Theory: Requests: manipulating data

13 minutes 0 / 5 problems solved Skip this topic Start practicing

599 users solved this topic. Latest completion was 3 days ago.

Most often we browse the web fetching some data, but when we log in to a social network, add an item to the cart in some online store or delete a picture posted earlier, we send information to the server to change its state. For this purpose we use such HTTP methods as POST, PUT and DELETE. In this topic, you'll learn how to send these common types of requests in Python.

§1. Setting up

First, you might need to install `requests`, a Python library for making HTTP requests:

```
pip install requests
```

To start using `requests` in your code, import the library. It will allow you to send different kinds of requests and get the responses to them from the server in a special **response** object.

§2. requests.post()

A POST request allows you to send some additional data to the server. Unlike GET requests, where you could specify additional parameters in the query string, POST requests pass additional data in the **body** of the message (request).

With the `requests` library, you can make a POST request using `requests.post`, specifying additional data with the `data` parameter, e.g. as a dictionary.

Imagine you want to add a new blog post to your personal blog. Then you would write something like this:

```
import requests


data = {'post_text': 'Today, I started learning Python. Exciting!'}
r = requests.post('http://bestblogever.com/posts', data=data)
```

§3. requests.put()

PUT method replaces the resource at the given URL with the resource specified within the request. If there is originally no resource to replace on the server, PUT will create one.

So if you need to update a specific post you published earlier, you will send a PUT request similar to this one:

```
import requests


data = {'post_text': 'some_updated_text'}
r = requests.put('http://bestblogever.com/posts/<post_id>', data=data)
```

§4. requests.delete()

You might also want to remove a post from your blog. In order to do so, you'll need to send a DELETE request, which deletes the resource identified by the request URL. This is straightforward with `requests.delete()`:

Current topic:

Requests: manipulating data

Topic depends on:

✗ HTTP messages Stage 1

✓ Dictionary 6★ Stage 1

✗ Pip Stage 1

Table of contents:

```
import requests


r = requests.delete('http://bestblogever.com/posts/<post_id>')
```

§5. Idempotency

An important property of some HTTP requests is idempotency. A request is called idempotent if sending the same request more than once doesn't introduce additional changes to the state of the server. Can you say which of the requests mentioned in this topic have this property?

Obviously, GET is idempotent because it just collects data from the server without introducing any changes to it. PUT is idempotent as well since if you PUT the same object multiple times, it will have no effect comparing to sending a PUT request just once. And, of course, if you DELETE a resource once, the state of the server will not change any further when you send the same request more times, meaning that DELETE is idempotent.

However, note that POST isn't idempotent, because a POST request may add new data to the server (e.g., a new record to a database), and doing so multiple times is different from adding it just once.

§6. Conclusions

Let's highlight the main points:

- POST is the most common request method used to send data to the server.
- To make a POST request with `requests`, use `requests.post(url, data)`.
- PUT requests are used to update the resource on the server. They can be sent with `requests.put(url, data)`.
- DELETE request `requests.delete(url)` removes a resource at the given URL from the server.
- An HTTP request is idempotent if sending several identical requests has the same result as sending just a single one.
- GET, PUT and DELETE are idempotent, while POST isn't.

Report a typo

60 users liked this theory. 2 didn't like it. What about you?

😍 🙂 😐 🙁 😡

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion