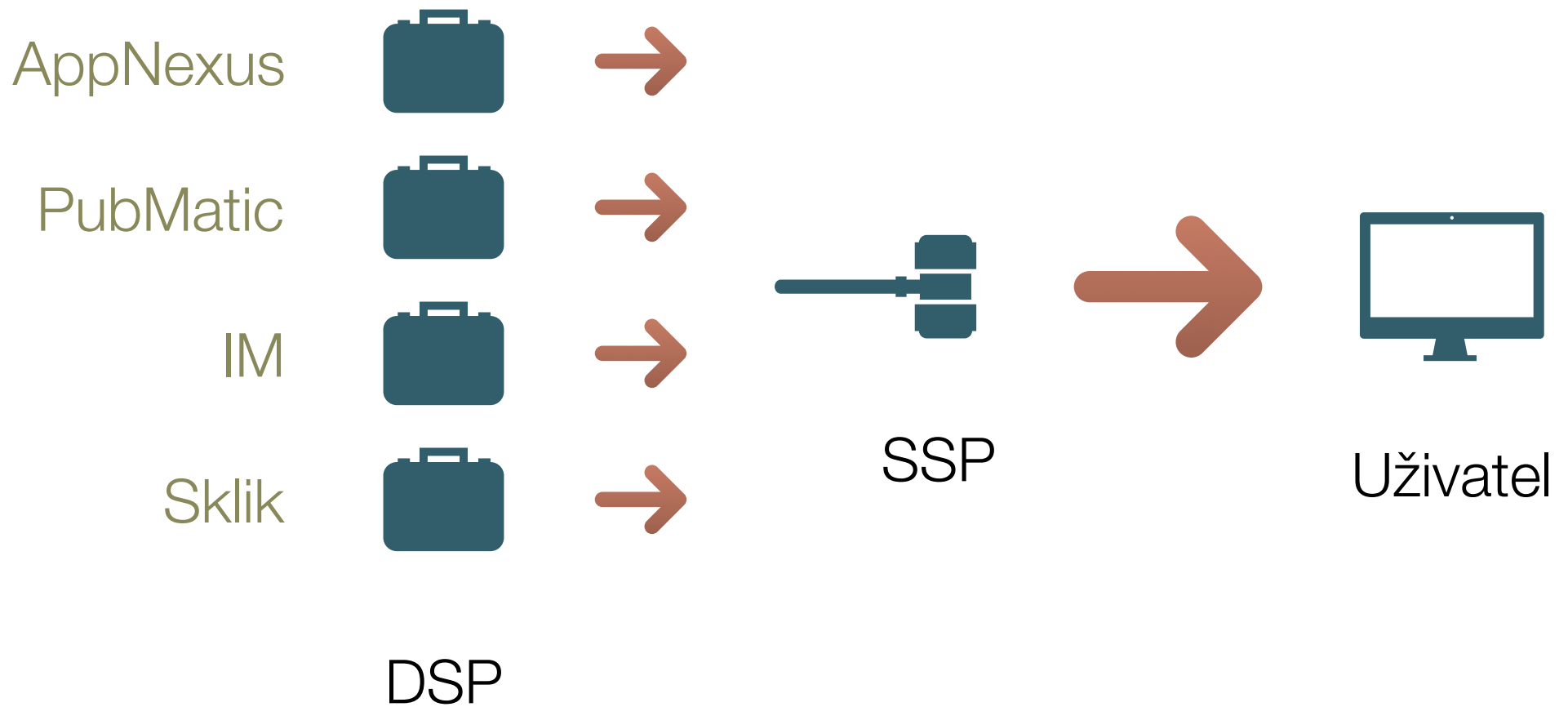


Kotlin na SSP serveru

Radek Miček

Aukce reklamy



Aukce reklamy

- Imprese od DSP se ukládají do:

```
data class DSPResponseImp(  
    val zoneId: Long,  
    val format: Dimensions,  
    val originalPrice: Double,  
    val price: Double,  
    val netPrice: Double,  
    val privateDeal: Boolean,  
    val currency: String,  
    val nonExpandedContent: String,  
    val nonexpandedWinNoticeURL: String  
)
```

Jak v Javě vytvářet třídy s mnoha fieldy?

- Chceme zaručit, že všechny fieldy budou inicializovány!

Jak v Javě vytvářet třídy s mnoha fieldy?

- Chceme zaručit, že všechny fieldy budou inicializovány!
- Konstruktor s parametrem pro každý field?

```
DSPResponseImp imp = new DSPResponseImp(  
    434,  
    new Dimensions(300, 300),  
    5.0, 125.0, 125.0,  
    false, "USD",  
    "foo", ""  
);
```

Jak v Javě vytvářet třídy s mnoha fieldy?

- Chceme zaručit, že všechny fieldy budou inicializovány!
- Spolehnout se na to, že programátor nezapomene inicializovat field?

Jak v Javě vytvářet třídy s mnoha fieldy?

- Chceme zaručit, že všechny fieldy budou inicializovány!
- Builder a při volání build() otestovat, že všechny fieldy byly inicializovány?

Pojmenované argumenty

- Řešení:

```
val imp = DSPResponseImp(  
    zoneId = 434,  
    format = Dimensions(width = 300, height = 300),  
    originalPrice = 5.0, price = 125.0, netPrice = 125.0,  
    privateDeal = false, currency = "USD",  
    nonExpandedContent = "foo", nonexpandedWinNoticeURL = ""  
)
```

- Kotlin navíc podporuje i defaultní argumenty

Nepovinné hodnoty

- V OpenRTB je hodně polí nepovinných, jak to reprezentovat v Javě?

