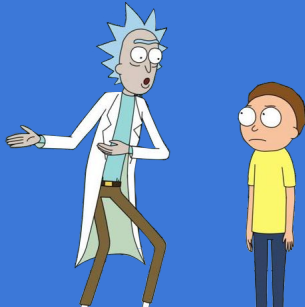


Ktor

Ruslan Ibragimov / ibragimov.by





Ktor



VS

Конкуренты



VS



Конкуренты



Можно использовать с Kotlin





Можно использовать с Kotlin





Асинхронные





Асинхронные





Корутины





Корутины



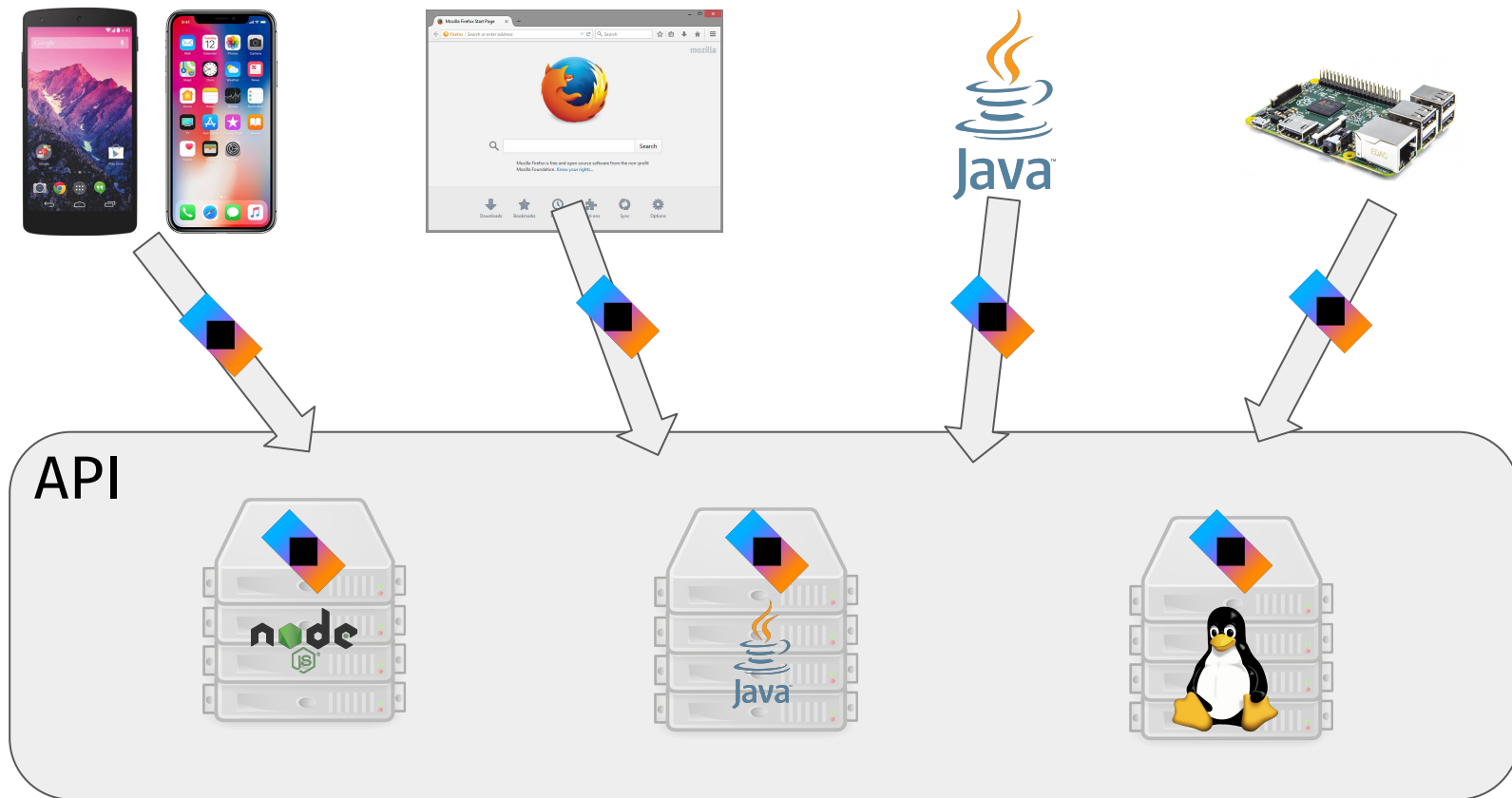


VS

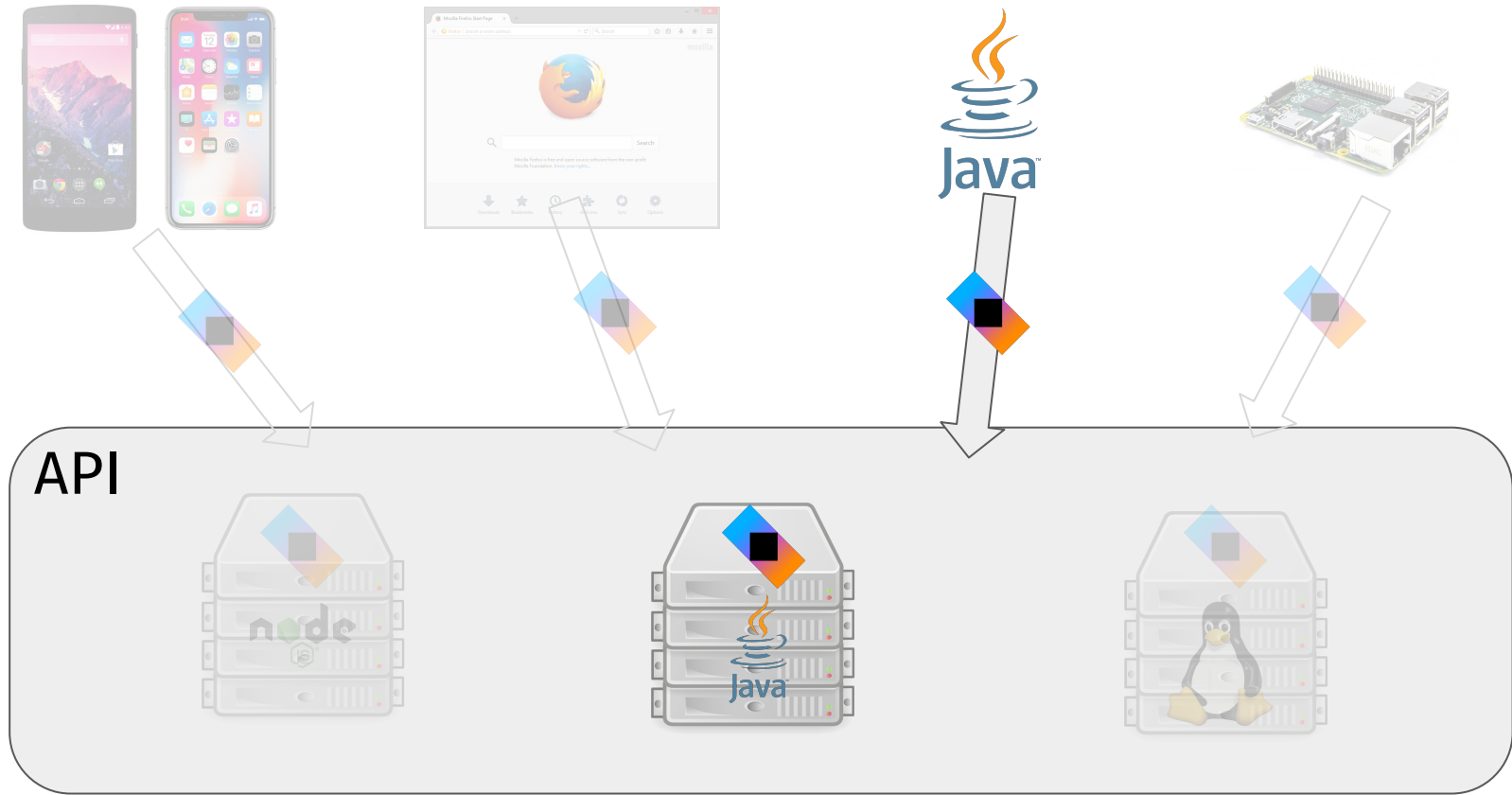


Конкуренты

One more thing...



Мультиплатформа



Java, Java, Java-Java Jing-Jing-Jing

Hello, World!



