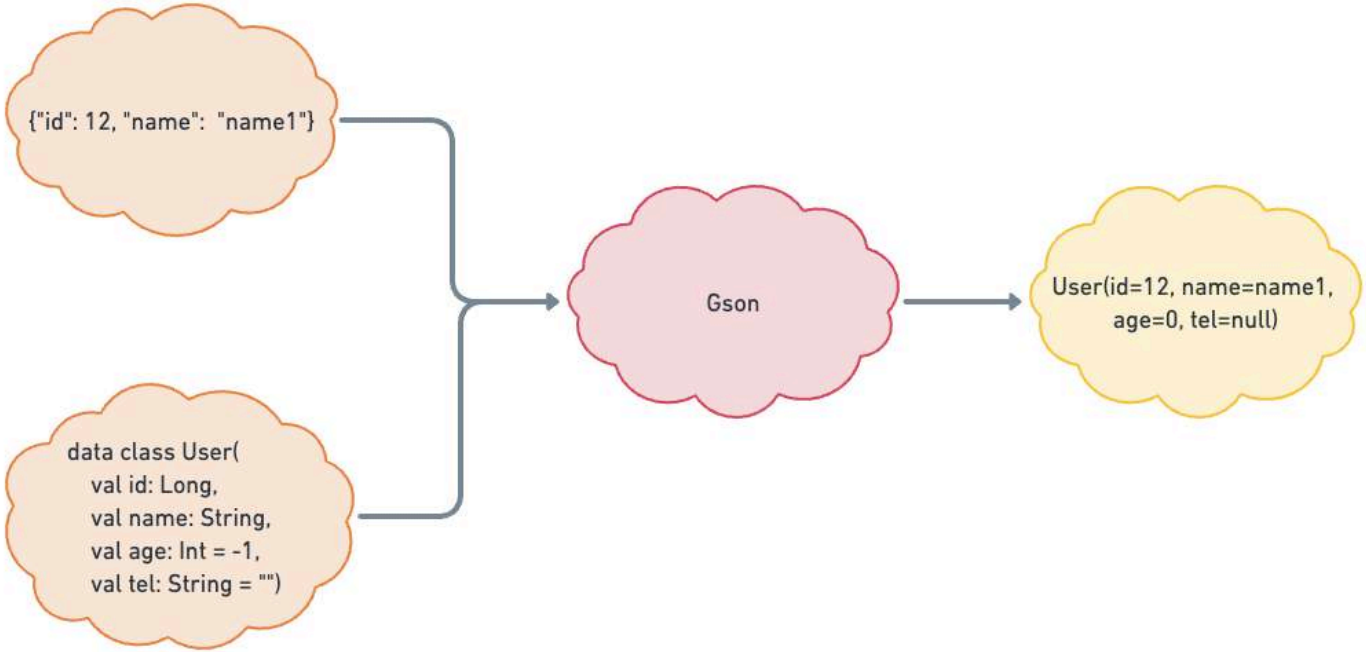
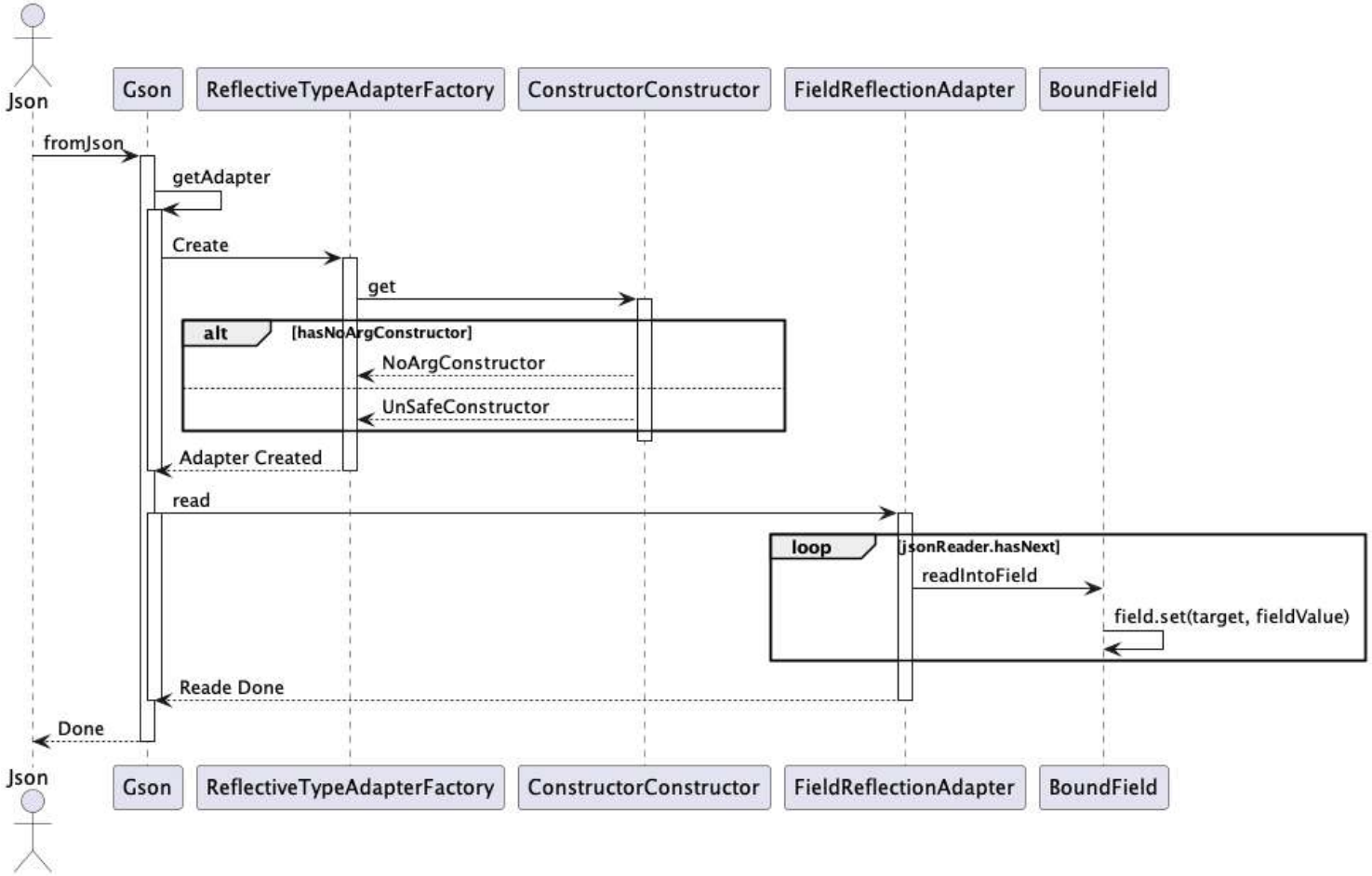
JET BRAINS Kotlin

