Server-Side Programing

# Ajax

ajax

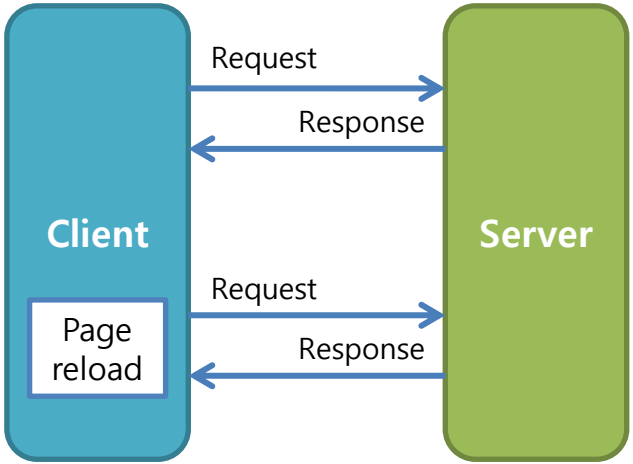
first coding

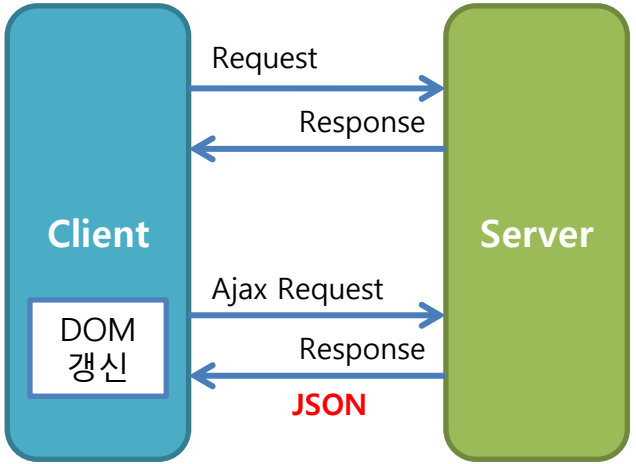
first
coding

# AJAX

# Ajax

- 자바스크립트를 이용해서 브라우저에서 서버에 비동기식으로 데이터를 요청하고 서버로부터 수신한 데이터를 사용하여 웹페이지의 화면을 동적으로 갱신하는 프로그래밍
- Web API를 이용하여 비동기 통신
  - XMLHttpRequest



# JSON

- 클라이언트와 서버간 HTTP 통신을 위한 텍스트 기반의 데이터 포맷.
- 특정 플랫폼이나 언어에 종속적이지 않음.
- 키는 반드시 큰따옴표를 사용해서 기술.
- 값은 객체 리터럴을 그대로 사용하고, 문자열은 반드시 큰따옴표를 사용.

```
{
    "name": "JIN",
    "age": 20,
    "alive": true,
    "hobby": [
        "Campping",
        "game"
    ]
}
```

# XMLHttpRequest

- 자바스크립트를 사용하여 HTTP 요청을 하기 위해서는 XMLHttpRequest를 사용
  - 요청 전송과 응답 수신을 위한 메소드와 프로퍼티 제공

- XMLHttpRequest 객체 생성

```
// 객체 생성
const xhr = new XMLHttpRequest();
```

# XMLHttpRequest

- Http 요청
  - open( method, url, [ async ]) : async 값은 기본값이 true 로 비동기 방식으로 동작
  - setRequestHeader() : Header값을 설정, open 메소드 호출 이후 설정
  - send() : 설정된 HTTP 요청을 서버로 전송

```
// 1. 객체 생성
const xhr = new XMLHttpRequest();

// 2. HTTP 요청 초기화
xhr.open('GET', '/users');

// 3. HTTP 요청 헤더 설정
// 클라이언트가 서버로 전송할 데이터의 MIME 타입 지정: json
xhr.setRequestHeader('content-type', 'application/json');

// 4. HTTP 요청 전송
xhr.send();
```