Gradle

```
buildscript {  
    repositories {  
        jcenter()  
    }  
  
    dependencies {  
        classpath("org.jetbrains.kotlin:gradle-plugin:$kotlinVersion")  
    }  
}
```


Gradle

```
apply plugin: "kotlin"
```

```
compileKotlin {  
    kotlinOptions.jvmTarget = "1.8"  
}
```

```
compileTestKotlin {  
    kotlinOptions.jvmTarget = "1.8"  
}
```

```
kotlin.experimental.coroutines = "enable"
```

Gradle

```
repositories {  
    jcenter()  
    maven { url "https://dl.bintray.com/kotlin/kotlinx" }  
    maven { url "https://dl.bintray.com/kotlin/ktor" }  
}  
  
dependencies {  
    compile("org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlinVersion")  
    compile("io.ktor:ktor-server-netty:$ktorVersion")  
}
```

Main

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```



Main

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Main

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Main

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Main

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Main

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
->                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```


Application::class

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) { this: Application  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Application::class

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) { this: Application  
        this.routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

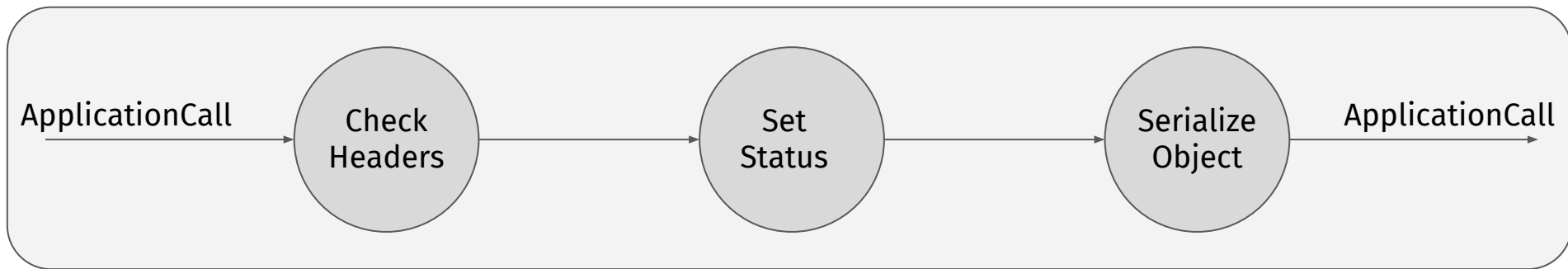
Application::class

Application это Pipeline (точнее несколько Pipeline'ов)

Application::class

Application это Pipeline (точнее несколько Pipeline'ов)

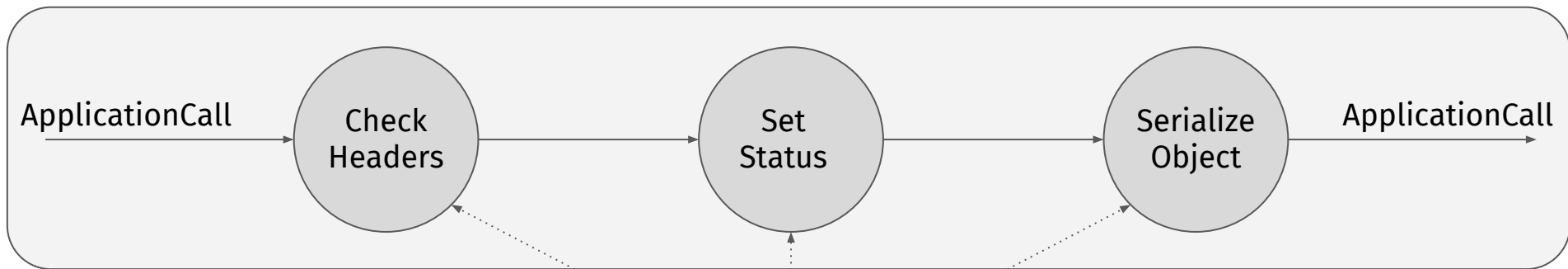
Pipeline



Application::class

Application это Pipeline (точнее несколько Pipeline'ов)

