

# 数据访问进阶


# Project Reactor 介绍

“在计算机中，响应式编程或反应式编程（英语：Reactive Programming）是一种面向数据流和变化传播的编程范式。这意味着可以在编程语言中很方便地表达静态或动态的数据流，而相关的计算模型会自动将变化的值通过数据流进行传播。”

概念类似于Excel的单元格函数


—— 维基百科

# Project Reactor




Create efficient Reactive systems

Reactor is a fourth-generation Reactive library for building non-blocking applications on the JVM based on the Reactive Streams Specification




### REACTIVE CORE

Reactor is a **fully non-blocking** foundation with efficient demand management. It directly interacts with Java *functional API*, *Completable Future*, *Stream* and *Duration*.



### TYPED [0|1|N] SEQUENCES

Reactor offers 2 **reactive composable API** Flux [N] and Mono [0|1] extensively implementing Reactive Extensions.



### NON BLOCKING IO

Suited for **Microservices Architecture**, Reactor offers **backpressure-ready network engines** for HTTP (including Websockets), TCP and UDP.

Reactive libraries, such as Reactor, aim to address these drawbacks of “classic” asynchronous approaches on the JVM while also focusing on a few additional aspects:

- Composability and readability
- Data as a flow manipulated with a rich vocabulary of operators
- Nothing happens until you subscribe
- Backpressure or the ability for the consumer to signal the producer that the rate of emission is too high
- High level but high value abstraction that is concurrency-agnostic

## Example of Callback Hell

```
userService.getFavorites(userId, new Callback<List<String>>() { 1
    public void onSuccess(List<String> list) { 2
        if (list.isEmpty()) { 3
            suggestionService.getSuggestions(new Callback<List<Favorite>>() { 4
                public void onSuccess(List<Favorite> list) { 4
                    UiUtils.submitOnUiThread(() -> { 5
                        list.stream()
                            .limit(5)
                            .forEach(uiList::show); 6
                    });
                }
            });
        } else { 7
            list.stream() 8
                .limit(5)
                .forEach(favId -> favoriteService.getDetails(favId, 9
                    new Callback<Favorite>() {
                        public void onSuccess(Favorite details) {
                            UiUtils.submitOnUiThread(() -> uiList.show(details));
                        }
                        public void onError(Throwable error) {
                            UiUtils.errorPopup(error);
                        }
                    }
                ));
        }
    }
    public void onError(Throwable error) {
        UiUtils.errorPopup(error);
    }
});
```

传统Callback方式程序可读性差

## reactor优化后程序

### Example of Reactor code equivalent to callback code

```
userService.getFavorites(userId) 1
    .flatMap(favoriteService::getDetails) 2
    .switchIfEmpty(suggestionService.getSuggestions()) 3
    .take(5) 4
    .publishOn(UiUtils.uiThreadScheduler()) 5
    .subscribe(uiList::show, UiUtils::errorPopup); 6
```

### Example of Reactor code with timeout and fallback

```
userService.getFavorites(userId)
    .timeout(Duration.ofMillis(800)) 1
    .onErrorResume(cacheService.cachedFavoritesFor(userId)) 2
    .flatMap(favoriteService::getDetails) 3
    .switchIfEmpty(suggestionService.getSuggestions())
    .take(5)
    .publishOn(UiUtils.uiThreadScheduler())
    .subscribe(uiList::show, UiUtils::errorPopup);
```



# 一些核心的概念

## Operators - Publisher / Subscriber

- Nothing Happens Until You subscribe()
- Flux [ 0..N ] - onNext()、onComplete()、onError()  
N个元素的序列
- Mono [ 0..1 ] - onNext()、onComplete()、onError()  
0或1个元素的序列

## Backpressure

- Subscription
- onRequest()、onCancel()、onDispose()  
每次订阅多少个元素                      onDispose 为完成，错误或取消信号而调用。

# 一些核心的概念

## 线程调度 Schedulers

在当前线程运行后面动作

新启线程

- `immediate()` / `single()` / `newSingle()`

使用可复用线程

- 线程池上操作
- `elastic()` / `parallel()` / `newParallel()`

空闲60s会被回收 跟cpu相同数目的固定线程池

## 错误处理

返回默认值

使用特定lambda表达式处理

- `onError` / `onErrorReturn` / `onErrorResume`

相当于try catch

- `doOnError` / `doFinally`



**“Talk is cheap, show me the code.”**

*Chapter 5 / simpler-reactor-demo*

# 通过 Reactive 的方式访问数据

## Redis

# Spring Data Redis

**Lettuce 能够支持 Reactive 方式**

**Spring Data Redis 中主要的支持**

- ReactiveRedisConnection
- ReactiveRedisConnectionFactory
- ReactiveRedisTemplate
  - opsForXxx() `ReactiveHashOperations<String, String, String> hashOps = redisTemplate.opsForHash();`

**“Talk is cheap, show me the code.”**

*Chapter 5 / reactive-redis-demo*

# 通过 Reactive 的方式访问数据

MongoDB

# Spring Data MongoDB

**MongoDB 官方提供了支持 Reactive 的驱动**

- mongodb-driver-reactivestreams

**Spring Data MongoDB 中主要的支持**

- ReactiveMongoClientFactoryBean
- ReactiveMongoDatabaseFactory
- ReactiveMongoTemplate



