Understanding HTML Form Encoding: URL Encoded and Multipart Forms

Dinesh Balaji Aug 11 '18 · 5 min read

#html #web #forms #webdev

The other day I was trying to write a REST endpoint in Go, which uploads the contents of a form submitted in a browser to another REST endpoint, in other words,

```
Form in Browser ----> My GO Rest API ----> Another REST API
```

While doing that I ended up learning some fundamentals of how HTML forms work. So thought it might be a good thing to share what I learned and hence the post.. :)

The encoding type of a form is determined by the attribute `enctype`. It can have three values,

- `application/x-www-form-urlencoded` - Represents an URL encoded form. This is the default value if `enctype` attribute is not set to anything.
- `multipart/form-data` - Represents a Multipart form. This type of form is used when the user wants to upload files
- `text/plain` - A new form type introduced in HTML5, that as the name suggests, simply sends the data without any encoding

Now, let us look at each form type with an example to understand them better.

URL Encoded Form

As the name suggests, the data that is submitted using this type of form is URL endcoded. Take the following form,

```
<form action="/urlencoded?firstname=sid&lastname=sloth" method="POST" enctyp
    <input type="text" name="username" value="sidthesloth"/>
    <input type="text" name="password" value="slothsecret"/>
    <input type="submit" value="Submit" />
</form>
```

Here, you can see that the form is submitted to the server using a POST request, this means that it has a body. But how is the body formatted? It is URL encoded. Basically, a long string of `(name, value)` pairs are created. Each `(name, value)` pair is separated from one another by a `& (ampersand)` sign, and for each `(name, value)` pair, the `name` is separated from the `value` by an `= (equals)` sign, like say,

```
key1=value1&key2=value2
```

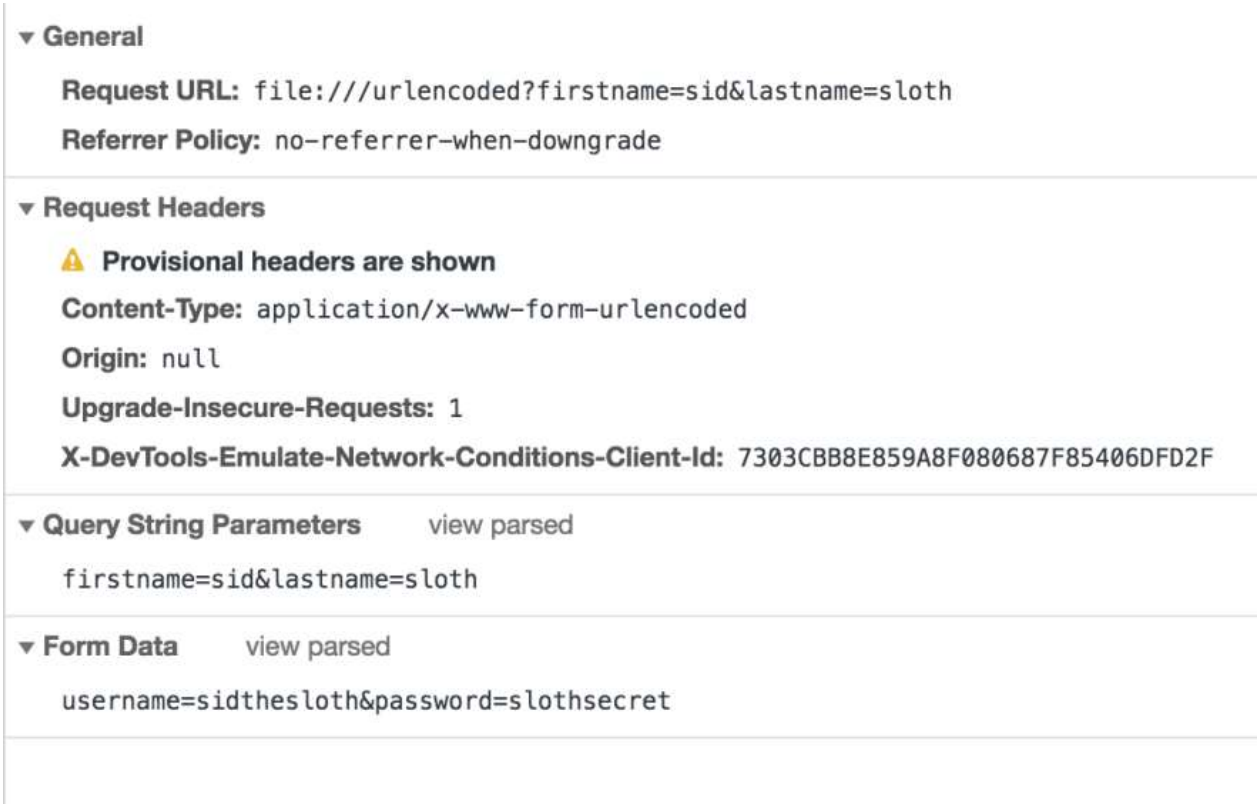
For the above form, it would be,

```
username=sidthesloth&password=slothsecret
```

Also, notice that we have some query parameters passed in the action URL, `/urlencoded?firstname=sid&lastname=sloth`.

