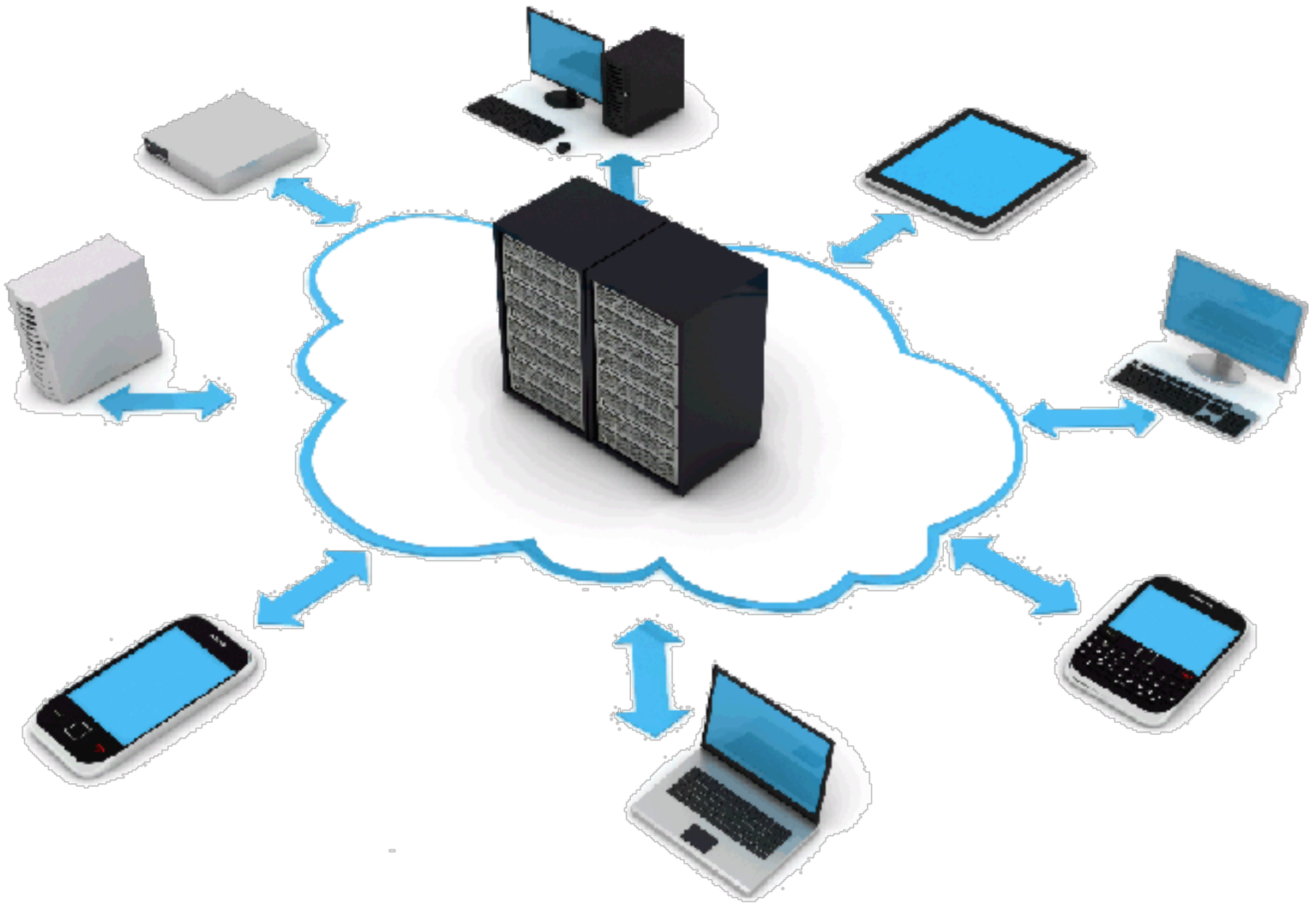
Network 기초

강사 주영민

Network

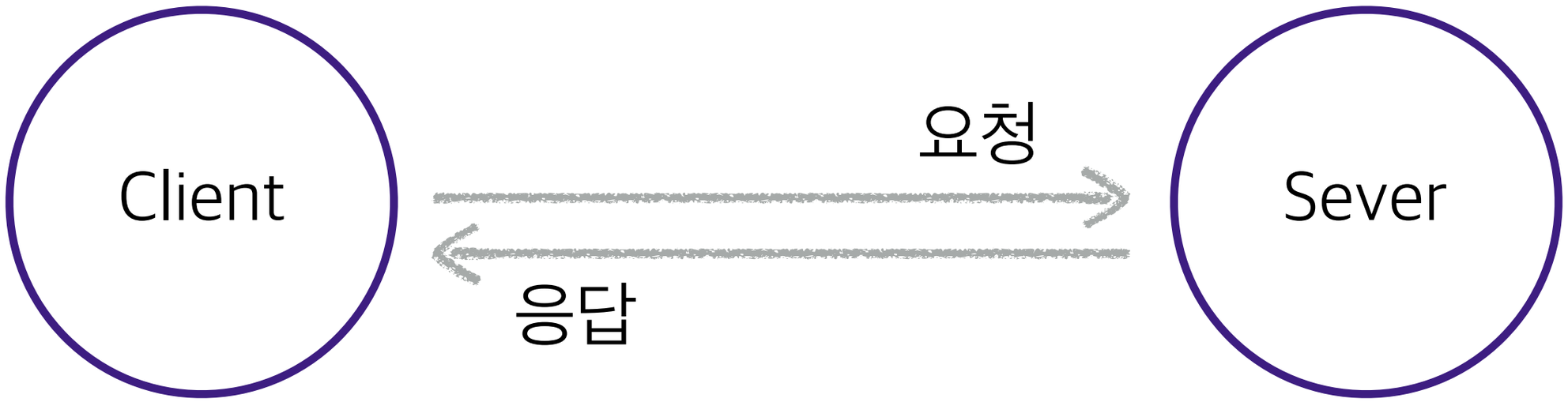
'어떤 연결을 통해 컴퓨터의 자원을 공유하는 것'.



클라이언트 서버 모델(client-server model)

- Network architecture 중 하나.
- Server : Client의 요청에 따라서 데이터를 제공해 주는 컴퓨터
- Client : 서버로부터 요청한 데이터를 받는 컴퓨터
- 각각의 컴퓨터가 Client, Server의 역할에 맞게 구성되어 Network통신이 이뤄진다면 우린 이걸 클라이언트 서버 모델이라고 부를수 있다.

클라이언트 서버 모델(client-server model)



서버에 요청은 어떻게 보낼까?