# XMLHttpRequest

- **HTTP 응답처리**
  - 서버에서 전송한 응답을 처리하기 위해서는 XMLHttpRequest 객체가 생성하는 이벤트를 확인해야 함.
  - XMLHttpRequest 의 이벤트 핸들러
    - onreadystatechange, onload, onerror
  - onreadystatechange
    - http 요청 현재 상태를 나타내는 readystate 프로퍼티 값이 변경된 경우 이벤트 발생

# XMLHttpRequest

```
const xhr = new XMLHttpRequest();

// https://jsonplaceholder.typicode.com은 Fake REST API를 제공하는 서비스다.
xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos/1');

xhr.send();

// readystatechange 이벤트는 readyState(HTTP 요청의 현재 상태) 프로퍼티가 변경될 때마다 발생
xhr.onreadystatechange = () => {
  // readyState 프로퍼티 값이 4(XMLHttpRequest.DONE)가 아니면 서버 응답이 완료되지 상태다.
  // 만약 서버 응답이 아직 완료되지 않았다면 아무런 처리를 하지 않는다.
  if (xhr.readyState !== XMLHttpRequest.DONE) return;

  // status 프로퍼티는 응답 상태 코드를 나타낸다.
  // status 프로퍼티 값이 200이면 정상적으로 응답된 상태이고 200이 아니면 에러가 발생한 상태다.
  // 정상적으로 응답된 상태라면 response 프로퍼티에 서버의 응답 결과가 담겨 있다.
  if (xhr.status === 200) {
    console.log(JSON.parse(xhr.response));
    // {userId: 1, id: 1, title: "delectus aut autem", completed: false}
  } else {
    console.error('Error', xhr.status, xhr.statusText);
  }
};
```

first
coding

# 프로미스

# 프로미스

- **자바스크립트는 비동기 처리를 위한 방법으로 콜백함수를 사용**
  - 가독성이 좋지 않음
  - 비동기 처리 중 에러 처리가 어려움
  - 여러 개의 비동기 처리를 한번에 처리도 한계가 있음

- **ES6 에서는 비동기통신의 처리를 위해 프로미스 도입.**
  - 콜백 패턴의 가진 단점을 보완 : 처리 시점을 명확히 확인하여 처리

# 프로미스

- Promise 생성자 함수를 new 연산자와 함께 호출하여 객체 생성
  - 콜백 함수를 인자로 받음 : 콜백 함수는 resolve, reject를 받음

```
// 프로미스 생성
const promise = new Promise((resolve, reject) => {
  // Promise 함수의 콜백 함수 내부에서 비동기 처리를 수행한다.
  if (/* 비동기 처리 성공 */) {
    resolve('result');
  } else { /* 비동기 처리 실패 */
    reject('failure reason');
  }
});
```

