




발표자료는 다음 위치에서 받으실 수 있습니다:
<http://kivol.net/playnode2017.pdf>

GraphQL in Action

REST와 이별할 때 생각해야 하는 것들



Ep 0. 고통의 시대

디자인 vs DB

인간 vs 컴퓨터

프로덕트의 **디자인**이 바뀌면

데이터의 **저장방식**이 바뀌어야 할까?

아니면 **프로그래머**가 그 중간에서 **고통**받아야 할까?

서버란 무엇인가

변화무쌍한 비즈니스 로직을 처리
하면서도 저장된 데이터 구조의 변경은 최소화
하면서도 클라이언트가 쓰기 쉬워야
하면서도 일정 이상의 퍼포먼스가 보장되어야

.....

서버란 무엇인가

요약하자면,

결국 클라이언트의 인터페이스를 결정하는 문제

네트워크를 거친다는 사실을 빼면,

컴포넌트의 인터페이스를 설계하는 문제

XML이 우리를 구원해 주실거야

XML-RPC

RPC: Remote Procedure Call

서버의 특정 메소드를 실행하고 리턴값을 받는 방식의 표준
매우 일반적이고 domain-specific하지 않음

자매품: JSON-RPC

SOAP

Simple Object Access Protocol

필요한 도메인(프로덕트)의 구조에 맞춘 XML

어때요, 참 쉽죠?

```
POST /Quotation HTTP/1.0
Host: www.xyz.org
Content-Type: text/xml; charset = utf-8
Content-Length: nnn

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

  <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotations">
    <m:GetQuotation>
      <m:QuotationsName>MiscroSoft</m:QuotationsName>
    </m:GetQuotation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


A hammock with blue and green stripes is hanging in a lush green field. The background is a dense forest of green trees, and the foreground shows a field of green grass with some purple flowers. The hammock is made of a woven fabric and has a wooden spreader bar. The ropes are light-colored and are attached to trees on either side.

Ep 1. REST의 등장

객체지향이란 무엇인가

하나의 개념을 **구조**(모델)과 **동작**(메소드)으로 구성

XML-RPC: 메소드(Procedure)를 서버에 **전가**

SOAP: **정의**하기 나름

→ 변화무쌍한 비즈니스로직 대응은 여전히 **고통**

관심사의 분리

서버는 **모델과 DB**의 구조화에만 집중

서버는 데이터를 저장된 **구조 그대로** 뿌려주고
클라이언트가 변화무쌍한 **비즈니스 로직**을 처리하게?

REST API

어디에서나 쉽게 파악할 수 있는 **단일한** 인터페이스
: **Uniform** Interface

[동작] /[리소스 구조]/id

ex/ GET /users/posts/42

REST API

쉽게 모든 리소스에 대한 자세한 동작을 정의
간단하고 배우기 쉽다 = 좋은 인터페이스
많은 OpenAPI들이 채용

Ep 2. REST시대의 통증의학



N+1 쿼리 문제

Q: 어떤 유저가 쓴 **모든 글의 내용**을 한 번에 보고 싶어요

초보: 일단 유저가 쓴 모든 글을 가져오고,

`/users/42`

그 내용의 리스트를 돌며 **한 번씩 fetch**

`/articles/11, /articles/22, /articles/33,`

아 그건 퍼포먼스가 좀...

Q: 어떤 유저가 쓴 모든 글의 내용을 한 번에 보고 싶어요

SQL맨: 나는 경기도 안양의 SQL맨이다!

```
SELECT a.content  
FROM articles as a, users as b  
WHERE a.user_id=b.id AND b.id=42
```



이 **REST**맨이 왔으니 안심하라구!

날퀴리는 관리가 안되잖아요...

Q: 어떤 유저가 쓴 모든 글의 내용을 한 번에 보고 싶어요

REST맨: 그냥 users 리소스에 전부 다 담으면 된단다!

```
GET /users/42
```

```
{ "id": 42, "name": "foo", "articles": [  
  { "id": 11, contents: "bar"},  
  { "id": 22, contents: "baz"},  
  { "id": 33, contents: "qux"} ]}
```




모든 정보가 항상 필요한건 아닌데...

Q: **필요한** 필드만 가져오고 싶어요. 트래픽과 퍼포먼스
낭비가 너무 심해요. **Pagination**을 하고 싶어요.

REST맨: Query Parameter를 이용하면 되지!

