이용한 암호화 알고리즘으로 두 개의 큰 소수를 기반으로 두개의 키를 생성하고, 이를 이용해 메세지를 암호화/복호화한다. (키가 있어야 복원 가능)

```
두 소수 p=163, q=167을 기반으로 생성한 암호키 e=847, d=127를 선택
평문 = hello
평문을 숫자로 표현 = 104 101 108 108 111
위 숫자를 e를 통해 암호화 = 16078 6642 1822 1822 13509
암호문을 d를 통해 복호화 = hello
```

이제 인코딩, 해싱, 암호화를 구분 할 수 있다면, 암호화의 두가지 방식에 대해서 알아보겠습니다.

대칭키 암호화

먼저 기원전부터 쓰인 대칭키 암호화 방식의 원리는 발신자와 수신자가 동일한 알고리즘과 그 열쇠를 공유하는 방식입니다. 쉽게는 알파벳을 1칸 앞으로(secret key) 옮기는 방식의 암호 알고리즘을 이용하면 hello는 ifmmp로 암호화 될 수 있으며, 마찬가지로 암호 키를 알고 있다면 역으로 복호화 할 수도 있습니다. 근대의 대칭키 암호화 역시 근본적으로 위와 같은 원리를 사용합니다.

대칭키 방식은 속도가 빠르고 연산량이 적지만, 키가 유실되면 보안 체계가 그대로 무너지기 때문에 키의 분배가 힘들다는 단점을 갖습니다.

비대칭키 암호화

대칭키의 키 분배 문제를 해결한 비대칭키 암호화 방식은 비대칭키와 달리 공개키(Public Key), 개인키(Private Key)라고 불리는 두개의 키를 생성합니다. 이 때 공개키로 암호화한 암호문은 개인키로만 복호화가 가능하며, 개인키로 암호화한 암호문은 공개키로만 복호화가 가능합니다.

이를 통해 발신자는 사전에 입수한 수신자의 공개키로 메세지를 암호화하고, 수신자는 전달 받은 암호문을 개인키로 복호화 할 수 있습니다.

비대칭키 방식은 서버-클라이언트와 같은 일대다 구조에서 키의 분배 문제를 해결할 수 있지만, 속도가 느리고 연산량이 많은 단점이 있습니다.

3. 전자서명 (Digital Signature)

대표적인 비대칭키 알고리즘인 RSA는 대표적으로 전자서명에 많이 이용되고 있습니다. 전자서명은 말 그대로 데이터의 작성자를 증빙하기 위해 데이터에 전자적인 서명을 첨부하는 것을 뜻합니다. 물론 전자서명은 수기서명과 동일한 효력을 지니고 있습니다.

<Digital Signature with RSA>

전자서명의 메커니즘은 비대칭키 암호화에 그 기반을 둡니다. 우선 원본 데이터를 공개된 방법으로 해싱하여 특정 길이의 문자열로 만듭니다. 그 해시값을 작성자의 개인키로 암호화한 전자서명을 생성하여 원본 데이터에 첨부하고, 수신자에게 발송합니다. 이후에 수신자는 받은 전자서명을 작성자의 공개키로 복호화하고 이를 받은 데이터를 해싱한 해시값과 비교합니다. 이때 두 해시값이 동일한지 비교하여 데이터의 작성자와, 데이터가 변조되지 않았음을 검증할 수 있습니다.

4. 인증서 (Certificate)

자, 데이터를 안전하게 암호화하고 또 전자서명하여 교환하는데 성공했습니다. 하지만 한가지 더 문제가 남아있습니다. 단순히 전자서명만으로는 데이터 생성자, 즉 공개키 소유자의 실제(오프라인) 신원을 증빙할 수 없다는 문제입니다.

여기서 공개키/개인키 키 쌍의 소유자의 신원을 보증하기 위해서 인증서(Certificate)라는 개념이 등장합니다. 사용자들은 암호화나 전자서명에 이용할 키 쌍을 정부기관이나 은행 등 공인된 기관에서 개인정보를 인증하며 발급 받을 수 있습니다. 이 때 발급자는 기관으로부터 생성된 개인키와 그에 대한 인증서를

받게 됩니다. 이 때 그 인증서에는 일반적으로 아래와 같은 정보가 포함됩니다.

<공인인증서>

- 발급자의 신원 정보
- 발급자의 공개키
- 인증 유효기간
- 인증기관 정보
- 위 모든 내용에 대한 인증기관의 전자서명

즉, 인증서란 공개키와 그에 대한 인증기관의 보증으로 볼 수 있습니다. 다시말해서 본인의 인증서를 제공한다는 의미는, 본인의 공개키와 그 보증을 전달하는 것과 같습니다. 인증서와 그에 포함된 공개키를 통해 전자서명이나 암호화시에 데이터의 무결성을 검증 할 수 있습니다. (물론 인증서 발급 기관을 신뢰 할 수 있을 때)

5. 인증기관 (CA; Certification Authority)

비대칭키 암호화의 속성을 이용하면, 데이터를 암호화하고, 데이터에 전자서명을 첨부하고, 인증서를 통해서 데이터 생성자의 신원을 보증 할 수 있습니다. 이처럼 공개키 및 인증서를 활용하는 소프트웨어, 플랫폼, 정책, 제도 등을 통틀어 공개키 기반 구조 (PKI; Public Key Infrastructure)라고 합니다. HTTPS, SFTP, SSH, SCP 등 SSL 위에서 작동하는 프로토콜, 또는 공인 인증서를 이용한 금융거래까지 모두 PKI에 속한다고 볼 수 있겠습니다.