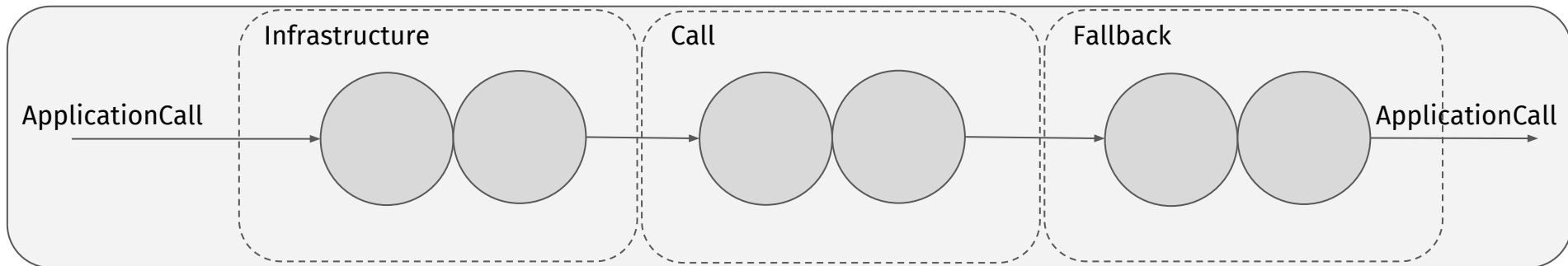
Pipeline



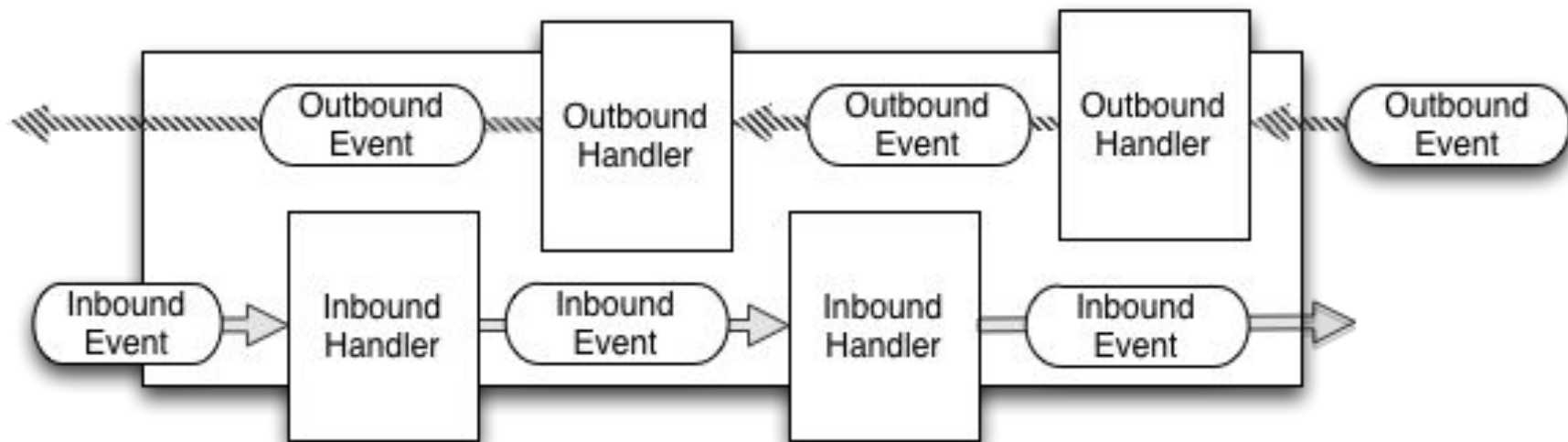
Interceptors

PipelinePhase

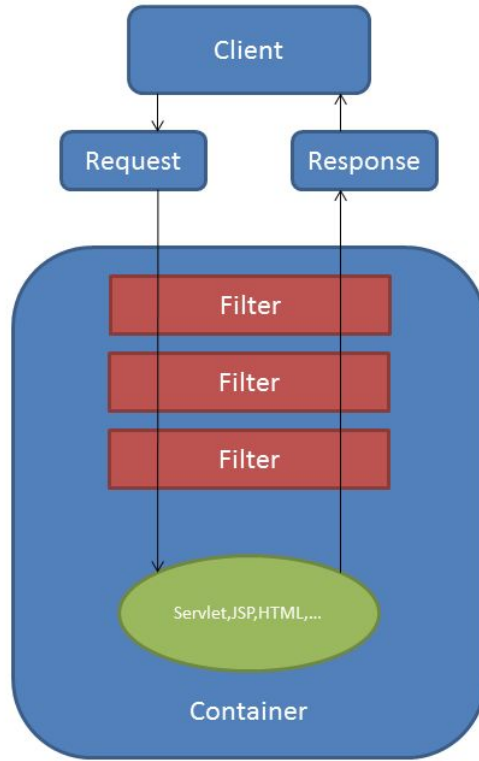
Application : ApplicationCallPipeline



Netty ChannelPipeline



Servlet



Pipelines

ApplicationCallPipeline

- ApplicationReceivePipeline
- ApplicationSendPipeline

Pipelines

ApplicationCallPipeline (ApplicationCall)

- ApplicationReceivePipeline (ApplicationReceiveRequest, ApplicationCall, IncomingContent)
- ApplicationSendPipeline (ApplicationCall, OutgoingContent)

Application::class

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) { this: Application  
        this.routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Application.intercept

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        intercept(ApplicationCallPipeline.Infrastructure) {  
            // log request headers  
            call.request.headers  
                .forEach { name, values -> println("$name: ${values.joinToString()}") }  
        }  
  
        //...  
    }.start(wait = true)  
}
```

Application.intercept

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        intercept(ApplicationCallPipeline.Infrastructure) {  
            // log request headers  
            call.request.headers  
                .forEach { name, values -> println("$name: ${values.joinToString()}") }  
        }  
  
        //...  
    }.start(wait = true)  
}
```

Application.intercept

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        intercept(ApplicationCallPipeline.Infrastructure) {  
