Kotlin Serialization

A viable alternative to Jackson?

Alex

@alexswilliams

Novice climber 🧗‍♂️
Passable cyclist 🚵‍♂️
Almost buoyant swimmer 🏊‍♂️
Deploys on Fridays 👨‍💻
Homo-romantic demisexual 🌈
Intolerant to 🥛🥝🍳

Alex Williams

- JVM and Kotlin Engineer
- Based in Leeds
- Twitter: @alexswilliams
- GitHub: alexswilliams

Contents

Motivation for Kotlin Serialization

Code examples comparing to Jackson

Timing performance

Summary

What is Jackson?

A Java object mapper

That speaks JSON

The default for every single spring project

Mature and highly configurable

What is Kotlin Serialization?

A Kotlin object mapper

Understands JSON

Fully Kotlin aware

A pluggable framework

Understands ProtoBuf

Understands CBOR

Could understand XML, Avro, etc

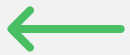
A work in progress

Has not yet reached v1.0.0

Why Kotlinx Serialization?

Multiplatform

Kotlin-specific



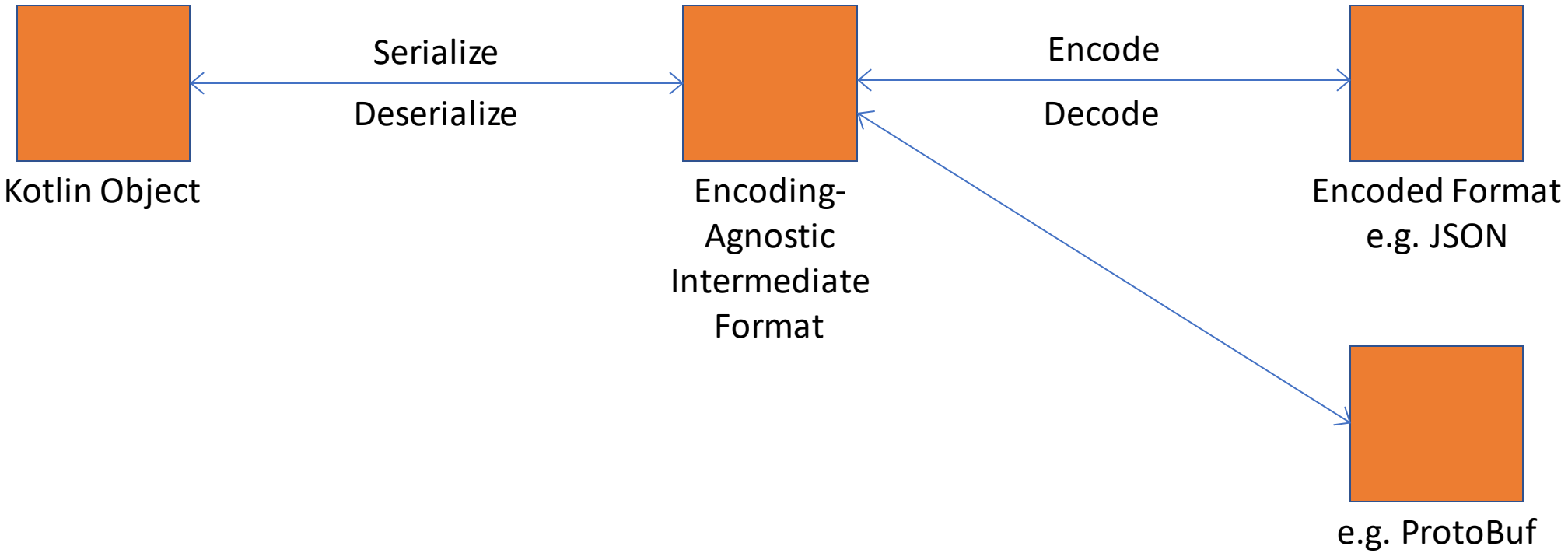
Push errors left

Compiler plugin = compile-time error

Customisable

Framework, not library

Kotlin Serializers and Codecs



Serializers

```kotlin
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: String
)

Cat.Companion.serializer()
```

```java
public static final class Companion {
   private Companion() {
   }

   // $FF: synthetic method
   public Companion(DefaultConstructorMarker $constructor_marker) { this(); }

   @NotNull
   public final KSerializer serializer() { return (KSerializer)Cat.$serializer.INSTANCE; }
}
```

```java
@NotNull
public Cat deserialize(@NotNull Decoder decoder) {
   Intrinsics.checkParameterIsNotNull(decoder, "decoder");
   SerialDescriptor var2 = $$serialDesc;
   boolean var4 = false;
   int var5 = 0;
   String var6 = null;
   int var7 = 0;
   float var8 = 0.0F;
   String var9 = null;
   CompositeDecoder decoder = decoder.beginStructure(var2, new KSerializer[0]);

   while(true) {
      int var3 = decoder.decodeElementIndex(var2);
      switch(var3) {
      case -2:
         var4 = true;
      case 0:
         var6 = decoder.decodeStringElement(var2, 0);
         var5 |= 1;
```

From JSON to Object

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: String
)
```

```
{
  "Name": "Marmalade",
  "Age": 3,
  "Weight": 10.4,
  "Colour": "Ginger"
}
```

Kotlin Serialization

```
val jsonMapper = Json(JsonConfiguration.Stable)