<X.509 구조>

앞서 다룬바처럼 PKI에서 본인의 신원(공개키)을 인증서로 인증 할 필요가 있는 경우엔 공인된 인증서가 필요합니다. 이러다보니 PKI와 그 인증기관에 따라서 인증서의 포맷이 제각각일 수 있습니다. 이러한 상황에서 인증서의 범용성을 높히고, 웹(https) 같은 범국제적인 PKI를 위해서 X.509라는 인증서의 국제적인 표준 규격이 자리잡았습니다.

<연쇄적인 인증 구조>

X.509에서 인증서는 기본적으로 최상위 인증기관(CA; Certification Authority)의 루트 인증서를 시작으로 하위 인증기관들의 인증서를 타고 내려가 일반 사용자들의 인증서까지 연쇄적으로 상위 기관이 하위 기관을 인증하는 트리 구조를 띄고 있습니다. 따라서 사용자가 Root CA의 인증서(최상위 인증서는 자가 서명되어 있음)를 신뢰 한다면, 그 하위의 모든 인증서를 신뢰 할 수 있는 구조가 됩니다.

> 한국의 공인인증서 및 개인키 역시 파일 양식 자체는 국제표준을 따르고 있지만, 그 파일들이 보관, 저장되는 위치와 방법이 독특하여 웹브라우저로는 사용이 불가능하다. 그 결과, 한국의 공인인증서를 이용하려면 이용자가 추가프로그램을 반드시 설치해야만 한다. 인증서는 원래 금융거래에만 사용되는 것이 아니라, 모든 전자적 거래(금전적이건 비금전적이건)에서 당사자의 신원을 확인하거나, 전자서명을 하는 용도로 사용될 수 있고, 한국의 공인인증서도 물론 그런 다양한 용도로 사용될 수 있다. 그러나 현실적으로 공인인증서는 전자금융거래에서 주로 사용되고 있다. 금융위원회는 전자금융거래에 "공인인증서 등"을 사용하도록 강제하고 있다가 의무사용 폐지됐다. ... (위키백과)

6. SSL HandShake

SSL의 목적

1. 패킷의 감청 방지
2. 패킷의 변조 방지
3. 서버의 신원 인증

다시 본래의 주제 SSL로 돌아와서, SSL의 목적을 생각해보면, 이제 그 해답지가 보입니다. 비대칭키를 이용해 TCP 패킷들을 암호화(패킷의 감청 방지)하고, 패킷에 전자서명을 첨부하며(패킷의 변조 방지), 서버 측에서 공개키에 대한 인증서를 제공(서버의 신원 인증)한다면 모든 목적을 달성 할 수 있겠습니다.
(하지만 SSL이라고 완벽하진 않습니다. 지속적인 보안 취약점이 발생하고 있습니다.)

<SSL Session Flow>

SSL 연결 과정 (SSL HandShake)

1. 서버는 클라이언트에게 인증서(+공개키)를 전달
2. 이때 클라이언트는 서버의 인증서를 본인이 미리 알고있는 CA 목록을 통해 검증해서, 신뢰할 수

없는 서버의 경우에 유저에게 경고 할 수 있음

3. 클라이언트는 대칭키 하나를 생성해 서버의 공개키로 암호화하여 서버에 전달
4. 이후 양측간에 주고 받는 데이터는 대칭키로 암호화함

SSL에서는 비대칭키를 통해서 최초로 연결을 맺지만, 실제 통신에서는 서버측의 연산량을 절감하기 위해서 대칭키를 사용합니다. (비대칭키의 복호화 연산량은 대칭키 보다 약 1,000배 크다고 함)

7. HTTPS (HTTP over SSL)

<HTTPS (Verified)>

여기까지 인터넷 암호화 및 전자서명에 대한 기본적인 배경지식을 습득했습니다. 이제 본연의 웹으로 돌아가서, 웹서버에 SSL을 적용한 HTTPS를 구현해보도록 하겠습니다.

<HTTPS (Not verified)>

본인 스스로 서명한 인증서(Root Certificate)를 생성해서 HTTPS 웹서버를 구축 할 수도 있습니다. 하지만 위처럼 공인된 인증서가 아니기 때문에 클라이언트의 신뢰를 얻기 힘듭니다. 따라서 공인 인증서를 발급하게 되는데, 이때 일반적으로 인증서는 유료로 발급 받을 수 있습니다.

