KOTLIN 도입해볼까

11st Front dev. team
우여명

목차

- Kotlin 소개
- Java, Scala 와 Kotlin 비교
- Kotlin 코드 보기
- 코틀린의 대박 기능
- 그냥 코틀린으로 바꿔보자
- 예상되는 장단점
- 나의 결론

코틀린이란

- 정적 타입 언어
- JVM과 자바스크립트 대상
- 자바를 대체할 (공존할) 범용 언어
- Open Source Software
- 외부 기여자가 있지만,
  주로 JetBrains 담당팀에서 관리

왜 코틀린을 만들었나

- 자바는 현재 가장 많이 사용되고,
  충분히 좋은 언어이다.
- 그러나 일부 한계가 있고, 하위호환 이슈 때문에
  수정이 불가능한 문제가 많다.
- 레거시 문제에서 벗어나 개발자가 필요한 기능을
  가지고 있는 JVM 대상 정적 타입 언어가 필요하다.

코틀린 주요 설계 목적

- 자바와 호환되는 언어
- 적어도 자바만큼 빠르게 컴파일
- 자바보다 안전 (특히 null 문제)
- 변수 타입 추론, 고차 함수(클로저), 확장 함수, 믹스인, 일급 위임과 같은 걸 지원해서 자바보다 간결
- 스칼라와 같은 성숙한 경쟁 언어 보다 간결하지만 유용한 수준의 표현력

로컬에 코틀린 설치하기

- 인텔리 J
  - preferences > plugin > search “kotlin" > install
- 이클립스
  - help > install new software … >
    add “https://dl.bintray.com/jetbrains/kotlin/eclipse-plugin/last/” >
    install
- stand alone
  - download kotlin complier
  - $ brew update
    $ brew install kotlin

JAVA 와 KOTLIN

자바

- 체크드 익셉션
- 클래스가 아닌
  기본 타입 정적 멤버
- private이 아닌 필드
- 와일드카드-타입

코틀린

- 람다식 + 인라인 함수 = 훌륭한 커스텀 제어 구조 확장 함수
- null 안정성
- 스마트 타입 변환
- 문자열 템플릿
- 프로퍼티
- 주요 생성자
- 필드-클래스 위임
- 변수와 프로퍼티 타입을 위한 타입 추론 싱글톤
- Declaration-site variance & Type projections
- Range 식
- 연산자 오버로딩
- 컴페니언 오브젝트
- 데이터 클래스
- 읽기 전용 컬렉션과 변경 가능 콜렉션 인터페이스 분리

SCALA 와 KOTLIN

스칼라

- 암묵적 변환, 파라미터, etc 오버라이딩할 수 있는 타입 멤버
- 경로-의존 타입(Path-dependent types)
- 매크로
- Existential types
- 트레잇 초기화를 위한 복잡한 로직
- 커스텀 오퍼레이션 심볼
- XML 기본 지원
- Structural types
- 값 타입(Value types) Yield 연산자
- 액터
- 병렬 콜렉션

코틀린

- 오버헤드 없는 null-안정성
- 스마트 변환
  - 스칼라는 신택틱(syntactic)하고 런타임 래퍼인 Option을 갖는다.
- 코틀린의 인라인 함수는 non-로컬 점프를 쉽게할 수 있다
- 일급 위임. 또한 외부 플러그인인 Autoproxy로 구현된다.
- 멤버 참조 (자바 8에서도 지원한다).

코틀린 코드 보기

- Realm 코틀린 간단 소개
  - https://realm.io/kr/news/kotlin-1/
  - https://realm.io/kr/news/kotlin-2/
  - https://realm.io/kr/news/kotlin-3/
- 코틀린 온라인 튜토리얼
  - http://try.kotlinlang.org/#/Examples/Hello,%20world!/Simplest%20version/Simplest%20version.kt

대박 기능 1 - NULL SAFE

- The Billion Dollar Mistake

- 코틀린 타입 시스템에서 NPE가 발생하는 경우
  - 직접 throw NullPointerException()을 실행
  - !! 연산자 사용
  - 외부 자바 코드에서 발생
  - 초기화에 관한 데이터 불일치 존재
    (생성자에서 초기화하지 않은 this를 어딘가에서 사용)

