Tornado

14.1 Tornado

This section is a summary of the tornado framework material from lectures:

The Server

- Tornado is a web framework which gives convenient access to a number of features helpful for developing web applications.
- In particular, we care about Tornado's routing, templating and its ability to get data from users via **`GET`** and **`POST`** requests.
- Tornado comes with a built in web server which we can use for development.
- To run the development server, we must set up an application, which includes a list of URL's that the server knows how to handle
- We can run the development server by asking the application to listen on a port for incoming connections and loop:

```
app = Application()
app.listen(<PORT>)
tornado.ioloop.IOLoop.instance().start()
```

Applications

- In order to make a page, we need to decide what URL to serve that page on and write a handler for that page.
- Handlers extend the **`tornado.web.RequestHandler`** class and should contain a **`get()`** or **`post()`** method to handle output.
- Output is written out to the browser by calling the **`self.write()`** method.
- A simple handler might look like this:

```
class HelloWorldHandler(tornado.web.RequestHandler):
  def get(self):
    self.write("Hello, World!")
```

  Note that we used self.write to do output instead of using a print statement.

- We can bind a handler to a URL by adding it to the list of handlers given in the application definition:

```
class Application(tornado.web.Application):
  def __init__(self):
    handlers = [
      (r"/hello/", HelloWorldHandler),
    ]
    tornado.web.Application.__init__(self, handlers)
```

- We can also pass paramters to the handler in the URL using regular expressions in our route. For example, to pass a lowercase word to our **`HelloWorldHandler`** we might do something like:

```
<snip>
  def get(self, name):
    self.write("Hello, {}!".format(name))
<snip>
  (r"/hello/([a-z]+)/", HelloWorldHandler)
<snip>
```

Getting User Data

- Tornado can retrieve data from both **`GET`** and **`POST`** requests.
- If we want to handle data from a post request, we need to write a **`post`** function in the **`RequestHandler`**.
- To retrieve data from a **`GET`** or **`POST`** request, we can use the **`self.get_argument`** function.
- We can also access arguments using the **`self.request.arguments`** dictionary.
- We must check whether a field exists, otherwise the server will throw an error.

```
if "name" in self.request.arguments:
  name = self.get_argument(name)
else:
  name = "World"
self.write("Hello, {}!".format(name))
```

Debugging

- When we are writing code it is helpful to turn on debugging output.
- When debugging mode is on exceptions are printed out to the web browser and automatic script reloading is enabled.
- If we do not enable debugging mode, changes to code will **not** become visible until the server is restarted.
- To enable debugging add the debug parameter to the application:

```
class Application(tornado.web.Application):
    def __init__(self):
        handlers = [
            (r"/", HelloWorldHandler),
        ]
        settings = {
            "debug": True,
        }
        tornado.web.Application.__init__(self, handlers, **settings)
```

- Debugging should never be enabled on production servers - showing the error messages to users is a major security risk since it could leak information about the program internals. It is much better to have errors go to a log file on a real system.

14.2 Walkthroughs

Note: Walkthrough 1 is critical for Stage 3 of the major project as it explains how to set up a Tornado server. Please ask us if you have any problems getting it to work.

14.2.1 Walkthrough 1: Hello Web

In this walkthrough we download the provided tornado skeleton and set up a simple web application to print out hello world.

In contrast to previous weeks exercises, we want to be able to generate a page using a python program on the fly, as opposed to serving a static (unchanging) HTML file. This allows for customization and dynamic updating of the content of the page.

1. First, we need to download the tornado skeleton from the course website at `http://rp-www.cs.usyd.edu.au/~info1903/lab11/tornado.zip`. Unzip this file to your home directory.

2. Confirm that you can run the development server. Run the **`app.py`** file by double clicking on it. Navigate to `http://localhost:8888/` and confirm that you see "Hello, World!" on screen. Close the server.

3. Before we begin development, we want to put the server into **`debug`** mode. Edit the **`app.py`** file, and add the value `"debug": True` to the settings dictionary in the **`Application`** object. Confirm the server starts again.