# 프로미스

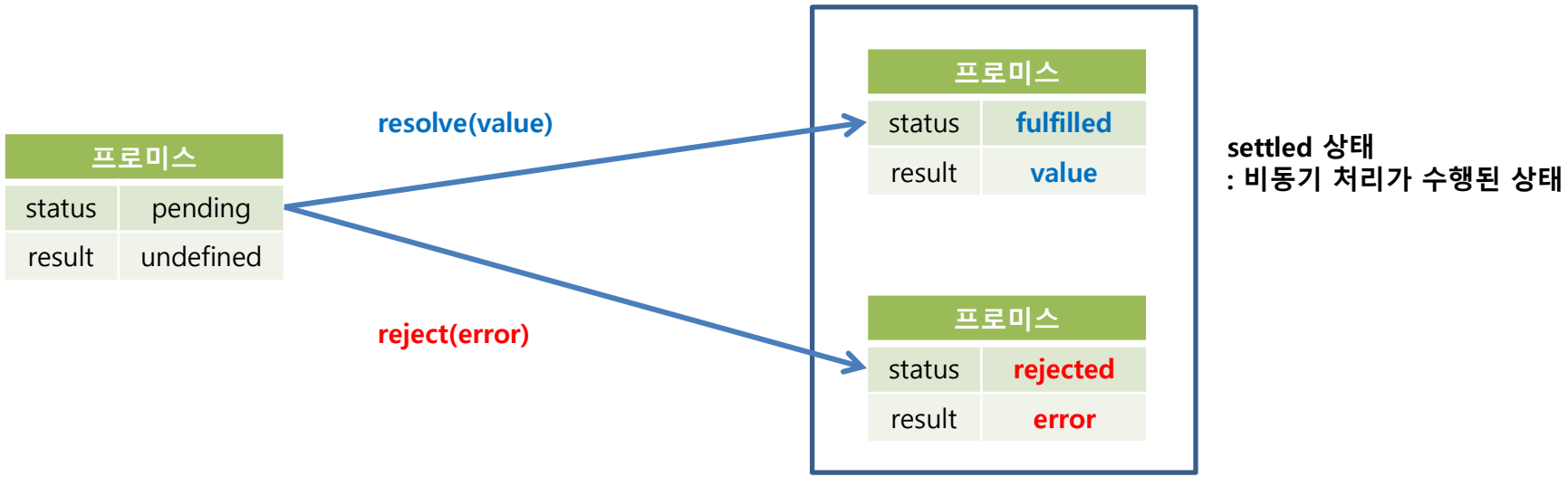
- get 방식 요청하는 비동기 통신을 하는 코드를 프로미스로 구현

```
// GET 요청을 위한 비동기 함수
const promiseGet = url => {
  return new Promise((resolve, reject) => {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);
    xhr.send();

    xhr.onload = () => {
      if (xhr.status === 200) {
        // 성공적으로 응답을 전달받으면 resolve 함수를 호출한다.
        resolve(JSON.parse(xhr.response));
      } else {
        // 에러 처리를 위해 reject 함수를 호출한다.
        reject(new Error(xhr.status));
      }
    };
  });
};

// promiseGet 함수는 프로미스를 반환한다.
promiseGet('https://jsonplaceholder.typicode.com/posts/1');
```

# 프로미스

- 프로미스 상태

| 프로미스의 상태정보 | 의미 | 상태변경 조건 |
|---|---|---|
| pending | 비동기 처리가 아직 수행되지 않은 상태 | 프로미스가 생성된 직후 상태 |
| fulfilled | 비동기 처리가 수행된 상태(성공) | resolve 함수 호출 |
| rejected | 비동기 처리가 수행된 상태(실패) | reject 함수 호출 |



# 프로미스

- **프로미스 후속 메소드**
  - .then()
    - 두 개의 콜백 함수를 전달 받음
    - 첫 번째 콜백 함수 : fulfilled 상태가 되면 호출
      → 이 때 콜백 함수의 인자는 프로미스 실행 결과를 받음
    - 두 번째 콜백 함수 : rejected 상태가 되면 호출
      → 프로미스의 에러를 인수로 받음
    - 프로미스 반환

# 프로미스

- **프로미스 후속 메소드**
  - .catch()
    - 한 개의 콜백 함수를 전달 받음
    - 프로미스 rejected 상태인 경우에만 호출
    - 프로미스 반환

# 프로미스

- **프로미스 후속 메소드**
  - .finally()
    - 한 개의 콜백 함수를 전달 받음
    - 프로미스의 상태와 상관 없이 무조건 한번 호출, 공통적으로 꼭 실행해야 하는 처리 수행
    - 프로미스 반환

# 프로미스

- 에러처리
  - then() 메소드에서 처리보다는 catch() 메소드에서 처리하는 것이 좋다
    - 비동기 처리 에러 뿐 아니라 then() 메소드 내부에서 처리되는 에러도 캐치 가능

```
promiseGet('https://jsonplaceholder.typicode.com/todos/1')
    .then(res => console.log(res))
    .catch(err => console.error(err)); // TypeError: console.xxx is not a function
```

