Checking whether a URI refers to a Web site root

devblogs.microsoft.com/oldnewthing/20241127-00

November 27, 2024

Let's say you need to check whether a URI refers to a Web site root. For example, the Cross-Origin Resource Sharing (CORS) specification tracks *origin* in this way.

You might be tempted to use a regular expression to do your validation. However, parsing URIs is quite a complicated process, and regular expressions are not the right tool for the job. In general, you should try to defer the job of parsing things to the experts at parsing those things. (For example, your regular expression probably doesn't deal with things like %-escapes and IDN.) This means that instead of rolling your own URI or JSON or XML parser, you should use an existing, well-established (and therefore well-debugged) URI or JSON or XML parser.

Okay, so I've convinced you not to use your own parser. But how do you check whether a URI refers to a Web site root?

Well, to see if a URI refers to a Web site at all, you can check whether the scheme is `http` or `https`. And you might be tempted to check whether it refers to a Web site root by creating a new URI constructed from the scheme and host name and then comparing them.

```
// C#
// Do not use - This code isn't correct
static bool IsWebSiteRoot(string uriString)
{
    if (!Uri.TryCreate(uriString, UriKind.Absolute, out var uri))
    {
        return false;
    }
    var scheme = uri.Scheme;
    if (scheme != "http" && scheme != "https")
    {
        return false;
    }
    if (!Uri.TryCreate(scheme + "://" + uri.Host,
                       UriKind.Absolute, out var domainUri))
    {
        return false;
    }
    return uri.Equals(domainUri);
}

// C++/WinRT
// Do not use - This code isn't correct
bool IsWebSiteRoot(winrt::hstring const& uriString)
{
    try {
        auto uri = Uri(uriString);

        auto scheme = uri.SchemeName();
        if (scheme != L"http" && scheme != "https") {
            return false;
        }

        auto domainUri = Uri(scheme + L"://" + uri.Host());
        return (uri.Equals(domainUri));
    } catch (...) {
        return false;
    }
}
```

Again, the code is trying to be too smart. It’s not dealing with the possibility of a nondefault port or a nondefault user name. Keep trying to make somebody else do the work. In this case, we can ask the URI parser to strip the URI to the root.

```
// C#
// Still not quite there, but see remarks.
static bool IsWebSiteRoot(string uriString)
{
    if (!Uri.TryCreate(uriString, UriKind.Absolute, out var uri))
    {
        return false;
    }
    var scheme = uri.Scheme;
    if (scheme != "http" && scheme != "https")
    {
        return false;
    }
    if (!Uri.TryCreate(uri, "/", out var domainUri))
    {
        return false;
    }
    return uri.Equals(domainUri);
}
```

```
// C++/WinRT
// Still not quite there, but see remarks.
bool IsWebSiteRoot(winrt::hstring const& uriString)
{
    try {
        auto uri = Uri(uriString);

        auto scheme = uri.SchemeName();
        if (scheme != L"http" && scheme != "https") {
            return false;
        }

        auto domainUri = uri.CombineUri(L"/");
        return (uri.Equals(domainUri));
    } catch (...) {
        return false;
    }
}
```

This preserves any password, userid, and port.

There is still a hole here, namely that the original URI might end up being the root of a Web site, but only as a consequence of normalization. For example, `http://example.com/sub/..` will be treated as a Web site root according to the above function. Maybe that’s correct, since it does refer to the root of `http://example.com`.

For this customer’s usage, that wasn’t a problem. The requirement that the URI be the root of the Web site is more to avoid confusion over what the URI will be used for, and not as a security measure: These URIs are kept on a list of Web sites that a particular feature will use.

In the customer’s use case, they were keeping a list of Web sites, so we should go one step further and put the normalized URI on the list. That way, `http://example.com/` and `http://%45xample.com:80` are treated as the same URI.