4. Let's create a new handler that will use a name passed in the URL to print out **`Hello, <name>!`**. Add the following below the **`MainHandler`** class:

```
class HelloNameHandler(tornado.web.RequestHandler):
  def get(self, name):
    self.write("Hello, {}!".format(name))
```

5. Now that we have a new handler, we need to configure the web server to redirect requests to it. Add the following entry to the **`handlers`** list in the **`Application`** object:

```
  (r"/hello/([a-zA-Z]+)/", HelloNameHandler),
```

6. We should now be able to navigate to the url at `http://localhost:8888/hello/Seb/` and see **`Hello, Seb!`** printed out. Confirm that changing the name in the url gives you the appropriate response.

14.2.2 Walkthrough 2: Hello Web, who are you?

In this walkthrough, we will modify the previous example to print out a HTML document using the tornado templating engine. Although we could construct HTML in python code as we go, it is generally neater to separate logic and presentation from each other. Instead, we use python to prepare all the necessary data and write a template which contains our page layout.

1. Firstly we need to create a HTML template, into which we will place the name we got using code. Our template should look something like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello, World!</title>
</head>
<body>
  <h1>Hello, {{ name }}!</h1>
</body>
</html>
```

2. Save this file as **`hello.html`** in the **`templates`** directory of your application. Notice that we left a **template expression** which the templating engine will replace with data from our code.

3. Next we need to modify our handler to output using the template instead of outputting a string. In the **`app.py`** file, modify the **`HelloNameHandler`** to the following:

```
class HelloNameHandler(tornado.web.Application):
  def main(self, name):
    self.render("hello.html", name=name)
```

4. Point your browser to `http://localhost:8888/hello/NAME` and check that it all works as expected.

5. The templating language is quite powerful and you should familiarise yourself with its syntax. The full documentation is available at `http://www.tornadoweb.org/en/stable/template.html`.

14.2.3 Walkthrough 3: HTML Forms

Finally, let's run through a simple example posting data using a HTML form. This is the basis by which we can create complex responsive Tornado applications, it would be quite limiting if there was no way to handle user input!

1. First, we need to create a static HTML form which will query for a users name. This could look something like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello, who are you?</title>
</head>
<body>
<h1>Hello, who are you?</h1>
<form action="/forms/" method="post">
  <div>
    Please enter your name: <input type="text" name="who">
  </div>
  <div>
    <input type="submit">
  </div>
</form>
</body>
</html>
```

2. Save this file in the **`static`** directory as **`hello.html`**. Note that the action property points to the URL that will contain our hander.

3. Now we need to write a handler in app.py which can respond to the data posted by the form. In the **`app.py`** file, create a new handler:

```
class FormHandler(tornado.web.RequestHandler):
  def post(self):
    if "name" in self.request.arguments:
      name = self.get_argument(name)
    else:
      name = "World"
    self.render("hello.html", name=name)
```

4. Finally, add the new handler to the list of handlers for the application in the **`Application`** object:

```
  handers = [
    ....
    ("/forms/", FormHandler),
    ....
  ]
```

5. Finally, point your browser at your HTML page and test your new form `http://localhost:8888/static/hello.html`

Make sure you are familiar with creating forms in HTML. This was covered in the basic HTML tutorial you did in the last lab but if you didn't get to that bit, you should read through the forms section at
`http://www.w3schools.com/html/html_forms.asp`
By the time you finish this section you should be able to create your own basic form and know how to point it at a Tornado handler script.

14.3 Exercises

These exercises combine what we have done previously in Python with tornado.

14.3.1 Exercise 1: Words to digits

Write a tornado handler and form that takes a number as a series of words (e.g. one two three five seven) and converts it a numerical expression (e.g. 12357) which is displayed in a webpage.

14.3.2 Exercise 2: Times Tables

Write a tornado handler to take an integer value n as a parameter in a URL, and output a HTML table containing the 1 to 12 times tables for n. As an example I should be able to navigate to `http://localhost:8888/times/2` to get the 2 times table.

14.3.3 Exercise 3: Crossword solver

Write a tornado and form which returns a list of words matching a given pattern for crossword solvers, where **`?`** is used to indicate unknown letters. In other words, the user should be able to enter `h??se` and get back a list of words matching this pattern.

14.4 Capability checklist

When you've finished this lab, check that you know how to...

1. Write a website that includes a form for user input
2. Write a basic tornado handler in Python
3. Write a template for tornado and use a handler to display it
4. Get user data using **`GET`** and **`POST`** requests
5. Debug tornado scripts

If you don't know how to do any of these things once you have completed the lab, please come and ask us.