왜냐면 CA은 발급자에게 서비스를 제공해야하며, 인증서 발급자의 도메인 소유 정보를 확인해야하고, 발급자들의 인증 정보를 갱신하며, 웹브라우저들에게 본 기관의 Root Certificate를 내장하도록 부탁해야 하는 등, CA의 역할을 맡는데 여러 비용이 수반되기 때문입니다. 대표적인 국제 공인 CA 중에는 Comodo, Symantec (Verisign), GoDaddy 등이 있습니다.

국내에서는 국가 공개키 기반 구조 (NPKI; National Public Key Infrastructure)라고 불리는 한국인터넷진흥원(KISA)을 Root CA로 두는 PKI가 자리잡고 있습니다. 하지만 KISA에서 아직까지 국제 Root CA 검증 감사를 받지 않아서, 국내 CA에서 발급 받은 인증서는 특정 웹 브라우저들에게 신뢰받지 못 할 수도 있습니다.

Let's Encrypt

<Let's Encrypt>

그러던 와중 2016년도에 Mozilla, Sisco, Akamai 등의 기업들이 Let's Encrypt라는 비영리 CA를 설립하고, HTTPS의 보급을 위해서 무료로 인증서를 발급해주고 있습니다. https://letsencrypt.org (https://letsencrypt.org)에서 기부 및 안내를 볼 수 있고, https://certbot.eff.org (https://certbot.eff.org)에서 인증서를 발급 받는 방법을 설명 받을 수 있습니다.

위의 certbot 웹사이트로 접속하면 운영중인 웹서버 종류 및 서버 OS를 선택 할 수 있습니다. 이후에 certbot 이라는 인증서 발급 및 갱신 프로그램을 설치하고, 실제로 발급 받는 방법에 대한 안내를 받을 수 있습니다. 여기서는 AWS 리눅스 환경에서 Node.js(Express) 웹서버에 SSL을 적용하는 과정을 적어보겠습니다.

1. certbot 설치 및 인증서 발급

```
# ssh로 도메인이 연결된 서버에 접속합니다.
dehypnosis-mac:keys dehypnosis$ ssh ec2-user@benzen.io

# wget은 주어진 URL에서 파일을 다운로드하는 프로그램입니다.
[ec2-user@benzen ~]$ wget https://dl.eff.org/certbot-auto

# 실행 권한을 주고
[ec2-user@benzen ~]$ chmod a+x certbot-auto

# 관리자 권한으로 실행하면, 이메일 주소 입력, 약관 동의, 이메일 공개(옵션)의 과정을 거칩니다.
# 이후에 certbot 프로그램이 자동으로 도메인 소유권을 인증하고, 인증서 및 개인키를 생성합니다.
# 서버 머신에 이미 Apache나 다른 웹서버가 설치되어 있따면, [1]이나 [3]의 옵션을 사용 할 수 있습니다.
# 그 외의 경우에는 단순히 [2]를 선택하면 certbot이 임시로 80 또는 443 포트에 웹서버를 가동하여 도메인 소
[ec2-user@benzen ~]$ ./certbot-auto certonly
FATAL: Amazon Linux support is very experimental at present...
if you would like to work on improving it, please ensure you have backups
and then run this script again with the --debug flag!
Alternatively, you can install OS dependencies yourself and run this script
again with --no-bootstrap.
[ec2-user@benzen ~]$ sudo ./certbot-auto certonly
Saving debug log to /var/log/letsencrypt/letsencrypt.log

How would you like to authenticate with the ACME CA?
-------------------------------------------------------------------------------
1: Apache Web Server plugin - Beta (apache)
2: Spin up a temporary webserver (standalone)
3: Place files in webroot directory (webroot)
-------------------------------------------------------------------------------
Select the appropriate number [1-3] then [enter] (press 'c' to cancel): 2
Enter email address (used for urgent renewal and security notices) (Enter 'c' to
cancel):kim@benzen.io

-------------------------------------------------------------------------------
```

```
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.1.1-August-1-2016.pdf. You must agree
in order to register with the ACME server at
https://acme-v01.api.letsencrypt.org/directory
-------------------------------------------------------------------------------
(A)gree/(C)ancel: a

-------------------------------------------------------------------------------
Would you be willing to share your email address with the Electronic Frontier
Foundation, a founding partner of the Let's Encrypt project and the non-profit
organization that develops Certbot? We'd like to send you email about EFF and
our work to encrypt the web, protect its users and defend digital rights.
-------------------------------------------------------------------------------
(Y)es/(N)o: y
Please enter in your domain name(s) (comma and/or space separated)  (Enter 'c'
to cancel):workshop.benzen.io
Obtaining a new certificate
Performing the following challenges:
tls-sni-01 challenge for workshop.benzen.io
Waiting for verification...
Cleaning up challenges

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at
   /etc/letsencrypt/live/workshop.benzen.io/fullchain.pem. Your cert
   will expire on 2017-08-27. To obtain a new or tweaked version of
   this certificate in the future, simply run certbot-auto again. To
   non-interactively renew *all* of your certificates, run
   "certbot-auto renew"
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory will
   also contain certificates and private keys obtained by Certbot so
   making regular backups of this folder is ideal.
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

2. 인증서 및 개인키 확인

