Fix dynamic content issue when using AJAX

After an AJAX request is done, the old content is replaced with the new content, causing dynamic content and handlers are destroyed, such as items were selected by javascript query, Mermaid code blocks, disqus comments, registered event for clicking, etc. A solution is to re-activate dynamic elements after the AJAX content is loaded.

#html #ajax #javascript

Last update: 2021-06-14 15:57:19

Table of Content

1. Instant loading

As mentioned in the Material for Mkdocs guide, this theme supports Instant Loading feature which intercepts internal links and load content via XHR request without fully reloading the page.

The resulting page is parsed and injected and all event handlers and components are rebound automatically. This means that the site behaves like a Single Page Application, which is especially useful for large documentation sites that come with a massive search index, as the search index will now remain intact in-between document switches.

However, using XHR in an AJAX site leads to a problem.

> **Dynamic Javascript-enabled elements do not work after AJAX content is loaded.**
>
> After an AJAX request is done, the old content is replaced with the new content, causing dynamic content and handlers are destroyed, such as items were selected by javascript query, Mermaid code blocks, disqus comments, registered event for clicking, etc.

Here are problems I get after click in an internal link:

- Zoom-in Images function does not work
- External links is not opened in a new tab
- Disqus section does not load new thread comments

The solution is to detect internal navigation, and then re-register all dynamic elements.

2. Detect internal navigation

When the site fetches new page content, it will change the URL location too, therefore there are some methods to detect an internal navigation:

- Detect URL change:
  - A while loop to check if URL is changed every 1s, or
  - A handler that listens to browser history change
- Detect content change:
  - Observe an element and detect its content is changed

2.1. Detect URL change

Here is a fix using URL detection approach, but the handler will be called multiple time causing bad performance, e.g. when a page has a lot of images. This method redefines the handler which

calls the old handler (`f` in lambda) and fires new events using `window.dispatchEvent` function:

assets\extra.js

```
function reactivateElements() {
    /* do something here */
}

history.pushState = ((f) =>
    function pushState() {
        var ret = f.apply(this, arguments);
        window.dispatchEvent(new Event("pushstate"));
        window.dispatchEvent(new Event("locationchange"));
        return ret;
    })(history.pushState);

history.replaceState = ((f) =>
    function replaceState() {
        var ret = f.apply(this, arguments);
        window.dispatchEvent(new Event("replacestate"));
        window.dispatchEvent(new Event("locationchange"));
        return ret;
    })(history.replaceState);

window.addEventListener("popstate", () => {
    window.dispatchEvent(new Event("locationchange"));
});

window.addEventListener("locationchange", () => {
    // can check actual change of URL but the content may not loaded
    console.log("locationchange");
    reactivateElements();
});
```

2.2. Detect content change

Another method uses an observer to watch the `<HEAD>` tag, because whenever a new page is loaded, the title and page's information will be changed inside the `<HEAD>` tag. Just need to monitor the children state of this tag.

assets\extra.js

```
var currentLocation = document.location.href;

function reactivateElements() {
    /* do something here */
}

const observer = new MutationObserver(() => {
    if (currentLocation !== document.location.href) {
        console.log("URL changed!");
        currentLocation = document.location.href;
```

```
            reactivateElements();
        }
    });

    observer.observe(document.getElementsByTagName("HEAD")[0], { childList: true });
```

I chose this method.

3. Activate dynamic elements

This step is to wrap all actions that need to be run:

- when the page is fully loaded at the first time; and
- when the content of the page is loaded after an XHR request

3.1. Re-direct external links

External link should be shown in a new tab, without any tracking information. To do that, I write a function to get all external links in the page, then set `target = "_blank"` and add attribute `rel = "noopener noreferrer"`.

assets\extra.js

```
function activateExternalLinks() {
    /* open external links in new tab */
    var links = document.links;
    for (var i = 0, linksLength = links.length; i < linksLength; i++) {
        if (links[i].hostname != window.location.hostname) {
            links[i].target = "_blank";
            links[i].setAttribute("rel", "noopener noreferrer");
            links[i].className += " externalLink";
        } else {
            links[i].className += " localLink";
        }
    }
}
```

Finally add this function to the re-activation chain:

assets\extra.js

```
function reactivateElements() {
    activateExternalLinks();
}
```

3.2. Re-register images

Firstly, wrap the handler in a function which will be re-called. Note that, `ViewBigimg` object will be created once as a constant object.

assets\extra.js

```
var viewer = new ViewBigimg();
function activateBigImg() {
    /* enable zoom-in */
    var figures = document.querySelectorAll("img");
    for (var i = 0; i < figures.length; i++) {
        figures[i].onclick = (e) => {
            if (e.target.nodeName === "IMG") {
                viewer.show(e.target.src);
            }
        };
    }
    /* click to close zoomed image */
    var containers = document.querySelectorAll("#iv-container .iv-image-view");
    for (var i = 0; i < containers.length; i++) {
        containers[i].onclick = () => {
            if (!dragged) {
                viewer.hide();
            }
        };
    }
}
```

Then add this function to the re-activation chain:

assets\extra.js

```
function reactivateElements() {
    activateExternalLinks();
    activateBigImg();
}
```

