

Asynchronous Programming in Play 2.0

for Non Functional Developers

James Roper
@jroper

Why Asynchronous?

- OS schedulers are good at scheduling blocking threads
- Asynchronous programming can be hard to debug and understand
- Callbacks introduce lots of boiler plate in Java (not so in Scala)

Why Asynchronous?

- Long polling comet applications
- More fine grained control over resources (and resource contention)
- Better performance characteristics under high load
- Play makes it easy!

Promise<T>

- Similar abstraction to Java's Future<T>
- I promise to give you something of type T at some point in future

Promise<T>

- Waiting for the promise value:
 - `T value = promise.get();`
 - `T value = promise.get(timeout);`

Promise<T>

- Asynchronous handling of the result value:

```
promise.onRedeem(new Callback<T>() {  
    public void invoke(T value) {  
        // do something  
    }  
});
```

Promise<T>

- `get()` and `onRedeem()` not usually useful
 - If you use `get()`, you're no longer asynchronous
 - If you use `onRedeem()`, you can't easily return the result to anything
- Use `async()`, `map()`, `flatMap()` and other methods

map()

- Map converts a Promise of one type to a Promise of another type
- eg. `Promise<Flour> -> Promise<Dough>`

flatMap()

- Combines two functional concepts
 - Map
 - Flatten

flatMap()

- The map function maps from type A to a promise of type B
- eg. Dough -> Promise<Bread>
- If using ordinary map, this would result in Promise<Promise<Bread>>

flatMap()

- Flatten converts `Container<Container<A>>` to `Container<A>`
- eg `List<List<String>> -> List<String>`
 - `{{"a", "b"}, {"c", "d"}} -> {"a", "b", "c", "d"}`
- `Promise<Promise<Bread>> -> Promise<Bread>`

recover()

- If a `Promise<A>` is unable to fulfill its promise, `recover` can be used to map an exception to an `A`
- For example, HTTP status code not returned, exception thrown in a `map()` function

waitAll()

- Used to wait for multiple promises in parallel
- Returns a Promise of a List of the results of each promise

Pure promises

- You may need to return a promise for something that you have now
 - eg, a cache hit
- `return Promise.pure(value);`

Using promises in play

- Actions
- WS API
- Akka API

Actions

- An action must return a Result
- AsyncResult is a result that wraps a Promise<Result>, and handles it asynchronously
- async() convenient static method for wrapping

WS API

- WS API returns `Promise<WS.Response>`:
 - `WS.url("http://google.com").get()`
- Use in combination with `map()` to provide an interface that returns promises for domain objects
- Or, map directly to a result from an action

Akka

- Akka can be used to offload expensive tasks to dedicated threads, or even other machines
- At a minimum, ensures your request processing isn't adversely impacted by slow requests

Take aways

- Play 2.0 promise API allows very simple asynchronous programming in Java
- Even simpler in Scala – try it yourself!

Thankyou Questions?

James Roper
@jroper