```
# 인증서 및 개인키(실제 파일의 바로가기 파일; 또는 symbolic link)는 다음 경로에 생성되어 있습니다.
[ec2-user@benzen ~]$ sudo ls /etc/letsencrypt/live/workshop.benzen.io
README  cert.pem  chain.pem  fullchain.pem  privkey.pem

# privkey.pem: 개인키를 Base64로 인코딩한 텍스트 파일입니다.
# fullchain.pem: 서버의 인증서와 상위 기관의 인증서를 Base64로 인코딩한 텍스트 파일입니다.
# cert.pem, chain.pem: cert.pem + chain.pem = fullchain.pem (일반적으로 fullchain.pem이

[ec2-user@benzen ~]$ sudo cat /etc/letsencrypt/live/workshop.benzen.io/README
This directory contains your keys and certificates.

`privkey.pem`  : the private key for your certificate.
`fullchain.pem`: the certificate file used in most server software.
`chain.pem`    : used for OCSP stapling in Nginx >=1.3.7.
`cert.pem`     : will break many server configurations, and should not be used
                 without reading further documentation (see link below).

We recommend not moving these files. For more information, see the Certbot
User Guide at https://certbot.eff.org/docs/using.html#where-are-my-certificates.
```

Let's Encrypt, certbot을 떠나서, 인증서 발급 과정에서 기억 할 점은, CA가 도메인 소유권을 인증하고, 인증서 및 개인키를 생성해준다는 점입니다.
CA에 따라서 인증서 발급 과정이 웹 UI상에서 이루어질 수도 있고, Let's Encrypt처럼 셸 프로그램을 제공 할 수도 있습니다.

3. 서버에 HTTPS 적용

```
// Express.js의 경우엔 아래처럼 443 포트에 https 서버를 실행시킬 수 있습니다.
// http의 기본 포트가 80인 것처럼, https의 기본 포트는 443입니다.
const path = require('path');
const fs = require('fs');
const https = require('https');
const app = express();

// ... 서버 로직

const CERT_DIR = path.join(__dirname, 'ssl');
const sserver = https.createServer({
  key: fs.readFileSync(CERT_DIR + '/privkey.pem'),
  cert: fs.readFileSync(CERT_DIR + '/fullchain.pem'),
}, app).listen(443);
console.log('Secure server running at port 443');