            // log request headers  
            call.request.headers  
                .forEach { name, values -> println("$name: ${values.joinToString())"} }  
        }  
  
        //...  
    }.start(wait = true)  
}
```

ApplicationCall::class

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        intercept(ApplicationCallPipeline.Infrastructure) {  
            // log request headers  
            call.request.headers  
                .forEach { name, values -> println("$name: ${values.joinToString())"} }  
        }  
  
        //...  
    }.start(wait = true)  
}
```

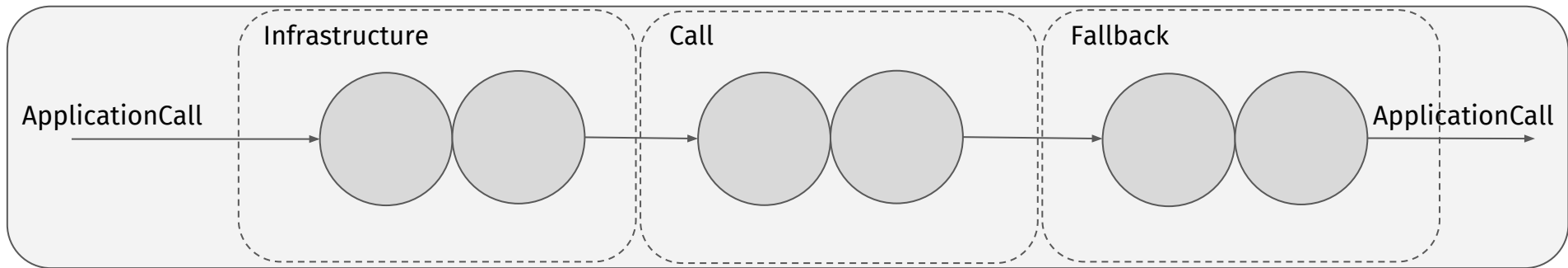
ApplicationCall::class

ApplicationCall

- ApplicationRequest
- ApplicationResponse
- Attributes

Application

ApplicationCallPipeline



Feature

- Роутинг
- Аутентификация
- Логирование запросов
- Проставление заголовков
- CORS
- Метрики
- Сессии
- и т.д.
- см. ApplicationFeature

Feature

```
routing {  
  get("/") {  
    call.respondText("I am Groot!", ContentType.Text.Html)  
  }  
}
```

Feature

```
install(Routing) {  
  get("/") {  
    call.respondText("I am Groot!", ContentType.Text.Html)  
  }  
}
```

Feature

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        install(DefaultHeaders)  
        install(CallLogging)  
  
        //..  
    }.start(wait = true)  
}
```

Features

```
intercept(ApplicationCallPipeline.Infrastructure) {  
  // log request headers  
  call.request.headers  
    .foreach { name, values -> println("$name: ${values.joinToString()}") }  
}
```

Features

