Different Types Of Encoding Schemes – A Primer

03/08/2009 · 1176 words · 6 min read

As a software developer and especially as a web developer you likely see/use different types of encoding every day. I know I come across all sorts of different encodings all the time. However since encoding is never really a central concept, it is often glossed over and it can sometimes be confusing which encoding is which and when each one is relevant. Well, to put the confusion to bed once and for all, here is a quick primer on the different types of encoding schemes you're likely to come across and when each one is relevant.

HTML Encoding

Html encoding is mainly used to represent various characters so that they can be safely used within an HTML document (as the name might suggest). As you know there are various characters that are part of the HTML markup itself (such as <, > etc.). To use these within the document as content you need to HTML encode them. There are two ways to HTML encode characters.

1. HTML encoding defines several entities to represent characters that can be part of the markup e.g.:

   ” - "
   ‘ - '
   > - >
   < - <
   & - &

2. You can also HTML encode any character using its ASCII code by prefixing it with &# and then using the ASCII decimal value, or prefixing it with &#x and using the ASCII hex value e.g.:

   ’ - '
   < - <
   > - >
   ‘ - "

So, any time you need to encode characters within an html document itself, those are the ways to do it. This one is sometimes confused with URL encoding which is somewhat

different.

URL Encoding

When dealing with URLs, they can only contain printable ASCII characters (these are characters with ASCII codes between decimal 32 and 126, i.e. hex 0x20 – 0x7E). However, some characters within this range may have special meanings within the URL or within the HTTP protocol. URL encoding comes into play when we have either some characters with special meaning in the URL or want to have characters outside the printable range. To URL encode a character we simply prefix its hex value with a % e.g.:

```
% - %25
space - %20
tab - %09
= - %3D
```

Note that as part of the URL encoding scheme you can also represent a space using +. This is often confused with Unicode encoding due to the fact that both begin with %, however Unicode encoding is a little bit more involved.

Unicode Encoding

Unicode encoding can be used to encode a character from any language or writing system in the world. Unicode encoding encompasses several encoding schemes.

1. 16 bit Unicode encoding is somewhat similar to URL encoding (which is why they can sometimes be confused). To encode a character in 16 bit Unicode, you start with %u and then append the *code point*, of the Unicode character that you want to encode. A code point is basically just a 4 digit hexadecimal number that maps to a particular character that you're trying to represent according to the Unicode standard (as you can imagine there are a LOT of these, have a look) e.g.:

```
@ - %u0040
∞ (infinity)- %u221E
```

2. UTF-8 is another type of Unicode encoding that uses one or more bytes to represent each character. When we need to UTF-8 encode characters for transmission over HTTP, we simply prefix each byte of the UTF-8 representation with a %. You can look up the UTF-8 representations for various characters by following the same link as ab ove.

```
± (plus minus) - %c2%b1
© (copyright) - %c2%a9
```

As you can imagine there is much, much more that could be said about Unicode, but this is only a quick primer so I'll leave it up to you to discover more for yourself.

Base64 Encoding

This one is fun and always useful to know about. Base64 is used to represent binary data using only printable characters. Usually it is used in basic HTTP authentication to encode user credentials, it is also used to encode e-mail attachments for transmission over SMPT. In addition Base64 encoding is sometimes used to transmit binary data inside cookies and other parameters, as well as often simply being used to make various data unreadable (or less easily readable) to prevent easy tampering.

Base64 looks at data in blocks of 3 bytes which is 24 bits. These 24 bits are then divided into 4 chunks of 6 bits each, and each of those chucks is then converted to its corresponding base64 value. Since it deals with 6 bit chunks there are 64 possible characters each chunk can map to (hence the name). If the end of the string that is being encoded contains less than 3 characters (which means we can make 4 base64 characters), then we make as many base64 characters as we can and pad the rest with = characters. You can have a look at the base 64 alphabet here as well as many other places. Lets have a look at a quick example.

If we want to convert the word ‘cake’ to base 64, we simply convert each of the characters to it’s ASCII decimal value and then get the binary value of each decimal value. In our case:

```
cake =  01100011011000010110101101100101
```

We now need to break up our binary string into chucks of 6 bits each, and since every 3 characters must make 4 base64 characters, we can pad with 0’s if we don’t have enough:

```
011000 110110 000101 101011 011001 010000 000000 000000
```

We now convert each one of the 6 bit binary values into it’s corresponding character, and in the case of an all zero value we use the padding character (=). In our case we get:

```
Y2FrZQ==
```

Which is the base64 encoded value of the word ‘cake’.

Base64 encoding is usually pretty easy to spot as it looks fairly distinctive especially when it contains padding characters (=) on the end of the string.

Hex Encoding

This one is easy, sort-of :). Basically with hex encoding, we simply use the hex value of every character to represent a collection of characters. So if we wanted to represent the word ‘hello’ it would be:

```
68656C6C6F
```

Pretty straight forward right? Well, it is only simple when we’re encoding printable ASCII characters (I mentioned those above). As soon as we need to encode international

characters of some sort, it becomes more complicated as we have to go back to Unicode. I wouldn't worry too much about this one, you're a lot less likely to encounter it in your day-to-day web development and you now know what it might be if you see a bunch hex numbers stuck together.

Hopefully this was helpful and you can refer to this – as a starting point – any time you have some encoding issues; to sort out any confusion. If you think there are other encoding schemes (relevant to our day to day lives as software developers) I should have mentioned, feel free to let me know in the comments.

#coding #encoding #encoding scheme

‹ **Are You The Best Developer In The World?**

Installing And Using SQLite With Ruby On Windows ›