val parsedCat = jsonMapper.parse(Cat.Companion.serializer(), inputString)
```

Jackson

```
val jsonMapper = ObjectMapper().registerModule(KotlinModule())
val parsedCat = jsonMapper.readValue(inputString, Cat::class.java)
```

Cat(Name=Marmalade, Age=3, Weight=10.4, Colour=Ginger)

Missing Fields

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: String?
)
```

```
{
  "Name": "Marmalade",
  "Age": 3,
  "Weight": 10.4
}
```

Kotlin Serialization

Field 'Colour' is required, but it was missing

```
val Colour: String? = null
```

Jackson

Cat(Name=Marmalade, Age=3, Weight=10.4, Colour=null)

Extra Fields

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: String
)
```

```
{
  "Name": "Marmalade",
  "Age": 3,
  "Weight": 10.4,
  "Colour": "Ginger",
  "Personality": "Useless"
}
```

Kotlin Serialization

```
val jsonMapper = Json(JsonConfiguration.Stable.copy(strictMode = false))
```

Jackson

```
val jsonMapper = ObjectMapper().registerModule(KotlinModule()).apply {
    disable(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES)
}
```

Cat(Name=Marmalade, Age=3, Weight=10.4, Colour=Ginger)

Enums

Yup, they work.

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: CatColour
)
enum class CatColour {
    Ginger, Black, White
}
```

```
{
  "Name": "Marmalade",
  "Age": 3,
  "Weight": 10.4,
  "Colour": "Ginger"
}
```

Collections

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: CatColour
)
enum class CatColour {
    Ginger, Black, White
}
```

```
[
 {
  "Name": "Marmalade",
  "Age": 3,
  "Weight": 10.4,
  "Colour": "Ginger"
 },
 {
  "Name": "Paprika",
  "Age": 6,
  "Weight": 12.2,
  "Colour": "Ginger"
 },
 {
  "Name": "Mr MistoffeLeeds",
  "Age": 2,
  "Weight": 7.1,
  "Colour": "Black"
 }
]
```

Kotlin Serialization

```
val parsedCats =
  jsonMapper.parse(Cat.serializer().list, inputString)
```

```
[
 Cat(Name=Marmalade, Age=3, Weight=10.4, Colour=Ginger),
 Cat(Name=Paprika, Age=6, Weight=12.2, Colour=Ginger),
 Cat(Name=Mr MistoffeLeeds, Age=2, Weight=7.1, Colour=Black)
]
```

Collections

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: CatColour
)
enum class CatColour {
    Ginger, Black, White
}
```

```
[
  {
    "Name": "Marmalade", …
  },
  {
    "Name": "Paprika", …
  },
  {
    "Name": "Mr MistoffeLeeds", …
  }
]
```

