CS1520 Recitation: RESTful Flask

...

Jeongmin Lee

Plan for Today

- Install RESTful-Flask
- Simple Example
- Resourceful Routing
- Add resource
- Argument Parsing
- Data Formatting
- Full Example

Install Flask-RESTful

- >> pip install flask-restful

Simple Example

- 1. Save it as api.py

```
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class HelloWorld(Resource):
    def get(self):
        return {'hello': 'world'}

api.add_resource(HelloWorld, '/')

if __name__ == '__main__':
    app.run(debug=True)
```

- 2. Run it

```
$ python api.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

- 3/ Get it

```
$ curl http://127.0.0.1:5000/
```

Plan for Today

- Install RESTful-Flask
- Simple Example
- Resourceful Routing
- Add resource
- Argument Parsing
- Data Formatting
- Full Example

Resourceful Routing

- **Resources are building block of Flask-RESTful.**
- Resources are built on top of Flask pluggable views
  - Giving you easy access to **multiple HTTP methods** just by defining methods on your resource

Basic CRUD example (crud_example.py)

```python
from flask import Flask, request
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)
todos = {}
class TodoSimple(Resource):
    def get(self, todo_id):
        return {todo_id: todos[todo_id]}

    def put(self, todo_id):
        todos[todo_id] = request.form['data']
        return {todo_id: todos[todo_id]}

api.add_resource(TodoSimple, '/<string:todo_id>')

if __name__ == '__main__':
    app.run(debug=True)
```

```
python crud_example.py
```

```
$ curl http://localhost:5000/todo1 -d "data=Remember
the milk" -X PUT
{"todo1": "Remember the milk"}

$ curl http://localhost:5000/todo1
{"todo1": "Remember the milk"}

$ curl http://localhost:5000/todo1 GET
{"todo1": "Remember the milk"}

$ curl http://localhost:5000/todo2 -d "data=Change my
brakepads" -X PUT
{"todo2": "Change my brakepads"}

$ curl http://localhost:5000/todo2
{"todo2": "Change my brakepads"}
```

Plan for Today

- Install RESTful-Flask
- Simple Example
- Resourceful Routing
- Add resource
- Argument Parsing
- Data Formatting
- Full Example

add_resource

- Many times in an API, your resource will have multiple URLs.
- You can pass multiple URLs to the add_resource() method on the Api object. Each one will be routed to your Resource

```
api.add_resource(HelloWorld,
    '/',
    '/hello')
```

add_resource

- **add_resource**(resource, *urls, **kwargs)
  - Adds a resource to the api.
  - **resource** (Resource) – the class name of your resource
  - **urls** (str) – one or more url routes to match for the resource. Any url variables will be passed to the resource method as args.
  - **endpoint** (str) – identifier that is used in determining what logical unit (e.g., a function) of your code should handle the request (Detail explanation: https://stackoverflow.com/a/19262349/6697629)

Plan for Today

- Install RESTful-Flask
- Simple Example
- Resourceful Routing
- Add resource
- **Argument Parsing**
- Data Formatting
- Full Example

Argument Parsing

- Argparse is a python library that helps you to **parse input argument**. Learning by example!

```
import argparse

parser = argparse.ArgumentParser(description='Short sample app')

parser.add_argument('-a', action="store_true", default=False)
parser.add_argument('-b', action="store", dest="b")
parser.add_argument('-c', action="store", dest="c", type=int)

print parser.parse_args(['-a', '-bval', '-c', '3'])

>> Namespace(a=True, b='val', c=3)
```

Argument Parsing

- In Flask-RESTful, it's a pain to **validate** form data
- Flask-RESTful has built-in support for request data validation using a library similar to argparse

```
from flask_restful import reqparse

parser = reqparse.RequestParser()
parser.add_argument('rate', type=int, help='Rate to charge for this
resource')
args = parser.parse_args()
```

Plan for Today

- Install RESTful-Flask
- Simple Example
- Resourceful Routing
- Add resource
- Argument Parsing
- Data Formatting
- Full Example

Data Formatting

- By default, all fields in your return iterable will be rendered as-is.
- While this works great when you’re just dealing with Python data structures

Data Formatting

- But, it can become very frustrating when working with objects.
- To solve this problem, Flask-RESTful provides the fields module and the marshal_with() decorator.
- You can use the fields module to describe the structure of your response