```
GET /users/42?field=id,name,picture,articles(page=3)
```

CRUD에서 벗어나는 행위

Q: 유저가 포인트 쿠폰을 쓰면 포인트를 올려주고 싶어요.

이럴 때는 PUT인가요 DELETE인가요?

REST 초보: **두 번** 하면 됩니다!

PUT /users/42

DELETE /coupons/33

클라에서 그런걸 하면 보안이...

Q: 유저가 포인트 쿠폰을 쓰면 포인트를 올려주고 싶어요.

이럴 때는 PUT인가요 DELETE인가요?

REST맨: 보안은 어떻게 **때운다** 치고 **배치**를 만들면 되지!

```
POST /batch
```

```
[{ "uri": "users/42", "method": "PUT"},  
{"uri": "coupons/33", "method": "DELETE"}]
```



꼬리를 무는 질문들

Q: 어떤 모델의 필드가 배열이어야 하는데, 그 배열 안에 다양한 모델이 들어가야 하는 경우는 어떻게 URL을 설계해야 하나요?

Q: 같은 리소스에 여러 가지의 특정한 행위를 구체적으로 요청하려면 어떻게 설계해야 하나요?

Q: 이런 경우는 어떻게 URL을 설계해야 RESTful한가요?

.....

이게 RESTful한게 맞긴 한가요?

애초에 우리는 왜
RESTful하기를 원했을까요?



Ep 3. 자유의 날개

GraphQL의 역사

2015년 Facebook이 발명

...했지만 페이스북도 아직은 REST API만 제공

Facebook Graph API와는 다릅니다

Graph 자료구조와 크게 관계는 없음

Github이 GraphQL API를 채택



하나의 Endpoint

하나의 Endpoint에 **모든** CRUD,
혹은 그에 속하기 애매한 **모든 행위**를 담음
POST의 경우 HTTP **Payload**를 적극적으로 활용

application/json

application/**graphql**

바로 이런 식으로

정의

```
type Article {  
  name: String  
  content: String  
  comments: [Comment]  
}
```

요청

```
{  
  article(name: "GraphQL") {  
    content  
  }  
}
```

응답

```
{  
  "article": {  
    "content": "GraphQL is awesome!"  
  }  
}
```


뭔가 고치고 싶다면?

정의

```
mutation CreateCommentForArticle($by: User!, $content: String!) {  
  createComment(by: $by, content: $content) {  
    content  
  }  
}
```

요청

```
{  
  "by": { "id": "42", "name": "foo" },  
  "content": "GraphQL is awesome indeed!"  
}
```

N+1 쿼리 문제는?

Q: 어떤 유저가 쓴 모든 글의
내용을 한 번에 보고 싶어요

GraphQL맨: 이렇게 간단히!

```
{  
  user(id: "42") {  
    articles {  
      id  
      content  
    }  
  }  
}
```

조건이 복잡해진다면?

Pagination

```
{  
  articles(first: 10) {  
    content  
    cursor  
  }  
}
```

더 복잡한 쿼리

```
type Query {  
  searchArticles(includes: String!, from: Int): [Articles]  
}
```

A hand is shown placing a white block onto a pyramid of white blocks. The pyramid is composed of several layers of blocks, with the top layer having two blocks. The hand is positioned above the pyramid, and the block being placed is held between the thumb and index finger. The background is a plain, light-colored wall.

Ep 4. 구현체

graphql.js

Facebook의 레퍼런스 구현체(=파서+etc.)

→ 무려 레퍼런스가 node.js입니다!

스키마와 실제 resolver로 express 서버 구현

```
type Article {  
  name: String  
  content: String  
  comments: [Comment]  
}
```