Protocol

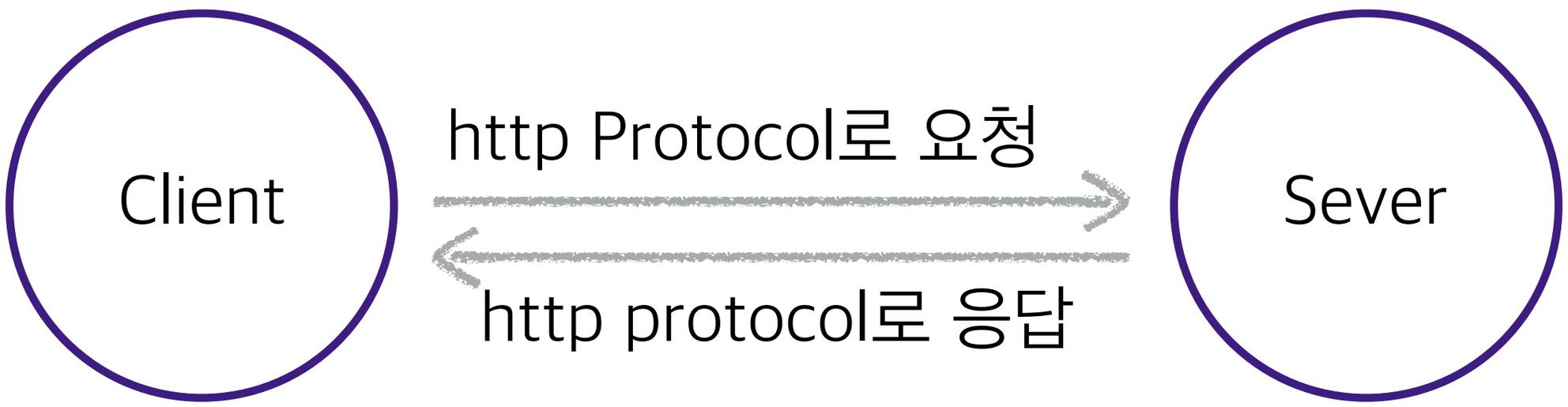
- 프로토콜(protocol)은 컴퓨터끼리 또는 컴퓨터와 단말기 사이에 상호통신 할때 데이터를 에러없이 원활하고 신뢰성있게 주고받기 위해 필요한 약속을 규정하는 것으로서 통신규약이라고도 한다.

- Protocol종류

1. HTTP : Hyper Text Transfer Protocol
2. HTTPS : Secure Hyper Text Transfer Protocol
3. FTP : File Transfer Protocol
4. SMTP : Simple Mail Transfer Protocol
5. SSH : Secure Shell
6. 등등

HTTP

HTTP (Hypertext Transfer Protocol) — a protocol or "handshake" that defines **how messages (and data) are sent and received** by computers; the underlying protocol for **the World Wide Web**



*TCP와 UDP는 무엇인가?

URL

- URL(Uniform Resource Locator, 문화어: 파일식별자, 유일자원지시기)은 네트워크 상에서 자원이 어디 있는지를 알려주기 위한 규약이다.

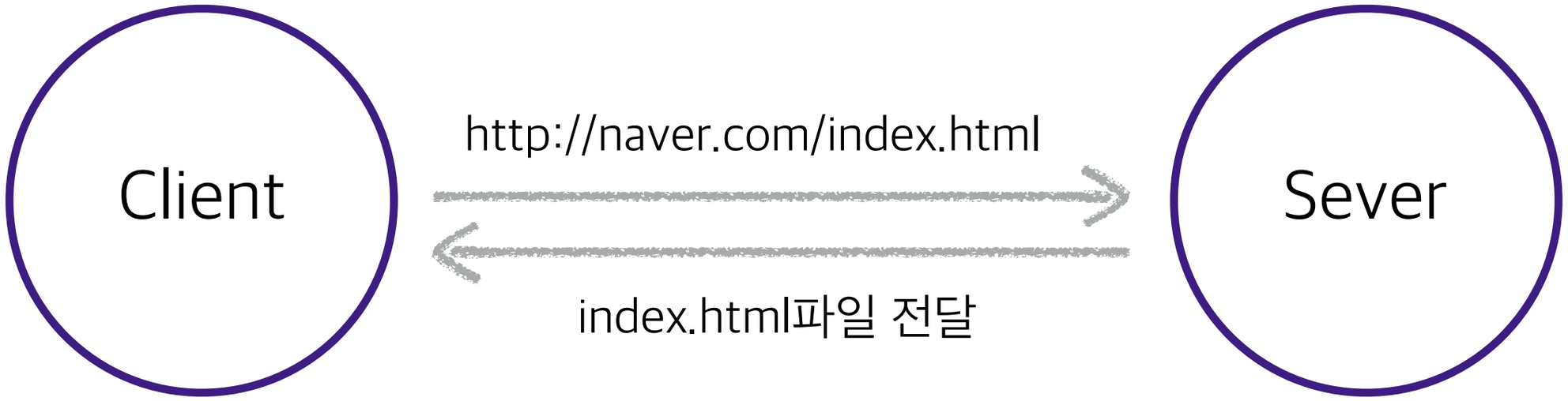
- URL구성

    1. URL은 제일 앞에 자원에 접근할 방법을 정의해 둔 프로토콜 이름을 적는다. ftp, http ..
    2. 프로토콜 이름 다음에는 프로토콜 이름을 구분하는 구분자인 ":"을 적는다.
    3. 만약 IP 혹은 Domain name 정보가 필요한 프로토콜이라면 ":" 다음에 "//"를 적는다.
    4. 프로토콜명 구분자인 ":" 혹은 "//" 다음에는 프로토콜 마다 특화된 정보를 넣는다.