Jackson

```
jsonMapper.readValue(inputString,  List::class.java)
```

```
[
  {Name=Marmalade, Age=3, Weight=10.4, Colour=Ginger},
  {Name=Paprika, Age=6, Weight=12.2, Colour=Ginger},
  {Name=Mr MistoffeLeeds, Age=2, Weight=7.1, Colour=Black}
]
```

Casting to List<Cat> doesn't help!

Collections

```kotlin
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: CatColour
)
enum class CatColour {
    Ginger, Black, White
}
```

```
[
  {
    "Name": "Marmalade", …
  },
  {
    "Name": "Paprika", …
  },
  {
    "Name": "Mr MistoffeLeeds", …
  }
]
```

Jackson

```kotlin
jsonMapper.readValue(inputString, Array<Cat>::class.java).toList()

jsonMapper.readValue(inputString, object : TypeReference<List<Cat>>() {})

jsonMapper.readValue<List<Cat>>(
    inputString,
    jsonMapper.typeFactory.constructCollectionType(List::class.java, Cat::class.java)
)
```

```
[
  Cat(Name=Marmalade, Age=3, Weight=10.4, Colour=Ginger),
  Cat(Name=Paprika, Age=6, Weight=12.2, Colour=Ginger),
  Cat(Name=Mr MistoffeLeeds, Age=2, Weight=7.1, Colour=Black)
]
```

Collections

```
@Serializable
data class Cat(
    val Name: String,
    val Age: Int,
    val Weight: Float,
    val Colour: CatColour
)
enum class CatColour {
    Ginger, Black, White
}
```

```
[
  {
    "Name": "Marmalade", …
  },
  {
    "Name": "Paprika", …
  },
  {
    "Name": "Mr MistoffeLeeds", …
  }
]
```

Kotlin Serialization

```
val parsedCats =
  jsonMapper.parse(Cat.serializer().list, inputString)
```

Custom Serdes

```
@Serializable
data class Cat(
    val Name: String,
    val ID: UUID,
    val Age: Int,
    val Weight: Float,
    val Colour: String
)
```

```
{
 "Name": "Marmalade",
 "ID": "0e8f548a-e792-4b8a-aeba-8fc06ac810a3",
 "Age": 3,
 "Weight": 10.4,
 "Colour": "Ginger"
}
```

```
@Serializable
data class Cat(
    val Name: String,
    val ID: UUID,
    val Age: I
    val Weight
    val Colour
) {
    enum class CatColour {
        Ginger, Black, White
    }
}
```

Serializer has not been found for type 'UUID'. To use context serializer as fallback, explicitly annotate type or property with @ContextualSerialization

Custom Serdes

```
{
 "Name": "Marmalade",
 "ID": "0e8f548a-e792-4b8a-aeba-8fc06ac810a3",
 "Age": 3,
 "Weight": 10.4,
 "Colour": "Ginger"
}
```

```
@Serializable
data class Cat(
    val Name: String,
    @Serializable(with = UUIDSerializer::class) val ID: UUID,
    val Age: Int,
    val Weight: Float,
    val Colour: String
)


class UUIDSerializer : KSerializer<UUID> {
    override val descriptor: SerialDescriptor get() = StringDescriptor
    override fun deserialize(decoder: Decoder): UUID = UUID.fromString(decoder.decodeString())
    override fun serialize(encoder: Encoder, obj: UUID) = encoder.encodeString(obj.toString())
}
```

Stringifying - Going the other way

Kotlin Serialization

```
val jsonMapper = Json(JsonConfiguration.Stable)
val asJson = jsonMapper.stringify(Cat.serializer(), cat)
```

