

🔗 d4bafd9f1d ▾

...

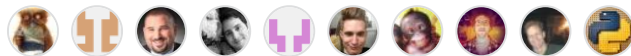
[oauthlib](#) / [oauthlib](#) / [oauth2](#) / [rfc6749](#) / [grant\\_types](#) / [base.py](#) / <> Jump to ▾



luhn Add CORS support for Refresh Token Grant.

🕒 History

👤 10 contributors



268 lines (222 sloc) | 10.7 KB

...

```

1  """
2  oauthlib.oauth2.rfc6749.grant_types
3  ~~~~~
4  """
5  import logging
6  from itertools import chain
7
8  from oauthlib.common import add_params_to_uri
9  from oauthlib.oauth2.rfc6749 import errors, utils
10 from oauthlib.uri_validate import is_absolute_uri
11
12 from ..request_validator import RequestValidator
13 from ..utils import is_secure_transport
14
15 log = logging.getLogger(__name__)
16
17
18 class ValidatorsContainer:
19     """
20     Container object for holding custom validator callables to be invoked
21     as part of the grant type `validate_authorization_request()` or
22     `validate_authorization_request()` methods on the various grant types.
23
24     Authorization validators must be callables that take a request object and
25     return a dict, which may contain items to be added to the `request_info`
26     returned from the grant_type after validation.
27
28     Token validators must be callables that take a request object and
29     return None.

```

```

30
31 Both authorization validators and token validators may raise OAuth2
32 exceptions if validation conditions fail.
33
34 Authorization validators added to `pre_auth` will be run BEFORE
35 the standard validations (but after the critical ones that raise
36 fatal errors) as part of `validate_authorization_request()`
37
38 Authorization validators added to `post_auth` will be run AFTER
39 the standard validations as part of `validate_authorization_request()`
40
41 Token validators added to `pre_token` will be run BEFORE
42 the standard validations as part of `validate_token_request()`
43
44 Token validators added to `post_token` will be run AFTER
45 the standard validations as part of `validate_token_request()`
46
47 For example:
48
49 >>> def my_auth_validator(request):
50 ...     return {'myval': True}
51 >>> auth_code_grant = AuthorizationCodeGrant(request_validator)
52 >>> auth_code_grant.custom_validators.pre_auth.append(my_auth_validator)
53 >>> def my_token_validator(request):
54 ...     if not request.everything_okay:
55 ...         raise errors.OAuth2Error("uh-oh")
56 >>> auth_code_grant.custom_validators.post_token.append(my_token_validator)
57 """
58
59 def __init__(self, post_auth, post_token,
60              pre_auth, pre_token):
61     self.pre_auth = pre_auth
62     self.post_auth = post_auth
63     self.pre_token = pre_token
64     self.post_token = post_token
65
66 @property
67 def all_pre(self):
68     return chain(self.pre_auth, self.pre_token)
69
70 @property
71 def all_post(self):
72     return chain(self.post_auth, self.post_token)
73
74
75 class GrantTypeBase:
76     error_uri = None
77     request_validator = None
78     default_response_mode = 'fragment'

```

```

79 refresh_token = True
80 response_types = ['code']
81
82 def __init__(self, request_validator=None, **kwargs):
83     self.request_validator = request_validator or RequestValidator()
84
85     # Transforms class variables into instance variables:
86     self.response_types = self.response_types
87     self.refresh_token = self.refresh_token
88     self._setup_custom_validators(kwargs)
89     self._code_modifiers = []
90     self._token_modifiers = []
91
92     for kw, val in kwargs.items():
93         setattr(self, kw, val)
94
95 def _setup_custom_validators(self, kwargs):
96     post_auth = kwargs.get('post_auth', [])
97     post_token = kwargs.get('post_token', [])
98     pre_auth = kwargs.get('pre_auth', [])
99     pre_token = kwargs.get('pre_token', [])
100     if not hasattr(self, 'validate_authorization_request'):
101         if post_auth or pre_auth:
102             msg = ("{} does not support authorization validators. Use "
103                  "token validators instead.").format(self.__class__.__name__)
104             raise ValueError(msg)
105         # Using tuples here because they can't be appended to:
106         post_auth, pre_auth = (), ()
107         self.custom_validators = ValidatorsContainer(post_auth, post_token,
108                                                     pre_auth, pre_token)
109
110 def register_response_type(self, response_type):
111     self.response_types.append(response_type)
112
113 def register_code_modifier(self, modifier):
114     self._code_modifiers.append(modifier)
115
116 def register_token_modifier(self, modifier):
117     self._token_modifiers.append(modifier)
118
119 def create_authorization_response(self, request, token_handler):
120     """
121     :param request: OAuthlib request.
122     :type request: oauthlib.common.Request
123     :param token_handler: A token handler instance, for example of type
124                          oauthlib.oauth2.BearerToken.
125     """
126     raise NotImplementedError('Subclasses must implement this method.')
127