대박 기능 1 - NULL SAFE

- nullable 레퍼런스와 non-null 레퍼런스
  - 코틀린에서 String은 non-null 레퍼런스,
    null이 들어가게 되면 **컴파일 에러**
  - 만약 String을 nullable 레퍼런스로 변경하고 싶다면,
    String? 이렇게 선언

```
var a : String = "abc"; //non-null
a = null; //컴파일 오류

var b : String? = "abc" //nullable
b = null;
```

대박 기능 1 - NULL SAFE

- 안전한 null 호출

```
b?.length
```

  - b가 null이 아니면 b.length를 리턴하고 아니면 null 리턴
  - 안전호출은 연속할 때 유용

```
bob?.department?.head?.name
```

- 엘비스 연산자

```
val l : Int = if (b != null) b.length else -1;
val l = b?.length ?: -1;
```

  - return 과 throw도 식이기 때문에 다음의 표현도 가능하다.

```
fun foo(node: Node): String? {
    val parent = node.getParent() ?: return null;
    val name = node.getName() ?:
               throw IllegalArgumentException("name expected")
    //. . .
}
```

대박 기능 1 - NULL SAFE

- !! 연산자
  - NPE 추종자를 위한 표현, 거의 쓸일이 없다 (없어야 한다)
  - b가 non-null이면 값을 리턴하고 null이면 NPE 발생

```
val n = b!!.length;
```

대박 기능 2 - 확장함수

StringExtensionFunction.kt

```
fun String?.toMapFromQueryString() : Map<String,String> {
    var map = HashMap<String, String>();
    val qureyString = this?: "";

    qureyString.split("&".toRegex()).filter{it.isNotEmpty()}
            .forEach {
                val splitIndex = it.indexOf("=")
                if (splitIndex > -1) {
                    val key = it.substring(0, splitIndex
                    val value = it.substring(splitIndex + 1)
                    map.put(key, value)
                }
            }
    return map;
}
```

대박 기능 2 - 확장함수

StringExtensionFunction.kt

```
@Test
fun toMapFromQueryString() {
    var map = "kwd=nike&ctgr=42412&brnd=1234,5212,3444&prtnr=1023212,4012312&inKwd=max"
            .toMapFromQueryString()
    assertEquals(map["kwd"], "nike")
    assertEquals(map["brnd"], "1234,5212,3444")
    assertEquals(map["prtnr"], "1023212,4012312")
    assertEquals(map["inKwd"], "max")

    var queryString = null;
    var emptyMap = queryString.toMapFromQueryString();
    assertTrue(emptyMap.isEmpty());
}
```

- 데코레이터와 같은 디자인 패턴을 사용하지 않고 기능을 확장할 수 있다.
- 확장 함수 비슷하게 확장 프로퍼티도 제공한다.
- null safe 하게 nullable 리시버 타입을 정의할 수 있다.

대박 기능 3 - 고차함수 및 람다

- 고차 함수란 파라미터로 함수를 받거나 함수를 리턴하는 함수

- 함수의 실행 시간을 측정하는 고차 함수

```
fun <T> calculateExecutionTime(func : () -> T ) : T {

    var startTime = System.currentTimeMillis();
    var result :T = func();
    var endTime = System.currentTimeMillis();

    println("function time check")
    println("execution Time : "
            + (endTime - startTime));

    return result;
}
```

대박 기능 3 - 고차함수 및 람다

```
@Test
fun testPrintTime() {
    calculateExecutionTime {
        for(i in 1 .. 100000)
            print("wym ")
    }

    fun sumNumber(limit : Int) : Long{
        var sum : Long = 0;
        for(i in 1 .. limit)
            sum += i;
        return sum
    }

    fun sumNumberLimit1000000() = sumNumber(1000000);

    assertEquals(
            calculateExecutionTime(::sumNumberLimit1000000 ),
            calculateExecutionTime { sumNumber(1000000) });
}
```

- 더 쉽게 횡단 관심사를 분리할 수 있다.

