Spring Framework 5 Themes & Trends

Preparing for 2017+

Juergen Hoeller
Spring Framework Lead
Pivotal

Spring Framework 4.3

- **Last 4.x feature release!**
- **Generally available since June 2016**

- **Extended support life until 2019**
  - on JDK 6, 7, 8
  - on Tomcat 6, 7, 8.0, 8.5
  - on WebSphere 7, 8.0, 8.5 and 9

- **Programming model refinements brought forward to JDK 6+**
  - DI & MVC refinements
  - composed annotations

The State of the Art: Component Classes

```
@Service
@Lazy
public class MyBookAdminService implements BookAdminService {

    // @Autowired
    public MyBookAdminService(AccountRepository repo) {
        ...
    }

    @Transactional
    public BookUpdate updateBook(Addendum addendum) {
        ...
    }
}
```

Configuration Classes with Autowired Constructors

```
@Configuration
public class MyBookAdminConfig {

    private final DataSource bookAdminDataSource;

    // @Autowired
    public MyBookAdminService(DataSource bookAdminDataSource) {
        this.bookAdminDataSource = bookAdminDataSource;
    }

    @Bean
    public BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(this.bookAdminDataSource);
        return service;
    }
}
```

Annotated MVC Controllers

```
@Controller
@CrossOrigin
public class MyRestController {

    @RequestMapping(path="/books/{id}", method=GET)
    public Book findBook(@PathVariable long id) {
        return this.bookAdminService.findBook(id);
    }

    @RequestMapping(path="/books/new", method=POST)
    public void newBook(@Valid Book book) {
        this.bookAdminService.storeBook(book);
    }
}
```

Precomposed Annotations for MVC Controllers

```
@RestController
@CrossOrigin
public class MyRestController {

    @GetMapping("/books/{id}")
    public Book findBook(@PathVariable long id) {
        return this.bookAdminService.findBook(id);
    }

    @PostMapping("/books/new")
    public void newBook(@Valid Book book) {
        this.bookAdminService.storeBook(book);
    }
}
```

Spring Framework 5

- **A new application framework generation for 2017+**
- **5.0: Q2 2017**
- **5.1: Q4 2017**

- **Major baseline upgrade**
  - JDK 8+, Servlet 3.1+, JPA 2.1+, JMS 2.0+
  - support for JUnit 5 (next to JUnit 4.12)

- **Key infrastructure themes**
  - JDK 9, Jigsaw, HTTP/2, Servlet 4, Kotlin
  - functional style and reactive architectures

spring by Pivotal

JDK 9

HTTP/2

Functional

Reactive

JDK 9: Not Just Jigsaw

- **Many general JVM improvements**
  - Compact Strings, G1 by default, TLS protocol stack
- **Jigsaw – module path as structured alternative to class path**
  - symbolic module names and requires/exports metadata for jar files
- **JDK 9 GA scheduled for July 2017**
  - comprehensive support coming in Spring Framework 5.1
- **For a smooth immediate upgrade, stay in class path mode!**
  - Spring 4.3 & 5.0 are generally compatible with JDK 9 already

spring by Pivotal

Using Jigsaw with Spring

- **Framework jars as Jigsaw-compliant modules on the module path**
  - internally declaring module-info for each jar (in consideration for 5.1+)
  - or as “automatic modules” (for the time being in 4.3 & 5.0)

- **Separate module namespace, following Maven Central jar naming**
  - spring-context, spring-jdbc, spring-webmvc

- **An application's module-info.java may refer to framework modules**

```
module my.app.db {
    requires spring.jdbc;
}
```

JDK 9

HTTP/2

Functional

Reactive

The Importance of HTTP/2 (RFC 7540)

- **Enormous benefits over HTTP 1.1 (1997 → 2017)**
  - binary protocol
  - TLS (SSL) everywhere
  - connection multiplexing
  - headers compression
  - request prioritization
  - push of correlated resources

- **Browsers already implement HTTP/2 over TLS**
  - major websites work with HTTP/2 already: Google, Twitter, etc
  - *We need to embrace it in Java land as well!*