// 같은 로직의 http 서버를 동시에 운영 할 수 도 있습니다.
const server = https.createServer(app).listen(80);
console.log('Non-Secure server running at port 80');
```

Apache, Nginx, Tomcat, IIS 등, 기타 상용 서버를 사용하는 경우에는 플랫폼에서 제공하는 방법에 따라서 개인키와 인증서를 연결하고 특정 포트에 https 서버를 실행 할 수 있습니다. 주의사항은 Let's Encrypt의 인증서의 유효기간은 90일이기 때문에, 만료일이 다가올 때마다 certbot을 이용해서(ex; sudo ./certbot-auto renew) 갱신하거나, Cron (https://ko.wikipedia.org/wiki/Cron) 등의 운영체제 서비스에 스크립트를 작성하여 이 과정을 자동화 할 필요가 있습니다.

8. (심화) SSH vs SSL/TLS에 대해서

이전에 공부했던 SSH 또한 SSL/TLS처럼 핸드쉐이킹에 비대칭키를, 패킷 통신에는 대칭키를 이용합니다. 또한 응용 프로세스를 위해서 TCP 패킷을 암호화하는 것도 동일합니다.

그렇다면 두 암호화 프로토콜이 어떻게 다르게 쓰이는 지 고민해볼 수 있겠습니다. 이 부분은 심화적인 내용이므로 사전에 설명하지 않은 용어들이 나올 수 있습니다. 네트워크, 암호화 및 SSH 프로토콜에 관심이 있는 분들이 읽어 보시길 바랍니다.

SSH는 일반적으로 엔드유저를 위한 ssh 클라이언트 프로그램을 의미하지만, 응용 프로토콜 하위의 하나의 암호화 프로토콜로도 볼 수 있겠습니다.

SSL/TLS는 인증서 포맷으로 X.509을, SSH는 독자적 포맷(ssh-keygen 프로그램을 통해서 생성)을 쓴다는 점을 제외하면, 모든 SSL과 SSH 응용은 대체가 가능합니다. FTPS(FTP over SSL)와 SFTP(Secure FTP), HTTPS와 SHTTP는 모두 비대칭키와 대칭키 암호화를 이용하므로 보안에 있어서 원리적인 차이가 없습니다.

하늘 아래 두 영웅이 있을 수 없다고 했는데요. 둘은 비슷한 역할을 수행하지만, 그 응용 방식에서 큰 차이가 납니다. SSH는 서버에서 sshd를 22포트에 열고 ssh 클라이언트와 암호화된 패킷을 주고 받으면서 포트 포워딩; 프로세스 간에 패킷을 중계하는 방식으로 응용됩니다. 실제로 sftp, scp, ssh 클라이언트의 트래픽은 모두 sshd의 22포트로 들어갑니다.

<SSH 터널링을 통해 패킷을 암호화하고 방화벽을 우회>

이처럼 SSH를 응용하는 프로그램은 항상 **SSH 터널링**(터널은 ssh 클라이언트로 쉽게 생성 할 수 있음)을 응용합니다. 또한 이러한 특성으로 **SSH는 방화벽 우회 및 프록시, 또는 패킷 암호화**에도 많이 쓰이고 있습니다.

이처럼 SSH는 여러 서버 호스트와의 연결 및 응용에 쓰이는 **프로토콜 내지는 활용도가 높은 프로그램**입니다. 이렇게 ssh 클라이언트가 다용도로 쓰이기 때문에, ssh 클라이언트는 핸드쉐이크 과정에서 (기본적으로) 로컬의 공개키를 무더기로 보냅니다.

반면에 SSL/TLS는 포트포워딩 방식보다는 각 응용 프로토콜마다 개별 포트를 바인딩하고, 서버 프로세스 자체에서 인증서를 갖고 핸드쉐이킹 과정을 처리하고 패킷을 주고 받습니다. 따라서 **SSL/TLS를 응용하기 위해서는 프로그램에서 내부적으로 프로토콜 수준부터 구현**해야 하겠습니다.

위 내용을 정리하자면 SSH나 SSL/TLS는 모두 암호화를 위한 프로토콜로 볼 수 있고, **SSH는 SSL/TLS 보다 좀 더 가벼운 용도로 손쉽게 응용** 할 수 있다는 생각입니다.

[1] Packet Sniffing
[2] DNS Spoofing
[3] SSL, Secure Sockets Layer
[4] TLS, Transfer Layer Security
[5] Encoding
[6] Hashing
[7] Encryption
[8] Public Key
[9] Private Key
[10] Certificate

[11] PKI, Public Key Infrastructure
[12] X.509
[13] CA, Certification Authority
[14] NPKI, National Public Key Infrastructure
[15] Let's Encrypt
[16] Cron

6 개발과 배포

6.6 비밀번호 해싱

1. 해싱의 필요성
2. 단순 해싱하기 (Hashing)
3. 소금 간하기 (Salting)
4. 해싱 반복하기 (Key Stretching)

> 고객의 개인정보를 보호하기 위해서 비밀번호를 해싱하는 방법을 알아봅니다.

1. 해싱의 필요성

개인정보보호법은 개인정보의 암호화에 대해서 다음과 같이 규정하고 있습니다.

> **비밀번호, 바이오정보, 주민등록번호** 등과 같은 주요 개인정보가 암호화되지 않고 개인정보처리 시스템에 저장되거나 네트워크를 통해 전송될 경우, 노출 및 위·변조 등의 위험이 있으므로 암호화 등의 안전한 보호조치가 제공되어야 한다.
> ※ "암호화"는 개인정보취급자의 실수 또는 해커의 공격 등으로 인해 개인정보가 비인가자에게 유·노출되더라도 그 내용 확인을 어렵게 하는 보안기술이다.

즉, 시스템이 인터넷에서 격리된 네트워크에 위치하는 경우나, 예외적인 개인정보 항목을 다루는 경우를 제외하고는 국가에서 권고하는 **상용 암호화 알고리즘을 이용해 개인정보를 암호화하도록** 법적으로 요구하고 있습니다.

또한 외부의 해킹 뿐만 아니라, 내부 DB 관리자의 개인정보 악용을 방지하기 위해서도 **개인정보를 저장하거나 통신 할 때** 암호화 할 필요가 있습니다.

특히 비밀번호가 노출될 경우, 일반적으로 수 많은 서비스에 같은 비밀번호를 쓰는 사람들이 많기 때문에 큰 피해를 초래 할 수 있습니다. 이번 챕터에서는 비밀번호를 안전하게 저장하는 방법에 대해서 알아보겠습니다.

- **DB에 저장시**

개인정보를 해싱하여 복원 할 수 없도록 함

- 통신시

  개인정보를 주고 받을 때(로그인/회원가입 페이지 등) SSL을 적용하여 암호화

일반적으로 개인정보를 처리 할 땐, 통신시에는 SSL을 적용하고, 저장 할 때는 데이터를 복원 할 수 없도록 해싱하여 저장합니다. 예를 들어 "1234"를 그대로 DB에 저장하는 것이 아니라, 특정 해싱 알고리즘을 이용하여 HASH("1234")를 계산, 저장하고, 추후에 로그인 과정에서는 HASH("입력값")과 해시값을 비교함으로써 인증 로직을 수행합니다.

이렇게 비교 검증에 쓰이는 해시값을 특히 Digest라고도 부릅니다.

> 해싱, 해시 함수는 **자료구조에서 빠른 자료의 검색**, 또는 **데이터의 위변조 체크**를 위해서 쓰이나, 복원이 불가능(알고리즘을 알아도 역으로 연산이 불가)한 단방향 해시함수의 경우에는 **암호학적 용도**로 쓰이고 있습니다.

2. 단순 해싱하기 (Hashing)

자 그럼 MD5나 SHA-1, SHA-256 (MD5, SHA-1은 모두 보안적 결함이 경고됨) 등의 해시 알고리즘을 써서 비밀번호를 해싱한 뒤 저장하도록 하겠습니다.