**“Talk is cheap, show me the code.”**

*Chapter 5 / reactive-mongo-demo*

# 通过 Reactive 的方式访问数据

## RDBMS

# Spring Data R2DBC

**R2DBC** (<https://spring.io/projects/spring-data-r2dbc>)

- Reactive Relational Database Connectivity

## 支持的数据库

- Postgres (io.r2dbc:r2dbc-postgresql)
- H2 (io.r2dbc:r2dbc-h2)
- Microsoft SQL Server (io.r2dbc:r2dbc-mssql)

:

Get started by picking a database driver and create a R2dbcEntityTemplate instance

- H2 (io.r2dbc:r2dbc-h2)
- MariaDB (org.mariadb:r2dbc-mariadb)
- Microsoft SQL Server (io.r2dbc:r2dbc-mssql)
- MySQL (dev.miku:r2dbc-mysql)
- jasync-sql MySQL (com.github.jasync-sql:jasync-r2dbc-mysql)
- Postgres (io.r2dbc:r2dbc-postgresql)
- Oracle (com.oracle.database.r2dbc:oracle-r2dbc)

## PostgreSQL Example

```
PostgresqlConnectionFactory connectionFactory = new
PostgresqlConnectionFactory(PostgresqlConnectionFactory.builder()
    .host(...)
    .database(...)
    .username(...)
    .password(...).build());

R2dbcEntityTemplate template = new R2dbcEntityTemplate(connectionFactory);

Mono<Integer> update = template.update(Person.class)
    .inTable("person_table")
    .matching(query(where("firstname").is("John")))
    .apply(update("age", 42));

Flux<Person> all = template.select(Person.class)
    .matching(query(where("firstname").is("John")
        .and("lastname").in("Doe", "White"))
        .sort(by(desc("id")))))
    .all();
```

[COPY](#)

## Repository Example

```
interface PersonRepository extends ReactiveCrudRepository<Person, String> {

    Flux<Person> findByFirstname(String firstname);

    @Modifying
    @Query("UPDATE person SET firstname = :firstname where lastname = :lastname")
    Mono<Integer> setFixedFirstnameFor(String firstname, String lastname);

    @Query("SELECT * FROM person WHERE lastname = :#{[0]}")
    Flux<Person> findByQueryWithExpression(String lastname);

}
```

[COPY](#)

# Spring Data R2DBC

## 一些主要的类

- ConnectionFactory
- DatabaseClient
  - `execute().sql(SQL)`
  - `inTransaction(db -> {})`
- R2dbcExceptionTranslator
  - `SqlErrorCodeR2dbcExceptionTranslator` 异常翻译



**“Talk is cheap, show me the code.”**

*Chapter 5 / simple-r2dbc-demo*

# R2DBC Repository 支持

## 一些主要的类

- `@EnableR2dbcRepositories`
- `ReactiveCrudRepository<T, ID>`
  - `@Table` / `@Id`
  - 其中的方法返回都是 Mono 或者 Flux
  - 自定义查询需要自己写 `@Query`

**“Talk is cheap, show me the code.”**

*Chapter 5 / r2dbc-repository-demo*

# 通过 AOP 打印数据访问层摘要

# Spring AOP 的一些核心概念

Spring 3 的时候使用CGLIB只能做一次增强，4以后就没有问题了

概念	含义
Aspect	切面
Join Point	连接点，Spring AOP里总是代表一次方法执行
Advice	通知，在连接点执行的动作
Pointcut	切入点，说明如何匹配连接点
Introduction	引入，为现有类型声明额外的方法和属性
Target object	目标对象 有接口的话使用JDK,但是没有接口，只有类使用CGLIB
AOP proxy	AOP 代理对象，可以是 JDK 动态代理，也可以是 CGLIB 代理
Weaving	织入，连接切面与目标对象或类型创建代理的过程

## 常用注解

Spring AOP文档: <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#spring-core>

- `@EnableAspectJAutoProxy` 使用这个之后才支持AspectJ注解
- `@Aspect` 使用这个时候还不是一个Bean,必须使用Java config自己声明一个Bean或者增加一个`@Component`
- `@Pointcut`
- `@Before`  
不管什么情况,结束就做
- `@After` / `@AfterReturning` / `@AfterThrowing`
- `@Around`
- `@Order` 指定执行顺序,越小优先级越高



# 如何打印 SQL

# HikariCP

- P6SQL, <https://github.com/p6spy/p6spy>

# Alibaba Druid

- 内置 SQL 输出
- [https://github.com/alibaba/druid/wiki/Druid中使用log4j2进行日志输出](https://github.com/alibaba/druid/wiki/Druid%20%E4%B8%AD%E7%94%A8log4j2%E8%BF%BB%E6%97%A5%E8%BE%BD%E8%BE%9C)

**“Talk is cheap, show me the code.”**

*Chapter 5 / performance-aspect-demo*

# SpringBucks 进度小结

## 本章小结

- Project Reactor 的基本用法
- 如何通过 Reactive 的方式访问 NoSQL
- 如何通过 Reactive 的方式访问 RDBMS
- Spring AOP 的基本概念
- 监控 DAO 层的简单方案

# SpringBucks 进度小结

- 通过 Reactive 的方式来保存数据与操作缓存