Spring 5 and HTTP/2

- **Servlet 4.0 specification coming in Q4 2017**
  - enforces support for HTTP/2 in Servlet containers
  - new PushBuilder API for pushing additional resources to a client
- **Native HTTP/2 support in current Servlet 3.1 containers**
  - Tomcat 8.5 / 9.0, Jetty 9.3 / 9.4, Undertow 1.3 / 1.4
  - special setup for ALPN on JDK 8 (comes out of the box in JDK 9)
- **Let's enable HTTP/2 as soon as possible...**
  - Spring's focus: native HTTP/2 on top of Tomcat / Jetty / Undertow
  - Spring Framework 5.1 will ship dedicated Servlet 4.0 support

spring by Pivotal

JDK 9

HTTP/2

Functional

Reactive

Functional Style vs Annotation Style

- **Spring 4.3 wraps up a well-established annotation story**
  - a comprehensive annotation-based component model
  - with loosely coupled, self-descriptive endpoint classes
- **Spring 5 provides functional-style APIs as an alternative**
  - programmatic bean registration and endpoint composition
  - no need for annotations or scanning, even avoiding reflection
- **First-class support for the Kotlin language out of the box**
  - Java 8 is a fine foundation for functional-style Java APIs
  - Spring's Kotlin extensions make code even more concise

spring by Pivotal

Programmatic Bean Registration with Java 8

```
// Starting point may also be AnnotationConfigApplicationContext
GenericApplicationContext ctx = new GenericApplicationContext();
ctx.registerBean(Foo.class);
ctx.registerBean(Bar.class,
        () -> new Bar(ctx.getBean(Foo.class)));

// Or alternatively with some bean definition customizing
GenericApplicationContext ctx = new GenericApplicationContext();
ctx.registerBean(Foo.class, Foo::new);
ctx.registerBean(Bar.class,
        () -> new Bar(ctx.getBean(Foo.class),
        bd -> bd.setLazyInit(true));
```

Programmatic Bean Registration with Kotlin

```
// Java-style usage of Spring's Kotlin extensions
val ctx = GenericApplicationContext()
ctx.registerBean(Foo::class)
ctx.registerBean { Bar(it.getBean(Foo::class)) }


// Gradle-style usage of Spring's Kotlin extensions
val ctx = GenericApplicationContext {
    registerBean<Foo>()
    registerBean { Bar(it.getBean<Foo>()) }
}
```

