Frameworks

Матвей Попов

План

- Что такое Frameworks и зачем нужны?
- Как собирать проект на Kotlin?
- Gradle
- Ktor
- Inversion of Control
- Livecoding

Frameworks

А что это еще вообще такое?

Framework - каркас, структура

Из чего он состоит?

- Большое количество библиотек/ модулей
- Набор шаблонов
- Определенные подходы к написанию функционала
- Определенные ограничения

Зачем он нужен?

- Увеличить скорость разработки
- Упростить работу
- Обеспечить безопасность
- Заставить использовать некоторый паттерн проектирования

С Frameworks вроде разобрались

Что дальше?

Зависимости

Что это и зачем?

- Мы хотим переиспользовать чужой код
- Мы хотим версионировать код
- Мы хотим управлять зависимостями

А что мы еще хотим?

- Автоматически собирать проект
- Писать свои скрипты автоматизации
- Иметь набор стандартизированных средств

Нам нужен ProjectManagmentTool!

Какие существуют?

Apache Ant

- Сценарий сборки - XML файл
- Наследник make
- Не зависима от платформы
- Создана в 2000 году
- Императивная сборка проекта

Как выглядит конфигурационный файл?

```xml
<?xml version="1.0"?>
<project default="build" basedir=".">
    <property name="name" value="AntBuildJar"/>
    <property name="src.dir" location="${basedir}/src"/>
    <property name="build" location="${basedir}/build"/>
    <property name="build.classes" location="${build}/classes"/>
    <path id="libs.dir">
    <fileset dir="lib" includes="**/*.jar"/>
    </path>
    <!-- Сборка приложения -->
    <target name="build" depends="clean" description="Builds the application">
        <!-- Создание каталогов -->
        <mkdir dir="${build.classes}"/>

        <!-- Компиляция исходных файлов -->
        <javac srcdir="${src.dir}"
               destdir="${build.classes}"
               debug="false"
               deprecation="true"
               optimize="true" >
            <classpath refid="libs.dir"/>
        </javac>

        <!-- Копирование необходимых файлов -->
        <copy todir="${build.classes}">
            <fileset dir="${src.dir}" includes="**/*.*" excludes="**/*.java"/>
        </copy>

        <!-- Создание JAR-файла -->
        <jar jarfile="${build}/${name}.jar">
            <fileset dir="${build.classes}"/>
        </jar>
    </target>

    <!-- Очистка -->
    <target name="clean" description="Removes all temporary files">
        <!-- Удаление файлов -->
        <delete dir="${build.classes}"/>
    </target>
</project>
```

Преимущества и недостатки

- Разработчики должны писать все команды
- Файлы часто получались очень большими
- Нет встроенной поддержки управления зависимостей
- Не было единого формата

Apache Maven

- Выпущен в 2008
- Использует свой формат POM
- Декларативный подход к сборке проекта
- Архитектура на основе плагинов

Как выглядит конфигурационный файл?

```xml
<project>
  <!-- версия модели для POM-ов Maven 2.x всегда 4.0.0 -->
  <modelVersion>4.0.0</modelVersion>

  <!-- координаты проекта, то есть набор значений, который
       позволяет однозначно идентифицировать этот проект -->

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- зависимости от библиотек -->

  <dependencies>
    <dependency>

      <!-- координаты необходимой библиотеки -->

      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- эта библиотека используется только для запуска и компилирования тестов -->

      <scope>test</scope>

    </dependency>
  </dependencies>
</project>
```

Преимущество и недостатки

- Позволяет сосредоточиться на том, что должна делать наша сборка
- Встроенная поддержка управления зависимостей
- Maven предписывает строгую структуру проекта
- Огромное количество плагинов

- Файлы имеют тенденцию становиться огромными
- Намного менее гибок чем Ant

Gradle

Was built upon the concepts of Ant and Maven

- Первый выпуск в 2010
- Основан на принципах Ant и Maven
- Имеет свой DSL
- Использует ациклический направленный граф для выполнения задач
- Поддерживает каскадную модель разработки

Как выглядит файл?