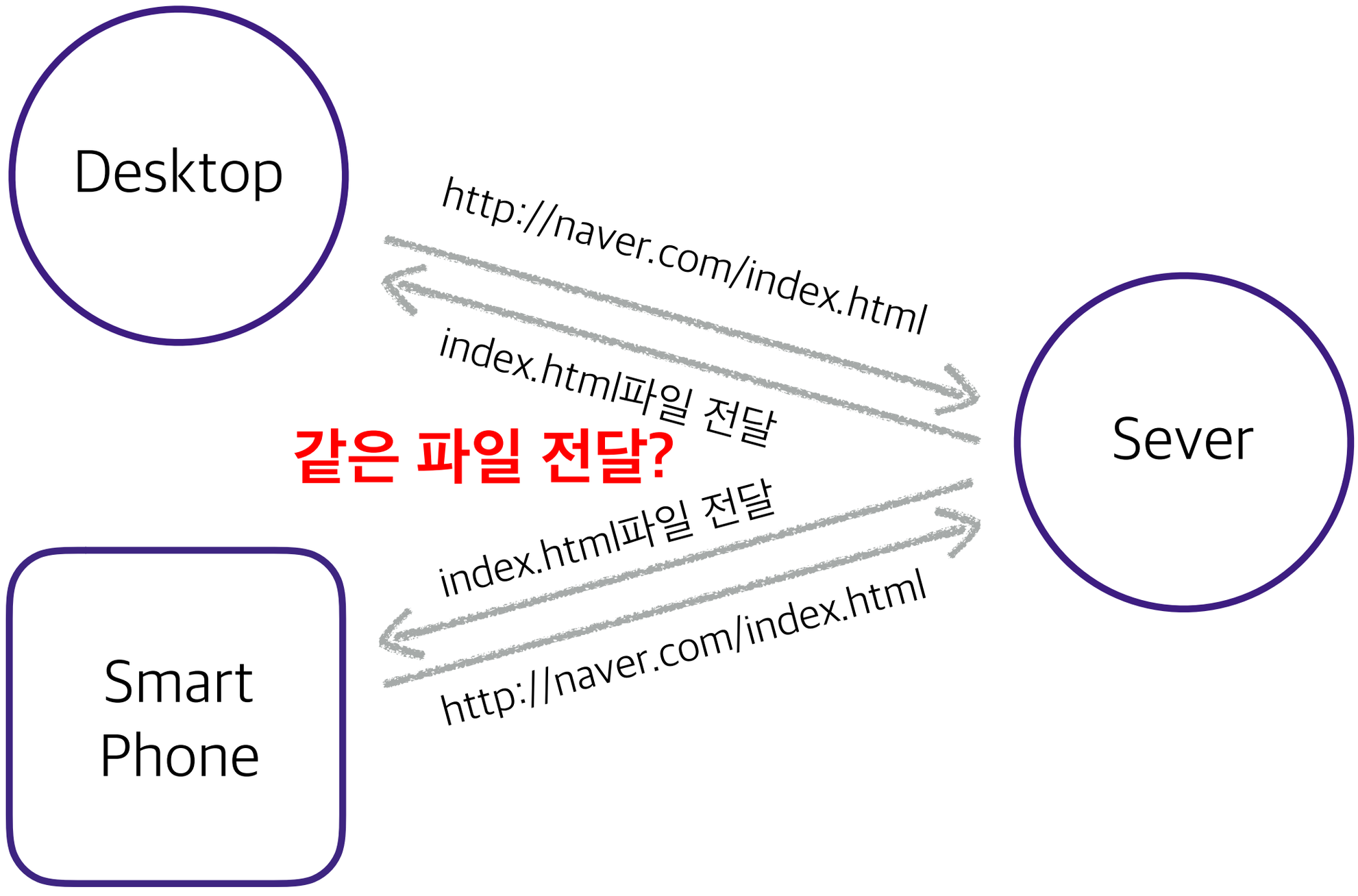
- 다음 URL을 실행하면 어떻게 될까? 얘기 해보자

http://naver.com/index.html

Network using http protocol



다양한 기기에서의 Network



URI

- 통합 자원 식별자(Uniform Resource Identifier, URI)는 인터넷에 있는 자원을 나타내는 유일한 주소이다. URI의 존재는 인터넷에서 요구되는 기본조건으로서 인터넷 프로토콜에 항상 붙어 다닌다.

- URI vs URL

    1. URL은 URI의 한 종류이다.
    2. URL은 특정 리소스의 정확한 위치를, URI는 자원을 나타내는 식별자 역할
    3. URL: 특정 자원의 위치값 (실제 파일이 있음)

    http://naver.com/index.html

    4. URI : Restful구조에서 특정자원을 나타내는 함수(실제 파일은 없음)

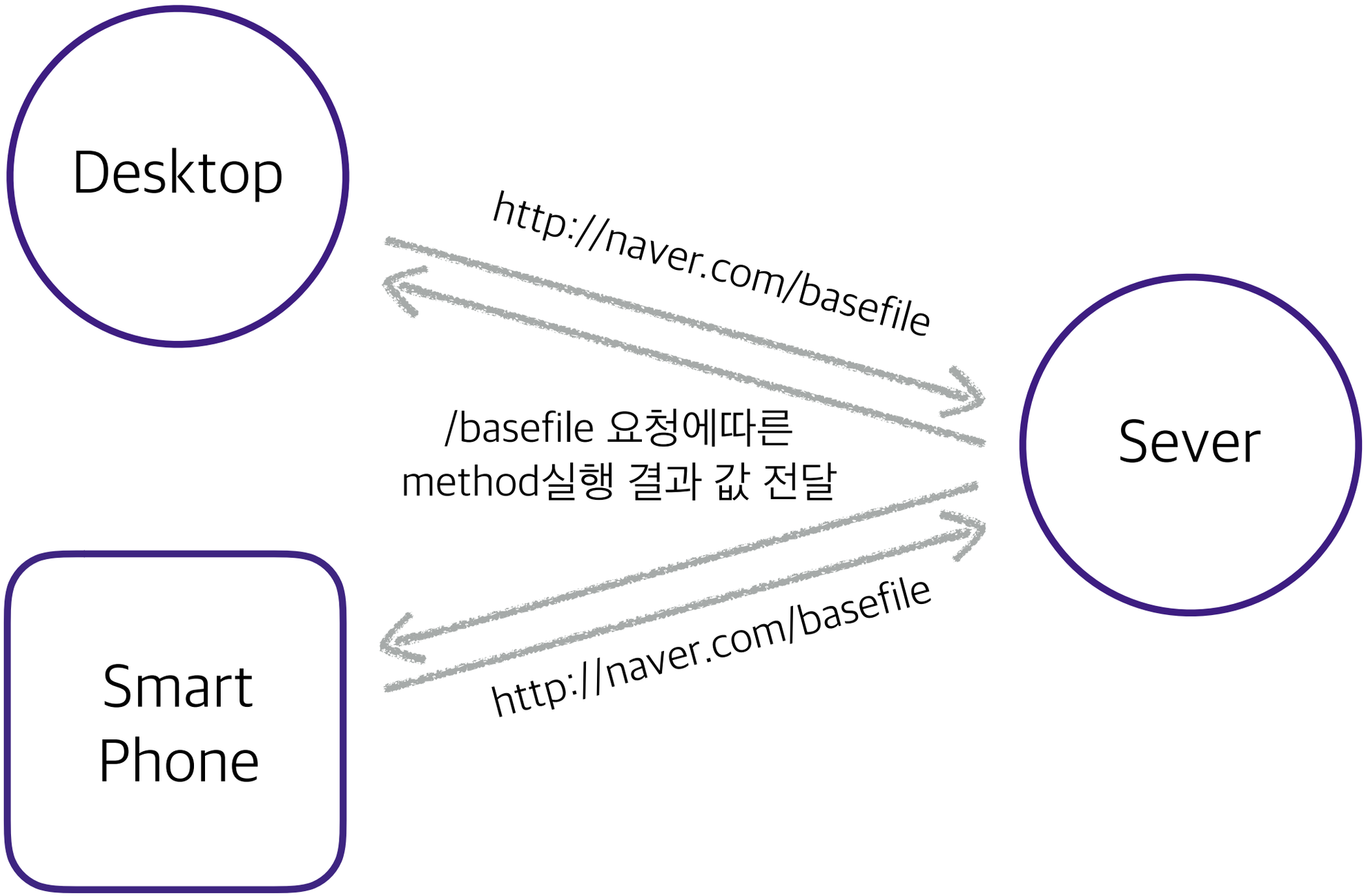
    http://naver.com/basefile

Rest

- REST(Representational State Transfer)는 월드 와이드 웹과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식이다.

