URL encoding - high level summary

According to the rules (RFC 3986), characters in a URL are limited to a defined set of ASCII characters.

This means that any other characters are not allowed in a URL.

But URL often contains characters outside the US-ASCII character set, so they must be converted to a valid ASCII format for worldwide interoperability

URL-encoding (also known as percent-encoding) is a process of encoding URL information so that it can be safely transmitted over the internet.
To map the wide range of characters that is used worldwide, this is what happens:

- At first, the data is encoded according to the encoding type you've set in the form (usually UTF-8 that you specify in your `accept-charset` attribute)
- If the character you are trying to encode corresponds to characters in the **reserved set,** then the browser will percent-encode the character like **%HH**, where **HH** is the hexadecimal value of the byte
- If the character you are trying to encode falls within the set of ASCII characters (and is not a reserved character) then this is the best outcome – nothing happens, and the character does not have to be URL Encoded

BUT (just to add more confusion)

- If the character falls outside of your `accept-charset` encoding type, then the browser will attempt to convert that character to a numerical character reference. We have seen how the browser does this in a previous lecture.

Want an example?

Let's say a user of your form types "hello" in Chinese, which is 你好 (English equivalent is Ni Hao).

Now, let's take 2 scenarios:

1. `accept-charset="utf-8"`

   In this case, the URL will be encoded to 你好.

Pretty easy huh. The character does not clash with any reserved set, and it falls within the encoding type. You may be surprised to see the actual Chinese character showing up in the URL. Didn't we just learn that the URL should only accept ASCII characters. Remember the browser will display the actual Chinese characters even though they fall outside of the ASCII character set due to the browser address bar UI feature. In the background the URL still has to be encoded.

NB: we will talk more about international characters shortly.

2. `accept-charset="ISO-8859-1"`

   In this case, the URL will be encoded as **%26%2320320%3B%26%2322909%3B**.

   But do you know why?

   Well, now we have a character that does not fall within the encoding type you've specified in your form. And remember what format the browser will encode this to? If you recall, the format is **%#D;.** The % symbol is encoded to %26, the # symbol is encoded to %23, and the ; is encoded to %3B. This is pretty easy right. The only difference is the numerical character reference number in our case which is now **2320320** for the first Chinese character 你 and **2322909** for the second Chinese character 好. Combining all of this gives us our URL encoded value of **%26%2320320%3B%26%2322909%3B**.

A few more things I want to say

My brain is starting to hurt.

So, I only want to say 3 more things before ending this article:

1. Remember that reserved characters only need to be encoded **in particular contexts**, not just whenever they appear. For example, a forward slash is perfectly valid as a separator between the elements of your URL path, however it is not valid as part of the value in a `name=value` pair in a URL query string.

2. There are many specs that talk about URLs in the browser. IETF published RFC 3986 which defines valid URLs. But the IETF has no enforcement authority. It creates voluntary standards by consensus and then publishes them for all to use. It is up to the vendors to decide to implement these standards. But we've seen that the W3 has accepted this standard, and today all browsers are W3C compliant. So indirectly all browsers have accepted RFC 3986.

3. When it comes to international characters, the RFC 3987 was created which defines an IRI. Most browsers have an address bar that give users a nice UI experience. Therefore, we get to see a web address written in Japanese for example, even though these characters fall outside of the ASCII encoding type.

Whew!

A lot to take in, but I am hoping this all makes sense.

Keep on going, and I will see you in the next lecture :)