lib/build.gradle

GROOVY

```groovy
plugins {
    id 'java-library' (1)
}

repositories {
    mavenCentral() (2)
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.9.3' (3)

    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'

    api 'org.apache.commons:commons-math3:3.6.1' (4)

    implementation 'com.google.guava:guava:32.1.1-jre' (5)
}

tasks.named('test') {
    useJUnitPlatform() (6)
}
```

lib/build.gradle.kts

```
plugins {
    `java-library` (1)
}

repositories {
    mavenCentral() (2)
}

dependencies {
    testImplementation("org.junit.jupiter:junit-jupiter:5.9.3") (3)

    testRuntimeOnly("org.junit.platform:junit-platform-launcher")

    api("org.apache.commons:commons-math3:3.6.1") (4)

    implementation("com.google.guava:guava:32.1.1-jre") (5)
}

tasks.named<Test>("test") {
    useJUnitPlatform() (6)
}
```

Преимущества и недостатки

- Состоит из плагинов
- Поддерживает множество языков
- Декларативное описание задач
- Поддержка “горячей” сборки
- Скорее всего взять Maven будет проще

- Хорошо подходит для сложных кодовых баз
- Требует специальных знаний для правильного использования

Что мы будем использовать?

Конечно Gradle!

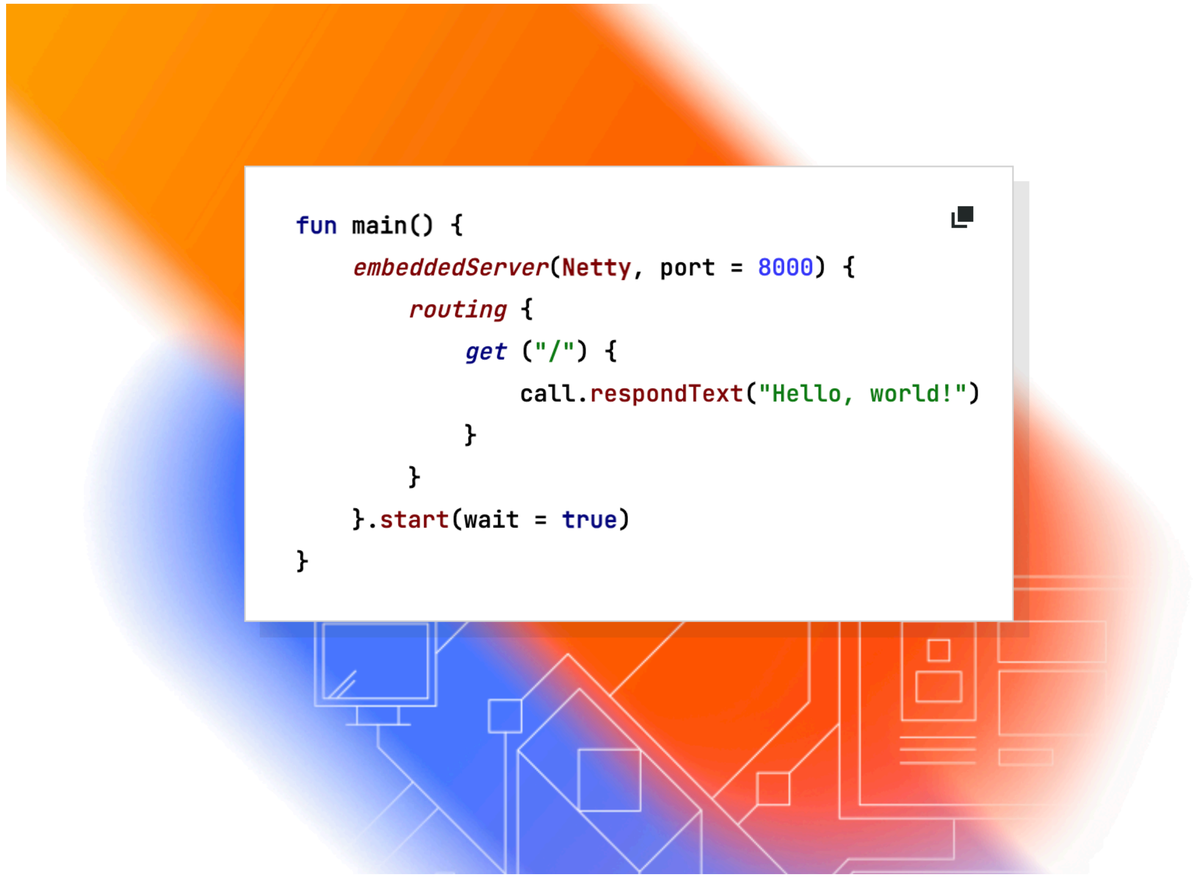
Почему?