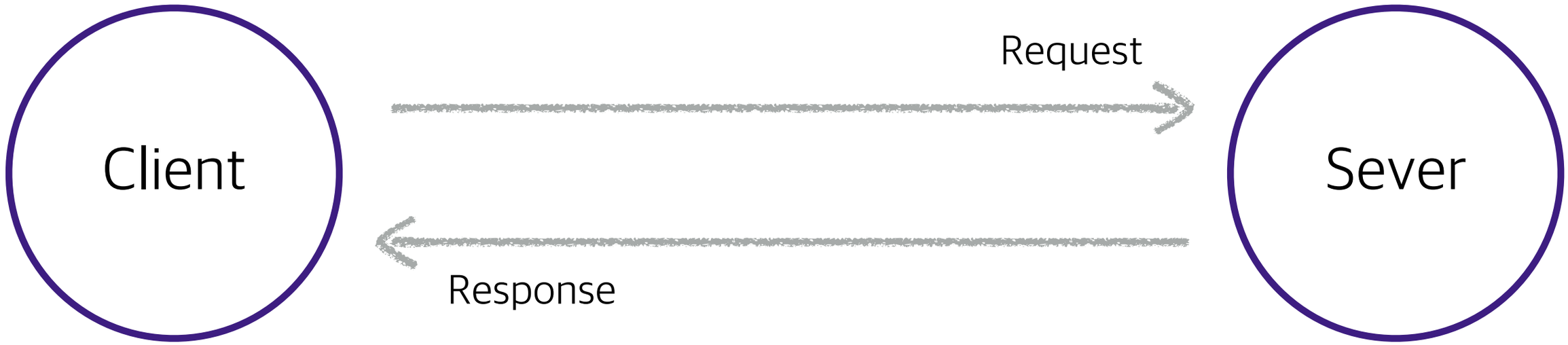
- 엄격한 의미로 REST는 네트워크 아키텍처 원리의 모음이다. 여기서 '네트워크 아키텍처 원리'란 자원을 정의하고 자원에 대한 주소를 지정하는 방법 전반을 일컫는다.

Rest



Rest

```
http://naver.com/basefile
```




```
app.get('/basefile', function(req, res) {
    if (req.user.divice == 아이폰)
    {
          res.send(m_index.hteml)
    }else
    {
          res.send(index.hteml)
    }
});
```

Rest 사용 원칙

- Resources : URI들은 쉽게 자원의 내용을 이해할수 있게 만들어야 한다.

- Representations : 객체와 속성을 나타내는 데이터를 **JSON**이나 XML구조로 전환해서 표현한다.

- Messages : **HTTP Methods**를 사용한다. (for example, GET, POST, PUT, and DELETE).

- Stateless : server와 clinet의 요청사항은 저장하지 않는다. client는 session 상태를 유지한다.

HTTP Request

- URI을 이용해서 Server에 데이터를 요청한다.
- 크게 header와 body로 구조를 나눌 수 있다.
- HTTP Method를 사용해서 요청 메세지를 보낸다.

HTTP Method	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Update/Modify
DELETE	Delete

참고 : http://goo.gl/xjH2ke

GET vs POST

- GET : 서버에 데이터를 요청하는 용도로 사용되는 Method
  간단한 데이터를 보내야 하면 URL뒤에 붙여서 보낸다.

www.example.com?id=mommoo&pass=1234

- POST : Request body에 데이터를 포함시켜서 서버에 보낸다.
  header에 contents-type을 추가, 보내는 타입을 명시.

HTTP Request 구조

Request-Line (ex: GET /index HTTP/1.1)
Header (general-header | **request-header** | entity-header)

[**message-body**]

Request Header

- Host : The domain name of the server (for virtual hosting), and the TCP port number on which the server is listening.
- Accept : Content-Types that are acceptable for the response. See Content negotiation.
- User-Agent : The user agent string of the user agent
- 기타 등등…

```
GET  /basefile   HTTP/1.1
Host: naver.com
Accept: text/plain
Content-Encoding "gzip"
```

Contents Type

- 서버, 클라이언트 간의 어떠한 데이터를 주고 받을수 있는지를 명시하는 타입

- 대표 Type 종류

1) Multipart Related MIME 타입
   - Content-Type: Multipart/related
2) XML Media의 타입
   - Content-Type: text/xml
   - Content-Type: Application/xml
3) Application의 타입
   - **Content-Type: Application/json**
   - **Content-Type: Application/x-www-form-urlencode**
4) 오디오 타입
   - Content-Type: audio/mpeg <-- MP3 or other MPEG audio
5) Multipart 타입
   - **Content-Type: multipart/formed-data** <-- 파일 첨부

…그외 다수

Application/x-www-form-urlencode

- 기본 데이터 타입
- Key, value로 이뤄져 있지만 Json과는 형태가 다르다

key=value&key=value

username=userid&password=1234456

Application/json

- JSON(JavaScript Object Notation)은 **속성-값 쌍으로 이루어진 데이터 오브젝트를 전달**하기 위해 인간이 읽을 수 있는 **텍스트를 사용하는 개방형 표준 포맷이다.**

{“key”:”value”,”key”:“value”}

{“username”:”userid”,"password”:“1234456”}

Multipart/formed-data type

- 복잡한 데이터 형식을 보낼때의 데이터 타입
- 파일 전송시 주로 사용

```
--YOUR_BOUNDARY_STRING

Content-Disposition: form-data; name="photo"; filename="calm.jpg"

Content-Type: image/jpeg



YOUR_IMAGE_DATA_GOES_HERE

--YOUR_BOUNDARY_STRING

Content-Disposition: form-data; name="message"



MESSAGEDATA

—YOUR_BOUNDARY_STRING—
```

Message Body

- GET HTTP Method의 파라미터는 URL에 포함시켜서 정보를 보낸다.
  ex) http://siteURI/age?firstName=joo&lastName=ym

- GET을 제외한 나머지 Method의 파라미터는 content-type에 맞는 형식으로 body message에 포함시켜 요청을 보낸다.

