Using JavaScript Promises and Python Futures

Al Crowley

TCD
POSITIVELY DISTINCT

Links

Links

Updated slides: https://sjtechmeetup.github.io/slides/async.pdf

Source Code: https://github.com/albertcrowley/promise-demo

Contact Me: albert.crowley@tcg.com

Sync / Async Programming

Synchronous Programming

Runs a sequence of operations one at a time in order.

Asynchronous Programming

Running a series of operations (seemingly) in parallel.

Two models:

- Callbacks
- Event Loops

Python asyncio

Python asyncio

asyncio is a library to write concurrent code using the async/await syntax.

- python.org

Python asyncio - Coroutiens and Tasks

Coroutines declared with async/await syntax is the preferred way of writing asyncio applications.

Tasks are used to run coroutines in event loops.

Python asyncio - Coroutiens and Tasks

```
async def work():
    # calculate pi
    return 22.0/7


co = work()   # co is a coroutine, not the value of pi
```

Python asyncio - Coroutiens and Tasks

```
async def calc_pi(n):
    pi = 0.0
    for i in range(n):
        pi = pi + (2 * i)/(2 * i - 1) * (2 * i)/(2 * i + 1)
    return pi


co = calc_pi(100)   # co is a coroutine, not the value of pi
```

Python asyncio - Coroutiens and Tasks

```
async def calc_pi(n):
    pi = 0.0
    for i in range(n):
        pi = pi + (2 * i)/(2 * i - 1) * (2 * i)/(2 * i + 1)
        await asyncio.sleep(0.00001)
    return pi


co = calc_pi(100)   # co is a coroutine, not the value of pi
```

Python asyncio - Coroutiens and Tasks

```
loop = asyncio.get_event_loop()
loop.run_until_complete(myCoroutine())
loop.close()
```

Python asyncio - Coroutiens and Tasks

```
task = asyncio.create_task( co() )  # Python 3.7
task = asyncio.ensure_future( co() ) # Python 3.6

# either of these will start the coroutien in the event loop
```

JavaScript Promises

Promise (JavaScript)

The Promise object represents the eventual completion (or failure) of an asynchronous operation, and its resulting value.

- MDN Web Documenation

Promise

A promise can be 'resolved' or 'rejected'

Internally a promise will keep track of its state.

The operation inside a promise will resolve or reject exactly once

Once a promise completes, you can still query its value or add "then"

Common Mistakes

Common Mistakes in asyncio

Forget to await:

```
async def work():
    #async work here

work()
```

Common Mistakes in asyncio

Close the loop too quickly:

```
for task in asyncio.Task.all_tasks():
    task.cancel()
loop = asyncio.get_event_loop()
loop.stop()
```

Common Mistakes in asyncio

```
async def exit():
    loop = asyncio.get_event_loop()
    loop.stop()

for task in asyncio.Task.all_tasks():
    task.cancel()
asyncio.ensure_future(exit())  # give asyncio a chance to run
```

Common Promise Mistakes

Promise Hell:

```
work().then(function(res1){
   work2(res1).then (function(res2) {
      work3(res2).then (function…..
```

Common Promise Mistakes

```
work()
.then(function(res1) {
  return res2
})
.then(function(res2) {
  return res3;
})
.then…..
```

Common Mistakes - Create extra Promises

```
p = new Promise( (res, rej) => {
   fetch(url)
      .then( (res) => {
        let data = work(res)
        resolve(data)
      })
      .catch( (err) => reject(err) )
```

Common Mistakes - Create extra Promises

```
p =  fetch(url)
     .then( (data) => work(data) )
```

Common Mistakes - Forget to return

```
fetch(url)
   .then( (data) => work(data) )
   .then( (data) => {
     more_work(data)    // Oops, no return here!
   })
   .then( (data) => other_work(data))
```