使用 KCP 打造更安全的 Gson 与更快的 Moshi

程序员江同学

Gson 的空安全与默认值问题

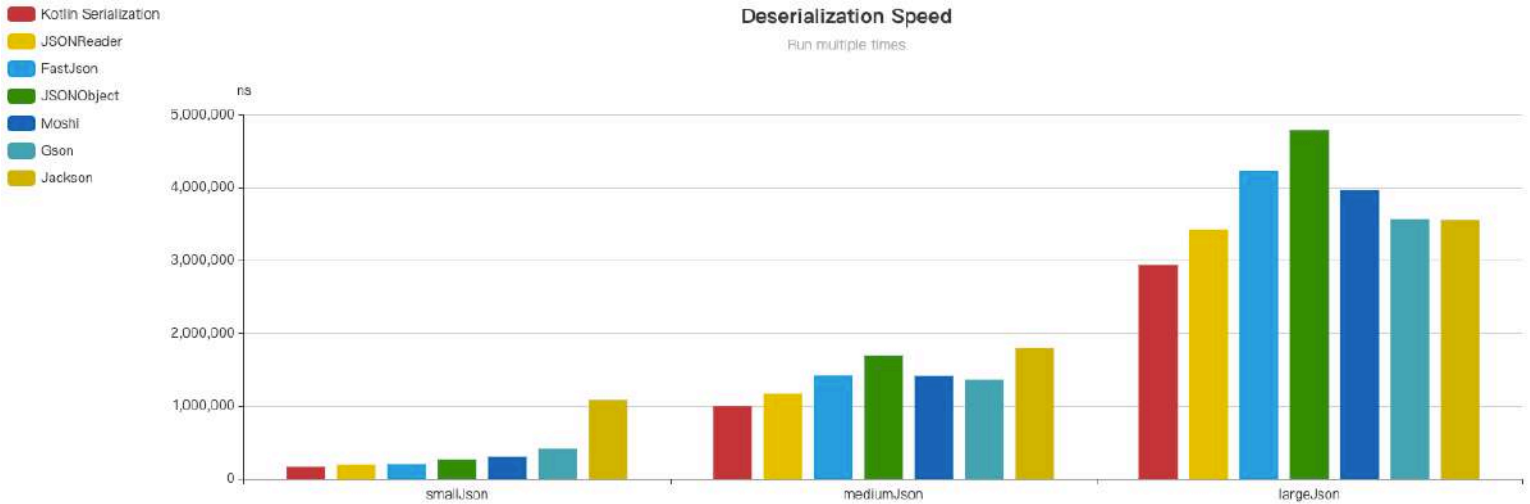



Json
Gson
ReflectiveTypeAdapterFactory
ConstructorConstructor
FieldReflectionAdapter
BoundField
fromJson
getAdapter
Create
get
alt
[hasNoArgConstructor]
NoArgConstructor
UnSafeConstructor
Adapter Created
read
loop
[jsonReader.hasNext]
readIntoField
field.set(target, fieldValue)
Reade Done
Done
Json
Gson
ReflectiveTypeAdapterFactory
ConstructorConstructor
FieldReflectionAdapter
BoundField

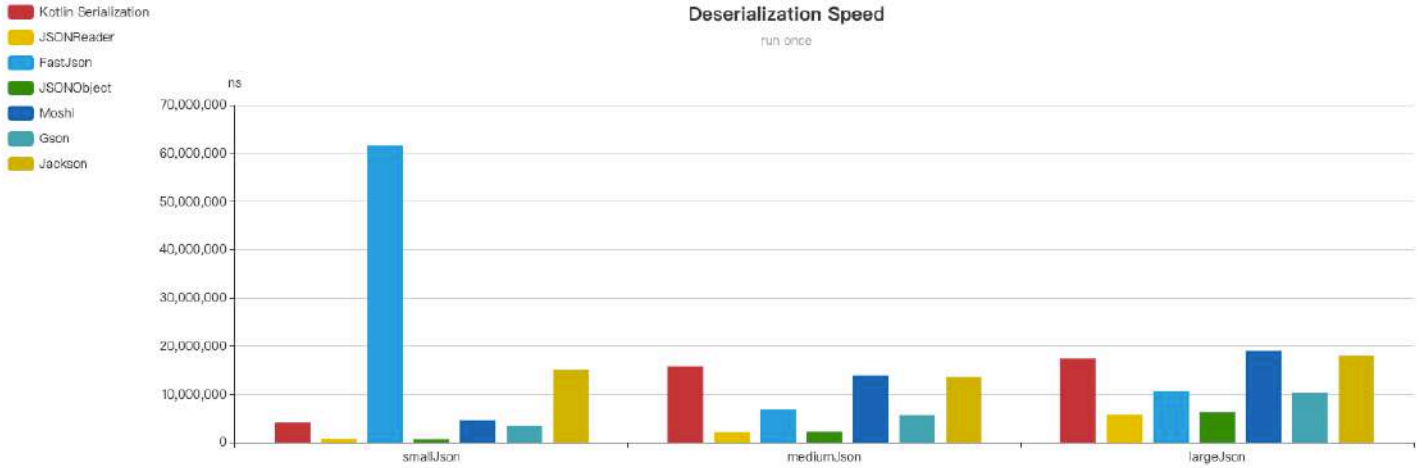
Gson 与 Moshi 性能怎么样？

- 通过 Jetpack Microbenchmark 库进行微基准测试，以避免 CPU 降频，JIT 优化对测试结果的影响
- 测试用例输入包括 12kb, 78kb, 238kb 大小的三个 json 文件，以测试 json 大小对反序列化速度的影响
- 测试结果分为多次运行充分预热与一次运行无预热两种情况，以测试在冷启动情况下反序列化速度的差异

多次运行测试结果


Kotlin Serialization
JSONReader
FastJson
JSONObject
Moshi
Gson
Jackson
Deserialization Speed
Run multiple times
ns
5,000,000
4,000,000
3,000,000
2,000,000
1,000,000
0
smallJson
mediumJson
largeJson

一次运行测试结果


Kotlin Serialization
JSONReader
FastJson
JSONObject
Moshi
Gson
Jackson
Deserialization Speed
run once
ns
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
smallJson
mediumJson
largeJson

Json 与 Protobuf 对比

- Protobuf 的反序列化速度相比之前表现最优秀的 Kotlin Serialization 也有 50% 左右的提升
- 在冷启动时，JSONReader 的表现还是遥遥领先，而 Protobuf 的表现比 Kotlin Serialization 还是要好一些。

多次运行测试结果

	small data	medium data	large data
Kotlin Serialization	171,134 ns	1,010,621 ns	2,945,403 ns
JSONReader	191,611 ns	1,165,174 ns	3,426,443 ns
Protobuf	108,520 ns	576,859 ns	1,674,978 ns

一次运行测试结果

	small data	medium data	large data
Kotlin Serialization	5,033,698 ns	15,217,709 ns	17,278,021 ns
JSONReader	519,167 ns	1,617,604 ns	5,623,749 ns
Protobuf	3,605,313 ns	6,869,480 ns	11,330,886 ns