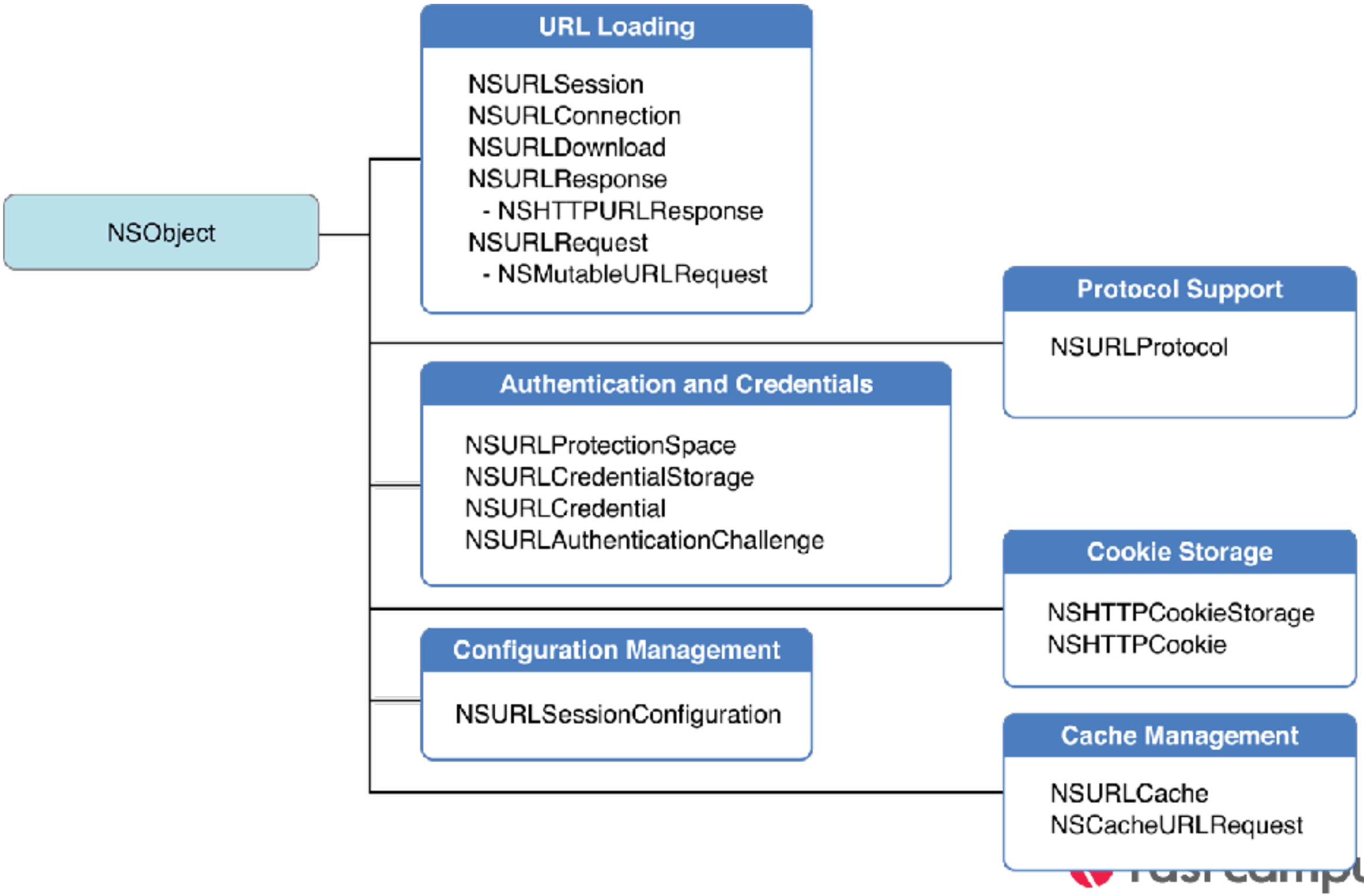
iOS Network

강사 주영민

URL Load System

- URL을 통해서 컨텐츠의 리소스를 받아올수 있는 가장 일반적인 방법으로 URLSession을 사용한다.

URL Load System 사용 Class



URLRequset

- URL Loading System에서 요청에 대한 추가 정보를 가지고 있는 클래스

- URL 및 프로토콜별 속성을 캡슐화 한 클래스

- 요청하는 정보에 대해서만 캡슐화 하고 있고, 서버에 요청하는 역할은 URLSession을 통해서 한다.

URLRequset - 주요 항목

```
public init(url: URL, cachePolicy: URLRequest.CachePolicy = default,
timeoutInterval: TimeInterval = default)

var cachePolicy: URLRequest.CachePolicy
var timeoutInterval: TimeInterval
var networkServiceType: URLRequest.NetworkServiceType

var httpMethod: String?
var httpBody: Data?
func addValue(_ value: String, forHTTPHeaderField field: String)
func setValue(_ value: String?, forHTTPHeaderField field: String)
```