대박 기능 3 - 고차함수 및 람다

- 람다식 작성

```
@Test
fun testLamda() {
    val sum = {x:Int, y:Int -> x+y};
    assertEquals(sum(2,3), 5);

    val add : (Int, Int) -> Int = {x , y ->  x + y};
    assertEquals(add(4,5), 9)
}
```

- 람다식의 파라미터 생략
  - 람다식은 파라미터를 한 개만 갖는 경우가 빈번
  - 코틀린이 시그니처를 알아낼 수 있으면 파라미터 생략 가능

```
list.filter{ it.isNotEmpty() } // 해당 리터럴 타입 '(it: String) -> Boolean)'
```

그냥 코틀린으로 바꿔보자

- 프로젝트에 kotlin-reflect.jar, kotlin-runtime.jar 의존 추가
- Intellij > 상단 메뉴 > Code > Convert Java File to Kotlin File
- 자동으로 변환된 파일 적절하게 수정

JAVA TO KOTLIN

NoResultJson.java

```java
import net.wym.common.util.CommonUtil.isNotEmpty;

public class NoResultJson {
    public static JSONObject convert(String alternativeKeyword) {
        JSONObject noResult = new JSONObject().put(“cellNm", "noResult");
        if (isNotEmpty(alternativeKeyword)) {
            if (isNotEmpty(alternativeKeyword)) {
                URIUtil uriUtil = SearchURIUtil.getInstance();
                noSearchData.put("alternativeKeyword", alternativeKeyword)
                            .put("uri", uriUtil.getSimpleSearchUrl(alternativeKeyword));
            }
        }
        return noResult;
    }

    public static JSONObject convert() {
        return convert("");
    }
}
```

JAVA TO KOTLIN

NoResultJson.kt

```kotlin
object NoResultJson {
    @JvmOverloads @JvmStatic
    fun convert(alternativeKeyword: String? = ""): JSONObject {
        val noResult = JSONObject().put("cellNm", "noResult")
        if (!recommendKeyword.isNullOrEmpty()) {
            val uriUtil = SearchURIUtil.getInstance()
            noSearchData.put("alternativeKeyword", alternativeKeyword)
                    .put("uri", uriUtil.getSimpleSearchUrl(alternativeKeyword))
        }
        return noResult
    }
}
```

JAVA TO KOTLIN

ParameterUtil.java

```java
public static List<String> toStringListFrom(String parameters) { return toStringListFrom(parameters, ",");}
public static List<String> toStringListFrom(String parameters, String delimiter) {
    if (isNotEmpty(parameters)) {
        List<String> result = new ArrayList<String>();
        String[] arr = parameters.split(delimiter);
        for (String value : arr) {
            if (isNotEmpty(value))
                result.add(value);
        }
        return result;
    }
    return Collections.emptyList();
}

public static List<Integer> toIntegerListFrom(String parameters) { return toIntegerListFrom(parameters, “,"); }
public static List<Integer> toIntegerListFrom(String parameters, String delimiter) {
    if (isNotEmpty(parameters)) {
        List<Integer> result = new ArrayList<Integer>();
        String[] arr = parameters.split(delimiter);
        for (String value : arr) {
            if (isNotEmpty(value)) {
                result.add(convertToInt(value));
            }
        }
        return result;
    }
    return Collections.emptyList();
}

public static String toStringFromList(List<?> list) { return toStringFromList(list, “,"); }
public static String toStringFromList(List<?> list, String delimiter) {
    StringBuilder string = new StringBuilder();
    if (isNotEmpty(list)) {
        for (Object obj : list) {
            string.append(delimiter + obj.toString());
        }
    }
    return string.toString().replaceAll("^," , "");
}
```

JAVA TO KOTLIN

ParameterUtil.kt

```kotlin
@JvmOverloads @JvmStatic
fun toStringListFrom(parameters: String, delimiter: String = ","): List<String> {
    return ArrayList(
            parameters.split(delimiter.toRegex())
            .filter { isNotEmpty(it) }.toTypedArray().toList());
}

@JvmOverloads @JvmStatic
fun toIntegerListFrom(parameters: String, delimiter: String = ","): List<Int> {
    val result  = ArrayList<Int>();`
    parameters.split(delimiter.toRegex()).forEach ({
                val num = convertToInt(it);
                if (num > 0) result.add(num)
            });
    return result;
}

@JvmOverloads @JvmStatic
fun toStringFromList(list: List<*>, delimiter: String = ","): String {
    return list.filter { isNotEmpty(it) }.joinToString(delimiter);
}
```