평문	Digest (SHA-256 해시값)
1234	03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4
benzen	e55fe3d91090865027e8c35b993cb30f7639b6910bf7f46920ec3cb883e4780d
qwer1234!@	6da4b98bfb06b104ff34b6e0806df208f78a16cec2a001c0964be9fc4c18c42b

<SHA-256 해싱>

자 이제는 데이터가 노출되어도 원본 데이터에 대해서 안심 할 수 있겠습니다.

MD5 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
md5_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	52 GB 64 GB	Perfect Non-perfect	Perfect Non-perfect
md5_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	460 GB 576 GB	Perfect Non-perfect	Perfect Non-perfect
md5_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,080	99.9 %	127 GB 160 GB	Perfect Non-perfect	Perfect Non-perfect
md5_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	680 GB 864 GB	Perfect Non-perfect	Perfect Non-perfect
md5_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,481,889,716,084	99.9 %	65 GB 80 GB	Perfect Non-perfect	Perfect Non-perfect
md5_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	316 GB 396 GB	Perfect Non-perfect	Perfect Non-perfect

SHA1 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
sha1_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	52 GB 64 GB	Perfect Non-perfect	Perfect Non-perfect
sha1_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	460 GB 576 GB	Perfect Non-perfect	Perfect Non-perfect
sha1_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,080	99.9 %	127 GB 160 GB	Perfect Non-perfect	Perfect Non-perfect
sha1_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	680 GB 864 GB	Perfect Non-perfect	Perfect Non-perfect
sha1_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,481,889,716,084	99.9 %	65 GB 80 GB	Perfect Non-perfect	Perfect Non-perfect
sha1_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	316 GB 396 GB	Perfect Non-perfect	Perfect Non-perfect

<Rainbow Table>

그런데 이게 뭘까요, 해커들이 특정 길이의 문자열들에 대해서 모든 해시값을 계산해두었습니다. 이 Rainbow Table이라는 자료에 해시값을 대조해보면 원본 데이터를, 특히 비밀번호처럼 짧은 데이터를 유추하는 것이 어렵지 않습니다.

이 Rainbow Table을 판매하는 사이트 (http://project-rainbowcrack.com/table.htm)뿐만 아니라, 온라인 상에서 각종 해시 알고리즘의 Rainbow Table을 이용해 해시값을 유추해주는 사이트 (https://crackstation.net/) 또한 존재합니다. 이 사이트에서 위 테이블의 해시값들을 입력해보면, 평문을 바로 확인 할 수 있습니다.

3. 소금 간하기 (Salting)

<Salting>

위처럼 단순 해시값은 해킹에 쉽게 노출될 수 있습니다. 이 때문에 Salting이라는 아이디어가 생겨났습니다. 비밀번호와 임의로 생성한 문자열(Salt)를 합쳐서 해싱하여 이 해시값를 저장하는 방법입니다.

평문	Salt (랜덤값)	Digest (SHA-256 해시값)
1234	54abcf98296c1	cd862644758055f28204d6248c6babb18f97dd3532c9633e27771df41c7839a1
benzen	f2f6e9faf7f27	6eae2b57421159cb8bd0d3783740650b72806968b519e1ab4692d9aca8dc72b1
qwer1234!@	e7822b7e9e0ad	62c57bb55d422a8fa06d87abb4939cd4b4baca6bbe0c671f6d6099dd1bb52849

<SHA-256 해싱>

이 때는 물론 해시값과 소금을 같이 보관해야 합니다. 그래야 인증시 HASH("입력값" + "소금")과 해시값의 비교 로직을 수행 할 수 있기 때문입니다.

위에서 소금 간을 해서 생성한 해시값은 위의 해시값을 유추해주는 사이트 (https://crackstation.net/)에서 복원되지 않습니다. 단순히 소금 간을 하는 로직으로 보안 수준을 확연히 높혔습니다.

4. 해싱 반복하기 (Key Stretching)

<Key Stretching>

자 여기서 한술 더떠서 위 해시 과정을 여러번 반복한다면(Key Stretching), 해커는 패스워드를 무작위 대입하며(Brute Force) 해시값과 비교하는 과정에서 훨씬 많은 계산량을 필요로하게 됩니다. 즉, 해싱 반복 횟수가 많아 질수록 보안 수준을 높힐 수 있습니다. 물론 이 때는 정상적인 유저의 로그인 요청에 대한 응답 시간 또한 늦어질 수 있습니다.

<Bcrypt Digest>

위와 같은 해싱 과정을 검증된 알고리즘(SHA-256 등)을 이용해서 직접 구현 할 수도 있지만, 기존에 이미 안전성이 검증된 비밀번호 해싱 라이브러리들이 구현되어 있습니다. 그 중 대표적으로 쓰이고 있는 bcrypt 라이브러리를 소개하고자 합니다. bcrypt는 Node.js 뿐만 아니라 다양한 플랫폼에서 이미 구현되어있어, 쉽게 프로그램에 적용 할 수 있습니다.

또한 bcrypt는 Digest 자체에 소금값과 해시값 및 반복 횟수를 같이 보관하기 때문에, 비밀번호 해싱을 적용하는데 있어서 DB 설계를 복잡하게 할 필요가 없습니다.

회원가입 예시