```
class HeaderLoggingFeature(configuration: Configuration) {
    val exclusions = configuration.exclusions

    class Configuration {
        var exclusions: List<String> = listOf()
    }

    fun log(call: ApplicationCall) {
        call.request.headers
            .filter { name, _ -> !exclusions.contains(name) }
            .forEach { name, values -> println("$name: ${values.joinToString()}" ) }
    }

    companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")

        override fun install(pipeline: ApplicationCallPipeline, configure: Configuration.() -> Unit): HeaderLoggingFeature {
            val configuration = HeaderLoggingFeature.Configuration().apply(configure)
            val feature = HeaderLoggingFeature(configuration)

            pipeline.intercept(ApplicationCallPipeline.Infrastructure) {
                feature.log(call)
            }
            return feature
        }
    }
}
```

Features

```
class HeaderLoggingFeature(configuration: Configuration) {  
    val exclusions = configuration.exclusions  
  
    class Configuration {  
        var exclusions: List<String> = listOf()  
    }  
  
    fun log(call: ApplicationCall) {  
        call.request.headers  
            .filter { name, _ -> !exclusions.contains(name) }  
            .forEach { name, values -> println("$name: ${values.joinToString()}" ) }  
    }  
}
```

companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration, HeaderLoggingFeature> {

override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")

override fun install(**pipeline**: ApplicationCallPipeline, **configure**: Configuration.() -> Unit): HeaderLoggingFeature {

val configuration = HeaderLoggingFeature.Configuration().apply(**configure**)

val feature = HeaderLoggingFeature(**configuration**)

pipeline.intercept(ApplicationCallPipeline.**Infrastructure**) {

feature.log(**call**)

}

return feature

}

}

Features

```
class HeaderLoggingFeature(configuration: Configuration) {  
    val exclusions = configuration.exclusions  
  
    class Configuration {  
        var exclusions: List<String> = listOf()  
    }  
  
    fun log(call: ApplicationCall) {  
        call.request.headers  
            .filter { name, _ -> !exclusions.contains(name) }  
            .forEach { name, values -> println("$name: ${values.joinToString()}" ) }  
    }  
}
```

companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration, HeaderLoggingFeature> {

override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")

override fun install(*pipeline*: ApplicationCallPipeline, *configure*: Configuration.() -> Unit): HeaderLoggingFeature {

val configuration = HeaderLoggingFeature.Configuration().*apply*(*configure*)

val feature = HeaderLoggingFeature(*configuration*)

pipeline.intercept(ApplicationCallPipeline.Infrastructure) {

feature.log(*call*)

 }

return feature

 }

}

Features

```
class HeaderLoggingFeature(configuration: Configuration) {  
    val exclusions = configuration.exclusions
```

```
    class Configuration {  
        var exclusions: List<String> = listOf()  
    }
```

```
    fun log(call: ApplicationCall) {  
        call.request.headers  
            .filter { name, _ -> !exclusions.contains(name) }  
            .forEach { name, values -> println("$name: ${values.joinToString()}" ) }  
    }
```

```
    companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration,  
HeaderLoggingFeature> {
```

```
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
```

```
        override fun install(pipeline: ApplicationCallPipeline, configure: Configuration.() -> Unit): HeaderLoggingFeature {
```

```
            val configuration = HeaderLoggingFeature.Configuration().apply(configure)
```

```
            val feature = HeaderLoggingFeature(configuration)
```

```
            pipeline.intercept(ApplicationCallPipeline.Infrastructure) {
```

```
                feature.log(call)
```

```
            }
```

```
            return feature
```

```
        }
```

```
    }
```

```
}
```

Features

```
class HeaderLoggingFeature(configuration: Configuration) {
    val exclusions = configuration.exclusions

    class Configuration {
        var exclusions: List<String> = listOf()
    }

    fun log(call: ApplicationCall) {
        call.request.headers
            .filter { name, _ -> !exclusions.contains(name) }
            .forEach { name, values -> println("$name: ${values.joinToString()}" ) }
    }

    companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")

        override fun install(pipeline: ApplicationCallPipeline, configure: Configuration.() -> Unit): HeaderLoggingFeature {
            val configuration = HeaderLoggingFeature.Configuration().apply(configure)
            val feature = HeaderLoggingFeature(configuration)

            pipeline.intercept(ApplicationCallPipeline.Infrastructure) {
                feature.log(call)
            }
            return feature
        }
    }
}
```

Features

