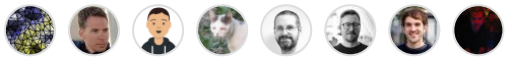
Evilebot Tnawi feat: the `resourceQuery` is passed to the `interpolateName` me... ... ✓ History

8 contributors 

151 lines (119 sloc) | 3.69 KB ...

```
1  'use strict';
2
3  const path = require('path');
4  const emojisList = require('emojis-list');
5  const getHashDigest = require('./getHashDigest');
6
7  const emojiRegex = /[\uD800-\uDFFF]./;
8  const emojiList = emojisList.filter((emoji) => emojiRegex.test(emoji));
9  const emojiCache = {};
10
11 function encodeStringToEmoji(content, length) {
12   if (emojiCache[content]) {
13     return emojiCache[content];
14   }
15
16   length = length || 1;
17
18   const emojis = [];
19
20   do {
21     if (!emojiList.length) {
22       throw new Error('Ran out of emoji');
23     }
24
25     const index = Math.floor(Math.random() * emojiList.length);
26
27     emojis.push(emojiList[index]);
28     emojiList.splice(index, 1);
29   } while (--length > 0);
```

```
30
31    const emojiEncoding = emojis.join('');
32
33    emojiCache[content] = emojiEncoding;
34
35    return emojiEncoding;
36  }
37
38  function interpolateName(loaderContext, name, options) {
39    let filename;
40
41    const hasQuery =
42      loaderContext.resourceQuery && loaderContext.resourceQuery.length > 1;
43
44    if (typeof name === 'function') {
45      filename = name(
46        loaderContext.resourcePath,
47        hasQuery ? loaderContext.resourceQuery : undefined
48      );
49    } else {
50      filename = name || '[hash].[ext]';
51    }
52
53    const context = options.context;
54    const content = options.content;
55    const regExp = options.regExp;
56
57    let ext = 'bin';
58    let basename = 'file';
59    let directory = '';
60    let folder = '';
61    let query = '';
62
63    if (loaderContext.resourcePath) {
64      const parsed = path.parse(loaderContext.resourcePath);
65      let resourcePath = loaderContext.resourcePath;
66
67      if (parsed.ext) {
68        ext = parsed.ext.substr(1);
69      }
70
71      if (parsed.dir) {
72        basename = parsed.name;
73        resourcePath = parsed.dir + path.sep;
74      }
75
76      if (typeof context !== 'undefined') {
77        directory = path
78          .relative(context, resourcePath + '_')
```

```
        .replace(/\\/g, '/')
        .replace(/\.\.(\/)?/g, '_$1');
      directory = directory.substr(0, directory.length - 1);
    } else {
      directory = resourcePath.replace(/\\/g, '/').replace(/\.\.(\/)?/g, '_$1');
    }

    if (directory.length === 1) {
      directory = '';
    } else if (directory.length > 1) {
      folder = path.basename(directory);
    }
  }

  if (loaderContext.resourceQuery && loaderContext.resourceQuery.length > 1) {
    query = loaderContext.resourceQuery;

    const hashIdx = query.indexOf('#');

    if (hashIdx >= 0) {
      query = query.substr(0, hashIdx);
    }
  }

  let url = filename;

  if (content) {
    // Match hash template
    url = url
      // `hash` and `contenthash` are same in `loader-utils` context
      // let's keep `hash` for backward compatibility
      .replace(
        /\[(?:([^:\]]+):)?(?:hash|contenthash)(?::([a-z]+\d*))?(?::(\d+))?\]/gi,
        (all, hashType, digestType, maxLength) =>
          getHashDigest(content, hashType, digestType, parseInt(maxLength, 10))
      )
      .replace(/\[emoji(?::(\d+))?\]/gi, (all, length) =>
        encodeStringToEmoji(content, parseInt(length, 10))
      );
  }

  url = url
    .replace(/\[ext\]/gi, () => ext)
    .replace(/\[name\]/gi, () => basename)
    .replace(/\[path\]/gi, () => directory)
    .replace(/\[folder\]/gi, () => folder)
    .replace(/\[query\]/gi, () => query);

  if (regExp && loaderContext.resourcePath) {
```

```
    const match = loaderContext.resourcePath.match(new RegExp(regExp));

    match &&
      match.forEach((matched, i) => {
        url = url.replace(new RegExp('\\[' + i + '\\]', 'ig'), matched);
      });
  }

  if (
    typeof loaderContext.options === 'object' &&
    typeof loaderContext.options.customInterpolateName === 'function'
  ) {
    url = loaderContext.options.customInterpolateName.call(
      loaderContext,
      url,
      name,
      options
    );
  }

  return url;
}

module.exports = interpolateName;
```