```python
from flask_restful import fields, marshal_with

resource_fields = {
    'task':   fields.String,
    'uri':    fields.Url('todo_ep')
}

class TodoDao(object):
    def __init__(self, todo_id, task):
        self.todo_id = todo_id
        self.task = task

        # This field will not be sent in the response
        self.status = 'active'

class Todo(Resource):
    @marshal_with(resource_fields)
    def get(self, **kwargs):
        return TodoDao(todo_id='my_todo',
                task='Remember the milk')
```

Assemble a value as a string

```python
from flask_restful import fields, marshal_with

resource_fields = {
    'task':   fields.String,
    'uri':    fields.Url('todo_ep')
}

class TodoDao(object):
    def __init__(self, todo_id, task):
        self.todo_id = todo_id
        self.task = task

        # This field will not be sent in the response
        self.status = 'active'

class Todo(Resource):
    @marshal_with(resource_fields)
    def get(self, **kwargs):
        return TodoDao(todo_id='my_todo',
                task='Remember the milk')
```

A **string representation** of a Url. It takes an **endpoint name** and generates a URL for that endpoint in the response.

```python
from flask_restful import fields, marshal_with

resource_fields = {
    'task':   fields.String,
    'uri':    fields.Url('todo_ep')
}

class TodoDao(object):
    def __init__(self, todo_id, task):
        self.todo_id = todo_id
        self.task = task

        # This field will not be sent in the response
        self.status = 'active'

class Todo(Resource):
    @marshal_with(resource_fields)
    def get(self, **kwargs):
        return TodoDao(todo_id='my_todo',
                       task='Remember the milk')
```

apply the transformation described by **resource_fields**.

Plan for Today

- Install RESTful-Flask
- Simple Example
- Resourceful Routing
- Add resource
- Argument Parsing
- Data Formatting
- Full Example

Full Example

- By default, all fields in your return iterable will be rendered as-is.
- While this works great when you're just dealing with Python data structures
- http://flask-restful.readthedocs.io/en/latest/quickstart.html#full-example

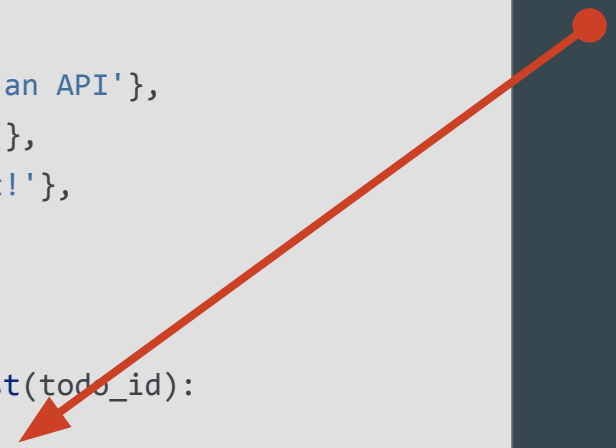
```python
from flask import Flask
from flask_restful import reqparse, abort, Api, Resource

app = Flask(__name__)
api = Api(app)

TODOS = {
    'todo1': {'task': 'build an API'},
    'todo2': {'task': '?????'},
    'todo3': {'task': 'profit!'},
}


def abort_if_todo_doesnt_exist(todo_id):
    if todo_id not in TODOS:
        abort(404, message="Todo {} doesn't
exist".format(todo_id))

parser = reqparse.RequestParser()
parser.add_argument('task')
```

Data structure we will use for saving TODOs

```
from flask import Flask
from flask_restful import reqparse, abort, Api, Resource

app = Flask(__name__)
api = Api(app)

