통 합 구 현

REST 이해

한국기술교육대학교
온라인평생교육원

학습내용

- REST 이해
- REST API 설계

학습목표

- REST 의 정의 및 구성 요소, 특성에 대해 설명할 수 있다.
- REST API를 설계할 수 있다.

1. REST 이해

1) REST 기본

(1) REST

① 정의

REST의 정의

- Representational Safe Transfer 의 약어로 웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍처 (네트워크 상의 자원을 정의하고 자원에 대한 주소를 지정하는 방법)

- Stateless 방식의 Client/server 구조 지원
- HTTP와 JSON을 함께 사용해 OPEN API를 구현하는 방법으로 주로 활용

② 구성요소

- 리소스, 메서드, 메시지

예시

"이름이 tom인 사용자를 생성한다."

리소스 : 사용자 (URI표현)
메서드 : 생성한다 (POST)
메시지 : 이름이 tom

리소스 표현

HTTP POST , http://myweb/users/
{
"users":{
"name":"tom"
}
}

1. REST 이해

2) REST 구성 요소 및 예제

(1) 구성 요소

① 메서드

- 행위에 대한 메서드로 HTTP 메서드를 그대로 사용
- HTTP의 다양한 메서드 중 CRUD에 해당하는 4개의 메서드만 사용

메서드	의미	Idempotent
POST	Create	No
GET	Read	Yes
PUT	Update	Yes
DELETE	Delete	Yes

- Idempotent(멱등성) : 여러 번 수행해도 결과가 같은 경우
  - Idempotent 하지 않은 메서드의 경우 API 호출이 실패하였을 때, 재호출 시 기존 상태를 복구한 상태에서 호출을 시도해야 하는 등 트랜잭션에 대한 처리에 주의 필요
  - stateless 서비스 구현에 필요한 원칙

1. REST 이해

2) REST 구성 요소 및 예제

(1) 구성 요소

② 리소스

- REST는 모든 것을 리소스, 즉 명사형으로 표기
- 명사형으로 표기한 리소스와 ID로 표기한 세부 리소스로 정의됨
  - 사용자라는 리소스 타입 : http://myweb/user
  - 사용자중 tom 라는 ID를 가진 리소스는 http://myweb/user/tom 로 정의
- 리소스를 URI로 정의하고 HTTP 메소드를 이용해서 CRUD 명령 구현
  - “푸시 메시지를 보낸다” 라는 형태보다 “푸시 메시지 요청을 생성한다” 형태로 명령 구현

예시

- **/web/sendmessage (x) - 동사형 사용**
- **POST/web/message (o) - HTTP 메서드 이용 및 명사형 사용**

1. REST 이해

2) REST 구성 요소 및 예제

(2) REST API 예제

① 간단한 URI로 리소스를 정의하고, HTTP 메시지를 이용해 메소드를 구현하며, 메시지는 JSON 등으로 표현해 HTTP Body에 기입해서 전송

- 사용자 생성
  - 사용자 이름과 주소 내용(메시지)를 해당 리소스에 생성(POST)

```
HTTP POST, http://myweb/users/
{
        "name" : "tom",
        "address" : "seoul"
}
```

- 사용자 조회
  - 사용자 리소스 중에서 이름에 대한 사용자 정보 조회(GET)

```
HTTP GET, http://myweb/users/tom
```

- 업데이트
  - 사용자 리소스에서 사용자 이름에 대한 주소 정보 수정(PUT)

```
HTTP PUT, http://myweb/users/tom
{
        "name" : "tom",
        "address" : "cheonan"
}
```

- 삭제
  - 사용자 정보를 삭제(DELETE)

```
HTTP DELETE, http://myweb/users/tom
```

1. REST 이해

3) REST 특성

(1) 특성

① 유니폼 인터페이스(Uniform Interface)

- HTTP 표준에만 따른다면 어떠한 기술이든 사용할 수 있는 인터페이스 스타일(특정 언어나 기술에 종속 받지 않고 사용가능)

② 무상태성(Stateless)

- 컨텍스트 저장소에 HTTP 세션과 같은 상태 정보를 저장하지 않는 형태
- 요청에 대한 메시지만 처리하며, 세션 등 컨텍스트 정보를 고려할 필요가 없어 구현이 단순해짐

③ 캐시 가능(Cacheable)

- 웹 표준을 사용하기 때문에 웹의 인프라를 그대로 활용할 수 있음
- HTTP 기반인 로드 밸런서, SSL, 캐싱 등을 적용할 수 있음
  - 캐싱 : 서버에 트랜잭션 발생이 감소하여 성능 향상에 기여

예시

클라이언트와 서버간에 Last-Modified 값을 이용하여 캐싱을 처리

[Last Modified 필드를 이용한 캐싱 처리 방식]

1. REST 이해

3) REST 특성

(1) 특성

④ 자체 표현 구조(Self-descriptiveness)

- REST API는 API 메시지만 보고도 이를 이해할 수 있는 직관적인 구조
- 메시지 포맷인 JSON 도 {key: value} 형태로 직관적으로 이해할 수 있는 구조
- REST 기반의 Open API는 최소한의 문서로도 API를 이해할 수 있도록 디자인

⑤ 클라이언트 서버 구조

- REST 서버 : API 제공하며 제공된 API 이용해 비즈니스 로직 처리 및 저장
- 클라이언트 : 사용자 인증 및 컨텍스트(세션, 로그인 정보)를 직접 관리하고 책임지는 역할
  - 클라이언트와 서버의 개발 내용이 명확히 구별되면서 개발의 의존성을 줄임

⑥ 계층형 구조(Layered System)

- REST 서버는 API 서버와 그 앞 단에 사용자 인증, 암호화, 로드 밸런싱 계층 등을 여러 계층으로 구현해 구조상의 유연성을 둘 수 있음

2. REST API 설계

1) REST API 디자인 가이드

(1) REST URI

- 단순하고 직관적으로 URI 표현
- 긴 URL 보다는 최대 2단계 정도로 간단하게 만듦
  - /dogs
  - /dogs/puddle (o)
- 리소스는 동사보다 명사형을 사용하여, HTTP 메서드의 대상이 되는 개체가 되도록 함
- 단수형 명사보다는 복수형 명사 사용
  - POST /dogs (o)
  - POST /getDogs (x)
  - POST /feedDogs (x)

리소스	POST	GET	PUT	DELETE
	create	read	update	delete
/dogs	새로운 dogs 등록	Dogs 목록을 반환	Bulk로 여러 dogs 정보를 업데이트	모든 dogs 정보를 삭제
/dogs/baduk	에러	Baduk이라는 이름의 dogs 정보를 반환	Baduk이라는 이름의 dogs 정보를 업데이트	Baduk이라는 이름의 dogs 정보를 삭제

2. REST API 설계

1) REST API 디자인 가이드

(2) 리소스 간의 관계 표현

- 서브 리소스로 표현 : 소유(has)의 관계를 묵시적으로 표현할 때 사용

> /"리소스명"/"리소스 id"/"관계가 있는 다른 리소스명" 형태
> HTTP GET : /users/{name}/devices
> {name}의 devices 목록 반환

- 서브 리소스에 관계를 명시하는 방법 : 관계명이 애매하거나 구체적인 표현 필요할 때 사용

> HTTP GET : /users/{name}/likes/devices
> {name}이 좋아하는 devices 목록 반환

(3) 에러 처리

- HTTP 응답 코드를 사용한 후 응답 보디에 에러에 대한 내용 서술
  - 내부 개발 중이나 디버깅 시에는 에러에 대한 스택 정보 포함시켜 활용 가능함 (dev 모드와 production 모드의 구분 중요)
- Google의 gdata 서비스는 10개, 넷플릭스는 9개, Digg는 8개의 응답 코드를 사용하는 등 필요에 따라 선택적으로 사용

> 200 - 성공
> 400 Bad Request - field validation 실패 시
> 401 Unauthorized - API 인증, 인가 실패
> 404 Not Found - 해당 리소스가 없음
> 500 Internal Server Error - 서버 에러

2. REST API 설계

1) REST API 디자인 가이드

(4) 페이징 및 부분 응답 처리

- 큰 사이즈의 리스트 형태 응답 처리를 위해 페이징 처리와 부분 응답 처리가 필요

페이징 예시 : 100 ~ 125 레코드 출력

- 페이스북 API 스타일 : /record?offset=100&limit=25
  → 100번째 레코드에서부터 25개 레코드 출력
- 트위터 API 스타일 : /record?page=5&rpp=25
  → rpp(record per page) : 페이지 당 레코드 수
  → page 5는 100~125 레코드

부분 응답 처리 예시 : 응답 메시지에 대해 모든 필드를 포함할 필요가 없는 경우

- 페이스북 API 스타일 : /tom/friends?fields=id,name
- 구글 API 스타일 : ?fields=title,
  media:group(media:thumbnail)
  → JSON의 Sub-Object 개념 지원

2. REST API 설계

1) REST API 디자인 가이드

(5) 검색 (전역 검색과 지역 검색)

- 일반적으로 HTTP GET에서 쿼리 스트링 검색 조건 정의
- name=cho이고 region=seoul 인 사용자 검색
  /users?name=cho&region=seoul&offset=20&limit=10
- 문제점 : offset, limit가 검색조건인지 페이징 조건인지 알 수 없음
- 쿼리 조건을 하나의 쿼리 스트링으로 정의 (3D는 =의 ascii 값)