```

```

128     def create_token_response(self, request, token_handler):
129         """
130         :param request: OAuthlib request.
131         :type request: oauthlib.common.Request
132         :param token_handler: A token handler instance, for example of type
133                             oauthlib.oauth2.BearerToken.
134         """
135         raise NotImplementedError('Subclasses must implement this method.')
136
137     def add_token(self, token, token_handler, request):
138         """
139         :param token:
140         :param token_handler: A token handler instance, for example of type
141                             oauthlib.oauth2.BearerToken.
142         :param request: OAuthlib request.
143         :type request: oauthlib.common.Request
144         """
145         # Only add a hybrid access token on auth step if asked for
146         if not request.response_type in ["token", "code token", "id_token token", "code id_token t
147             return token
148
149         token.update(token_handler.create_token(request, refresh_token=False))
150         return token
151
152     def validate_grant_type(self, request):
153         """
154         :param request: OAuthlib request.
155         :type request: oauthlib.common.Request
156         """
157         client_id = getattr(request, 'client_id', None)
158         if not self.request_validator.validate_grant_type(client_id,
159                                                         request.grant_type, request.client, requ
160             log.debug('Unauthorized from %r (%r) access to grant type %s.',
161                     request.client_id, request.client, request.grant_type)
162             raise errors.UnauthorizedClientError(request=request)
163
164     def validate_scopes(self, request):
165         """
166         :param request: OAuthlib request.
167         :type request: oauthlib.common.Request
168         """
169         if not request.scopes:
170             request.scopes = utils.scope_to_list(request.scope) or utils.scope_to_list(
171                 self.request_validator.get_default_scopes(request.client_id, request))
172         log.debug('Validating access to scopes %r for client %r (%r).',
173                 request.scopes, request.client_id, request.client)
174         if not self.request_validator.validate_scopes(request.client_id,
175                                                         request.scopes, request.client, request):
176             raise errors.InvalidScopeError(request=request)

```

```

177
178 def prepare_authorization_response(self, request, token, headers, body, status):
179     """Place token according to response mode.
180
181     Base classes can define a default response mode for their authorization
182     response by overriding the static `default_response_mode` member.
183
184     :param request: OAuthlib request.
185     :type request: oauthlib.common.Request
186     :param token:
187     :param headers:
188     :param body:
189     :param status:
190     """
191     request.response_mode = request.response_mode or self.default_response_mode
192
193     if request.response_mode not in ('query', 'fragment'):
194         log.debug('Overriding invalid response mode %s with %s',
195                 request.response_mode, self.default_response_mode)
196         request.response_mode = self.default_response_mode
197
198     token_items = token.items()
199
200     if request.response_type == 'none':
201         state = token.get('state', None)
202         if state:
203             token_items = [('state', state)]
204         else:
205             token_items = []
206
207     if request.response_mode == 'query':
208         headers['Location'] = add_params_to_uri(
209             request.redirect_uri, token_items, fragment=False)
210         return headers, body, status
211
212     if request.response_mode == 'fragment':
213         headers['Location'] = add_params_to_uri(
214             request.redirect_uri, token_items, fragment=True)
215         return headers, body, status
216
217     raise NotImplementedError(
218         'Subclasses must set a valid default_response_mode')
219
220 def _get_default_headers(self):
221     """Create default headers for grant responses."""
222     return {
223         'Content-Type': 'application/json',
224         'Cache-Control': 'no-store',
225         'Pragma': 'no-cache',

```

```

226     }
227
228     def _handle_redirects(self, request):
229         if request.redirect_uri is not None:
230             request.using_default_redirect_uri = False
231             log.debug('Using provided redirect_uri %s', request.redirect_uri)
232             if not is_absolute_uri(request.redirect_uri):
233                 raise errors.InvalidRedirectURIError(request=request)
234
235             # The authorization server MUST verify that the redirection URI
236             # to which it will redirect the access token matches a
237             # redirection URI registered by the client as described in
238             # Section 3.1.2.
239             # https://tools.ietf.org/html/rfc6749#section-3.1.2
240             if not self.request_validator.validate_redirect_uri(
241                 request.client_id, request.redirect_uri, request):
242                 raise errors.MismatchingRedirectURIError(request=request)
243         else:
244             request.redirect_uri = self.request_validator.get_default_redirect_uri(
245                 request.client_id, request)
246             request.using_default_redirect_uri = True
247             log.debug('Using default redirect_uri %s.', request.redirect_uri)
248             if not request.redirect_uri:
249                 raise errors.MissingRedirectURIError(request=request)
250             if not is_absolute_uri(request.redirect_uri):
251                 raise errors.InvalidRedirectURIError(request=request)
252
253     def _create_cors_headers(self, request):
254         """If CORS is allowed, create the appropriate headers."""
255         if 'origin' not in request.headers:
256             return {}
257
258         origin = request.headers['origin']
259         if not is_secure_transport(origin):
260             log.debug('Origin "%s" is not HTTPS, CORS not allowed.', origin)
261             return {}
262         elif not self.request_validator.is_origin_allowed(
263             request.client_id, origin, request):
264             log.debug('Invalid origin "%s", CORS not allowed.', origin)
265             return {}
266         else:
267             log.debug('Valid origin "%s", injecting CORS headers.', origin)
268             return {'Access-Control-Allow-Origin': origin}

```