# 프로미스

- 프로미스 체이닝

```
const url = 'https://jsonplaceholder.typicode.com';

// id가 1인 post의 userId를 취득
promiseGet(`${url}/posts/1`)
    // 취득한 post의 userId로 user 정보를 취득
    .then(value => {
        console.log('userId', value.userId)
        return promiseGet(`${url}/users/${value.userId}`)
    })
    .then(userInfo => console.log(userInfo))
    .catch(err => console.error(err));
```

| 후속 처리 메소드 | 콜백 함수의 인수 | 후속처리 메소드의 반환 값 |
|---|---|---|
| then | promiseGet 함수가 반환한 프로미스가 resolve 한 값<br>( id 가 1인 post ) | 콜백 함수가 반환한 프로미스 |
| then | 첫 번째 then 메소드가 반환한 프로미스가 resolve한 값<br>( post의 userId로 취득한 user 정보 )) | 콜백 함수가 반환한 값을 resolve 한 프로미스 |
| catch | promiseGet 함수 또는 후속 처리 메소드에서 반환한 프로미스의 rejected한 값 | 콜백 함수가 반환한 값을 resolve 한 프로미스 |

first
coding

# fetch

# fetch

- HTTP 요청 전송 기능을 제공하는 Web API
  - 사용법이 간단하고 프로미스를 지원
  - HTTP 요청 URL과 HTTP요청 메소드, HTTP 요청 헤더, 페이로드(쿼리스트링)등을 설정한 객체를 전달

```
const promise = fetch(url [, options])
```

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
    .then(response => console.log(response));
```

```
Response {type: 'cors', url: 'https://jsonplaceholder.typicode.com/todos/1', redirected:
false, status: 200, ok: true, …}
  body: (...)
  bodyUsed: false
  headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: ""
  type: "cors"
  url: "https://jsonplaceholder.typicode.com/todos/1"
  [[Prototype]]: Response
```

# fetch

- HTTP 요청 전송 기능을 제공하는 Web API

```
fetch('https://jsonplaceholder.typicode.com/todos/1')
  // response는 HTTP 응답을 나타내는 Response 객체이다.
  // json 메서드를 사용하여 Response 객체에서 HTTP 응답 몸체를 취득하여 역직렬화한다.
  .then(response => response.json())
  // json은 역직렬화된 HTTP 응답 몸체이다.
  .then(json => console.log(json));
  // {userId: 1, id: 1, title: "delectus aut autem", completed: false}
```

# fetch

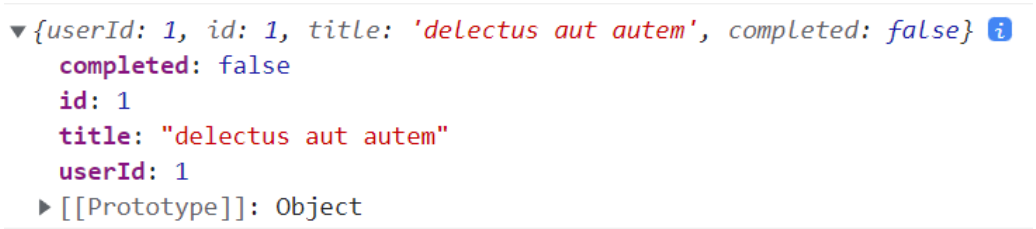
- HTTP 요청 전송 기능을 제공하는 Web API

```
const request = {
    get(url) {
        return fetch(url);
    },
    post(url, payload) {
        return fetch(url, {
            method: 'POST',
            headers: { 'content-Type': 'application/json' },
            body: JSON.stringify(payload)
        });
    },
    delete(url) {
        return fetch(url, { method: 'DELETE' });
    }
};
```

# fetch

- HTTP 요청 전송 기능을 제공하는 Web API

```
request.get('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => {
    if (!response.ok) throw new Error(response.statusText);
    return response.json();
  })
  .then(todos => console.log(todos))
  .catch(err => console.error(err));
// {userId: 1, id: 1, title: "delectus aut autem", completed: false}
```



# fetch

- HTTP 요청 전송 기능을 제공하는 Web API

