INF5190 - Intégration d'API Web

Jean-Philippe Caissy

2 octobre 2019

Intégration d'API Web

- L'intégration d'une API Web consiste à consommer l'API et l'exposer au client
- Abstraction des points d'entrées, de la sérialisation, du mécanisme de communication
- Similaire au patron `Active Record`, on peut exposer en objets les ressources distants
- Une application Web (serveur) qui doit utiliser une API Web devient un client
  - Une application Web peut être à la fois un client et un serveur d'API

Intégration d'API Web

Client

- Un client d'API Web s'occupe d'intégrer une API web distante
- Responsabilités :
  - Respecter le format de sérialisation (JSON, XML, etc)
  - Utiliser le bon canal de communication (HTTP, socket, etc)
  - Effectuer une validation initiale des entrants (champs obligatoires, types, etc)
  - S'authentifier (si nécessaire)
  - Gérer les erreurs
  - Effectuer la mise en cache lorsque possible

Intégration d'API Web

Client

```
from mon_projet.api import Client, Compte

client = Client("http://api.example.com/")

compte = client.Compte.get_by_id(456)
# GET api.example.com/compte/456
# Content-Type: application/json

compte.id
# 456
```

Intégration d'API Web

Client

Une certaine validation peut avoir lieu sur le client en respectant les contraintes de l'API.
e.g. : les identifiants de comptes `id` de l'API sont des entiers.

```
from mon_projet.api import Client, Compte

client = Client("http://api.example.com/")

compte = client.Compte.get_by_id("toto")
# Traceback (most recent call last):
#  File "<stdin>", line 1, in <module>
# Exception: L'identification du compte doit être un
↪  entier
```

La validation des données par un client d'API n'est pas obligatoire, mais permet d'éliminer des appels inutiles

Intégration d'API Web

Client

Dans le cas d'une API REST, HTTP est utilisé comme canal de communication.

- Sérialiser les données (au besoin)
- Faire la requête HTTP (URL, méthode http, données)
- Désérialiser la réponse
- Si erreur : gérer le(s) erreur(s) retournées si présentes

Intégration d'API Web

Exemple

API officiel de Github (https://github.com/PyGithub/PyGithub)

```
from github import Github

g = Github("user", "password")

for repo in g.get_user().get_repos():
    print(repo.name)

repo = g.get_user().get_repo("PyGithub/PyGithub")

issue = repo.get_issue(number=874)
# Issue(title="PyGithub example usage", number=874)

repo.create_issue(title="This is a new issue",
 ↪  body="This is the issue body")
# Issue(title="This is a new issue", number=XXX)
```

Intégration d'API Web

Exemple

La méthode `create_issue` de la classe `Repository`
https://github.com/PyGithub/PyGithub/blob/c9ed82b211bbd97dea805c0b63971c424832e924/github/Repository.py#L1046-L1085

Intégration d'API Web

Exemple

```
    def create_issue(self, title, body=None, [...])
        """
        :calls: `POST /repos/:owner/:repo/issues
↪   <http://developer.github.com/v3/issues>`
        :param title: string
        :param body: string
        [...]
        :rtype: :class:`github.Issue.Issue`
        """
```

Documentation! De type similaire à JavaDoc en tant que commentaire

- Identifie l'URL et la méthode HTTP
- Définition des paramètres
- Type d'objet retourné

Intégration d'API Web

Exemple

N.B. : L'exemple est simplifié par rapport au code de la librairie

```
def create_issue(self, title, body=None, [..]):
    [...]
    assert isinstance(title, str), title
    assert body == None or isinstance(body, str),
    ↪  body
    [...]
```

Validation des paramètres :

- `title` doit être un `string`
- `body` peut être nulle, ou un `string`

Intégration d'API Web

Exemple

N.B. : L'exemple est simplifié par rapport au code de la librairie

```
def create_issue(self, title, body=None, [..]):
    [...]
    post_parameters = {
        "title": title,
    }
    if body is not github.GithubObject.NotSet:
        post_parameters["body"] = body
    [...]
```

Construction du corps de la requête :

- `title` est obligatoire et sera présent
- `body` est facultatif

Intégration d'API Web

Exemple

```
    def create_issue(self, title, body=None, [..]):
        [...]
        headers, data =
↪   self._requester.requestJsonAndCheck(
            "POST",
            self.url + "/issues",
            input=post_parameters
        )
        return github.Issue.Issue(self._requester,
        ↪  headers, data, completed=True)
```

Envoie de la requête :