```
const bcrypt = require('bcrypt');

// ... 유저의 요청 데이터를 파싱 한 후

// 임의의 소금을 생성하고, 10번의 키 스트레칭을 통해 해시를 생성
bcrypt.hash(request.password, 10)
  .then(hash => {
    request.password = hash;
    return User.create(request); // ORM으로 보시면 되겠습니다.
  })
  .then(user => {
    // ...
  });
```

로그인 예시

```
const bcrypt = require('bcrypt');

// ... 유저의 요청 데이터를 파싱 한 후, 유저의 저장된 bcrypt 해시값을 가져옴

bcrypt.compare(request.password, stored_hash)
  .then(okay => {
    if (!okay) {
      throw new Error('PASSWORD_NOT_MATCHED');
      // ...
    }
    // ...
  })
  .then(() => {
    // ...
  });
```

[1] Digest

[2] Rainbow Table

[3] Salting

[4] Key Stretching

[5] Brute Force

[6] bcrypt

7 다른 플랫폼으로

1. 다른 플랫폼들 (ing) ... 478

2. GUI 프로그램 아키텍쳐, MVC 패턴 ... 480

새로운 플랫폼을 두려워 할 필요가 없습니다. 이론이 충족된 상태라면 어느 플랫폼에도 빠르게 적응 할 수 있습니다. 다른 웹 백엔드 플랫폼인 PHP 등, 또 SPA 프레임워크, 스마트폰 플랫폼 Android나 iOS, 하이브리드 앱 기술 등, 새로운 플랫폼에 어떻게 접근해야 할 지 알아봅니다.

7 다른 플랫폼으로

7.1 다른 플랫폼들 (ing)

1. 작성중

> Node.js와 웹 브라우저 플랫폼에서 나아가서 다른 웹 백엔드 플랫폼, SPA, 스마트폰 플랫폼, 하이브리드 앱 등 다른 플랫폼의 소프트웨어 구조에 대해서 알아봅니다.

1. 작성중

...

7 다른 플랫폼으로

7.2 GUI 프로그램 아키텍쳐, MVC 패턴

1. MVC 패턴
2. 웹 서비스의 MVC 패턴
3. 응용프로그램의 MVC 패턴과 그 파생

GUI 프로그램의 MVC, MVP, MVVM 아키텍쳐에 대해서 알아봅니다.

1. MVC 패턴

일반적으로 최종사용자(End-User)를 위한 응용프로그램들은 GUI 기반의 프로그램들입니다. 웹 서비스 또한 웹 서버와 웹 브라우저의 네트워킹에 기반을 두고 있지만 사용자 입장에서는 하나의 GUI 프로그램처럼 작동합니다. 또한 OS 응용프로그램들, 모바일 프로그램들 역시 GUI에 기반을 두고 있습니다.

<MVC 패턴>

이런 GUI 프로그램의 구조에 대해서 생각해보면, 일반적으로 유저의 입력에 반응하여, 데이터를 조작 또는 추출하고, 그래픽을 통해 가공해 보여주는 구조로 생각 할 수 있습니다. 이러한 GUI 프로그램의 개발은 방법론의 발전에 따라서, 개발의 편의성과 재사용성, 확장성, 즉 추상화를 따르다보니, 위 세가지 역할들을 분리하여 고도로 추상화하는 방향으로 발전했습니다.

- 모델 (Model)
  프로그램에서 다룰 데이터들을 조작하고 추출 할 수 있도록 추상화한 요소.
- 뷰 (View)
  뷰는 UI를 구성하며, 모델의 변화를 직접 감지하거나, 통보 받아서 UI에 반영하는 요소.
- 컨트롤러 (Controller)
  컨트롤러는 플랫폼에 따라 다양한 모습으로 나타나며, 일반적으로 사용자의 입력/요청을 받아서,

보여줄 뷰를 선택/교체하거나, 모델에게 데이터의 조작을 요청하는 요소.

MVC 패턴은 개발자의 철학이나, 프로그램이 작성되는 플랫폼에 따라서 다양한 면모를 띌 수 있습니다. 하지만 공통되는 원칙은 데이터를 제어하는 로직, 화면을 제어하는 로직, 그외의 로직을 분리하는 작성하는 것입니다.

2. 웹 서비스의 MVC 패턴

<웹서비스의 MVC 패턴>

고전적인 웹 서비스(Client-Side의 JavaScript; Ajax, DOM 조작 등을 제외해보면)의 전체적인 흐름을 모식하면 위 그림과 같습니다. 서버는 사용자가 보낸 HTTP Request를 적절히 라우팅하여 데이터를 조작하거나 한 뒤, 특정 뷰를 HTTP Response로 응답해줍니다.

3. 응용프로그램의 MVC 패턴과 그 파생

<MVC 패턴>

MVC 패턴