URLSession

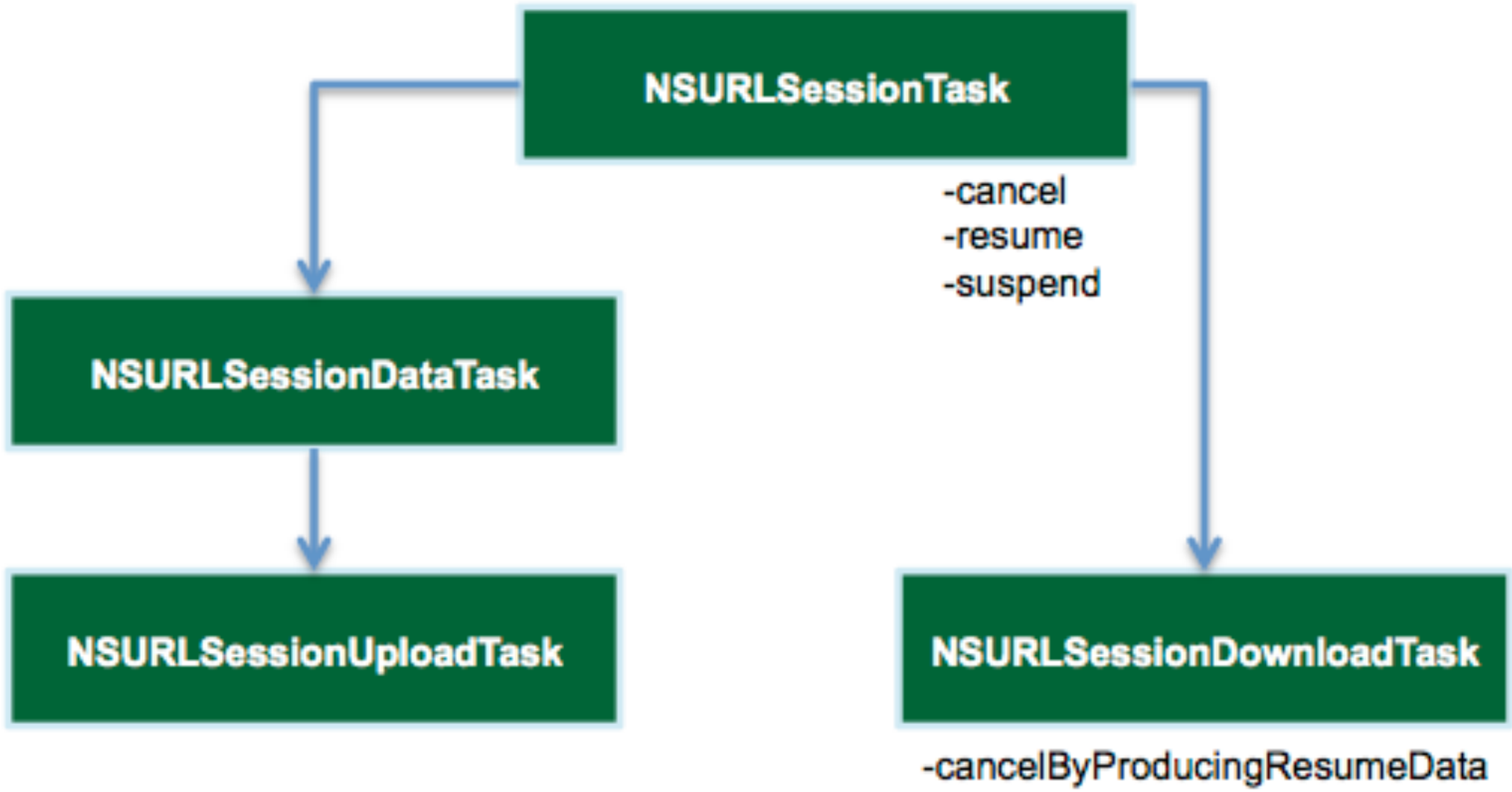
- URL Loading System에서 컨텐츠를 검색하는 가장 일반적인 클래스
- URLSession은 HTTP requests를 통해 데이터를 보내고, 받는데 API를 제공하는 클래스.
- APP이 실행되지 않은 상태에서도 백그라운드에서 Upload 및 Download 기능을 제공한다.
- 지원 가능 URL
    - File Transfer Protocol (`ftp://`)
    - Hypertext Transfer Protocol (`http://`)
    - Hypertext Transfer Protocol with encryption (`https://`)
    - Local file URLs (`file:///`)
    - Data URLs (`data://`)

URLSession - 주요 항목

```
class var shared: URLSession { get }
init(configuration: URLSessionConfiguration)


func dataTask(with request: URLRequest,
    completionHandler:@escaping(Data?, URLResponse?, Error?) -> Swift.Void)
        -> URLSessionDataTask

func dataTask(with url: URL,
    completionHandler:@escaping(Data?, URLResponse?, Error?) -> Swift.Void)
    -> URLSessionDataTask

func uploadTask(with request: URLRequest, from bodyData: Data?,
    completionHandler:@escaping(Data?, URLResponse?, Error?) -> Swift.Void)
    -> URLSessionUploadTask

func downloadTask(with request: URLRequest,
   completionHandler:@escaping(URL?, URLResponse?, Error?) -> Swift.Void)
   -> URLSessionDownloadTask
```

URLSessionTask

- URLSessionTask는 URLSession의 작업 하나(Task)를 나타내는 추상클래스로 URLSession을 통해서만 생성 가능
(ex : dataTask(with:)



URLSessionTask

- NSURLSessionDataTask: HTTP GET요청으로 서버로부터 데이터를 가져오는 작업을 수행합니다.

- NSURLSessionUploadTask: 디스크로부터 웹서버로 파일을 업로드하는 작업을 수행합니다. 주로 HTTP POST, PUT 메소드를 이용합니다.

- NSURLSessionDownloadTask: 서버로부터 파일을 다운로드하는 작업을 수행합니다.

URLSessionConfiguration

- URLSessionConfiguration은 Session의 **설정**에 관련된 클래스. 다음 3가지 중 하나 값으로 생성되며, 타임아웃, 캐시 정책등의 프로퍼티를 설정가능

1. default: 디폴트 configuration 객체를 생성합니다. 디폴드 값으로는 파일로 다운로드 될때를 제외하고는 Disk에 캐쉬를 저장하며, 키체인에 자격을 저장합니다.
2. ephemeral: 디폴트 configuration과 설정은 동일하다 session관련 데이터가 메모리에 올라갑니다.
3. background: Session이 백그라운드에서 다운로드 작업과 업로드 작업을 마져 수행할 수 있도록 합니다.

네트워크 실행 순서

1. URLRequest 인스턴스 생성. (요청생성)

2. URLSession 인스턴스 생성.
   (*URLConfiguration 설정은 옵션)

3. URLSession의 메소드를 통해 URLSessionTask 생성

4. Task 실행 : 네트워크 요청

5. 요청한 task에 대한 응답(respond)처리 (Delegate or Closure)

6. JSONSerialization or JSONDecoder(Swift 4)를 통해 알맞은 데이터 인스턴스로 변환

URLRequest 인스턴스 생성. (요청생성)

```
let url = URL(string:"http://naver.com")!
//request instanse
var request = URLRequest(url: url)
//config = default
request.httpMethod = “POST"
//request.httpMethod = “GET”
//bodyData
let dataStr = "{username:\(userID),password:\(pw)}"
let data = dataStr.data(using: .utf8)
request.httpBody = data
```

URLSession 인스턴스 생성.

```
let url = URL(string:"http://naver.com")!
//request instanse
var request = URLRequest(url: url)
//config = default
request.httpMethod = “POST"
//request.httpMethod = “GET”
//bodyData
let dataStr = "{username:\(userID),password:\(pw)}"
let data = dataStr.data(using: .utf8)
request.httpBody = data

let urlSession = URLSession.shared
```

URLSessionTask 생성

```
let url = URL(string:"http://naver.com")!
//request instanse
var request = URLRequest(url: url)
//config = default
request.httpMethod = “POST"
//request.httpMethod = “GET”
//bodyData
let dataStr = "{username:\(userID),password:\(pw)}"
let data = dataStr.data(using: .utf8)
request.httpBody = data

let urlSession = URLSession.shared

urlSession.dataTask(with: request) { (data, response, error) in



}.resume()
```

