互联网应用开发技术

Web Application Development

第9课
WEB前后台通信-CORS & FETCH

Episode Nine

CORS and Fetch API

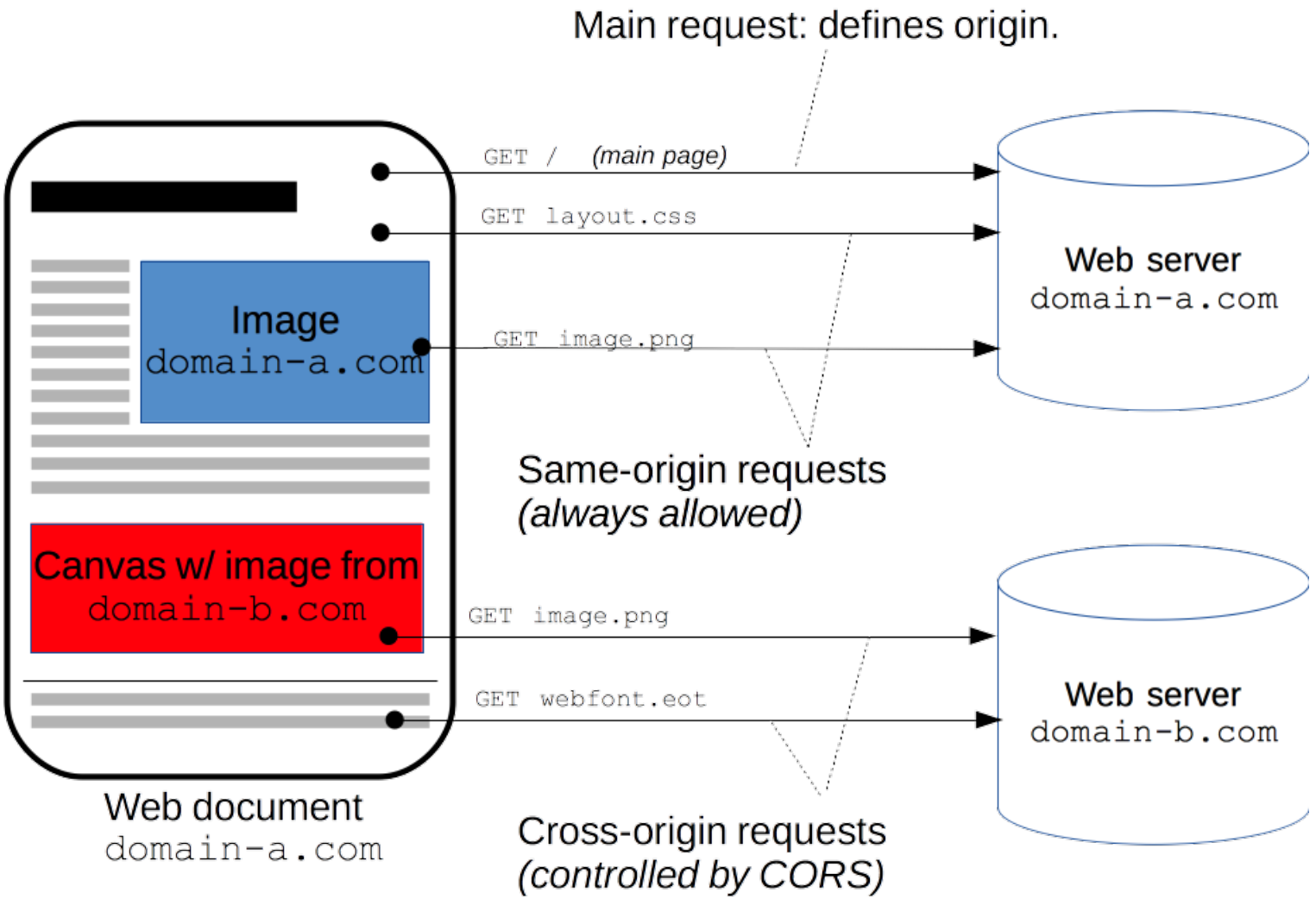
陈昊鹏

chen-hp@sjtu.edu.cn

- **Cross-Origin Resource Sharing** (CORS)
  - is a mechanism that uses additional HTTP headers to tell browsers to give a web application running at one origin, access to selected resources from a different origin.
  - A web application executes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, or port) from its own.

