

Usando Java 8 no Android

Eduardo Bonet

Motivação

- Java 8 foi lançado há mais de 2 anos, trazendo diversas novidades, mas que não chegaram ao Android.
- A nova toolchain (Jack & Jill) trouxe um suporte limitado de algumas features, porém deixou outras de fora e algumas são suportadas apenas de Android N para frente.

Java 8

- Redução de verbosidade
- Novas API's:
 - `java.time.*`
 - `java.util.stream.*`
 - `java.util.function.*`
- Mudanças na linguagem:
 - Default Methods e métodos estáticos para Interfaces
 - Lambda
 - Method References

Streams

API melhorada para manuseio de coleções, facilitando execução paralela

// Java 7

```
names = new ArrayList<>();  
for (Person p : people) {  
    if(p.age > 16)  
        names.add(p.name);  
}
```

// Java 8

```
names = people.stream()  
    .filter(p -> p.age > 16)  
    .map(p -> p.name)  
    .collect(Collectors.toList());
```

Time API

Nova API para lidar com data e hora, substituindo as classes `java.util.Calendar` e `java.util.Date`.

```
// Java7
```

```
Date date, datePlusThreeDays;  
date = new GregorianCalendar(2014, Calendar.FEBRUARY, 11).getTime()  
  
Calendar c = Calendar.getInstance();  
c.setTime(date);  
c.add(Calendar.DATE, 3)  
datePlusThreeDays = c.getTime()
```

```
// Java 8
```

```
LocalDate otherDate, otherDatePlusThreeDays;  
otherDate = LocalDate.of(2014, Month.FEBRUARY, 11);  
otherDatePlusThreeDays = otherDate.plus(3, ChronoUnit.DAYS);
```

Lambdas

Lambdas fornecem uma sintaxe muito mais limpa para Functors.

```
// Java 7
```

```
v.setOnClickListener(new View.OnClickListener() {  
    @Override public void onClick(View view) {  
        Log.d(TAG, "onClick: ");  
    }  
});
```

```
// Java 8 Lambda
```

```
v.setOnClickListener(view -> Log.d(TAG, "onClick: "));
```

```
Observable.from(people)
    .filter(new Func1<Person, Boolean>() {
        @Override
        public Boolean call(Person person) {
            return person.age > 16;
        }
    })
    .map(new Func1<Person, String>() {
        @Override
        public String call(Person person) {
            return person.name;
        }
    })
    .subscribe(new Action1<String>() {
        @Override
        public void call(String s) {
            System.out.println(s);
        }
    });
```

```
Observable.from(people)
    .filter(person -> person.age > 16)
    .map(person -> person.name)
    .subscribe(s -> System.out.println(s));
```

Method References

Method References são uma versão ainda mais simplificada de lambdas, onde o argumento é simplesmente repassado para outra função

```
Observable.from(people)
    .filter(person -> person.age > 16)
    .map(person -> person.name)
    .subscribe(System.out::println); // .subscribe(s -> System.out.println(s));
```


Try-with-resources

Sintaxe melhorada para objetos que precisam ser fechados após seu uso.

// Java 7

```
BufferedReader br = new BufferedReader(new FileReader(path));  
try {  
    System.out.println(br.readLine());  
} finally {  
    if (br != null) br.close();  
}
```

// Java 8

```
try (BufferedReader br = new BufferedReader(new FileReader(path))) {  
    System.out.println(br.readLine());  
}
```

Default Methods para Interfaces

```
interface Vehicle {  
    default void print(){  
        System.out.println("I am a vehicle!");  
    }  
}  
  
class Car implements Vehicle {  
    public void print(){  
        Vehicle.super.print();  
        System.out.println("I am a car!");  
    }  
}  
  
class Boat implements Vehicle {  
}
```

Trazendo o Java 8 para o Android

Jack

- Nova ferramenta do Android para compilar código Java para .dex
- Introduzido no Android N
- Suporte a Lambdas e Method References para todas as versões
- Suporte a Default Methods e Streams para Android 24+
- **Não traz `java.time.*`**

Streams: LightweightStreams

Implementação da Streams API usando Collections do Java 7.

```
List<String> names = Stream.of(people)
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .collect(Collectors.toList());
```

- Method Count: 719 (1.1.2)

Streams: RxJava

Observáveis são fundamentalmente diferente de Streams (push vs pull), mas pode-se ter o mesmo resultado usando `Observable.from(myCollection)`.

```
List<String> names = Observable.from(people)
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .toList().toBlocking().first();
```

- Method Count: 5492 (1.1.8)

Streams

// Stream

```
people.stream()  
    .filter(p -> p.age > 16)  
    .map(p -> p.name)  
    .collect(Collectors.toList());
```

// LightweightStreams

```
Stream.of(people)  
    .filter(p -> p.age > 16)  
    .map(p -> p.name)  
    .collect(Collectors.toList());
```

// RxJava

```
Observable.from(people)  
    .filter(p -> p.age > 16)  
    .map(p -> p.name)  
    .toList().toBlocking().first();
```

java.time.*:

ThreeTenABP

- Versão otimizada para Android da biblioteca ThreeTenBP
- Mesma implementação do java.time.*, tornando fácil refatoração futura
- Method Count: 3280

Retrolambda

- Transformar código Java 8 em código compatível com Java 5, 6 e 7
- Opera em tempo de compilação
- Suporte completo a Lambdas, Try-With-Resources e Method References
- Suporte parcial a default methods

RetroLambda vs Jack

Code	Retrolambda 2.1.0	Retrolambda 2.3.0	Jack 24.0.1
<pre>new Runnable() { @Override public void run() { greeter.sayHi(); } }</pre>	2	2	2
<pre>() -> greeter.sayHi()</pre>	6 or 7	4	3
<pre>greeter::sayHi</pre>	4	3 or 4	3

<https://speakerdeck.com/jakewharton/exploring-hidden-java-costs-360-andev-july-2016?slide=126>

Resumo

	RL	Jack	RxJava	LS	TT
Streams		24+	✓	✓	
Default Methods	Parcial	24+			
Lambda	✓	✓			
Method References	✓	✓			
Try-With-Resources	✓	✓			
java.time.*					✓

- RL: Retrolambda, LS: LightweightStreams, TT: ThreeTenABP

Referências

- [Jack](#)
- [Jack e Java 8](#)
- [Retrolambda](#)
- [Lightweight Stream](#)
- [ThreeTenABP](#)
- [Estudo sobre API para date](#)
- [RxJava](#)
- [Retrolambda vs Jack](#)

Obrigado

Dúvidas?

[blog](#) | [github](#) | [linkedin](#)