Fluent API / LINQ style

JAVA TO KOTLIN

QueryString.java

```java
private void parse(String queryString) {
    String[] paramArray = queryString.split("&");

    for (String paramKeyValue : paramArray) {
        int splitIndex = paramKeyValue.indexOf("=");
        if (splitIndex > -1) {
            String key = paramKeyValue.substring(0, splitIndex);
            String value = paramKeyValue.substring(splitIndex + 1);
            parameters.put(key, value);
        }
    }
}

@Override
public String toString() {
    StringBuilder queryString = new StringBuilder();
    for(Map.Entry<String, String> entry : parameters.entrySet()) {
        queryString.append("&"+entry.getKey()+entry.getValue());
    }
    return queryString.toString().replaceAll("^[&]", "");
}
```

JAVA TO KOTLIN

QueryString.kt

```kotlin
private fun parse(queryString: String) {
    queryString.split("&".toRegex()).filter { !it.isEmpty() }
    .forEach {
        val splitIndex = it.indexOf("=")
        if (splitIndex > -1) {
            val key = it.substring(0, splitIndex)
            val value = it.substring(splitIndex + 1)
            parameters.put(key, value)
        }
    };
}

override fun toString(): String {
    return parameters.asIterable()
            .filter{ !it.value.isEmpty() }.joinToString("&");
}
```

개발서버에 적용해 보기

- 서버에 코틀린 설치
- 빌드 스크립트에 아래 코드 추가

```
<property name="kotlin.lib" value="{코틀린 컴파일러의 lib}"/>
...
<typedef resource="org/jetbrains/kotlin/ant/antlib.xml"
         classpath="${kotlin.lib}/kotlin-ant.jar" />

<target name="kotlin-compile" depends="init">
     <kotlinc src="${src.dir}" output="${classes.dir}" classpathref="classpath"  />
</target>

<target name="compile" depends="kotlin-compile">
     <!--기존과 동일-->
</target>
```

  - 원래 <javac {속성들}><withKotiln/></javac>를 하면 된다고 했는데 실패
  - 자바 컴파일 하기 전에 코틀린을 먼저 컴파일함.

- 개발서버 빌드!

예상되는 장점

- 자바의 단점을 극복하여 생산성 향상
  - 불필요한 일을 안하게됨
  - null에 좀 더 안전함
  - 높은 표현력으로 가독성 향상
- 최신 프로그래밍 언어 트렌드 습득
- REPL로 보다 빠른 개발

Terminal

```
# aymonwoo @ CP72008-MAC in ~/worktown/intellij/11st-listing-renewal-3rd [20:47:37]
$ kotlinc
Welcome to Kotlin version 1.0.2 (JRE 1.7.0_79-b15)
Type :help for help, :quit for quit
>>> println("hello world")
hello world
>>> fun plusAndMulti(a :Int, b:Int, c:Int) = (a + b) * c
>>> plusAndMulti(5, 5, 2)
20
>>>
```

너무 하고 싶었던 REPL

예상되는 단점

- 각자 IDE에 코틀린 플러그인을 설치해야한다.
  특히 이클립스…
- 새로운 언어에 대한 학습 비용
- 신생 언어이기 때문에 문법이 조금씩 변경됨
- 어노테이션 처리도구 미지원(논란이 많음)
- 여러분의 생각은?

저의 결론

- 우리 시스템의 변경을 최소화하고 도입가능한 최신 언어
- 스칼라보다는 배우기 쉽고 자바보다는 훨씬 편리한 언어
- 하지만 아직 남아있는 혹시 모르는 문제들
- 그래도 점진적으로 도입하기에 참 좋은 언어
- 자바보다는 클린코드에 좀더 가까운 언어

Q&A

감사합니다.

참고자료 : http://javacan.tistory.com/425