KUDOS

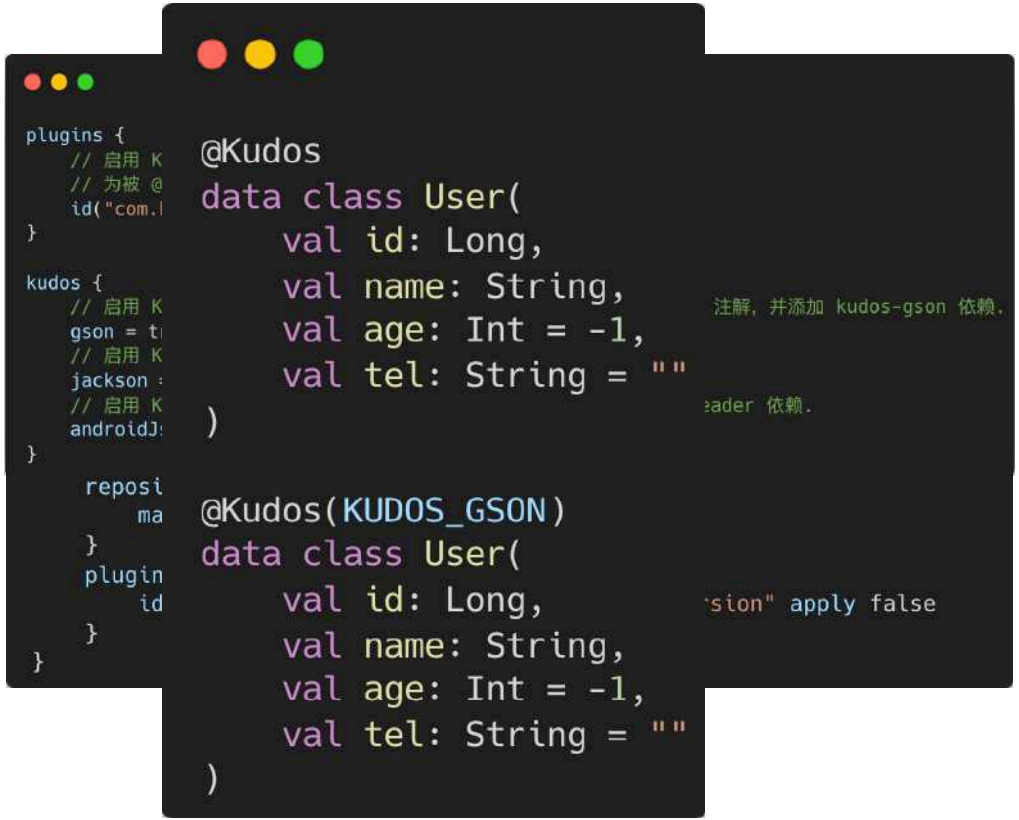
安全，易用，高性能的反序列化框架

KUDOS : Kotlin utilities for deserializing objects

https://github.com/kanyun-inc/Kudos

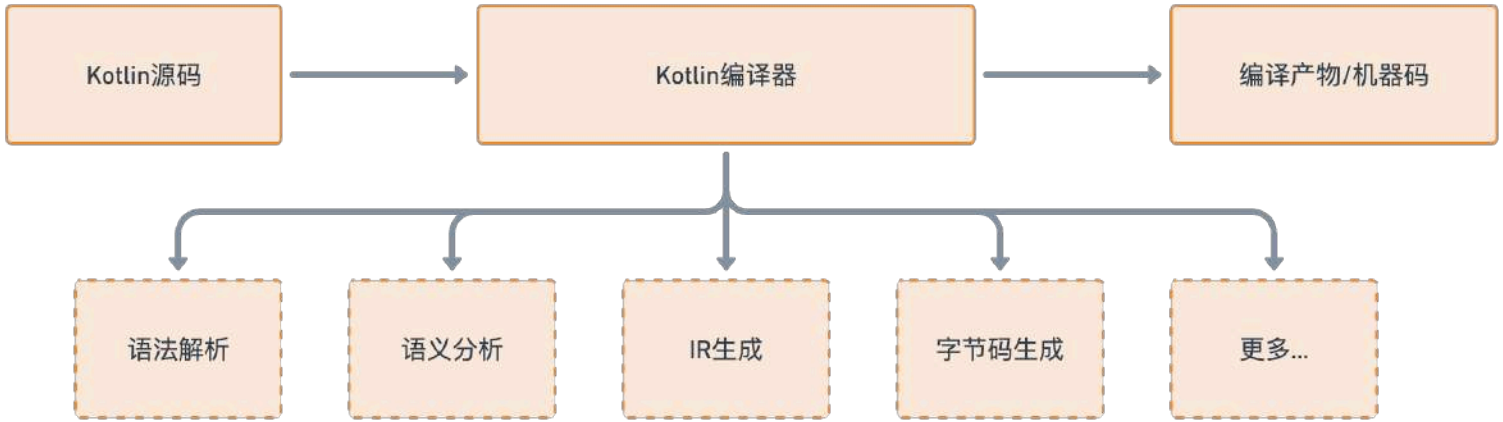
KUDOS 使用

- 添加插件到 classpath
- 应用插件
- 添加注解



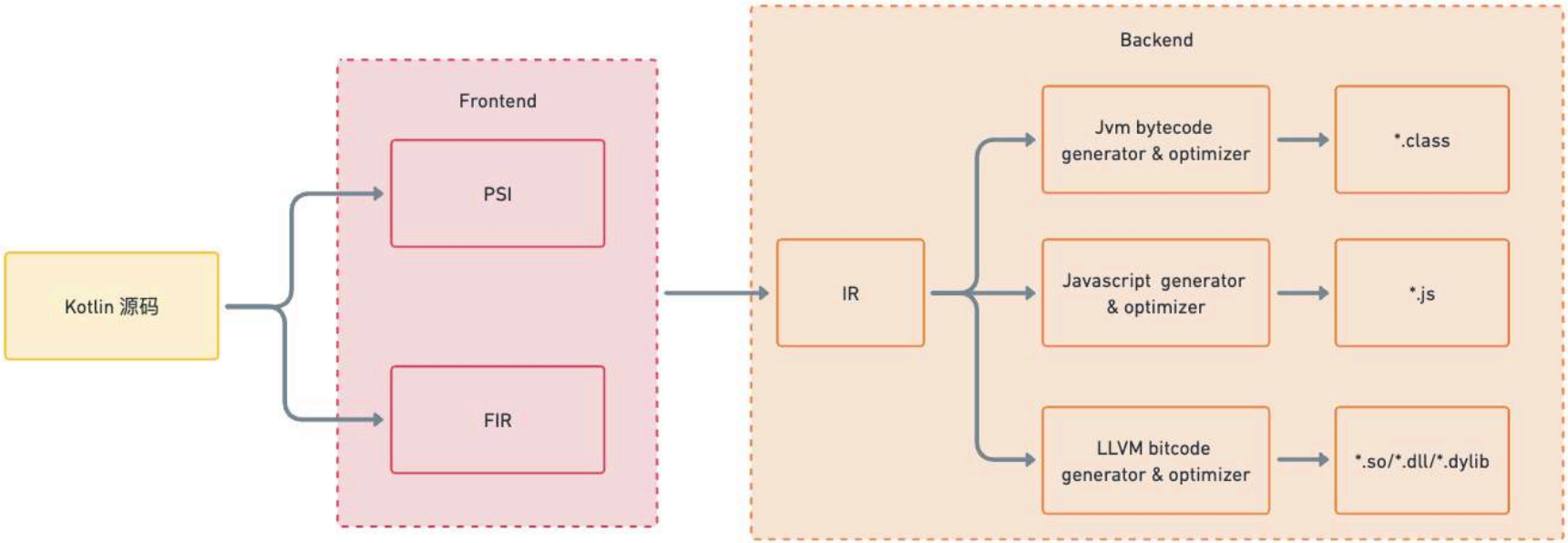
KUDOS原理解析

前置知识: Kotlin 编译过程



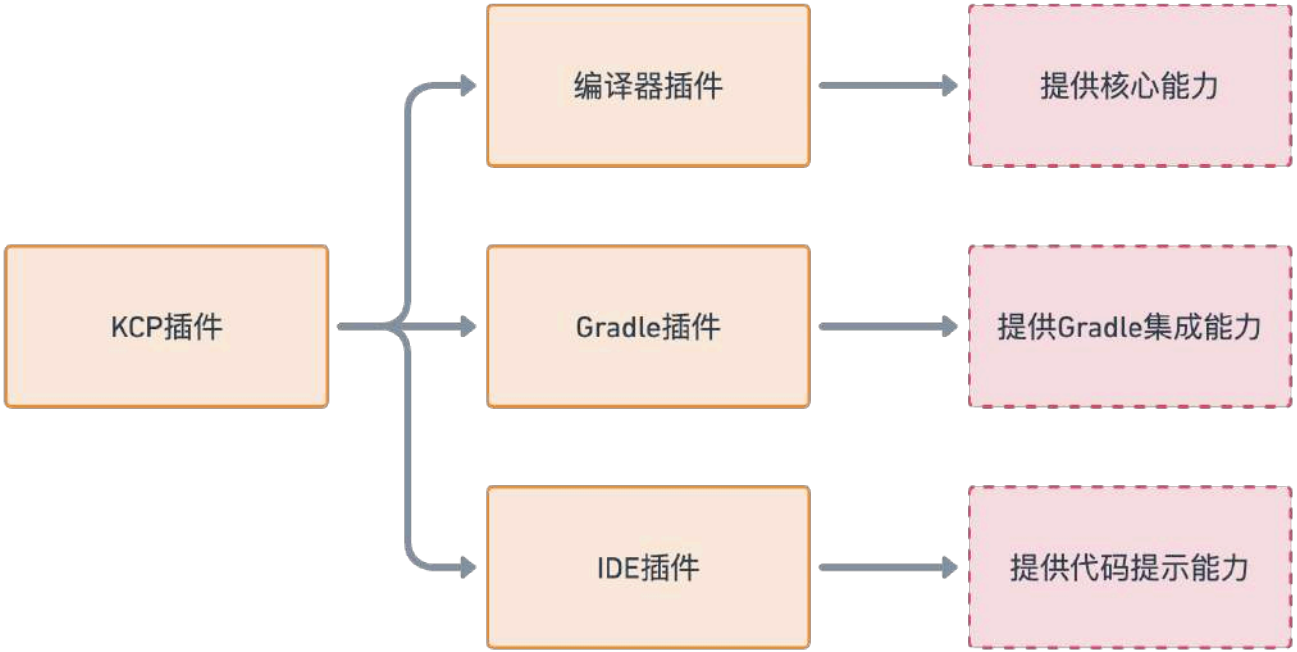
KUDOS原理解析

前置知识：K2 编译器与 K1 编译器



KUDOS原理解析

前置知识：Kotlin 编译器插件



KUDOS原理解析

前置知识：编译器插件扩展点

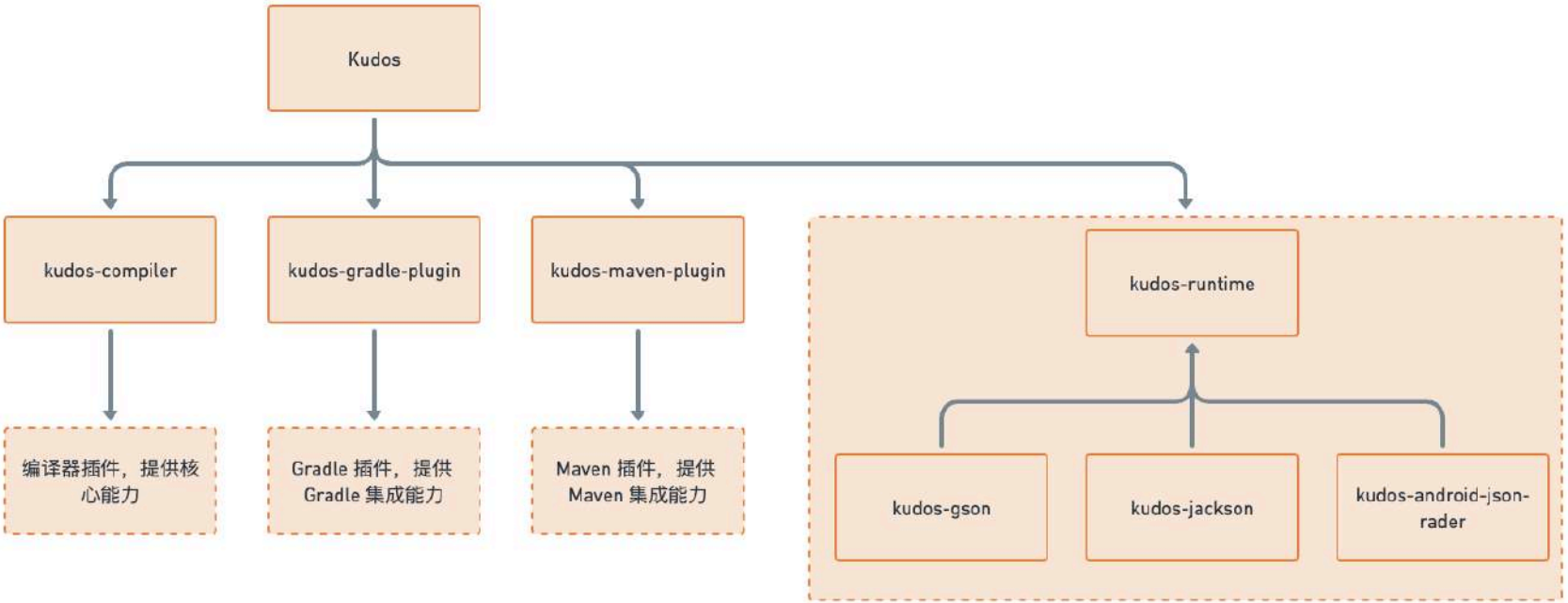
扩展的类名	编译阶段	功能说明	用例