Don't the URL encoded body and the query parameters passed in the action URL look awfully similar? It's because they are similar. They share the same format discussed above.

Try creating an HTML file with the above code and see how it's submitted in the dev tools. Here is a snap,



The things to notice here are the `Content-Type` header which says `application/x-www-form-urlencoded`, the query string and the form fields are transferred to the server in the format as discussed above.

> **Note:** Don't get confused by the term Form Data in the screen shot. It's just how Google Chrome represents form fields.

All is fine, but there is a little more to the encoding process. Let's introduce some spaces in the submitted values, take the

below form which is the same as the previous one but has the `firstname` value changed from `sid` to `sid slayer` and `username` value changed from `sidthesloth` to `sid the sloth`.

```
<form action="/urlencoded?firstname=sid slayer&lastname=sloth" method="POST"
    <input type="text" name="username" value="sid the sloth"/>
    <input type="text" name="password" value="slothsecret"/>
    <input type="submit" value="Submit" />
</form>
```

Now try to submit the form and see how the form fields are transferred in the dev tools. Here is a dev tools snap in Chrome.

Clearly, you can see that the spaces are replaced by either '%20' or '+'. This is done for both the query parameters and the form body.

Read this to understand when + and %20 can be used. This encompasses the URL encoding process.

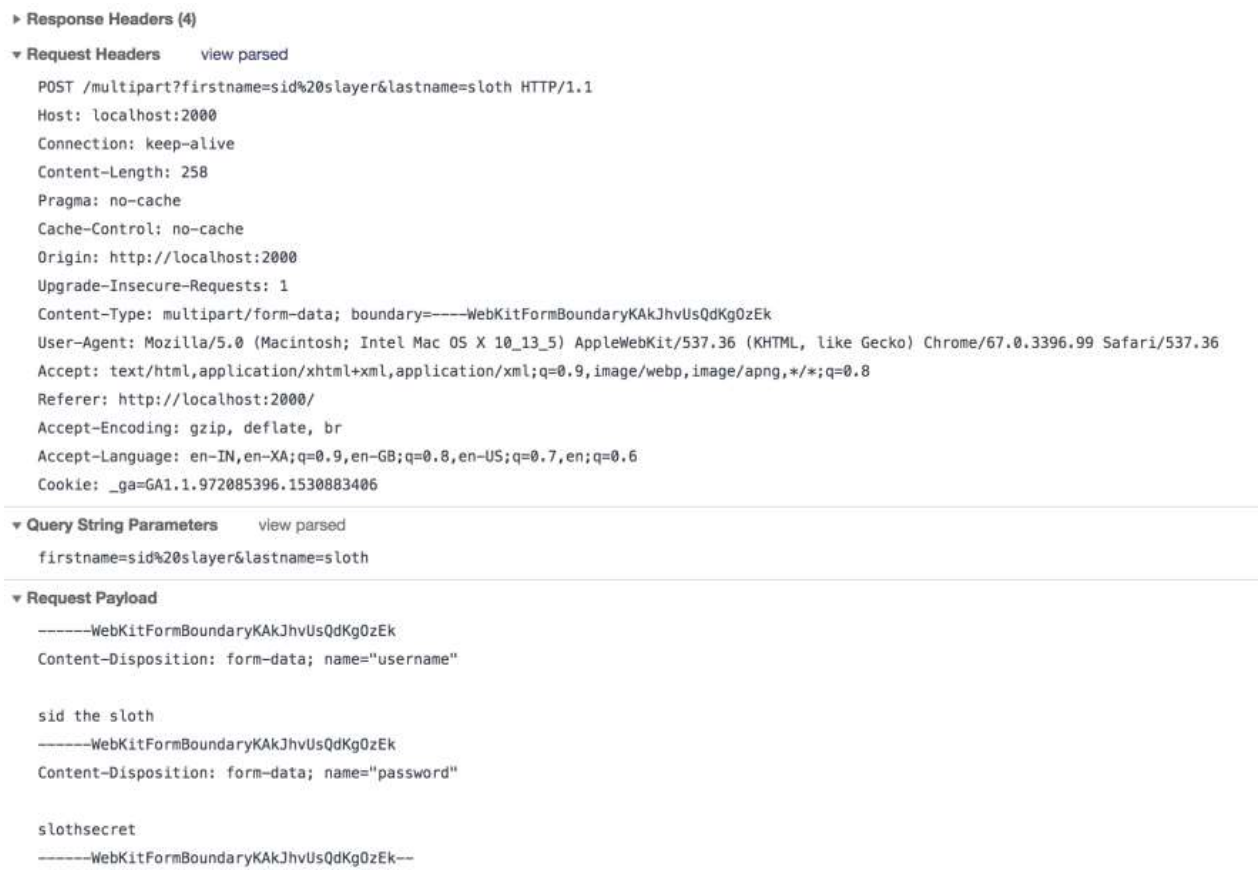
Multipart Forms

Multipart forms are generally used in contexts where the user needs files to be uploaded to the server. However, we'll just focus on simple text field based forms, as is it enough to understand how they work.

To convert the above form into a multipart form all you have to do is to change the `enctype` attribute of the form tag from `application/x-www-form-urlencoded` to `multipart/form-data`.

```
<form action="/multipart?firstname=sid slayer&lastname=sloth" method="POST"
    <input type="text" name="username" value="sid the sloth"/>
    <input type="text" name="password" value="slothsecret"/>
    <input type="submit" value="Submit" />
</form>
```

Let's go ahead and submit it and see how it appears in the dev tools.



There are the two things to notice here, the `Content-Type` header and the payload of the form request. Let's go through them one by one.

Content-Type Header

The value of the `Content-Type` header is obviously `multipart/form-data`. But it also has another value, `boundary`. The value for this in the example above is generated by the browser, but the user can very well define it as well, say for example, `boundary=sidtheslothboundary`. We'll get to see how it's useful in the next section.