TODOS = {
    'todo1': {'task': 'build an API'},
    'todo2': {'task': '?????'},
    'todo3': {'task': 'profit!'},
}


def abort_if_todo_doesnt_exist(todo_id):
    if todo_id not in TODOS:
        abort(404, message="Todo {} doesn't
exist".format(todo_id))

parser = reqparse.RequestParser()
parser.add_argument('task')
```

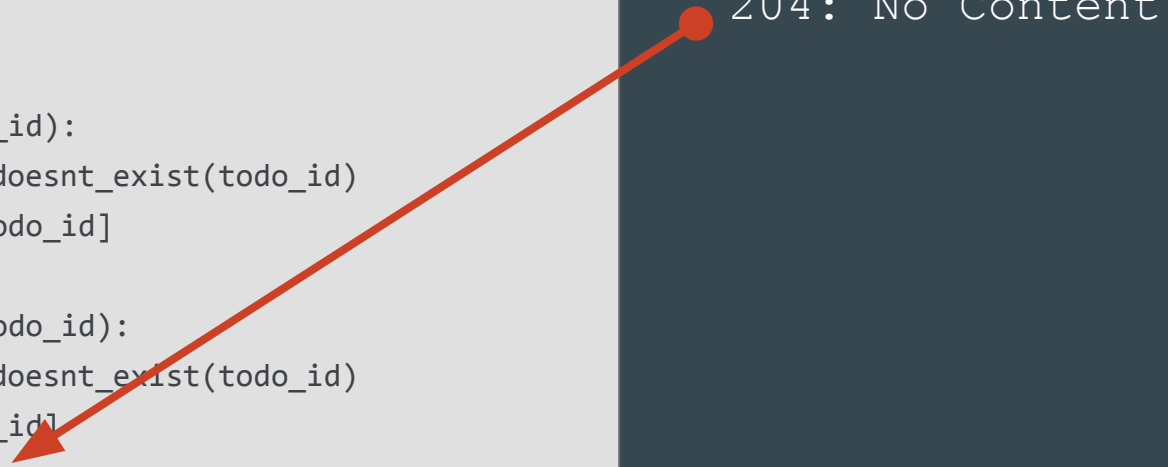
Check whether requested instance exists

```python
from flask import Flask
from flask_restful import reqparse, abort, Api, Resource

app = Flask(__name__)
api = Api(app)

TODOS = {
    'todo1': {'task': 'build an API'},
    'todo2': {'task': '?????'},
    'todo3': {'task': 'profit!'},
}


def abort_if_todo_doesnt_exist(todo_id):
    if todo_id not in TODOS:
        abort(404, message="Todo {} doesn't
exist".format(todo_id))

parser = reqparse.RequestParser()
parser.add_argument('task')
```

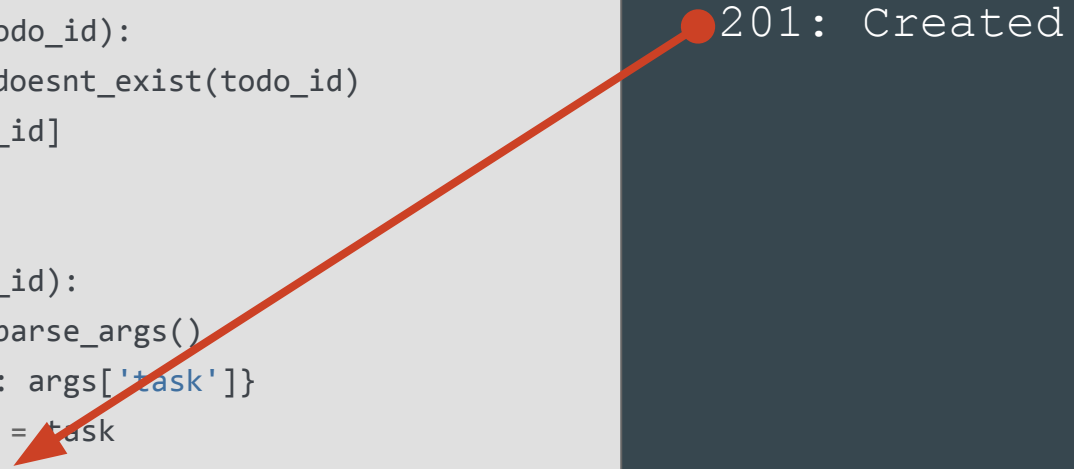


flask_restful.**abort**():
Raise a HTTPException for the given http_status_code.