SyntheticResolveExtension	前端	解析生成的类，函数等	Parcelize
FirExtensionRegistrar	前端	FIR 扩展，可用于提供代码声明信息与代码检查	Parcelize
StorageComponentContainerContributor	前端	可用于编译期代码检查	Compose
IrGenerationExtension	多平台 IR 后端	生成与修改 IR	Parcelize、Atomicfu、NoArg

KUDOS原理解析

Kotlin 编译器插件可以做什么？

KUDOS 原理解析

项目结构



KUDOS 原理解析

如何保证空安全与默认值？

```kotlin
@Kudos
data class User(
    val id: Long,
    val name: String,
    val age: Int = -1,
    val tel: String = ""
)
```

```kotlin
@Kudos
// 如果启用了 com.kanyun.kudos.gson 插件，则生成 @JsonAdapter 注解
@JsonAdapter(value = KudosReflectiveTypeAdapterFactory::class)
data class User(
    val id: Long,
    val name: String,
    val age: Int = -1,
    val tel: String = ""
) : KudosValidator {
    constructor() { // 生成的默认无参构造器
        super() // 调用父类默认无参构造器
        init<User>() // 调用 User 类内部的 init 块（包括定义在内部的属性初始化）
        this.age = -1 // 使用主构造器的参数默认值初始化属性
        this.tel = "" // 使用主构造器的参数默认值初始化属性
    }

    // 生成的用于校验字段空安全的函数
    override fun validate(status: Map<String, Boolean>) {
        validateField("id", status)
        validateField("name", status)
    }
}
```