```
request.post('https://jsonplaceholder.typicode.com/todos', {
        userId: 1,
        title: 'JavaScript',
        completed: false
    }).then(response => {
        if (!response.ok) throw new Error(response.statusText);
        return response.json();
    })
    .then(todos => console.log(todos))
    .catch(err => console.error(err));
    // {userId: 1, title: "JavaScript", completed: false, id: 201}
```

```
▼ {userId: 1, title: 'JavaScript', completed: false, id: 201}
    completed: false
    id: 201
    title: "JavaScript"
    userId: 1
  ▶ [[Prototype]]: Object
```

# fetch

- HTTP 요청 전송 기능을 제공하는 Web API

```
request.delete('https://jsonplaceholder.typicode.com/todos/1')
    .then(response => {
        if (!response.ok) throw new Error(response.statusText);
        return response.json();
    })
    .then(todos => console.log(todos))
    .catch(err => console.error(err));
```

```
▼ {userId: 1, title: 'JavaScript', completed: false, id: 201}
    completed: false
    id: 201
    title: "JavaScript"
    userId: 1
  ▶ [[Prototype]]: Object
```

first
coding

# Axios

# Axios

- **브라우저와 node.js에서 사용할 수 있는 Promise 기반 HTTP 클라이언트 라이브러리**
  - Axios는 node.js와 브라우저를 위한 Promise 기반 HTTP 클라이언트 입니다.
  - 특징
    - 브라우저를 위해 XMLHttpRequests 생성
    - Promise API를 지원
    - 요청 및 응답 데이터 변환
    - JSON 데이터 자동 변환

# 설치

- 설치
  - jsDelivr CDN 사용하기:
    - <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
  - unpkg CDN 사용하기:
    - <script src="https://unpkg.com/axios/dist/axios.min.js"></script>

# Axios API

- Axios API 레퍼런스
  - axios에 해당 config을 전송하면 요청이 가능합니다.
  - axios(config)

```
// POST 요청 전송
axios({
    method: 'post',
    url: '/user/12345',
    data: { firstName: 'Fred', lastName: 'Flintstone' }
});
```

  - axios(url[, config])

```
// GET 요청 전송 (기본값)
axios('/user/12345');
```

# Axios API

- Axios API 레퍼런스
  - 요청 메소드 명령어 : 편의를 위해 지원하는 모든 요청 메소드의 명령어를 제공합니다.
    - axios.request(config)
    - axios.get(url[, config])
    - axios.delete(url[, config])
    - axios.head(url[, config])
    - axios.options(url[, config])
    - axios.post(url[, data[, config]])
    - axios.put(url[, data[, config]])
    - axios.patch(url[, data[, config]])
  - 명령어 메소드를 사용시 'url', 'method', 'data' 속성을 config에서 지정할 필요가 없습니다.

# 기본 예제

- GET 요청

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

```
axios.get('/todos')
        .then(function (response) { // 성공 핸들링
            console.log(response);
        })
        .catch(function (error) { // 에러 핸들링
            console.log(error);
        })
        .then(function () { // 항상 실행되는 영역
        });

// 선택적으로 위의 요청은 다음과 같이 수행될 수 있습니다.
axios.get('/todos', { params: { userId: 1 } })
        .then(function (response) {
            console.log(response);
        })
        .catch(function (error) {
            console.log(error);
        })
        .then(function () { // 항상 실행되는 영역
        });
```

# 기본 예제

- POST 요청

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

```
axios.post('/todos', { userId: 5,
        title: 'Java',
        completed: false})
    .then(function (response) {
        console.log(response);
    })
    .catch(function (error) {
        console.log(error);
    });
```

```
▶ {data: {…}, status: 201, statusText: 'Created', headers: i, config: {…}, …}
```

# 기본 예제

- PUT 요청

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

```
axios.put('/todos/5', { id: 5, content: 'VUE', completed: true })
    .then(function (response) {
        console.log(response);
    })
    .catch(function (error) {
        console.log(error);
    });
```

# 기본 예제

- DELETE 요청

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

```
axios.delete('/todos/5')
    .then(function (response) {
        console.log(response);
    })
    .catch(function (error) {
        console.log(error);
    });
```