JSONSerialization로 Dictionary만들기

```
urlSession.dataTask(with: request) { (data, response, error)
in
    //바로 보내주기
    if let data = data
    {
        do {
            let dic = try JSONSerialization.jsonObject(with:
data, options: []) as! [String: Any]
            //..

        } catch let error {
            print("\(error.localizedDescription)")
        }
    }
}.resume()
```

Codable Data Model

```
struct UserModel:Codable {
    var name:String
    var userID:String


    enum CodingKeys: String, CodingKey {
        case userID = "userid"
        case name
    }

    func encode(to encoder: Encoder) throws
    {
        var container = encoder.container(keyedBy:
CodingKeys.self)
        try container.encode(userID, forKey: .userID)
        try container.encode(name, forKey: .name)
    }
}
```

JSONDecoder & JSONEncoder

- Swift4에서 사용 가능
- 데이터 모델 인스턴스를 encoding과 decoding을 통해 바로 json으로 포팅이 가능
- 데이터 모델에서 Codable Protocol 채택 후 사용

Json >> Dictionary

기존

```
do {
    let dic = try JSONSerialization.jsonObject(with: data,
                                               options: []) as! [String: Any]
    let user = UserModel(dicData:dic)
} catch let error {
    print("\(error.localizedDescription)")
}
```

Swift4

```
let decoder = JSONDecoder()
do {
    let user = try decoder.decode(UserModel.self, from: data)

} catch let error {
    print("\(error.localizedDescription)")
}
```

Json >> Array

기존

```
do {
    let list = try JSONSerialization.jsonObject(with: data,
                                                options: []) as! [[String: Any]]
    for dic in list{
        let user = UserModel(dicData:dic)
        users.append(user)
    }
}
```

Swift4

```
let decoder = JSONDecoder()
do {
    let users = try decoder.decode([UserModel].self, from: data)

} catch {
    print("\(error.localizedDescription)")
}
```

Codable Data Model - JSONDecoder

```
struct UserModel:Codable {
    var name:String
    var userID:String

    //CodingKeys를 통해 Customizing Key Names을 정할수 있다.
    //모든 항목을 case로 만들어야 한다.
    enum CodingKeys: String, CodingKey {
        case userID = "userid"
        case name
    }

    func encode(to encoder: Encoder) throws
    {
        var container = encoder.container(keyedBy:
CodingKeys.self)
        try container.encode(userID, forKey: .userID)
        try container.encode(name, forKey: .name)
    }
}
```

Codable Data Model - JSONEncoder

```
struct UserModel:Codable {
    var name:String
    var userID:String


    enum CodingKeys: String, CodingKey {
        case userID = "userid"
        case name
    }
    //encoding시 만들어질 key값 설정
    func encode(to encoder: Encoder) throws
    {
        var container = encoder.container(keyedBy:
CodingKeys.self)
        try container.encode(userID, forKey: .userID)
        try container.encode(name, forKey: .name)
    }
}
```

JSONEncoder

기존

```
let url = URL(string:"http://naver.com")!
//request instanse
var request = URLRequest(url: url)
//config = default
request.httpMethod = “POST"
//bodyData
let dataStr = "{username:\(userID),password:\(pw)}"
let data = dataStr.data(using: .utf8)
request.httpBody = data
```

Swift4

```
let url = URL(string:"http://naver.com")!
//request instanse
var request = URLRequest(url: url)
//config = default
request.httpMethod = “POST"
//bodyData
let encoder = JSONEncoder()
let data = try! encoder.encode(user)
request.httpBody = data
```

예제

- 날씨 정보 받아오기

Using URL Request

```
NSURL *thumbnailURL = [NSURL URLWithString:@"url string"];

NSURLSession *session = [NSURLSession sessionWithConfiguration:
                         [NSURLSessionConfiguration defaultSessionConfiguration]];
NSURLSessionTask *task = [session dataTaskWithURL:thumbnailURL
                            completionHandler:^(NSData * _Nullable data,
                                                NSURLResponse * _Nullable response,
                                                NSError * _Nullable error) {
                                if (data) {
                                    UIImage *image = [UIImage imageWithData:data];
                                    if (image) {
                                        //UI변경 코드
                                        //변경을 위해 GCD사용
                                    }
                                }
}];

[task resume];
```

실습 : 구글에 있는 이미지 다운받아 보기

Let’s play coding

보안

- 스누핑 - 제 3자가 전송중인 프로그램의 데이터를 스니핑하는 공격.
- 중간자 (Man-in-the-middle) 공격 - 제 3자가 프로그램과 서버 사이에 자체 컴퓨터를 삽입하는 공격. 중개자 공격에는 다음이 포함됩니다.

스푸핑 및 피싱 - 합법적 인 서버로 가장 한 가짜 서버 만들기.
조작 - 서버와 프로그램 간의 데이터 수정.
세션 하이재킹 - 인증 정보를 캡처하여 사용자로 사용합니다.

TLS

- TLS (Transport Layer Security) 프로토콜은 서버 및 (선택적으로) 클라이언트의 인증과 함께 소켓 기반 통신을 위한 데이터 암호화를 제공하여 스누핑을 방지합니다.