- Чтобы писать еще больше на Kotlin
- Смотреть на Kotlin приятнее чем на XML
- В новых проектах заметно чаще используется Gradle

Какой framework мы будем использовать?

Ktor

Что это?



Simple and fun

Create asynchronous client and server applications. Anything from microservices to multiplatform HTTP client apps in a simple way. Open Source, free, and fun!

Create Learn

Latest release: 2.3.5

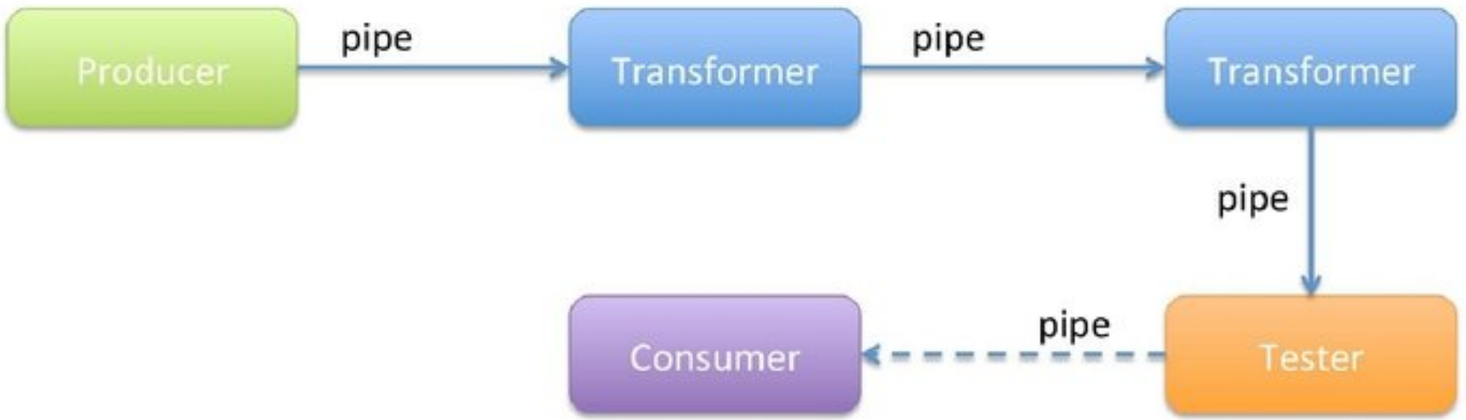
- Написан на Kotlin для Kotlin
- Поддержаны Coroutines на уровне платформы
- Мультиплатформенный
- Легковесный
- Расширяемый
- Спроектирован с помощью pipeline архитектуры
- Сделан в JetBrains