```python
# Todo
# shows a single todo item and lets you delete a todo
item
class Todo(Resource):
    def get(self, todo_id):
        abort_if_todo_doesnt_exist(todo_id)
        return TODOS[todo_id]

    def delete(self, todo_id):
        abort_if_todo_doesnt_exist(todo_id)
        del TODOS[todo_id]
        return '', 204

    def put(self, todo_id):
        args = parser.parse_args()
        task = {'task': args['task']}
        TODOS[todo_id] = task
        return task, 201
```



```python
# Todo
# shows a single todo item and lets you delete a todo
item
class Todo(Resource):
    def get(self, todo_id):
        abort_if_todo_doesnt_exist(todo_id)
        return TODOS[todo_id]

    def delete(self, todo_id):
        abort_if_todo_doesnt_exist(todo_id)
        del TODOS[todo_id]
        return '', 204

    def put(self, todo_id):
        args = parser.parse_args()
        task = {'task': args['task']}
        TODOS[todo_id] = task
        return task, 201
```



```python
# TodoList
# shows a list of all todos, and lets you POST to add new
tasks
class TodoList(Resource):
    def get(self):
        return TODOS

    def post(self):
        args = parser.parse_args()
        todo_id = int(max(TODOS.keys()).lstrip('todo')) + 1
        todo_id = 'todo%i' % todo_id
        TODOS[todo_id] = {'task': args['task']}
        return TODOS[todo_id], 201
```

```
##
## Actually setup the Api resource routing here
##
api.add_resource(TodoList, '/todos')
api.add_resource(Todo, '/todos/<todo_id>')


if __name__ == '__main__':
    app.run(debug=True)
```

Get the list

```
$ curl http://127.0.0.1:5000/todos

{"todo1": {"task": "build an API"}, "todo3": {"task": "profit!"}, "todo2":
{"task": "?????"}}
```

GET a single task

```
$ curl http://127.0.0.1:5000/todos/todo3

{"task": "profit!"}
```

DELETE a task

```
$ curl http://127.0.0.1:5000/todos/todo2 -X DELETE -v

> DELETE /todos/todo2 HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7
OpenSSL/0.9.8l zlib/1.2.3
> Host: localhost:5000
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 204 NO CONTENT
< Content-Type: application/json
< Content-Length: 0
< Server: Werkzeug/0.8.3 Python/2.7.2
< Date: Mon, 01 Oct 2012 22:10:32 GMT
```

Add a new task

```
$ curl http://127.0.0.1:5000/todos -d "task=something new" -X POST -v

> POST /todos HTTP/1.1
> User-Agent: curl/7.19.7 (universal-apple-darwin10.0) libcurl/7.19.7
OpenSSL/0.9.8l zlib/1.2.3
> Host: localhost:5000
> Accept: */*
> Content-Length: 18
> Content-Type: application/x-www-form-urlencoded
>
* HTTP 1.0, assume close after body
< HTTP/1.0 201 CREATED
< Content-Type: application/json
< Content-Length: 25
< Server: Werkzeug/0.8.3 Python/2.7.2
< Date: Mon, 01 Oct 2012 22:12:58 GMT
<
* Closing connection #0
{"task": "something new"}
```

Update a task

```
$ curl http://127.0.0.1:5000/todos/todo3 -d "task=something different" -X PUT
-v

> PUT /todos/todo3 HTTP/1.1
> Host: localhost:5000
> Accept: */*
> Content-Length: 20
> Content-Type: application/x-www-form-urlencoded
>
* HTTP 1.0, assume close after body
< HTTP/1.0 201 CREATED
< Content-Type: application/json
< Content-Length: 27
< Server: Werkzeug/0.8.3 Python/2.7.3
< Date: Mon, 01 Oct 2012 22:13:00 GMT
<
* Closing connection #0
{"task": "something different"}
```

Questions?