Request Body

The request payload contains the form fields themselves. Each `(name, value)` pair is converted into a MIME message part of the following format,

```
--<<boundary_value>>
Content-Disposition: form-data; name="<<field_name>>"

<<field_value>>
```

The above format is repeated for each `(name, value)` pair.

Finally, the entire payload is terminated by the `boundary` value suffixed with a `--`. So the entire request looks like,

```
--<<boundary_value>>
Content-Disposition: form-data; name="<<field_name>>"

<<field_value>>
--<<boundary_value>>
Content-Disposition: form-data; name="<<field_name>>"

<<field_value>>
--<<boundary_value>>--
```

Now, we see how the boundary value is used.

In the case of an `application/x-www-form-urlencoded` form, the `&` ampersand kind of acts as a delimiter between each `(name, value)` pair, enabling the server to understand when and where a parameter value starts and ends.

```
username=sidthelsloth&password=slothsecret
```

In the case of a `multipart/form-data` form, the boundary value serves this purpose. Say if the boundary value was `XXX`,

the request payload would look like,

```
--XXX
Content-Disposition: form-data; name="username"

sidthesloth
--XXX
Content-Disposition: form-data; name="password"

slothsecret
--XXX--
```

The hyphens themselves are not part of the boundary value but rather needed as part of the request format. The `Content-Type` header for the above request would be,

```
Content-Type: multipart/form-data; boundary=XXX
```

This allows the browser to understand, when and where each field starts and ends.

Text/plain Forms

These forms are pretty much the same as the URL encoded forms, except that the form fields are not URL encoded when sent to the server. These are not used widely in general, but they have been introduced as a part of the HTML 5 specification.

Avoid using them as they meant for human understanding and for machines.

As quoted from the spec,

> Payloads using the text/plain format are intended to be human readable. They are not reliably interpretable by computer, as the format is ambiguous (for example, there is no way to distinguish a literal newline in a value from the newline at
> the end of the value).

Hope, I was clear in explaining what I learnt..See you in the next one guys..Peace.. :)

Dinesh Balaji + FOLLOW

Web/Hybrid Mobile Application developer | Open Source Enthusiast | Cat and Ice Cream Lover

@sidthesloth92 sidthesloth92 sidthesloth92 dbwriteups.wordpress.com

fonzane

Jun 3 '19

How do I make the POST-Request and its details visible in the chrome dev-tools? Whenever I submit the form, I only see the requests to the files that belong to the "page not found" page in the network tab of chrome.

How did you do it?

1

REPLY