- **Cross-Origin Resource Sharing** (CORS)
  - An example of a cross-origin request: the front-end JavaScript code served from https://domain-a.com uses XMLHttpRequest to make a request for https://domain-b.com/data.json.
  - For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts.
    - For example, XMLHttpRequest and the Fetch API follow the same-origin policy.
    - This means that a web application using those APIs can only request resources from the same origin the application was loaded from, unless the response from other origins includes the right CORS headers.

CORS


Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
Web server
domain-a.com
Same-origin requests
(always allowed)
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-b.com
Web document
domain-a.com
Cross-origin requests
(controlled by CORS)

Same-origin policy

- The same-origin policy is a critical security mechanism that
  - restricts how a document or script loaded from one origin can interact with a resource from another origin.
  - It helps isolate potentially malicious documents, reducing possible attack vectors.

- Definition of an origin
  - Two URLs have the *same origin* if the protocol, port (if specified), and host are the same for both.

Same-origin policy

- The following table gives examples of origin comparisons with the URL
  - http://store.company.com/dir/page.html:

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

- This cross-origin sharing standard can enable cross-site HTTP requests for:
  - Invocations of the XMLHttpRequest or Fetch APIs, as discussed above.
  - Web Fonts (for cross-domain font usage in @font-face within CSS), so that servers can deploy TrueType fonts that can only be cross-site loaded and used by web sites that are permitted to do so.
  - WebGL textures.
  - Images/video frames drawn to a canvas using drawImage().
  - CSS Shapes from images.

CORS in Java Backend - Filter


```
@WebFilter(filterName = "corsFilter", urlPatterns = {"/*"})
public class CorsFilter implements Filter{
   @Override
   public void init(FilterConfig filterConfig) throws ServletException {  }

   @Override
   public void doFilter(ServletRequest servletRequest,
                        ServletResponse servletResponse, FilterChain filterChain)
                        throws IOException, ServletException {
      HttpServletResponse response = (HttpServletResponse) servletResponse;
      response.setHeader("Access-Control-Allow-Origin", "http://localhost:3000");
      response.setHeader("Access-Control-Allow-Methods",
                         "POST, GET, OPTIONS, DELETE, PUT");
      response.setHeader("Access-Control-Max-Age", "3600");
      response.setHeader("Access-Control-Allow-Headers",
                         "Content-Type,Access-Token,Authorization,ybg");
      filterChain.doFilter(servletRequest, servletResponse);
   }

   @Override
   public void destroy() { }
}
```

CORS in Java Backend - @CrossOrigin


```
@RestController
public class HelloWorldController {

    @Autowired
    JdbcTemplate jdbcTemplate;

    @CrossOrigin
    @RequestMapping("/")
    public String home() {
        final Logger log = LoggerFactory.getLogger(HelloWorldController.class);
        List<Book> result = new ArrayList<Book>();

        log.info("Querying Books");
        jdbcTemplate.query(
                "SELECT * FROM book",
                (rs, rowNum) -> new Book(rs.getLong("id"),
                        rs.getString("title"),
                        rs.getString("author"),
                        rs.getString("language"),
                        rs.getString("published"),
                        rs.getString("sales"))
        ).forEach(book -> {log.info(book.toString()); result.add(book);});
        Iterator<Book> it = result.iterator();
```

Fetch API

- The Fetch API provides an interface for fetching resources (including across the network).

- Fetch Interfaces
  - WindowOrWorkerGlobalScope.fetch(): The fetch() method used to fetch a resource.
  - Headers: Represents response/request headers, allowing you to query them and take different actions depending on the results.
  - Request: Represents a resource request.
  - Response: Represents the response to a request.

Fetch API

- A basic fetch request is really simple to set up. Have a look at the following code:

```
fetch('http://example.com/movies.json')
 .then((response) => {
   return response.json();
 })
 .then((data) => {
   console.log(data);
 });
```

- Here we are fetching a JSON file across the network and printing it to the console.
- The simplest use of fetch()
  - takes one argument — the path to the resource you want to fetch — and returns a promise containing the response (a Response object).