```
fields: {  
  comments: {  
    resolve: (parent, args) => { // }  
  }  
}
```


GraphQL Client

물론 curl이나 Postman으로 보낼 수도 있지만..

GraphiQL: 레퍼런스 GUI 클라이언트

스키마의 모든 계층구조를 보여줍니다

GraphQL Client

GraphQL

Prettify

1 query {
2 problem(id: "KR-PB-0000058543") {
3 id
4 unit {
5 name
6 }
7 }
8 }

QUERY VARIABLES

{
"data": {
"problem": {
"id": "KR-PB-0000058543",
"unit": {
"name": "이등변삼각형의 구성 요소 알기"
}
}
}
}

< Schema Query X

Q Search Query...

No Description

FIELDS

product(id: ID!): Product
curriculum(id: ID!): Curriculum
chapter(id: ID!): Chapter
lesson(id: ID!): Lesson
unit(id: ID!): Unit
chapterTest(id: ID!): ChapterTest
drillTest(id: ID!): DrillTest
levelTest(id: ID!): levelTest
partTest(id: ID!): PartTest
pattern(id: ID!): Pattern
video(id: ID!): Video
problem(id: ID!): Problem
tutorial(id: ID!): Tutorial



Apollo

React, **Vue** 등 플랫폼과 자연스러운 연동
쿼리, 캐싱, Mutation, 서버사이드 렌더링 등 쉽게 가능

웹소켓 연동서버

Express 외에도 다양한 **서버** 지원

node.js 외에도 다양한 **언어** 지원

GraphCool

Firestore와 같은, GraphQL을 위한 **BaaS**

클라우드 기반의 **서버리스** 호스팅

GUI 기반의 스키마 모델링과

GUI 기반의 DB **편집**

(다른 오픈소스도 잘하는 회사예요!)

Ep 5. 리얼 월드

Resolver

```
let graphql = require('graphql');

module.exports = new graphql.GraphQLObjectType({
  name: 'Query',
  fields: () => ({
    article: {
      type: articleType,
      args: {
        id: { type: new graphql.GraphQLNonNull(graphql.GraphQLID) }
      },
      async resolve(parent, args) {
        let data = await query(`SELECT * FROM articles where id=${args.id}`);
        // ...
      }
    },
  }),
});
```

1:N 관계인 경우?

```
let articleType = new graphql.GraphQLObjectType({
  name: 'Article',
  fields: () => ({
    id: { type: new graphql.GraphQLNonNull(graphql.GraphQLID) },
    name: { type: new graphql.GraphQLNonNull(graphql.GraphQLString) },
    comments: {
      type: new graphql.GraphQLList(commentType),
      async resolve(parent, args) {
        let data = await query(`SELECT * FROM comments
          WHERE article_id = ${parent.id}`);
      }
    }
  })
});
```


더 복잡한 쿼리

```
query {  
  user(id: 42) {  
    followers {  
      followers {  
        followers {  
          id  
        }  
      }  
    }  
  }  
}
```

복잡한 계층구조

N+1을 아득히 넘어서는

수많은 쿼리 필요

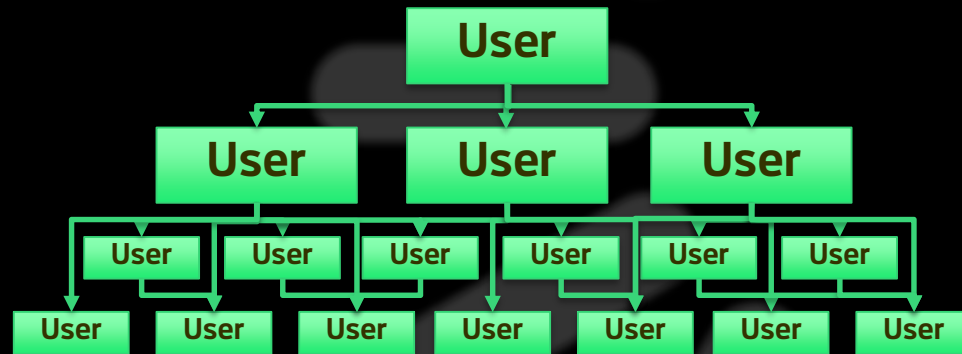
DataLoader

Written by Facebook

이미 fetch한 데이터는 캐싱된 것을 이용
같은 모델의 여러 개의 id를 가져와야 할 때 모아서 fetch
(한 리퀘스트 내에서)

모델에 id와 같은 PK가 있다고 가정

DataLoader



수 많은 쿼리



정리된 쿼리

Resolver의 대체