3.3. Re-register pagination links

Do the same step as described above to make a wrapped function to add into the re-activation chain:

assets\extra.js

```
function activatePaginationLinks() {
    var pagination = document.getElementById("pagination");
    var links = pagination.getElementsByClassName("page-number");
    if (links.length) {
        for (var i = 0; i < links.length; i++) {
            links[i].addEventListener("click", function () {
                var current = pagination.getElementsByClassName("active");
                console.log(current);
                if (current.length) {
                    current[0].className = current[0].className.replace(
```

```
                        " active",
                        ""
                    );
                }
                this.className += " active";
            });
        }
        links[0].click();
    }
}

function reactivateElements() {
    activateExternalLinks();
    activateBigImg();
    activatePaginationLinks();
}
```

3.4. Re-enable Disqus

Disqus can not see the URL changed by itself, and it provides an API to reset the Disqus section. There is a guide on Using Disqus on AJAX sites showing a snippet to reload Disqus thread:

```
DISQUS.reset({
    reload: true,
    config: function () {
        this.page.identifier = "newidentifier";
        this.page.url = "http://example.com/#!newthread";
    },
});
```

Checking the template of Material in `partials\integrations\disqus.html`, it gets *page id* and *page url* using jinja template:

```
<h2 id="__comments">{{ lang.t("meta.comments") }}</h2>
<div id="disqus_thread"></div>
<script>
    var disqus_config = function() {
        this.page.url =
            "{{ page.canonical_url }}",
        this.page.identifier =
            "{{ page.canonical_url | replace(config.site_url, '') }}"
    };
    window.addEventListener("load",function() {
        var e = document, i = e.createElement("script");
        i.src = "//{{ disqus }}.disqus.com/embed.js",
        i.setAttribute("data-timestamp", +new Date),
        (e.head||e.body).appendChild(i)
    });
</script>
```

Therefore, I can get those information in the same way by adding two `<div>` with data being the *page id* and *page url* in the `main.html` template:

main.html

```
{{ page.content }}
<div id="page_url"
    data-value="{{page.canonical_url }}">
</div>
<div id="page_identifier"
    data-value="{{ page.canonical_url | replace(config.site_url, '') }}">
</div>
```

Then, I created a function to reset Disqus which extracts *page id* and *page url* from above elements, and call to `DISQUS.reset()` :

assets\extra.js

```
function resetDisqusPlugin() {
    var page_url = document.getElementById("page_url");
    var page_identifier = document.getElementById("page_identifier");

    if (page_url && page_identifier) {
        page_url = page_url.dataset.value + "#!newthread";
        console.log(page_url);

        page_identifier = page_identifier.dataset.value;
        console.log(page_identifier);

        try {
            DISQUS_RECOMMENDATIONS.reset();
            DISQUS.reset({
                reload: true,
                config: function () {
                    this.page.identifier = page_identifier;
                    this.page.url = page_url;
                },
            });
        } catch (e) {
            console.log(e);
        }
    }
}
```

However, there is still a problem:

Disqus is not initialized in some pages

The Disqus section is controlled by:

- setting in `mkdocs.yml` file
- the `page.is_homepage` variable

- the field `disqus` in page's meta-data

If an user visits the first page with no disqus activated, the next page even with disqus included still fails to load disqus as the script `//{{ disqus }}.disqus.com/embed.js` is not loaded.

The solution is enable disqus on all pages, then hide the section on some specific pages. How to do it? Here are steps:

1. Remove default Disqus block by extending it with empty content:

```
{% block disqus %}
{% endblock %}
```

2. Wrap the page content inside a new block `page_content` :

```
{% block page_content %}
    ...
    {{ page.content }}
{% endblock %}
```

3. Append the customized Disqus block after the `page_content` :

```
{% block content %}
  {% block page_content %}
    ...
    {{ page.content }}
  {% endblock %}
  <div id="page_url"
    data-value="{{page.canonical_url }}">
  </div>
  <div id="page_identifier"
    data-value="{{page.canonical_url|replace(config.site_url,'')}}">
  </div>
  {% set disqus = config.extra.disqus %}
  {% if disqus %}
    {% if page and page.meta and page.meta.hide %}
      {% set hidden = "hidden" if "disqus" in page.meta.hide %}
    {% endif %}
    <div {{ hidden }}>
      {% set disqus = config.extra.disqus %}
      <h2 id="__comments">{{ lang.t("meta.comments") }}</h2>
      <div id="disqus_thread"></div>
      <script>var disqus_config = function () { this.page.url = "{{
page.canonical_url }}", this.page.identifier = "{{ page.canonical_url |
replace(config.site_url, '') }}" }; window.addEventListener("load", function
() { var e = document, i = e.createElement("script"); i.src = "//{{ disqus
}}.disqus.com/embed.js", i.setAttribute("data-timestamp", +new Date), (e.head
|| e.body).appendChild(i) })</script>
      </div>
  {% endif %}
```

```
{% endblock %}

{% block disqus %}
{% endblock %}
```

4. All other templates which extends the `main.html` template have to extend the `page_content` block, not the `content` block.
5. Finally, in the page where I don't want to see disqus section, just add `disqus` to the `hide` attribute in the meta-data section:

```
---
title: Recent updated posts
description: Latest news and recent posts
template: blog.html
hide:
    - disqus
---
```