KUDOS 原理解析

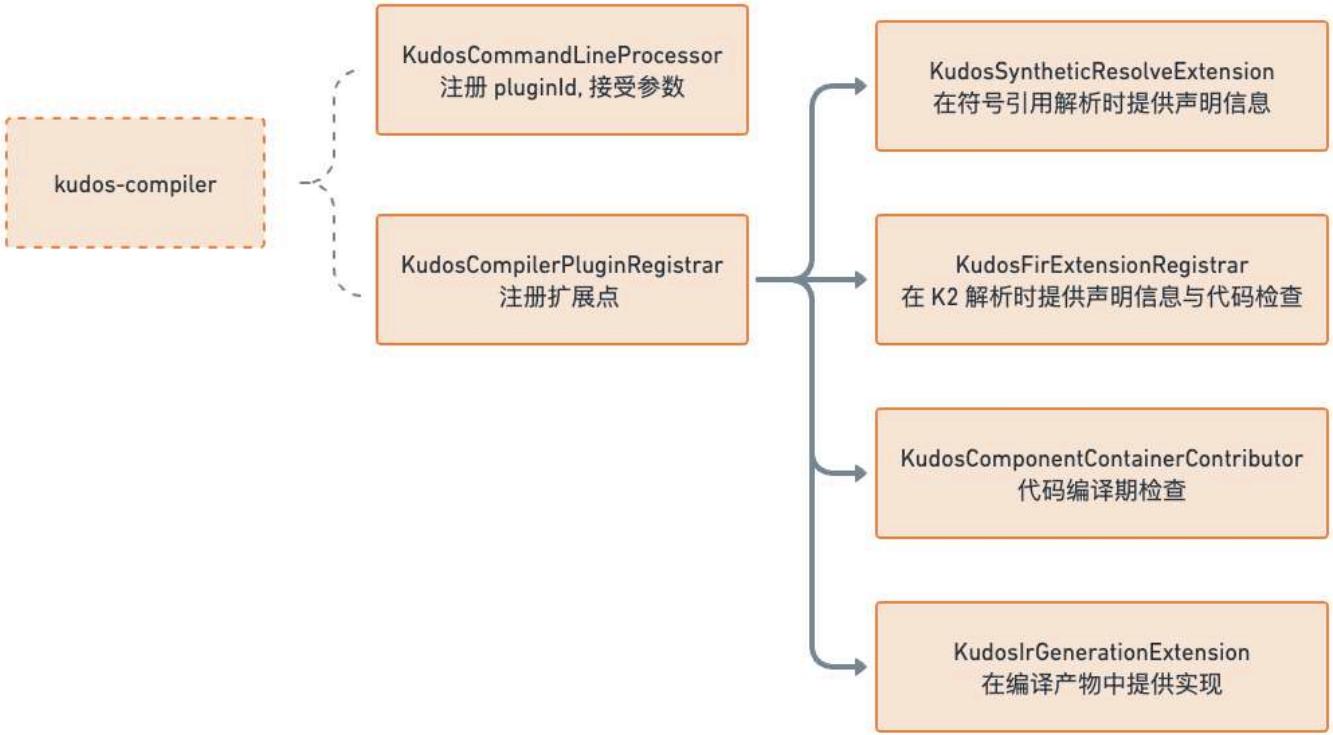
如何优化反序列化性能?

```
@Kudos
data class User(
    val id: Long,
    val name: String,
    val age: Int = -1,
    val tel: String = ""
)
```

```
@Kudos
data class User(
    val id: Long,
    val name: String,
    val age: Int = -1,
    val tel: String = ""
) : KudosJsonAdapter {
    private var kudosFieldStatusMap: Map<String, Boolean> = hashMapOf()

    override fun fromJson(jsonReader: JsonReader): User {
        jsonReader.beginObject()
        while (jsonReader.hasNext()) {
            val tmp0 = jsonReader.nextName()
            if (jsonReader.peek() == JsonToken.NULL) {
                jsonReader.skipValue()
                continue
            }
            when {
                tmp0 == "id" -> {
                    <this>.id = jsonReader.nextLong()
                    <this>.kudosFieldStatusMap.put("id", <this>.id != null)
                }
                tmp0 == "name" -> {
                    <this>.name = jsonReader.nextString()
                    <this>.kudosFieldStatusMap.put("name", <this>.name != null)
                }
                // ...
                else -> {
                    jsonReader.skipValue()
                }
            }
        }
        jsonReader.endObject()
        validate(<this>.kudosFieldStatusMap)
        return this
    }
}
```