```
let myBatchGetter = async function(keys) {  
    return await query(`SELECT * FROM mine WHERE id IN (${keys.join(',')})`);  
};  
  
let myDataLoader = new DataLoader(keys => myBatchGetter(keys));  
  
resolve(parent, args) {  
    return myDataLoader.load(args.ids);  
}
```

A close-up, low-key photograph of a person's hand holding a black semi-automatic handgun. The hand is positioned on the left side of the frame, with the thumb resting on the trigger guard. A silver-toned metal bracelet is visible on the wrist. Above the handgun, a single bullet is captured in mid-air, having just been fired. The background is dark and out of focus, with a hint of a greenish-grey wall or surface. The overall mood is serious and focused.

Ep 6. 트러블슈팅

스키마의 상속

강한 의미의 상속은 없지만,
Union이나 Interface를 사용하여 모델의 분류 가능

```
union SearchResult = Article | Comment | Notice

type Query {
  search(text: String!): [SearchResult]!
  ...
}
```

```
interface Searchable {
  searchPreviewText: String!
}

type Article implements Searchable {
  searchPreviewText: String!
  ...
}
```

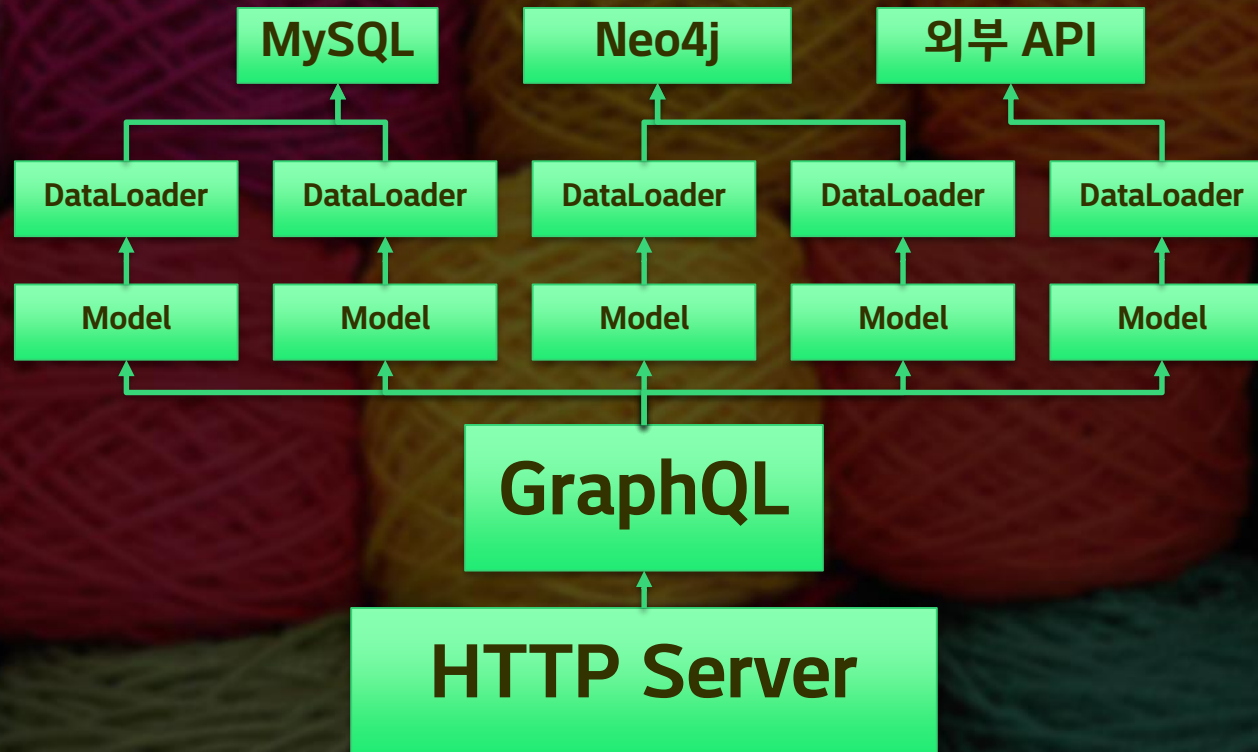

Recursive Query

