

<WE/BUILD>

December 12th, 2020
Block 71 Saigon

WeBuild Day 2020

Toan Tran

Google Developer Expert on Kotlin |  GDG

“Reactive programming in Android”





About speaker

Hello! I'm Toan

Kotlin GDE, Android folk @Lazada



<https://toan.mobi>



[toan_mobi](https://twitter.com/toan_mobi)



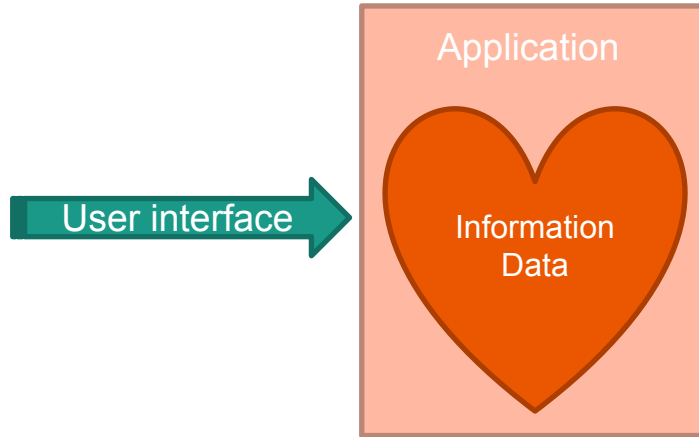
toantran-ea



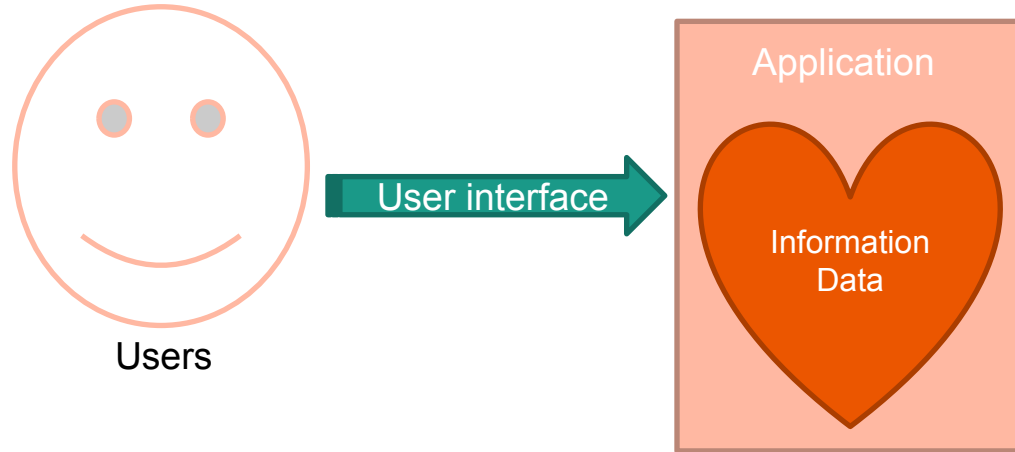
Reactive Programming in Kotlin|Android

1. The ultimate goal of app development.
2. Introducing Reactive Programming.
3. Realize RP with Kotlin in Android.
4. Some common patterns we may use daily.
5. Key-takeaways.

1. Let's talk about application development



1. Let's talk about application development



1. Let's talk about application development



Speed

Effective

Efficient

Scalable



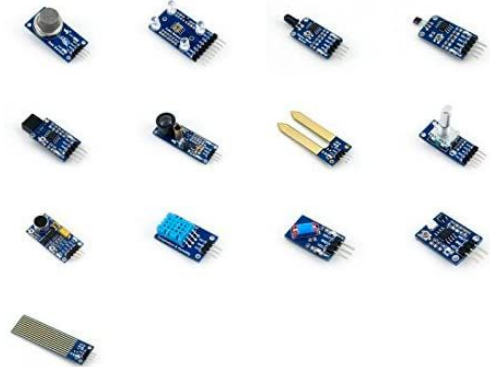
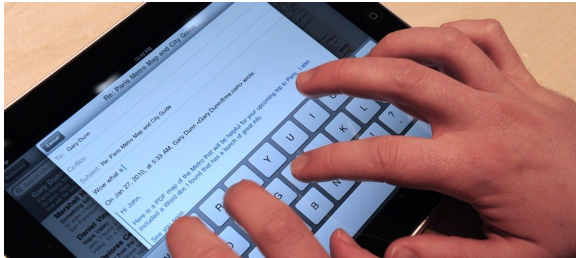


1. Introducing Reactive Programming

1. A programming paradigm.
2. Based on 3 pillars:
 - Data streams.
 - Asynchronous processing.
 - Functional programming.

1. Introducing Reactive Programming

Data streams: sequence of events. It could be anything!



1. Introducing Reactive Programming





1. Introducing Reactive Programming

Reactive Programming vs ...

```
Val data = getDataFromStream()  
If (data is matched condition) {  
    ui.display()  
}
```

```
getDataFromStream().  
    .filter( condition )  
    .subscribe {  
        ui.display()  
    }
```



1. Introducing Reactive Programming

Reactive Programming vs ...

```
Val remote = getDataFromStream()  
// blocking call  
Val local = readFromLocalDB()  
// blocking call  
  
If (remote is matched condition 1  
AND local matched condition 2 ) {  
    ui.display()  
}
```

```
getDataFromStream().  
concat(readFromLocalDB)  
.filter( condition1 AND condition 2 )  
.subscribe {  
    ui.display()  
}
```



1. Introducing Reactive Programming

Reactive Programming vs ...