```
Controller {
  init() {
    // 여러 뷰에 의존성을 갖습니다.
    this.view1.on('click', e => {
      // ..
      this.model1.update(..);
    });

    // 여러 모델에 의존성을 갖습니다.
    this.model1.on('update', value => {
      // ..
      this.view2.update(..);
    });
  }
}
```

이 기본적인 MVC 패턴에서 Model과 View를 접착하는 Controller의 역할에 따라 프로그램 아키텍쳐는 다양한 형태로 파생됩니다. 이에 따라서 Model과 View의 구조, 프로그램의 전체적인 구조가 영향을 받습니다.

기본적인 MVC 패턴에서는 컨트롤러가 직접 사용자의 입력을 받아 Model을 조작하고, View에 Model을 반영합니다. 이 때는 컨트롤러의 복잡도가 상당히 커지며 재사용이 불가능 할 수 밖에 없습니다.

<MVP 패턴>

MVP 패턴

```
View1 {
  // 프레젠터에 의존성을 갖습니다.
  onclick(e) {
    this.presenter.update(..);
  }
}
Presenter1 {
  init() {
    // 모델과 뷰에 의존성을 갖으며, 모델의 데이터를 뷰에 구체적으로 반영합니다.
    this.model1.on('update', value => {
      // ..
      this.view.update(..);
    })
  }
  update(..) {
    this.model1.update(..);
  }
}

```

나아가서 중앙 집중화된 컨트롤러의 복잡도를 줄이기 위해 MVP라고 불리는 패턴을 적용 할 수도 있습니다. 먼저 사용자의 다양한 입력을; 즉, 모든 이벤트에 대한 처리 로직을 담당하는 컨트롤러의 복잡도를 줄이기 위해서 각각의 입력 처리를 View 자체로 옮깁니다.
다음으로는 View와 Model을 이어주기 위한 Presenter라는 새로운 요소를 각 View마다 작성해줍니다. 컨트롤러의 무게를 줄이고 뷰와 모델을 분리 할 수 있지만, 뷰 단위로 각각의 컨트롤러(프레젠터)를 작성하는 느낌을 지울 수는 없습니다.

<MVVM 패턴>

MVVM 패턴

```
View1 {
  // 뷰모델에 의존성을 갖습니다.
  init() {
    this.viewModel1.on('update', (state, data) => {
      if (state == 'hide') {
        this.hide();
      }
      // ..

      this.textContent = ...;
      // ..
    });
  }
  onclick(e) {
    this.viewModel.trigger(..);
  }
}
ViewModel1 {
  // 여러 (데이터) 모델에 의존성을 갖습니다.
  // 또한 다중의 뷰에게 상태 변화를 알립니다. (이벤트 패턴 등으로 느슨하게)
  init() {
    this.model1.on('update', data => {
      // ..
      this.update(.., ..);
    });
  }
  trigger(..) {
    // ..
    this.model1.update(..);
  }
}

```

MVP 패턴에서 나아가, Presenter의 뷰에 대한 과한 결합도를 덜기위해 추상화한 것을 ViewModel 이라는 요소로 표현합니다. ViewModel은 Presenter와 달리 View 자체의 구체적인 형태는 알지 못합니다. ViewModel은 View의 구체적인 표현 로직을 Presenter에서 View로 옮긴 것으로 생각 할 수 있습니다.

<세가지 패턴에 대한 비교>

실무 경험이 많이 쌓이기 전이라면, 위의 의사 코드나 그 용어들이 아리송하고 서로 구분이 잘 가지 않으실 것 같습니다. 소프트웨어 디자인 패턴은 개발자들끼리도 이해하고 있는 바가 다른 경우가 많고, 실무에서는 여러 패턴이 뒤섞여가며 사용되는 것이 일반적입니다. 역할을 분담하고, 코드의 집중도를 분배하고, 재사용성을 높히기 위한 다양한 시도라고 생각하시면 좋겠습니다. 실제로 플랫폼과 개발하는 프로그램의 성격에 따라 적합한 패턴이 서로 다르기 마련입니다.

이런 정형화된 설명에 꼭 끼워 맞추기 보다는 역할 분담, 의존성의 분리, 재사용성이라는 핵심적인 원리를 기억하고 프로그램을 설계하는 것이 최선이라고 생각합니다.

[1] Model
[3] Controller
[5] MVP, Model, View, Presenter

workshop.benzen.io

BENZEN