```
class HeaderLoggingFeature(configuration: Configuration) {
    val exclusions = configuration.exclusions

    class Configuration {
        var exclusions: List<String> = listOf()
    }

    fun log(call: ApplicationCall) {
        call.request.headers
            .filter { name, _ -> !exclusions.contains(name) }
            .forEach { name, values -> println("$name: ${values.joinToString()}" ) }
    }

    companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration, HeaderLoggingFeature> {
        override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")

        override fun install(pipeline: ApplicationCallPipeline, configure: Configuration.() -> Unit): HeaderLoggingFeature {
            val configuration = HeaderLoggingFeature.Configuration().apply(configure)
            val feature = HeaderLoggingFeature(configuration)

            pipeline.intercept(ApplicationCallPipeline.Infrastructure) {
                feature.log(call)
            }
            return feature
        }
    }
}
```

Features

```
class HeaderLoggingFeature(configuration: Configuration) {  
    val exclusions = configuration.exclusions
```

```
    class Configuration {  
        var exclusions: List<String> = listOf()  
    }  
}
```

```
fun log(call: ApplicationCall) {  
    call.request.headers  
        .filter { name, _ -> !exclusions.contains(name) }  
        .forEach { name, values -> println("$name: ${values.joinToString()}") }  
}
```

```
companion object Feature : ApplicationFeature<ApplicationCallPipeline, HeaderLoggingFeature.Configuration, HeaderLoggingFeature> {  
    override val key = AttributeKey<HeaderLoggingFeature>("HeaderLoggingFeature")
```

```
    override fun install(pipeline: ApplicationCallPipeline, configure: Configuration.() -> Unit): HeaderLoggingFeature {  
        val configuration = HeaderLoggingFeature.Configuration().apply(configure)  
        val feature = HeaderLoggingFeature(configuration)
```

```
        pipeline.intercept(ApplicationCallPipeline.Infrastructure) {  
            feature.log(call)  
        }  
        return feature
```

```
    }  
}
```

Features

```
intercept(ApplicationCallPipeline.Infrastructure) {  
  // log request headers  
  call.request.headers  
    .forEach { name, values -> println("$name: ${values.joinToString()}") }  
}
```

```
install(HeaderLoggingFeature)
```

```
install(HeaderLoggingFeature) {  
  exclusions = listOf("User-Agent")  
}
```

Features

```
intercept(ApplicationCallPipeline.Infrastructure) {  
  // log request headers  
  call.request.headers  
    .forEach { name, values -> println("$name: ${values.joinToString()}") }  
}
```

```
install(HeaderLoggingFeature)
```

```
install(HeaderLoggingFeature) {  
  exclusions = listOf("User-Agent")  
}
```

Features

```
intercept(ApplicationCallPipeline.Infrastructure) {  
  // log request headers  
  call.request.headers  
    .forEach { name, values -> println("$name: ${values.joinToString()}") }  
}
```

```
install(HeaderLoggingFeature)
```

```
install(HeaderLoggingFeature) {  
  exclusions = listOf("User-Agent")  
}
```


Application и организация кода

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) {  
        routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Application и организация кода

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8080) { this: Application  
        this.routing {  
            get("/") {  
                call.respondText("I am Groot!", ContentType.Text.Html)  
            }  
        }  
    }.start(wait = true)  
}
```

Application и организация кода

```
fun Application.myApp() {  
    routing {  
        get("/") {  
            call.respondText("I am Groot!", ContentType.Text.Html)  
        }  
    }  
}
```

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        myApp()  
    }.start(wait = true)  
}
```

Application и организация кода

```
fun Application.myApp() {  
    routing {  
        get("/") {  
            call.respondText("I am Groot!", ContentType.Text.Html)  
        }  
    }  
}
```

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        myApp()  
    }.start(wait = true)  
}
```

Application и организация кода

```
fun Application.myApp() {  
    routing {  
        get("/") {  
            call.respondText("I am Groot!", ContentType.Text.Html)  
        }  
    }  
}
```

```
fun main(args: Array<String>) {  
    embeddedServer(Netty, 8081) {  
        myApp()  
    }.start(wait = true)  
}
```