- Sérialisation avec `requestJsonAndCheck`
- Méthode HTTP `POST` (création d'un issue)
- Objet de retour de type `Issue`

Intégration d'API Web

Communication

Sous Python 3, la libraire standard `urllib` permet d'instancier un client HTTP pour faire des requêtes

```
>>> from urllib.request import urlopen
>>> with urlopen('http://python.org/') as response:
...     html = response.read()
...     headers = response.headers
...     http_code = reponse.code
...     reason = reponse.reason
...
>>> headers['Content-Type']
'text/html; charset=utf-8'
>>> http_code, reason
(200, 'OK')
>>> html
b'<!doctype html>\n<!--[if lt IE 7]>[...]
```

Intégration d'API Web

Exemple `urllib`

Pour envoyer des données JSON, en premier on doit les sérialiser

```
>>> import json
>>> json_payload = {"toto": "tata", "foobar": [1, 2]}
>>> params = json.dumps(json_payload).encode('utf8')
```

Intégration d'API Web

Exemple `urllib`

Ensuite on utiliser `urllib` pour envoyer la requête

```
>>> from urllib.request import Request, urlopen
>>> req = urllib.request.Request(
...     "https://api.example.com/api", data=params,
...      headers={
...          'content-type': 'application/json'
...      }
... )
>>> with urlopen(req) as response:
...     data = response.read()
...     headers = response.headers
...     http_code = reponse.code
```

Intégration d'API Web

Exemple `urllib`

Finalement on vérifie le code d'erreur, on s'assure d'une réponse JSON et on désérialise

```
>>> assert http_code == 200
>>> assert headers['Content-Type'] ==
 ↪  "application/json"
>>> payload = json.loads(data)
```

Intégration d'API Web

REST API

Voici l'aperçu global d'une requête client pour un API REST

1. Validation des données
2. Sérialisation de la requête
3. Envoie de la requête HTTP
4. Gestion d'erreur
5. Désérialisation de la réponse

Exercice

#TODO

Intégration d'API Web

Gestion d'erreur

La gestion d'erreur se fait sur trois niveaux différents :

- Données du clients (mauvais types, champs manquants, etc)
- Canal de communication (fermeture du socket, timeout sur la lecture, etc)
- Les erreurs retournés par le serveur

Intégration d'API Web

Gestion d'erreur

Sert à identifier au client de l'API ce qui n'est pas valide.
Exemple commun d'erreurs :

- Champ manquant ou invalide
- Contrainte applicative non respectée

Intégration d'API Web

Gestion d'erreur

Code HTTP

Une bonne pratique est d'utiliser un code HTTP lors d'un erreur

Code HTTP	Description
`400 Bad Request`	Mauvaise requête (JSON invalide)
`401 Unauthorized`	Permission invalide
`404 Not Found`	Ressource non trouvée
`422 Unprocessable Entity`	Erreur sémantique (champ manquant)

Intégration d'API Web

Gestion d'erreur

Mauvais exemple

```
200 OK

{"errors": {
    "prenom": {
        "code": "missing",
        "text": "Champ manquant"
    },
    "date_naissance": {
        "code": "invalid",
        "text": "La date de naissance est invalide"
    }
}
```

Intégration d'API Web

Gestion d'erreur

Bon exemple

```
422 Unprocessable Entity
```

```
{"errors": {
    "prenom": {
        "code": "missing",
        "text": "Champ manquant"
    },
    "date_naissance": {
        "code": "invalid",
        "text": "La date de naissance est invalide"
    }
}
```

Intégration d'API Web

Gestion d'erreur

Mauvais exemple

```
200 OK

{"error": {
    "base": {
        "code": "does-not-exist",
        "text": "La ressource n'existe pas"
    }
}
```

Intégration d'API Web

Gestion d'erreur

Bon exemple

```
404 OK

{"error": {
    "base": {
        "code": "does-not-exist",
        "text": "La ressource n'existe pas"
    }
}
```

Intégration d'API Web

Gestion d'erreur

Bon exemple

```
404 OK
```

Pas besoin de corps de réponse, le code 404 est suffisant pour identifier une ressource non existante.

Intégration d'API Web

Gestion d'erreur

- Utiliser les code de statut HTTP
- Inclure un message d'erreur pour les développeurs (code)
- Inclure un message d'erreur pour l'utilisateur (texte)
- Différencier les erreurs de clients (400 Bad Request) des erreurs de serveurs (4xx)

Liens

- HOWTO Fetch Internet Resources Using The urllib Package
- Github REST API v3
- Exemple de client Python pour l'api REST de Github