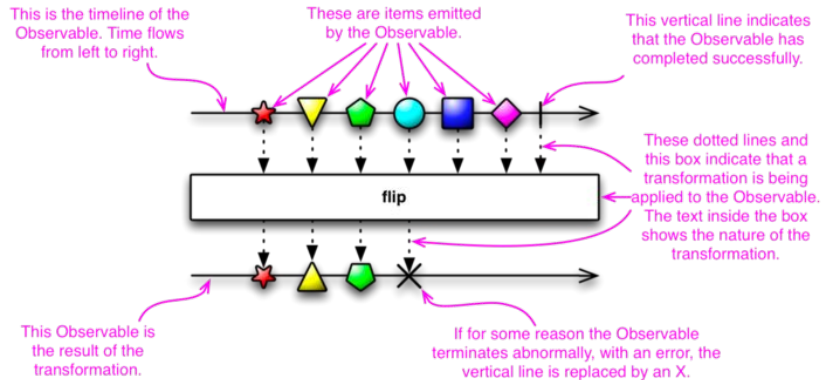
```
Callback<Remote> remote = getDataFromStream()  
Callback<Local> local = readFromLocalDB()
```

```
remote.execute(THREAD_IO).onResult { res1 ->  
    local.execute(THREAD_IO).onResult {res2 ->  
        if (condition 1 and condition 2) {  
            runOnnUiThread {  
                ui.display()  
            }  
        }  
    }  
}
```

```
getDataFromStream().  
concat(readFromLocalDB)  
.subscribeOn(THREAD_IO)  
.filter( condition1 AND condition 2 )  
.observeOn(UI_THREAD)  
.subscribe {  
    ui.display()  
}
```

3. Realize RP in Kotlin/Android

- <http://reactivex.io/>
- RxJava, RxJS, RxSwift, etc
- Early adopted by big names: Netflix, Microsoft, GitHub, SoundCloud, etc





3. Realize RP in Kotlin/Android

Main functionalities of Rx family



CREATE

Easily create event streams or data streams.



COMBINE

Compose and transform streams with query-like operators.



LISTEN

Subscribe to any observable stream to perform side effects.



3. Realize RP in Kotlin/Android

Better codebases



Functional

Avoid intricate stateful programs, using clean input/output functions over observable streams.



Less is more

ReactiveX's operators often reduce what was once an elaborate challenge into a few lines of code.



Async error handling

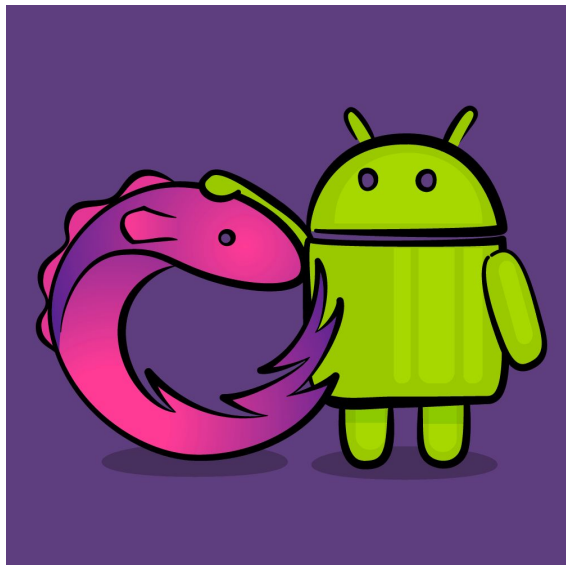
Traditional try/catch is powerless for errors in asynchronous computations, but ReactiveX is equipped with proper mechanisms for handling errors.



Concurrency made easy

Observables and Schedulers in ReactiveX allow the programmer to abstract away low-level threading, synchronization, and concurrency issues.

3. Realize RP in Kotlin/Android



RxJava + Android



3. Realize RP in Kotlin/Android

RxJava:

- Create
- Combine
- Subscribe



RxAndroid:

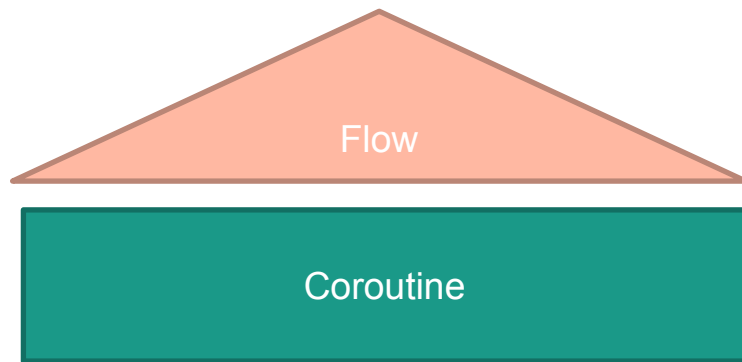
- Android
threading

3. Realize RP in Kotlin/Android





3. Realize RP in Kotlin/Android





4. Common patterns

Creating patterns: To create observables

- Create
- Defer
- From
- Just



4. Common patterns

Transforming patterns:

- Map
- FlatMap
- GroupBy
- Buffer



4. Common patterns

Filtering patterns:

- Filter
- Debounce
- First
- Last



4. Common patterns

Combine patterns:

- Concat
- Join
- Merge
- Zip



5. Key takeaways

1. Reactive Programming is paradigm based on async stream processing with the support of functional programming.
2. Realize with ReactiveX language implementations: RxJava, RxKotlin, RxSwift, RxJS
3. Rx provided with set of common tools/pattern to solve data manipulation in an asynchronous manner.



Thank you