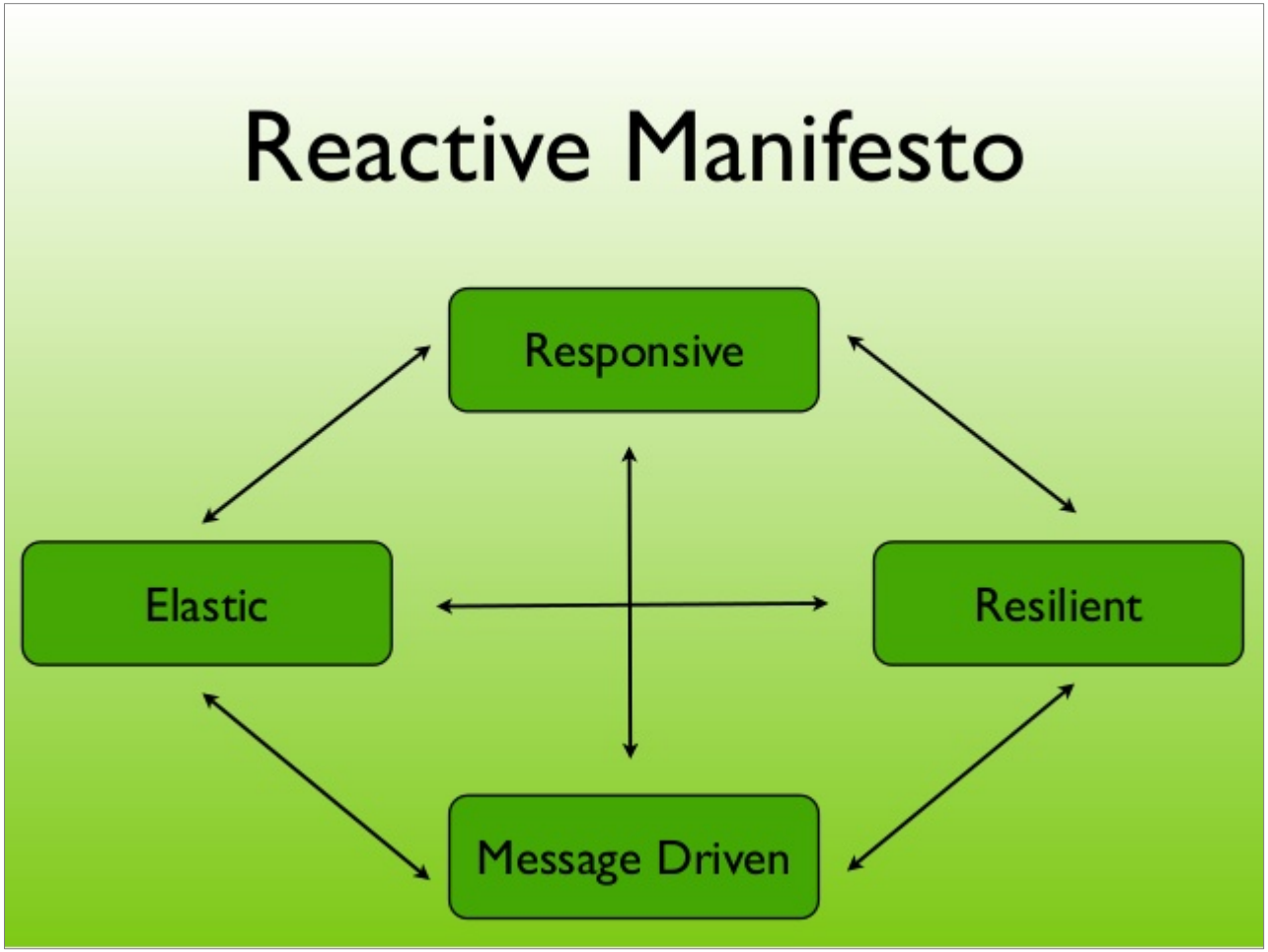
JDK 9

HTTP/2

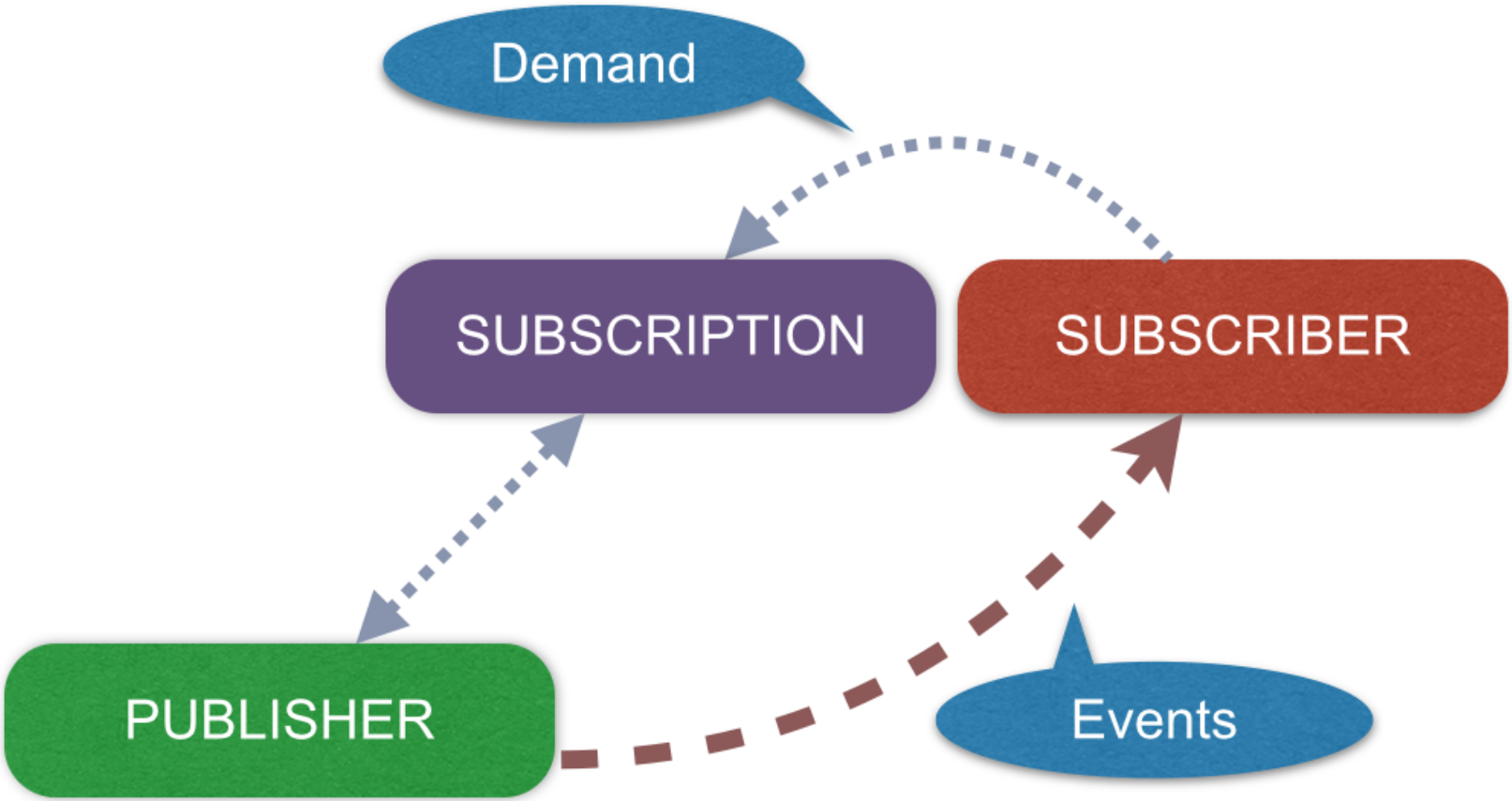
Functional

Reactive

The Importance of Reactive Architectures

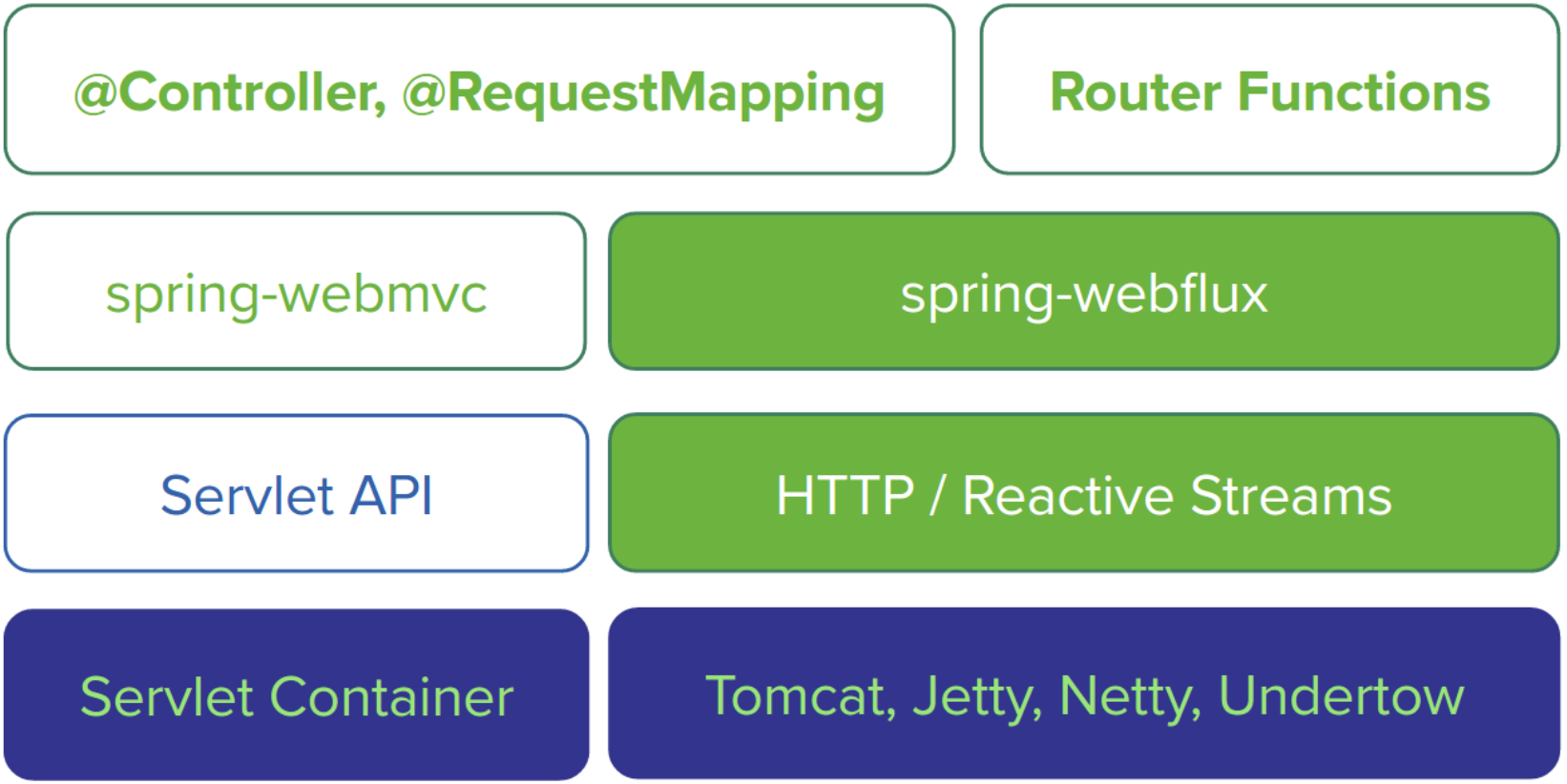


Reactor 3: Reactive Streams with Backpressure



spring by Pivotal

Spring MVC on Servlets ↔ Spring WebFlux on Reactor



Reactive Web Controller with Repository Interop

```
@Controller
public class MyReactiveWebController {

    private final UserRepository repository;

    public MyReactiveWebController(UserRepository repository) {
        this.repository = repository;
    }

    @GetMapping("/users/{id}")
    public Mono<User> getUser(@PathVariable Long id) {
        return this.repository.findById(id);
    }

    @GetMapping("/users")
    public Flux<User> getUsers() {
        return this.repository.findAll();
    }
}
```