```
{  
  messages {  
    ...CommentsRecursive  
  }  
}  
  
fragment CommentFields on Comment {  
  id  
  content  
}
```

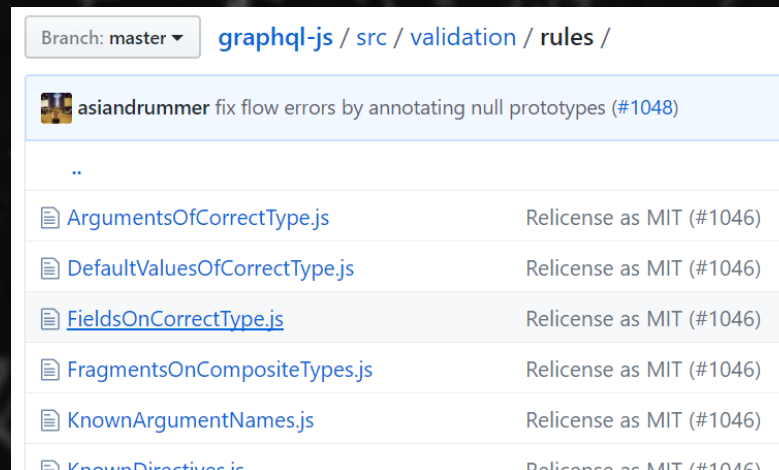
```
fragment CommentsRecursive on Message {  
  comments {  
    ...CommentFields  
    comments {  
      ...CommentFields  
      comments {  
        ...CommentFields  
      }  
    }  
  }  
}
```

완전한 Recursive Query는 불가능
정말 필요하다면 차라리 테이블 **전체**를 Fetch

다양한 DB와의 연결



공격: 복잡한 쿼리



Query validation & timeout

: 유효성을 확인하고 오래 걸리는 쿼리를 기록 or 거부

공격: 복잡한 쿼리

Query whitelisting

: 실제로 제품에서 쓸 쿼리의 DB를 만들어 놓고 관리

극단적으로는 쿼리가 아니라 쿼리의 ID를 날리기도
빌드시스템을 통한 자동화도 가능

공격: 인젝션

```
resolve: async (parent, args) => {  
  return await query(`SELECT * FROM articles WHERE name = '${args.name}';`);  
}
```

```
{  
  articles(name: "foo'; DROP TABLE articles; --") {  
    id  
  }  
}
```

여전히 Input Sanity 보장은 중요합니다
Validation이 방법 중 하나일 수 있습니다



Ep 7. 못 다한 이야기들

특허 이슈가 있다던데?

[기존]

BSD + **Patents**

방어적 특허라고는 하지만...

React 등과 함께 문제가 됨

[지금]

MIT

특허 **모두 포기**

Clear!!??

몇 가지 단점이나 오해들

Information Hiding
문제

필요한 필드만 스키마에 정의

모델의 버저닝을
어떻게?

버저닝은 되도록 하지 않습니다

@include(if: true), @skip 등의 Directive로 하위호환 보장

HTTP 캐싱이 어렵다

단점 맞습니다

필요한 경우 앞단에 Redis등의 캐싱

몇 가지 단점이나 오해들 (2)

스키마를 알아야 하면
SOAP과 뭐가 다름?

훨씬 직관적이고 문서화가 쉽습니다
REST도 결국 스키마는 알아야 했습니다

트래픽 사이즈 커짐
새로 배워야 함

인정? 어 인정!

Verification이
결국 문제가 되지 않나?

보안문제는 언제나 존재해왔고 우리의 숙명입니다
SI가 우리의 직업을 빼앗을 때까지는 받아들입니다

기존 REST API는 어떻게?

Step 1

테이블 JOIN이 필요한 로직을 제거합니다

Step 2

스키마를 정리하고 GraphQL 인터페이스를 만듭니다

Step 3

클라이언트가 바라보는 엔드포인트를 점진적으로 바꿔나갑니다

사실 저희 회사는 이렇게는 안 하고 한 번에 넘겼습니다
(컨텐츠 관련 내부 마이크로서비스 API)

REST는 죽을까요?

당분간은 **아직**.....

Facebook **본인**들도 아직은 REST API만 엽니다

하지만 **GraphQL**이 고통을 크게 덜어주는 것은 사실!

그리고 원모어뽕

우리와 함께하실 분을 찾습니다

프로덕트 엔지니어링
총괄 디렉터

큰 그림을 설계하고 관리하실 경험 많은 엔지니어

node.js 주니어
개발자

프로덕트 서버를 맡을 신선한 엔지니어

안드로이드 주니어
개발자

안드로이드 클라이언트를 맡을 신선한 엔지니어

Wanted 사이트를 참고해 주세요! (www.wanted.co.kr)

:wq!