KUDOS 原理解析

kudos-compiler 实现



KUDOS 原理解析

Benchmark

多次运行测试结果

	small json	medium json	large json
Gson	412,375 ns	1,374,838 ns	3,641,904 ns
Kudos-Gson	517,123 ns	1,686,568 ns	4,311,910 ns
Jackson	1,035,010 ns	1,750,709 ns	3,450,974 ns
Kudos-Jackson	1,261,026 ns	2,030,874 ns	3,939,600 ns
JsonReader	190,302 ns	1,176,479 ns	3,464,174 ns
Kudos-JsonReader	215,974 ns	1,359,587 ns	4,019,024 ns

一次运行测试结果

	small json	medium json	large json
Gson	3,974,219 ns	4,666,927 ns	8,271,355 ns
Kudos-Gson	4,531,718 ns	6,244,479 ns	11,160,782 ns
Jackson	12,821,094 ns	13,930,625 ns	15,989,791 ns
Kudos-Jackson	13,233,750 ns	15,674,010 ns	18,641,302 ns
JsonReader	662,032 ns	2,056,666 ns	4,624,687 ns
Kudos-JsonReader	734,907 ns	2,362,010 ns	6,212,917 ns

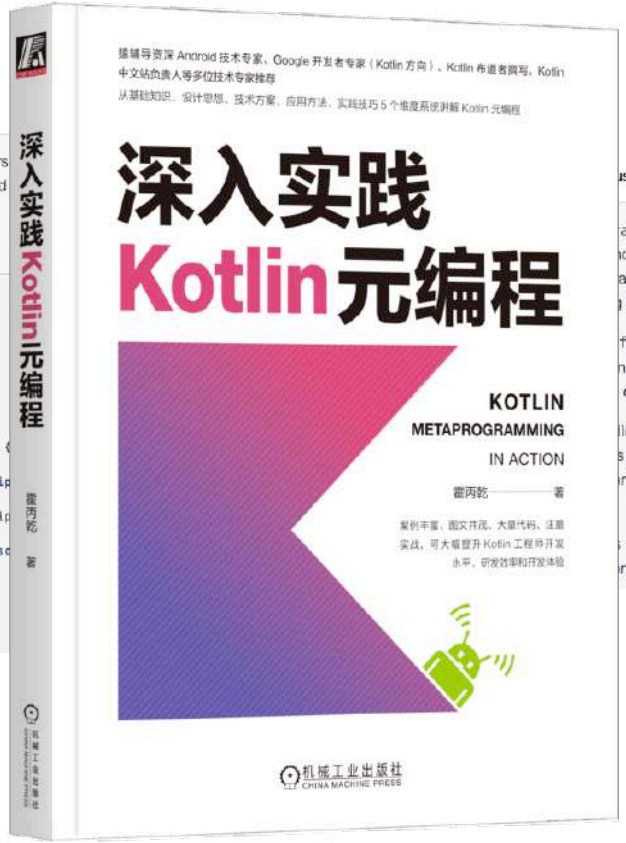
如何学习 KCP？

我 Give me an example of using SyntheticResolveExtension

SyntheticResolveExtension is a plugin provided by Kotlin compilers properties, which are element functions or properties that can be added synthetic declarations inside compiler plugins.

Here is an example:

```
open class MyBasePluginCliOptionProcessor :
    CommandLineProcessor {
    ....

    override val pluginId: String = "demo.plugin"
    override val pluginOptions: Collection<CliOption> = listOf()

}

val SyntheticResolveExtension = object : SyntheticResolveExtension {

    override fun addSyntheticSupertypes(thisDescriptor: ClassDescrip

    override fun getSyntheticCompanionObjectNameIfNeeded(thisDescrip

    override fun getSyntheticFunctionNames(thisDescriptor: ClassDesc
        if (thisDescriptor.name.asString() == "MyClass") {
            return listOf(Name.identifier("mySyntheticFunction"))
        }
        return emptyList()
    }
```



using FirDeclarationGenerationExtension

ationExtension is even a lower level extension point and involves Transformer-based
d requires understanding of the Fir tree, scoping, and resolving. The extension point is
ant to introduce new language constructs or change the behavior of existing language
new declarations at the frontend IR (FIR) phase.

find a simple example of using the FirDeclarationGenerationExtension in Kotlin
be used to modify the IR tree, add or remove declarations, and perform advanced
compile phase.

irectly working with Kotlin compiler, especially AST (Abstract Syntax Tree)
s a deep understanding of the Kotlin language, its compiler, and the IR system and it's
nced use cases like creating new language features or making experimental changes to

of present, this API is available internally at Kotlin compiler, and is subject to change. It
r public use, but to provide extension points for its internal usages.

加入北京 KUG

- 在这里可以和其它成员一起讨论 Kotlin 语言的各项技术与最新特性
- 我们还会不定期举行分享会，进行与 Kotlin 相关的主题分享与讨论
- 关注微信公众号“**程序员江同学**”，点击技术讨论，加入技术交流群

Ricardo

中国大陆

扫一扫上面的二维码图案，加我为朋友。

Thanks!
Have a nice Kotlin