> /user?q=name%3Dcho,
> region%3Dseoul&offset=20&limit=10
> (name=cho, region=seoul)을 URI 인코딩을 통해 쿼리 스트링 형성

- 전역 검색 : 전체 리소스에 대한 검색 (전역 검색 URI(/search 등) 사용)
  - /search?q=id%3Dtom
- 지역 검색 : 특정 리소스에 대한 검색 (리소스 명에 쿼리 조건 명시)
  - /users?q=id%3Dtom

(6) 단일 API 엔드포인트 활용

- 물리적으로 분리된 여러 개의 서버에서 작동할 때 API 서비스마다 URL을 분리해서 제공하기보다는 단일 URL을 사용하는 것이 바람직함
- 단일 URL로 리버스 프록시에 접속하고, 리버스 프록시에서 각 서비스로 라우팅하게 구성
- 개발 및 관리 관점에서 유연한 운용이 가능해짐

> user.apiserver.com, car.apiserver.com에 대해
> api.apiserver.com이라는 프록시 서버로 단일 URL을 구축한 후,
> api.apiserver.com/user가 user.apiserver.com으로 라우팅
> api.apiserver.com/car가 car.apiserver.com으로 라우팅

2. REST API 설계

2) REST API 안티 패턴

(1) REST API 디자인 시 사용하지 않아야 하는 디자인 패턴

- GET/POST를 이용한 터널링
- Self-descriptiveness 속성을 사용하지 않음
  - 잘못된 예 :
    http://myweb/users?method=update&id=tom
    (리소스 업데이트에 PUT을 사용하지 않고 GET에 쿼리 파라미터로 method=update를 넘겨 수정 메서드임을 명시함)
  - 쉽게 API를 이해하자는 기본 정신에 위배
- HTTP 응답 코드를 사용하지 않음
  - HTTP 응답 코드를 충실하게 따르지 않고, 별도의 코드를 정의해서 사용하는 경우

2. REST API 설계

3) REST API 취약점

(1) 취약점

- REST 사용시 HTTP + JSON 형태 뿐만 아니라 XML등도 이용이 가능하기 때문에 REST 사상을 정확하게 이해하고 사용해야 함
  - 리소스를 잘 정의하고, CRUD를 HTTP 메소드로 구현하고, 에러에는 HTTP 응답 코드 활용
- 메시지 구조 정의(SOAP의 WSDL 등), 서비스 관리 체계(UDDI 등), 메시지 규약(WS-I, WS-*) 등 표준 규약이 없어 관리가 어려움
  - API 개발 전후로 API 문서(Spec)을 만들어 사용
- REST는 전통적인 RDBMS에 적용하기가 어려움
  - 리소스가 계층구조를 가지기 때문에 기본 키를 부여하는데 어려움 존재
  → 여러 컬럼을 조합해서 기본 키값을 설정하면 리소스를 제대로 표현하기 어려움
  /userinfo/{ssn}/{region}/{name}
  - 키를 부여하는 등 문제 해결 위해 AK(Alternative Key)를 사용할 수 있으나(구글에서 AK 사용), 이는 전체적인 DB 설계 변경 의미
  - 최근 MongoDB, CouchDB, Riak 등의 문서 기반 NoSQL은 JSON 문서를 그대로 넣을 수 있는 구조를 갖춰 REST의 리소스 구조를 적용하기가 쉬워짐

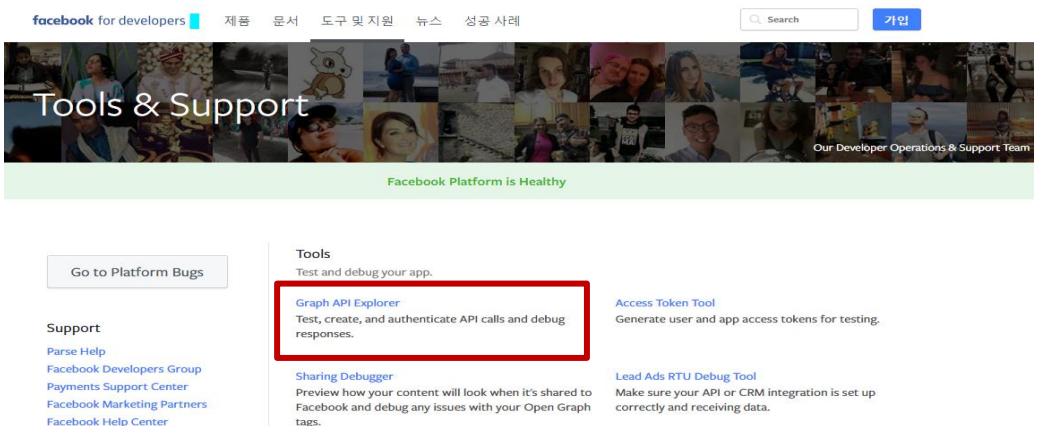
2. REST API 설계

4) REST API 설계 예시

(1) Facebook for Developers

① Graph API

- http://developer.facebook.com/tools-and-support
- Face book에서 구현한 API 호출을 테스트, 생성, 인증해보며, 각 호출에 대한 응답을 디버깅할 수 있도록 개발자에게 제공한 API
- 데이터를 입력 받고 출력하는 기본적인 Facebook 플랫폼