{"Name":"Fluffy","ID":"52bc9bc6-a193-411e-a1b8-f293eec794dd","Age":6,"Weight":6.9,"Colour":"White"}

```
val protoBufMapper = ProtoBuf()
val asProtoBuf = protoBufMapper.dump(Cat.serializer(), cat)
```

[10, 6, 70, 108, 117, 102, 102, 121, 18, 36, 102, 100, 55, 49, 49, 53, 49, 99, ...

```
val cborMapper = Cbor()
val asCbor = cborMapper.dump(Cat.serializer(), cat)
```

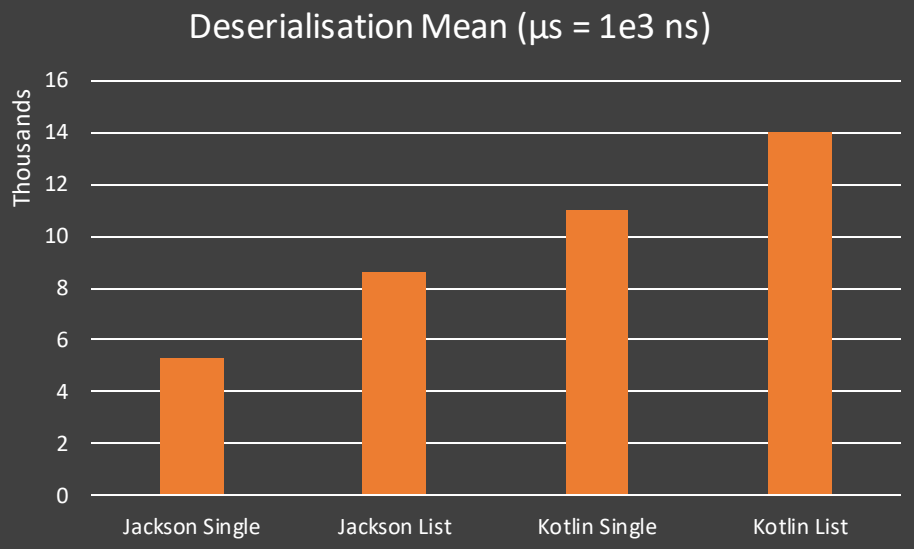
[-65, 100, 78, 97, 109, 101, 102, 70, 108, 117, 102, 102, 121, 98, 73, 68, 120, ...