Тестирование

```
dependencies {  
    compile("org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlinVersion")  
    compile("io.ktor:ktor-server-netty:$ktorVersion")  
  
    compile("ch.qos.logback:logback-classic:1.2.1")  
  
    testCompile("junit:junit:4.12")  
    testCompile("io.ktor:ktor-server-test-host:$ktorVersion")  
}
```

Тестирование

```
class AppKtTest {  
    @Test fun testIsIAmGroot() {  
        withTestApplication(Application::myApp) {  
            with(handleRequest(HttpMethod.Get, "/")) {  
                assertEquals(HttpStatusCode.OK, response.status())  
                assertEquals("I am Groot!", response.content)  
            }  
        }  
    }  
}
```

Тестирование

```
class AppKtTest {  
    @Test fun testIsIAmGroot() {  
        withTestApplication(Application::myApp) {  
            with(handleRequest(HttpMethod.Get, "/")) {  
                assertEquals(HttpStatusCode.OK, response.status())  
                assertEquals("I am Groot!", response.content)  
            }  
        }  
    }  
}
```


Тестирование

```
class AppKtTest {  
    @Test fun testIsIAmGroot() {  
        withTestApplication(Application::myApp) {  
            with(handleRequest(HttpMethod.Get, "/")) {  
                assertEquals(HttpStatusCode.OK, response.status())  
                assertEquals("I am Groot!", response.content)  
            }  
        }  
    }  
}
```

Тестирование

```
class AppKtTest {  
    @Test fun testIsIAmGroot() {  
        withTestApplication(Application::myApp) {  
            with(handleRequest(HttpMethod.Get, "/")) {  
                assertEquals(HttpStatusCode.OK, response.status())  
                assertEquals("I am Groot!", response.content)  
            }  
        }  
    }  
}
```

Autoreload

```
fun Application.myApp() {  
    routing {  
        get("/") {  
            call.respondText("I am Groot!", ContentType.Text.Html)  
        }  
    }  
}
```



Autoreload

```
ktor {  
    deployment {  
        port = 8080  
        watch = [ ktor-bkug ]  
    }  
  
    application {  
        modules = [ by.bkug.autoreload.AutoreloadKt.module ]  
    }  
}
```

Autoreload

`io.ktor.server.netty.DevelopmentEngine`

Autoreload

ApplicationEngineEnvironmentReloading

Autoreload

Demo?

Серверы

Netty (**ktor-server-netty**),

Jetty (ktor-server-jetty),

Tomcat (ktor-server-tomcat)

Servlet 3.0+ (ktor-server-servlet)

HTTP Клиент

Клиенты

Apache HTTP (**ktor-client-apache**),

Jetty (ktor-client-jetty)

Пример

```
fun Application.myApp() {  
    val client = HttpClient(Apache)  
}
```

Пример

```
fun Application.myApp() {  
    val client = HttpClient(Apache)  
  
    routing {  
        get("/call") {  
            val text = client.get<String>(  
                host = "localhost",  
                port = 8081,  
                path = "/text"  
            )  
  
            call.respondText(text)  
        }  
    }  
}
```

Пример

```
fun Application.myApp() {  
    val client = HttpClient(Apache)  
  
    routing {  
        get("/call") {  
->            val text = client.get<String>(   
                host = "localhost",  
                port = 8081,  
                path = "/text"  
            )  
  
->            call.respondText(text)  
        }  
    }  
}
```

Пример

```
val client = HttpClient(Apache) {  
    install(JsonFeature)  
}
```

Пример

```
val user = client.get<User>(
    host = "localhost",
    port = 8081,
    path = "/json"
)
```

Пример

```
val user = client.get<User>(  
    host = "localhost",  
    port = 8081,  
    path = "/json"  
)
```


Про что еще можно было бы рассказать

- Продвинутый роутинг
- JSON
- Статические данные (HTML, JS, ...)
- ExceptionHandling
- IoC (DI)
- HTTP/2
- WebSockets
- Как добавить \$server? (Undertow, ...)



Резюме

- Ktor - connected systems
- Application - Pipelines
- Pipeline - Interceptors
- Interceptors - Feature
- Авторелoad
- Тестирование
- Клиент

Вопросы?

- Ktor.io: ktor.io
- Awesome Kotlin: kotlin.link
- Belarus Kotlin User Group: bkug.by



COFFEE TIME

