PGS
SOFTWARE

PGS SOFTWARE

Spring Framework 5.0

Introduction to Reactive Programming Model

Jakub Madej · Dev.Java.Wro #2 · 12.09.2018

Agenda

Generations of reactive programming concept

Reactor Project

Code samples

Reactive drivers

Reactive JDBC?

Use cases

Lessons learned

Generations

reactive libraries evolved over time
let’s name some important projects

Reactive streams

* 2015, Netflix, Pivotal and Lightbend
* low level contracts, back-pressure
* incorporated in JDK 9 (java.util.concurrent.Flow)

ReactiveX/RxJava

* tools used by Netflix released as open source
* previously Netflix/RxJava

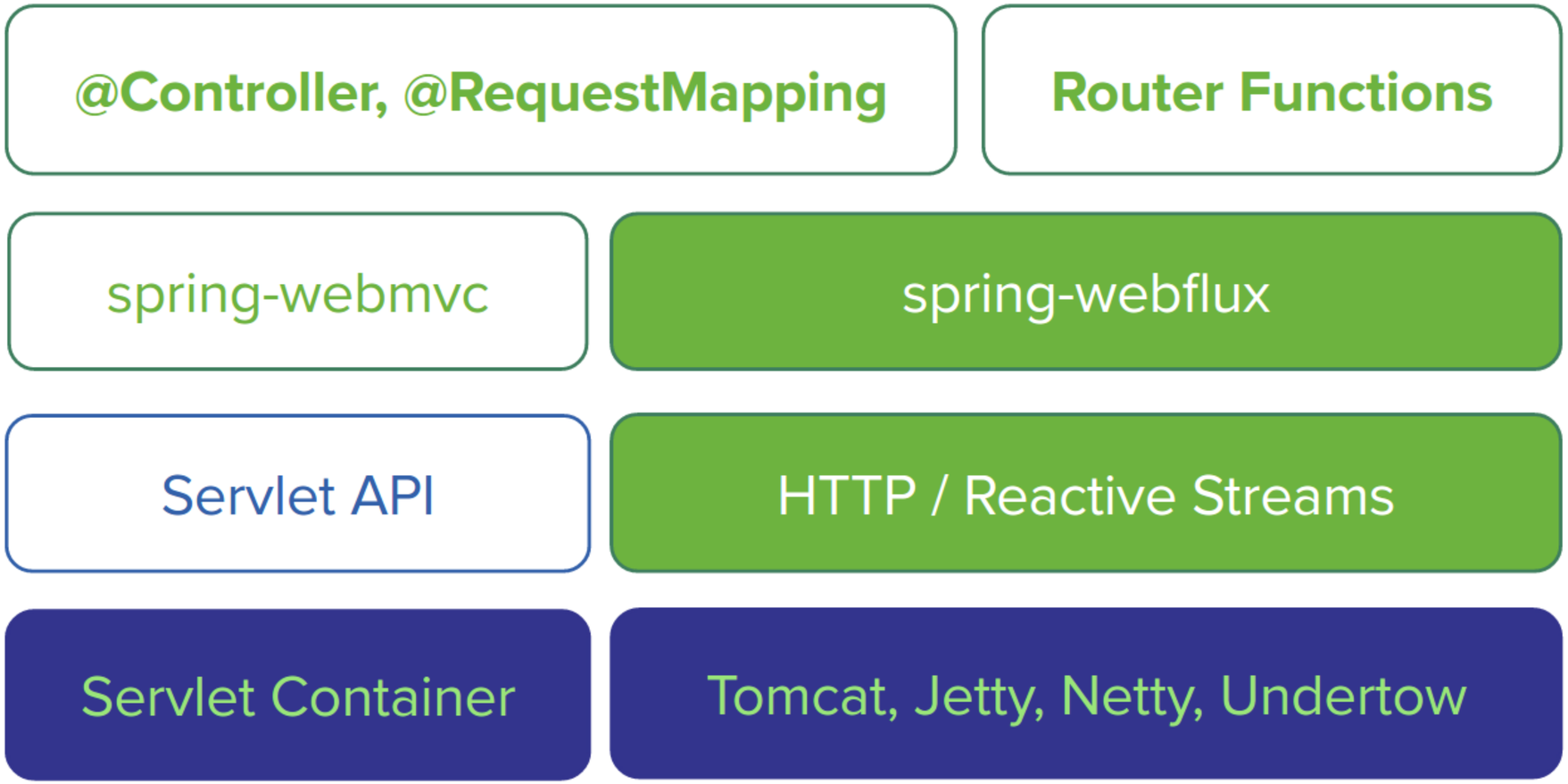
Reactor (Pivotal)

* Java framework, wrapper around low-level network runtimes

Spring Framework 5.0

* reactive features, built on Reactor (but we may use also RxJava)

Spring Framework 5.0



Reactive programming

asynchronous

non-blocking

event-driven

Back pressure

Consumer controls the flow

You can't overwhelm reactive consumer

You can't DDoS reactive consumer

Reactor project

2 specialized types:

Flux is a publisher which produces 1 to N values (unbounded)
Mono is a publisher which produces 0 or 1 value

Code samples

```
public interface ReactivePersonRepository
  extends ReactiveCrudRepository<Person, String> {

  Flux<Person> findByLastname(Mono<String> lastname);

  @Query("{ 'firstname': ?0, 'lastname': ?1}")
  Mono<Person> findByFirstnameAndLastname(String firstname,
String lastname);
}
```

```
@RestController
class PersonController {

  private final PersonRepository people;

  public PersonController(PersonRepository people) {
    this.people = people;
  }

  @GetMapping("/people")
  Flux<String> namesByLastname(@RequestParam Mono<String>
lastname) {

    Flux<Person> result =
repository.findByLastname(lastname);
    return result.map(it -> it.getFullName());
  }
}
```

```
RouterFunction<?> route = route(GET("/person/{id}"),
  request -> {
    Mono<Person> person =
Mono.justOrEmpty(request.pathVariable("id"))
      .map(Integer::valueOf)
      .then(repository::getPerson);
    return Response.ok().body(fromPublisher(person,
Person.class));
  })
  .and(route(GET("/person"),
    request -> {
      Flux<Person> people = repository.allPeople();
      return Response.ok().body(fromPublisher(people,
Person.class));
    }));
```

```
@EnableReactiveMongoRepositories
public class AppConfig extends
AbstractReactiveMongoConfiguration {

  @Bean
  public MongoClient mongoClient() {
    return MongoClients.create();
  }

  @Override
  protected String getDatabaseName() {
    return "reactive";
  }
}
```

Reactive drivers

* NoSQL:
Reactive MongoDB
Reactive Cassandra
Reactive Couchbase
Reactive Redis

Reactive JDBC

Asynchronous Database Access API (ADBA)
April 2018
https://blogs.oracle.com/java/jdbc-next:-a-new-asynchronous-api-for-connecting-to-a-database

Use cases

* external service calls (in REST-ful services)

Lessons learned

API contract vs reactive approach
reactiveness can be introduced step by step

There’s more...

Reactive Spring Security

@EnableWebFluxSecurity...

WebClient

* non-blocking, working over HTTP/1.1

Further reading

https://spring.io

Dávid Karnok
https://akarnokd.blogspot.com/2016/03/operator-fusion-part-1.html

Reactive Spring - Josh Long, Mark Heckler
(SpringOne Platform 2017)

PGS SOFTWARE

Thank you!

Go visit pgs-soft.com

Jakub Madej

PGS
SOFTWARE