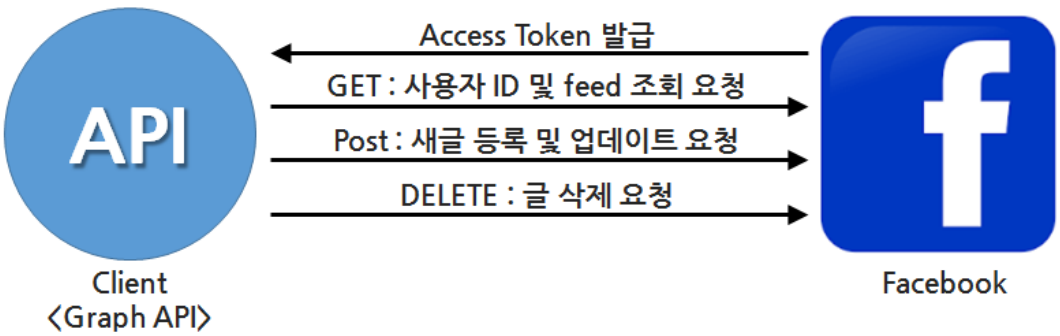


2. REST API 설계

4) REST API 설계 예시

(1) Facebook for Developers

② 실습 Flow



2. REST API 설계

4) REST API 설계 예시

(2) Graph API Explorer

① Access Token 발급

- “Get Token”의 “Get User Access Token” 에서 Graph API에서 접근이 가능하도록 할 사용자 정보를 선택한 후, 토큰 발급 진행
- Facebook 로그인 진행 및 쓰기 권한 설정

② Get Method - 조회

- Access Token 하단 필드는 HTTP 메소드와 API version 및 리소스 및 메시지를 입력할 수 있음
- REST의 설계 사상에 맞게 URI가 작성되었는지 확인해 볼 수 있음
- 좌측 화면의 Node에서는 해당 노드에서 선택 가능한 Field를 선택할 수 있음
- Search for a field를 누르면 해당 노드에서 사용 가능한 field list 출력

③ POST Method - 등록

- 본인의 Feed에 글을 등록해보는 예시
- Access Token을 얻을 때 Permissions에서 “Publish_actions”을 추가해야 사용할 수 있음
- GET을 클릭해 POST로 메소드를 바꾸고, 노드를 /me/feed로 입력
- POST로 바뀐 후 하단에 Add a Field 버튼이 생긴 것을 확인 한 뒤, 다음과 같이 messages 필드에 원하는 텍스트를 입력
- Submit 버튼을 클릭하면 id가 출력됨
- 출력된 id를 클릭해보면 메소드가 GET으로 바뀌며, field에 해당 id가 노드로 입력되어 다음과 같이 출력됨
- 또한 facebook feed에서 새로 등록한 글을 확인할 수 있음

2. REST API 설계

4) REST API 설계 예시

(2) Graph API Explorer

④ POST Method - 수정

- 등록한 글을 수정할 때도 POST 메소드 이용
- 노드에 해당 글의 id를 입력하고, message 필드에 수정된 내용을 입력한 후 submit
- 글이 성공적으로 수정되었다면 true가 반환됨
- Feed에서도 변경된 내용이 출력됨

⑤ POST Method - 삭제

- 생성한 글을 삭제하는 예시
- 메소드 입력하는 부분을 DELETE로 설정하고, field는 해당 글의 id로 입력함
- 성공적으로 삭제가 진행되면, 다음과 같이 success : true가 출력됨

(3) Graph API Docs

- http://developer.facebook.com/docs/graph-api
- Facebook에서 사용하는 node, field 등이 예시와 함께 제공되므로, 추후 활용 가능

핵심정리

1. REST 이해

- REST 정의
  - Representational Safe Transfer 의 약어로 웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍처 (네트워크 상의 자원을 정의하고 자원에 대한 주소를 지정하는 방법)
  - Stateless 방식의 Client/server 구조 지원
  - HTTP와 JSON을 함께 사용해 OPEN API를 구현하는 방법으로 주로 활용

2. REST API 설계

- REST URI
  - 단순하고 직관적으로 URI 표현
  - 긴 URL 보다는 최대 2단계 정도로 간단하게 만듦
  - 리소스는 동사보다 명사형을 사용하여, HTTP 메서드의 대상이 되는 개체가 되도록 함
  - 단수형 명사보다는 복수형 명사 사용