Архитектура

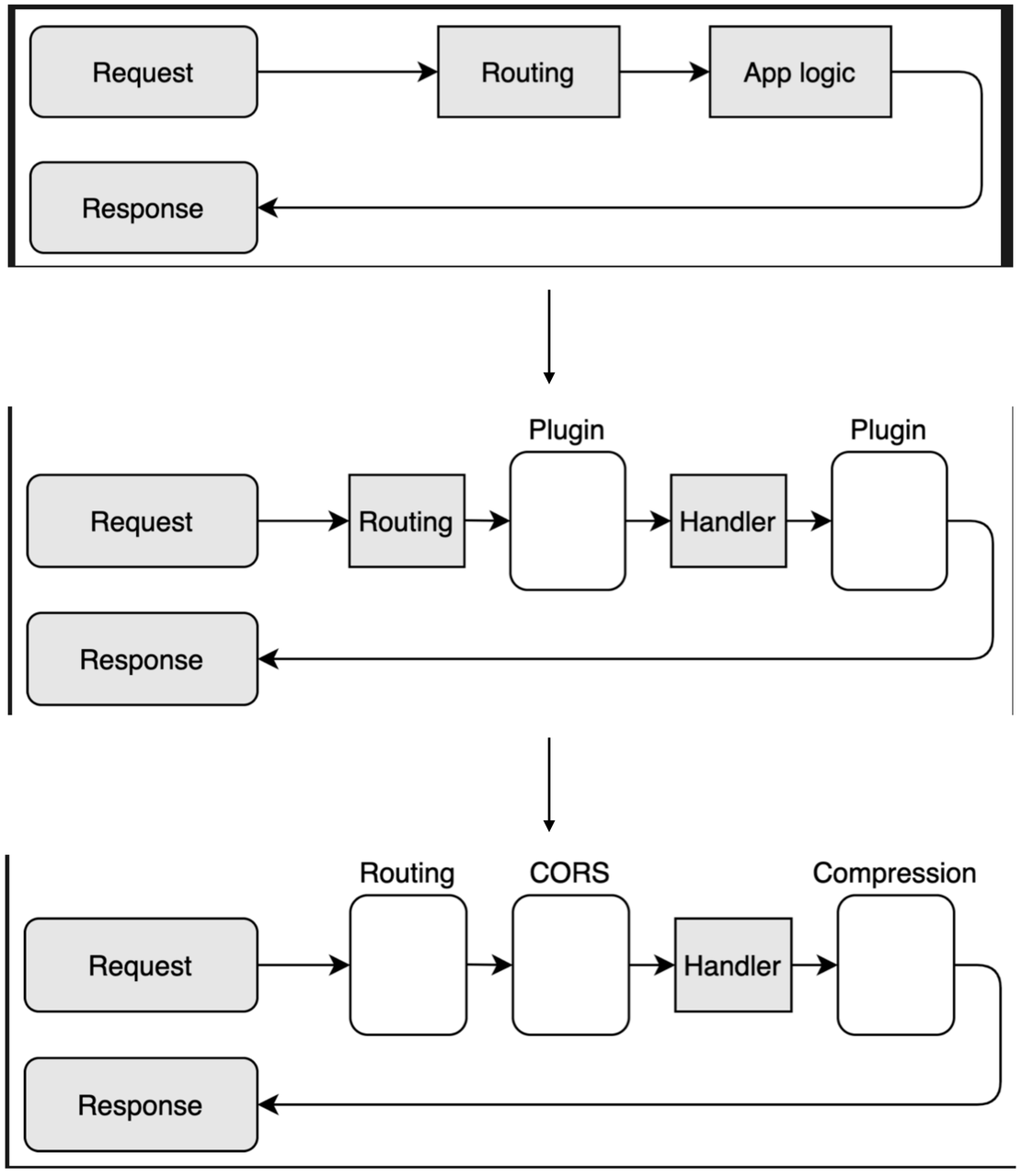
Pipeline architecture

Паттерн проектирования, основанный на множестве компонент, которые фильтруют или обрабатывают информацию и передают их дальше внутри некоторого pipe.

Pipeline architecture



Как это выглядит в контексте Ktor?


Request
Routing
App logic
Response
Plugin
Plugin
Request
Routing
Handler
Response
Routing
CORS
Compression
Request
Handler
Response

Как им пользоваться?

1. Добавить необходимые зависимости в build.gradle.kts файл
2. Написать необходимый код
3. Запустить! 🥳

Всем спасибо за внимание, лекция закончена!

Что еще нам необходимо знать?

Снова про зависимости

Inversion of Control

Инверсия управления

Важный принцип объектно-ориентированного программирования, используемый для уменьшения связанности(зацепления) в программах.

А проще можно?

Inversion of Control

Также известен как внедрение зависимостей(DI). Это процесс, при котором объекты определяют свои зависимости (то есть другие объекты, с которыми они работают) только через аргументы конструктора, аргументы фабричного метода или свойства, которые устанавливаются на экземпляр объекта после его создания или возврата фабричным методом. Затем контейнер внедряет эти зависимости.

Другими словами:
Объекты настраиваются внешними объектами

Что мы будем использовать?


Koin
The pragmatic Kotlin & Kotlin Multiplatform
Dependency Injection framework

Почему Koin?

- Очень простой
- Очень легковесный
- Внедрение зависимостей с помощью DSL
- Интегрированная поддержка с Ktor

Посмотрим как использовать этот ваш Ktor…

Livecoding: Пишем свой простой сервер

Выводы

Модно
Стильно
Молодежно

Домашнее задание

Начинаем делать проект!

Можно предложить свой сервис

Материалы

- Сайт Ktor: https://ktor.io/
- Простой курс по Ktor: https://clck.ru/369CcL
- Серия статей на Habr: https://clck.ru/369Cdv

Вопросы?

Всем спасибо за внимание!