Fetch API

- The fetch() method can optionally accept a second parameter, an init object that allows you to control a number of different settings:

```
// Example POST method implementation:
async function postData(url = '', data = {}) {
  // Default options are marked with *
  const response = await fetch(url, {
    method: 'POST', // *GET, POST, PUT, DELETE, etc.
    mode: 'cors', // no-cors, *cors, same-origin
    cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
    credentials: 'same-origin', // include, *same-origin, omit.
    headers: {
        'Content-Type': 'application/json'
        // 'Content-Type': 'application/x-www-form-urlencoded',
    },
    redirect: 'follow', // manual, *follow, error
    referrerPolicy: 'no-referrer', // no-referrer, *client
    body: JSON.stringify(data) // body data type must match "Content-Type" header
  });
  return response.json(); // parses JSON response into native JavaScript objects
}
postData('https://example.com/answer', { answer: 42 })
  .then((data) => {
    console.log(data); // JSON data parsed by `response.json()` call
  });
```

Fetch API

- Uploading JSON data

```
const data = { username: 'example' };

fetch('https://example.com/profile', {
  method: 'POST', // or 'PUT'
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
 .then((response) => response.json())
 .then((data) => {
    console.log('Success:', data);
})
 .catch((error) => {
   console.error('Error:', error);
});
```

Fetch API

- Uploading a file

```
const formData = new FormData();
const fileField = document.querySelector('input[type="file"]');

formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  body: formData
})
 .then((response) => response.json())
 .then((result) => {
   console.log('Success:', result);
 })
 .catch((error) => {
    console.error('Error:', error);
});
```

Fetch API

- Uploading multiple files
  - Files can be uploaded using an HTML <input type="file" multiple/> input element, FormData() and fetch().

```
const formData = new FormData();
const photos = document.querySelector('input[type="file"][multiple]’);

formData.append('title', 'My Vegas Vacation');
for (let i = 0; i < photos.files.length; i++) {
   formData.append('photos', photos.files[i]);
}

fetch('https://example.com/posts', {
   method: 'POST’,
   body: formData,
})
.then((response) => response.json())
.then((result) => { console.log('Success:', result); })
.catch((error) => { console.error('Error:', error);});
```

Fetch API

- Checking that the fetch was successful

```
fetch('flowers.jpg')
  .then((response) => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.blob();
  })
  .then((myBlob) => {
    myImage.src = URL.createObjectURL(myBlob);
  })
  .catch((error) => {
    console.error('There has been a problem with your fetch operation:', error);
  });
```

Fetch API

- Supplying your own request object

```
const myHeaders = new Headers();
const myRequest = new Request('flowers.jpg', {
    method: 'GET',
    headers: myHeaders,
    mode: 'cors',
    cache: 'default',
});

fetch(myRequest)
  .then((response) => response.blob())
  .then((myBlob) => {
      myImage.src = URL.createObjectURL(myBlob);
  });
```

Fetch API

- Response objects
  - The most common response properties you'll use are:
  - Response.status — An integer (default value 200) containing the response status code.
  - Response.statusText — A string (default value "OK"), which corresponds to the HTTP status code message.
  - Response.ok — seen in use above, this is a shorthand for checking that status is in the range 200-299 inclusive. This returns a Boolean.

```
const myBody = new Blob();

addEventListener('fetch', function(event) {
  // ServiceWorker intercepting a fetch
  event.respondWith(
    new Response(myBody, {
      headers: { 'Content-Type': 'text/plain' }
    })
  );
});
```

Fetch API

- Body
  - Both requests and responses may contain body data.

```
const form = new FormData(document.getElementById('login-form'));

fetch('/login', {
    method: 'POST',
    body: form
});
```

References

- Cross-Origin Resource Sharing (CORS)
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- 你所需要的跨域问题的全套解决方案都在这里啦！（前后端都有）
  - https://zhuanlan.zhihu.com/p/120764119
- Fetch API
  - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
  - https://www.w3cschool.cn/fetch_api/

- *Web*开发技术
- *Web Application Development*

Thank You!