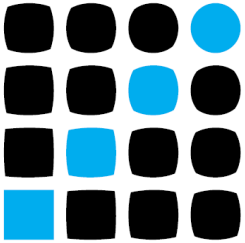


adaptTo()

APACHE SLING & FRIENDS TECH MEETUP

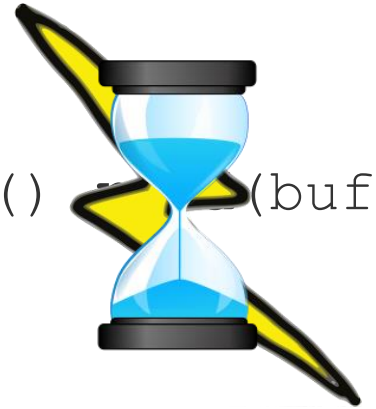
BERLIN, 22-24 SEPTEMBER 2014

OSGi Asynchronous Services: more than RPC

Michael Dürig, Adobe Research

Some typical code

```
socket.getInputStream() .(buffer);
```



Timings on typical hardware...

Execute CPU instruction	1ns
Fetch from L2 cache	7 ns
Fetch from memory	100 ns
Read from disk	8 ms
Network round-trip Europe-US	150 ms

source: http://norvig.com/21-days.html#answers

...in human terms

Execute CPU instruction	1 s
Fetch from L2 cache	7 s
Fetch from memory	1.6 min
Read from disk	13.2 weeks
Network round-trip Europe-US	4.8 years

source: http://norvig.com/21-days.html#answers

Outline

- OSGi RFC 206: Asynchronous Services
- Promise
- Example

Goals

- Improve resource utilisation
  - Parallelism
  - Non blocking IO
- Automatic thread management
  - Runtime vs. hard coded

OSGi RFC 206: Asynchronous Services

- Asynchronous method calls
  - Return `Promise<T>` instead of `T`
  - Direct implementations
  - Mediation through `Async`

Mediating a service

```
Async async = ...
ServiceReference<Weather> ref = ...

Weather weather = async.mediate(ref, Weather.class);
Promise<Boolean> sunnyPromise = async.call(weather.isSunny());
```

Promise<T>?

Promise

- Encapsulation of a value that
  - maybe available at some later time
  - can be read multiple times
  - immutable once resolved

Promise callback

```
sunnyPromise.then(...
```

Promise callback: success

```
weather.isSunny().then(
    success -> {
        if (success.getValue()) {
            println("Sun fun and nothing to do");
        } else {
            println("Singing in the rain");
        }},
...
```

Promise callback: failure

```
weather.isSunny().then(
    success -> {
        if (success.getValue()) {
            println("Sun fun and nothing to do");
        } else {
            println("Singing in the rain");
        }},
    failure -> {
        failure.getFailure().printStackTrace();});
```

Effects

- Promises capture the effects of
  - latency: when the call back happens
  - error: which call back happens

Promise<T>!

Vendor Service

```
interface Vendor {
    Promise<Offer> getOffer(String item);
}

class Offer {
    public Offer(Vendor vendor,
                 String item,
                 double price,
                 String currency) {
    ...
```

Exchange Service

```
Promise<Offer> convertToEuro1(Offer offer);

Promise<Offer> convertToEuro2(Offer offer);
```

Goals

- Get offer from vendor
- Convert to € with recovery
- Fail gracefully
- Non blocking!

getOffer...

```
Promise<Offer> getOffer(Vendor vendor, String item) {
    return vendor
        .getOffer(item);
}
```

flatMap

```
Promise<R> flatMap(T -> Promise<R>)
```

flatMap

```
Promise<R> flatMap(Offer -> Promise<R>)
```

flatMap

```
Promise<Offer> flatMap(Offer -> Promise<Offer>)
```

flatMap

```
Promise<Offer> flatMap(Offer -> Promise<Offer>)

Promise<Offer> convertToEuro1(Offer);
```

getOffer: convert to €

```
Promise<Offer> getOffer(Vendor vendor, String item) {
    return vendor
        .getOffer(item)
        .flatMap(offer -> convertToEuro1(offer));
}
```

recoverWith

```
Promise<R> recoverWith(Promise<?> -> Promise<R>)
```

recoverWith

```
Promise<Offer> recoverWith(Promise<?> -> Promise<Offer>)
```

recoverWith

```
Promise<Offer> recoverWith(Promise<?> -> Promise<Offer>)

Promise<Offer> convertToEuro2(Offer);
```

getOffer: fallback

```
Promise<Offer> getOffer(Vendor vendor, String item) {
    return vendor
        .getOffer(item)
        .flatMap(offer -> convertToEuro1(offer)
        .recoverWith(failed -> convertToEuro2(offer)));
}
```

recover

```
Promise<R> recover(Promise<?> -> R)
```

recover

```
Promise<Offer> recover(Promise<?> -> Offer)
```

getOffer: fail gracefully

```
Promise<Offer> getOffer(Vendor vendor, String item) {
    return vendor
        .getOffer(item)
        .flatMap(offer -> convertToEuro1(offer)
        .recoverWith(failed -> convertToEuro2(offer))
        .recover(failed -> Offer.NONE));
}
```

Non blocking

Act on `Promise<T>` instead of `T`

Demo

Summary

- Promises capture latency and error
  - Non blocking
  - Focus on main path
- Decouple parallelisation
  - Better resource utilisation
  - Application scalability

Resources

http://goo.gl/pB9fza

Appendix

Deferred

- Encapsulation of a value that
  - should be made available at some later time
  - can be written once
  - immutable once resolved

What about j.u.concurrent.Future?

- Blocking semantics
  - No callbacks
- Not composable
  - No map/flatMap/filter/...

Resources

- Support material: https://github.com/mduerig/async-support/wiki
  - Session slides
  - Runnable demo
  - Links to further reading
- RFC 206: https://github.com/osgi/design/tree/master/rfcs/rfc0206
  - Public draft of OSGi RFC 206: Asynchronous Services
- OSGi Alliance: http://www.osgi.org/

Resources

- RxJava: https://github.com/ReactiveX/RxJava
  - Observables: taking Promises one step further
- Akka: http://akka.io/
  - Toolkit for concurrent and distributed applications on the JVM