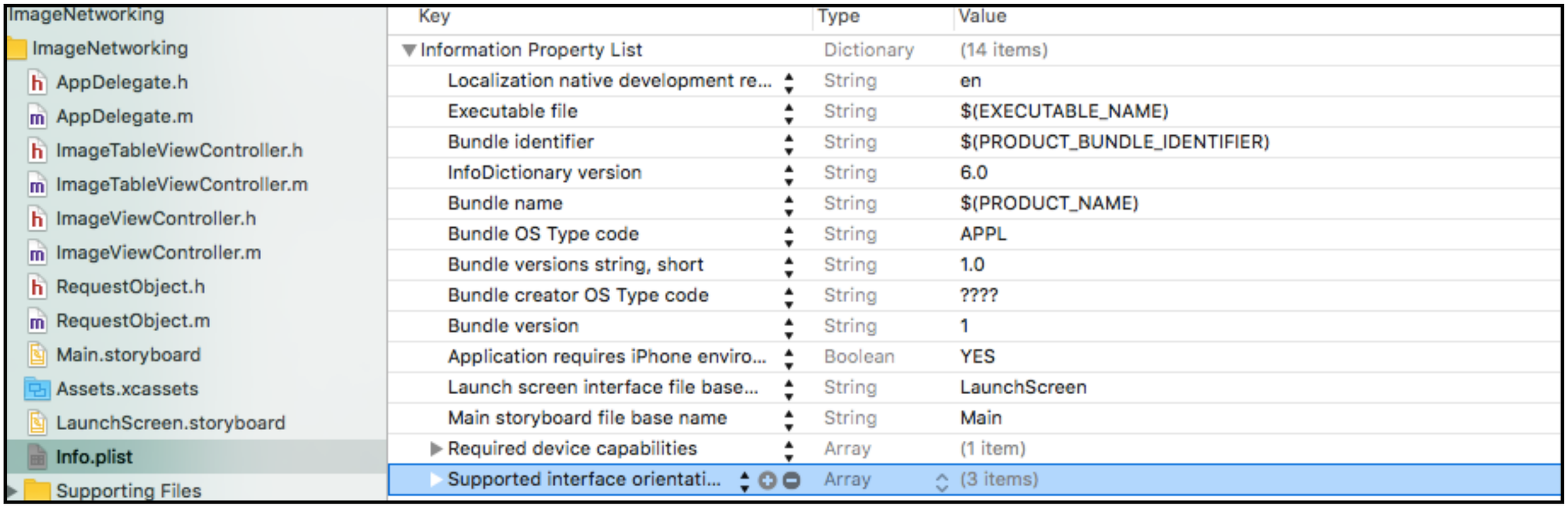
- http대신 https를 사용하면 자동으로 TSL를 사용하여 접속

ATS

- ATS(App Transport Security)
- iOS9 이상의 버전에는 ATS 기술이 기본적으로 적용된다. ATS는 앱과 웹 서비스 간 연결보안을 강화하는 기술로 이 기술이 적용되면 기존에 iOS 앱에서 사용하던 암호화되지 않은 HTTP 통신은 OS 내부에서 강제적으로 차단된다.

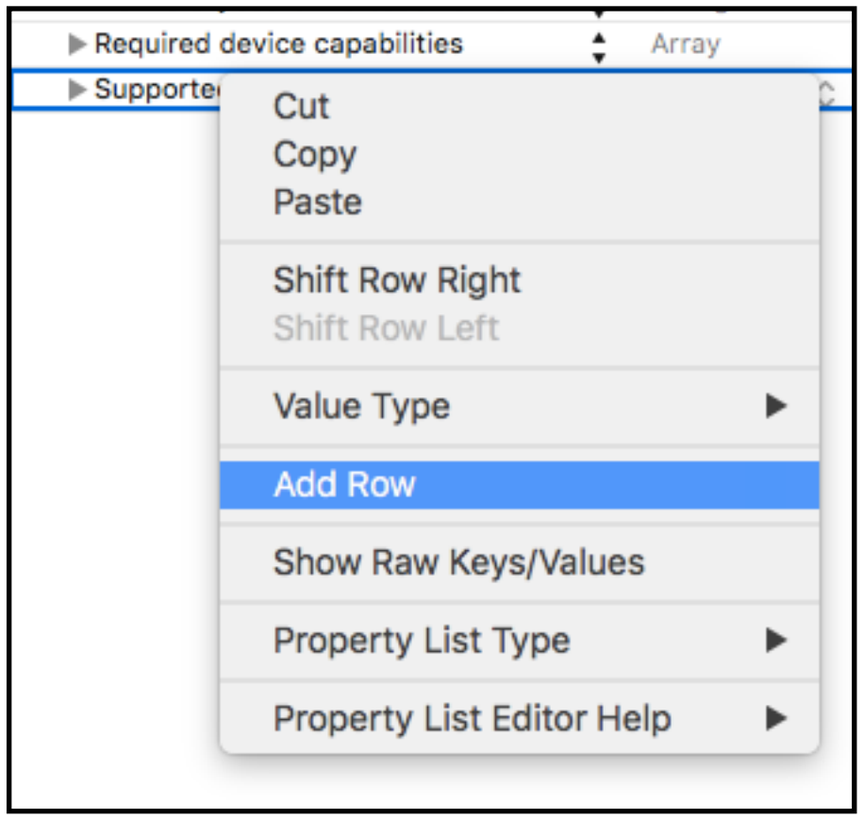
ATS 해제 - http request 허용

1. info.plist 파일 열기

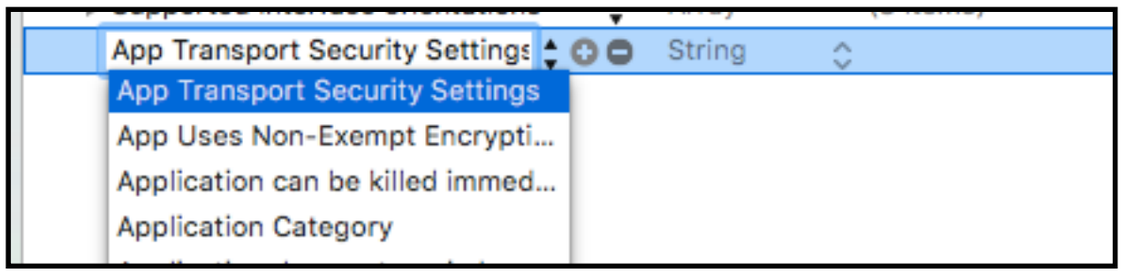


ATS 해제 - http request 허용

2. 새로운 Row 추가



3. App Transport Security Settings 항목 추가



ATS 해제 - http request 허용

4. Allow Arbitrary loads 항목 추가

5. Yes로 변경 : ATS해제

Multipart form

강사 주영민

Data Form

```
--YOUR_BOUNDARY_STRING

Content-Disposition: form-data; name="photo"; filename="calm.jpg"

Content-Type: image/jpeg



YOUR_IMAGE_DATA_GOES_HERE
```

<< Contents Disposition

```
--YOUR_BOUNDARY_STRING

Content-Disposition: form-data; name="message"



My first message
```

<< Contents Disposition

```
--YOUR_BOUNDARY_STRING

Content-Disposition: form-data; name="user"



1
```

```
—YOUR_BOUNDARY_STRING—
```

<< End Line

Contents Disposition

```
--YOUR_BOUNDARY_STRING
Content-Disposition: form-data; name="photo"
Content-Type: image/jpeg        (일반 스트링이면 생략 가능)

YOUR_IMAGE_DATA_GOES_HERE       (실제 데이터)
```

예제

```
NSString *boundary = @“YOUR_BOUNDARY_STRING”;

NSMutableData *body = [NSMutableData data];

[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];

[body appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data; name=\"photo\";
filename=\"%@.jpg\"\r\n", self.message.photoKey] dataUsingEncoding:NSUTF8StringEncoding]];

[body appendData:[@"Content-Type: application/octet-stream\r\n\r\n"
dataUsingEncoding:NSUTF8StringEncoding]];

[body appendData:[NSData dataWithData:imageData]];

[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];

[body appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data;
name=\"message\"\r\n\r\n%@", self.message.message] dataUsingEncoding:NSUTF8StringEncoding]];



[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];
```

추가로 더 알아보기

- https://goo.gl/gxE9Ce