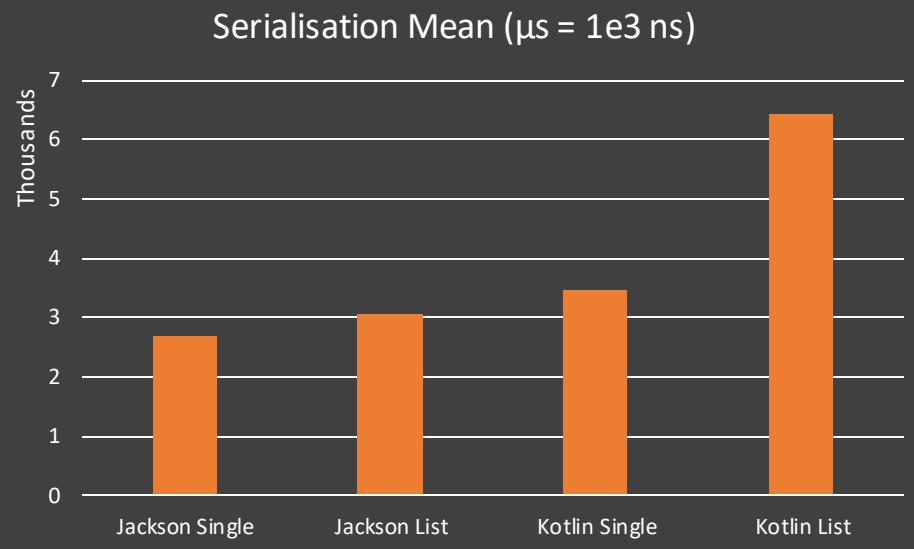
Jackson

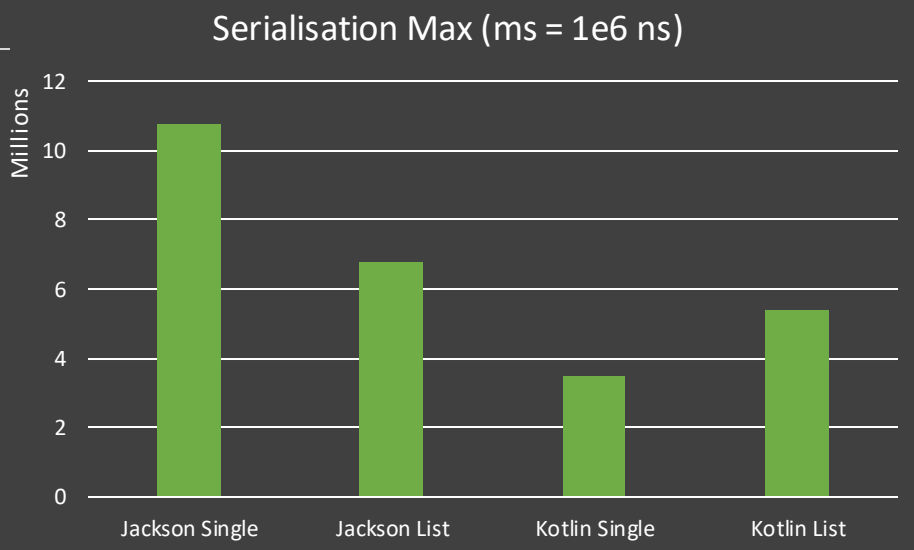
```
val jsonMapper = ObjectMapper()
val asJson = jsonMapper.writeValueAsString(cat)
```

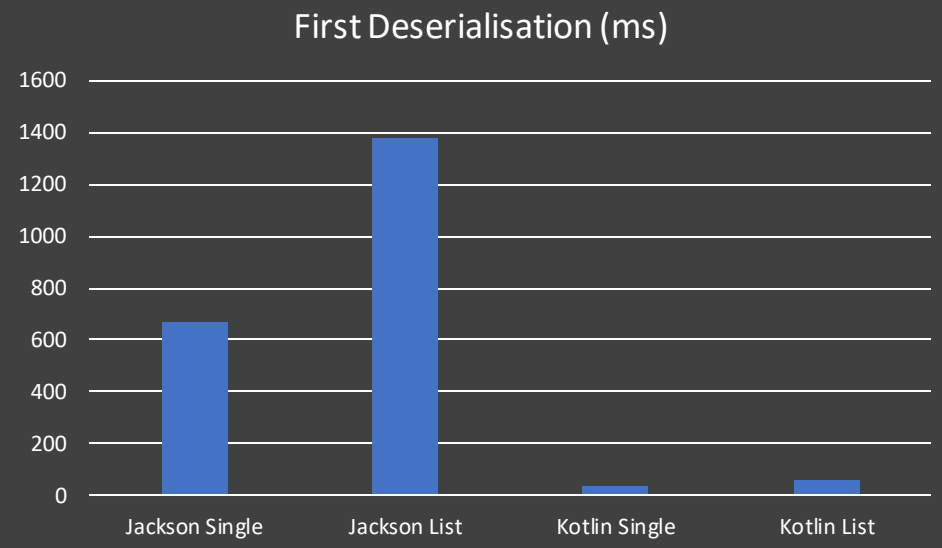
{"name":"Fluffy",
"age":6,
"weight":6.9,
"colour":"White",
"id":"bc1d3823-8618-4000-8b1e-041d5b6e14af"}

Performance









Performance

So which should I use?

Summary

It depends!