Nepovinné hodnoty

- V OpenRTB je hodně polí nepovinných, jak to reprezentovat v Javě?
- Pomocí hodnoty null?

Nepovinné hodnoty

- V OpenRTB je hodně polí nepovinných, jak to reprezentovat v Javě?
- Pomocí hodnoty null + pole anotovat @Nullable resp. @NotNull?

Nepovinné hodnoty

- V OpenRTB je hodně polí nepovinných, jak to reprezentovat v Javě?
- Pomocí typu Optional z Javy 8?

Bezpečná práce s null v Kotlinu

- Typ String nepřipouští null:

```
var a: String = "abc"  
a = null // Nepřeloží se
```

- Typ String? připouští null:

```
var b: String? = "abc"  
b = null
```

```
b.length // Nepřeloží se
```

```
b?.length ?: -1
```

Bezpečná práce s null v Kotlinu

- Bezpečná volání pomocí ?. lze řetězit:

```
imp?.banner?.width
```

Extension funkce a vlastnosti

- K existujícím třídám můžeme “přidávat” metody.
- Např. pro řetězce můžeme psát

```
str.isNullOrEmpty()
```

místo

```
str == null || str.isEmpty()
```

- Deklarace:

```
fun CharSequence?.isNullOrEmpty(): Boolean =  
    this == null || this.isEmpty()
```

Použít Scalu?

Použít Scalu?

- Co dělá následující kód?

```
Map(8 -> "osm").map(_._1)
```

Použít Scalu?

- Co dělá následující kód?

```
Map(8 -> "osm").map(_._1)
```

- Jaké metody se volají?
- Jaké objekty se vytvářejí?
- Instance jaké třídy je vrácena?
- Jaký je typ návratové hodnoty?

Použít Scalu?

- Co dělá následující kód?

```
Map(8 -> "osm").map(_._1)
```

- Dělá:

```
Map.apply(ArrowAssoc(8).->("osm"))  
  .map(x => x._1)(Iterable.newBuilder)
```

- Je vrácena instance třídy List[String]
- Typ návratové hodnoty je Iterable[String]

removeParams

```
@NotNull
public List<NameValuePair> removeParams(
    @NotNull Set<String> remove,
    @NotNull List<NameValuePair> params) {

    return params
        .stream()
        .filter(p -> !remove.contains(p.getName()))
        .collect(Collectors.toList());

}
```

removeParams

```
@NotNull
public List<NameValuePair> removeParams(
    @NotNull Set<String> remove,
    @NotNull List<NameValuePair> params) {

    return params
        .stream()
        .filter(p -> !remove.contains(p.getName()))
        .collect(Collectors.toList());
}

    ↓

fun removeParams(
    remove: Set<String>,
    params: List<NameValuePair>): List<NameValuePair> {


    return params
        .stream()
        .filter { p -> !remove.contains(p.name) }
        .collect(Collectors.toList())
}
```

removeParams

```
@NotNull
public List<NameValuePair> removeParams(
    @NotNull Set<String> remove,
    @NotNull List<NameValuePair> params) {

    return params
        .stream()
        .filter(p -> !remove.contains(p.getName()))
        .collect(Collectors.toList());
}

    
```



```
fun removeParams(
    remove: Set<String>,
    params: List<NameValuePair>): List<NameValuePair> {


    return params
        .stream()
        .filter { p -> !remove.contains(p.name) }
        .toList()
}
    
```

removeParams

```
@NotNull
public List<NameValuePair> removeParams(
    @NotNull Set<String> remove,
    @NotNull List<NameValuePair> params) {

    return params
        .stream()
        .filter(p -> !remove.contains(p.getName()))
        .collect(Collectors.toList());
}


```



```
fun removeParams(
    remove: Set<String>,
    params: List<NameValuePair>): List<NameValuePair> {

    return params
        .stream()
        .filter { p -> p.name !in remove }
        .toList()
}

```


removeParams

```
@NotNull
public List<NameValuePair> removeParams(
    @NotNull Set<String> remove,
    @NotNull List<NameValuePair> params) {

    return params
        .stream()
        .filter(p -> !remove.contains(p.getName()))
        .collect(Collectors.toList());
}

fun removeParams(
    remove: Set<String>,
    params: List<NameValuePair>): List<NameValuePair> {

    return params
        .stream()
        .filter { it.name !in remove }
        .toList()
}
```



removeParams

```
@NotNull
public List<NameValuePair> removeParams(
    @NotNull Set<String> remove,
    @NotNull List<NameValuePair> params) {

    return params
        .stream()
        .filter(p -> !remove.contains(p.getName()))
        .collect(Collectors.toList());
}
```



```
fun removeParams(
    remove: Set<String>,
    params: List<NameValuePair>): List<NameValuePair> =

    params
        .stream()
        .filter { it.name !in remove }
        .toList()
```

Pár dalších věcí, co Kotlin umí

- Pro rovnost se používá `==` (nikoliv `equals`)
- Typové aliasy
- Typy pro lambdy (není třeba psát rozhraní)
- Možnost mít část kódu v Javě a část v Kotlinu (není třeba vše přepsat naráz)
- Kompilace do JavaScriptu a brzy do nativního kódu
- IDEA umí převést Javu do Kotlinu (a skoro to funguje)
- Korutiny (např. pro asynchronní kód)

Díky za pozornost! Otázky?