Functional Web Endpoints with Method References

```
RouterFunction<?> router =
    route(GET("/users/{id}"), handlerDelegate::getUser)
    .andRoute(GET("/users"), handlerDelegate::getUsers);


public class MyReactiveHandlerDelegate {

    public ServerResponse<String> getUser(ServerRequest request) {
        Mono<User> user = Mono.justOrEmpty(request.pathVariable("id"))
            .map(Long::valueOf).then(repository::findById);
        return ServerResponse.ok().body(user, User.class);
    }

    public ServerResponse<String> getUsers(ServerRequest request) {
        Flux<User> users = repository.findAll();
        return ServerResponse.ok().body(users, User.class);
    }
}
```

Functional Web Endpoints in Lambda Style

```
UserRepository repository = ...;

RouterFunction<?> router =
    route(GET("/users/{id}"),
        request -> {
            Mono<User> user = Mono.justOrEmpty(request.pathVariable("id"))
                .map(Long::valueOf).then(repository::findById);
            return ServerResponse.ok().body(user, User.class);
        })
    .andRoute(GET("/users"),
        request -> {
            Flux<User> users = repository.findAll();
            return ServerResponse.ok().body(users, User.class);
        });
```

Spring Framework 5.0

RC1 in April 2017

JDK 8+ baseline
functional style: Java 8, Kotlin
reactive web endpoint model

spring by Pivotal

Consider reactive datastores: Spring Data “Kay”

RC1 in April 2017

built on Spring Framework 5.0
reactive repository model

Stay tuned for...
Spring Boot 2.0!

M1 in April 2017

built on Spring Framework 5.0
reactive web starters

Spring Framework 5.1

Q4 2017

Servlet 4.0, Bean Validation 2.0
comprehensive JDK 9 support
foundation for Boot 2.0 GA

spring by Pivotal