*

Python JSON Emoji Crash Story

Sebastian Pipping <sebastian@pipping.org>

--

Berlin, 2020-02-18, v4

DISCLAIMER

Slides were done with (GNOME pinpoint and) very tight time constraints.

My apology, better slides next time!

1.

Django in Berlin at ~170 companies

https://github.com/hartwork/django-berlin#companies

2.

Who has a friend running...

- Django 3 <3.0.1
- Django 2 <2.2.9
- Django 1 <1.11.27

?

Please consider upgrading!

`CVE-2019-19844`

Potential **account hijack** via password reset form https://www.djangoproject.com/weblog/2019/dec/18/security-releases/

3.

Who has a friend running...

`settings.DEBUG == True`

accessible by public internet?

4.

Actual talk

Python JSON Emoji Crash Story

Tell a story

Point out a problem

Questions + Discussion

Environment:

- Django backend...
  with Django REST Framework

- A JavaScript frontend
  POST'ing JSON

Flow of data

1. User input
2. Form / HTML DOM
3. JavaScript
4. JSON (= ECMA-404)
5. HTTP request with body
6. Django REST Framework
7. `rest_framework.parsers.JSONParser`
8. De-serialization
9. *Some action* (e.g. store into database)

Unicode

`U+0000 — U+ffff`
Basic **M**ultilingual **P**lane

`U+10000 — U+10ffff`
16 "astral" planes

Emoji

beyond U+`ffff`

i.e. need more than 4 hex digits

Example:

Character 'GRINNING FACE'

Code point: `U+1F600`

Example glyph: 😀

Unicode characters in JSON

a) character itself as UTF-8
   (except `U+0` to `U+1f`, U+22, U+5C)

b) escaped a la (regex:) \\u[0-9a-fA-F]{4}

Works, Python:

```
In [1]: import json

In [2]: json.loads('"😀"')  # plain UTF-8
Out[2]: '😀'

In [3]: json.loads('"\\ud83d\\ude00"')
Out[3]: '😀'
```

1024 "high" surrogates (`U+D800–U+DBFF`)
1024 "low" surrogates (`U+DC00–U+DFFF`)

Pair of surrogates allows "addressing" any of the astral characters.

This is the very idea behind UTF-16.

```
(2**20 + 2**16 == 2**16 * 17)
```

Length of a string

Python:

```
In : len('😀')
Out: 1
```

JavaScript:

```
>> '😀'.length
<- 2
```

JavaScript:

```
>> '😬'.split('')
<- Array [ "\ud83d", "\ude00" ]
```

What if buggy code italified like this?:

JavaScript:

```
>> input_text.replace(/./g, '<em>$&</em>')
```

We send *single surrogates*

to the backend

JavaScript:

```
>> '😀'.replace(/./g, '[$&]').split('')
<- Array(6) [ "[", "\ud83d", "]", "[", "\ude00", "]" ]
```

How does *Python* deal with this?

Python:

```
In : json.loads('"[\\ud83d][\\ude00]"')
Out: '[\ud83d][\ude00]'
```

Surrogates in isolation

==

invalid characters

Python:

```
In : json.loads('"[\\ud83d][\\ude00]"').encode('utf-8')
[..]
UnicodeEncodeError: 'utf-8' codec can't encode character
  '\ud83d' in position 1: surrogates not allowed
```

Fixed for `CharField`

in next release (3.12.0?) of

Django REST Framework

https://github.com/encode/django-rest-framework/pull/7067

https://github.com/encode/django-rest-framework/issues/7026

"Unfixed" in CPython's JSON decoder

Considered a feature upstream

Potentially for good reasons

https://docs.python.org/3/library/json.html#character-encodings

Playing with surrogate characters

```
# pip3 install surrogates
```

https://github.com/hartwork/surrogates#usage

Consequences?

- Produce error 500 on any(?) DRF deployed today
- (Read secrets if `DEBUG=True`)
- Catch early once 👍 or late everywhere 👎
- ?

Coping strategies?

Thank you!

Sebastian Pipping <sebastian@pipping.org>