Android Annotation Processing @

Wai Yan Phyoe

Android Developer @AdventSoft

AdventSoft

Annotation

- **Annotations** are a class of metadata
- Associated with classes, methods, fields, and even other annotations

Familiar Annotations

- @Override
- @Deprecated
- @SupressWarning (value = “type”)
- @SafeVarargs
- @FunctionalInterface

Improve code inspection with annotations

```
implementation 'androidx.annotation:annotation:1.0.0'
```

Nullness Annotations

```
@Nullable
@NonNull
```

Resources Annotations

`@StringRes`

`@DrawableRes`

`@DimenRes`

`@ColorRes`

`@ColorInt`

`@InterpolatorRes`

`@AnyRes` (can be any type of `R` resource)

Thread Annotations

```
@MainThread
@UiThread
@WorkerThread
@BinderThread
@AnyThread
```

Value Constraint Annotations

`@IntRange`

`@FloatRange`

`@Size`

- Minimum size (such as `@Size(min=2)`)
- Maximum size (such as `@Size(max=2)`)
- Exact size (such as `@Size(2)`)
- A number of which the size must be a multiple (such as `@Size(multiple=2)`)

Permission Annotations

```
@RequiresPermission
```

Return Value Annotations

```
@CheckResult
```

CallSuper Annotations

```
@CallSuper
```

Typedef Annotations

```
@IntDef
@StringDef
```

Combining constants with flags

```
@IntDef(flag = true,value={})
@StringDef(flag = true,value={})
```

Keep Annotation

```
@Keep
```

Android Annotation Libraries

- Butterknife
- Dagger 2
- Room Persistence
- ObjectBox
- GreenDAO
- Green EventBus
- Glide
- Lombok

Android Annotation Libraries

- Retrofit
- Parceler
- Dart
- Icepick
- Android Annotation

Introduction to Annotation

- Java Version 5 Support (Sep 2004)
- Interface
- Able to write Metadata in source code
- Work in Compile Time and Runtime
- ***Generate files during compilation***

Components

1. Annotation
2. Processor
3. APT (Annotation Processing Tool) / [kapt (kotlin android processing tool)]
4. Annotated Source

Benefits

- ❏ Write your code generator (processor) **once**
- ❏ **Trust** the generated code
- ❏ Eliminate boilerplate codes
- ❏ Improve productivity

Java Boilerplate Codes

```
public class User {

    private String firstName;
    private String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(firstName, user.firstName) &&
                Objects.equals(lastName, user.lastName);
    }

    @Override
    public int hashCode() {

        return Objects.hash(firstName, lastName);
    }

    @Override
    public String toString() {
        return "User{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                '}';
    }
```

Eliminate Boilerplate Codes

```
@Data
public class User {

    private String firstName;
    private String lastName;
}
```

Use : https://projectlombok.org/

Custom Annotation

- Use @Interface

Constraints

- Can't use parameters in **method**
- Can't use throw in **method**

Meta Annotations

- @Retention(RetentionPolicy. -)
- @Target(ElementType. -)

Retention Policy

- SOURCE
- CLASS
- RUNTIME

RetentionPolicy.SOURCE

- analyses by compiler and never stored
  (visible by neither the compiler nor the runtime)

RetentionPolicy.CLASS

- stored into class file and not retained in runtime (visible by the compiler)

RetentionPolicy.RUNTIME

- store into class file and usable in runtime (can inspect via reflection)
  (visible by the compiler and the runtime)

Element Type

- ❏ TYPE
- ❏ FIELD
- ❏ LOCAL_VARIABLE
- ❏ CONSTRUCTOR
- ❏ METHOD
- ❏ PACKAGE
- ❏ PARAMETER
- ❏ ANNOTATION_TYPE
- ❏ TYPE_PARAMETER
- ❏ TYPE_USE

Optional Meta Annotations

- @Documented
- @Inherited
- @Repeatable

Annotation processor

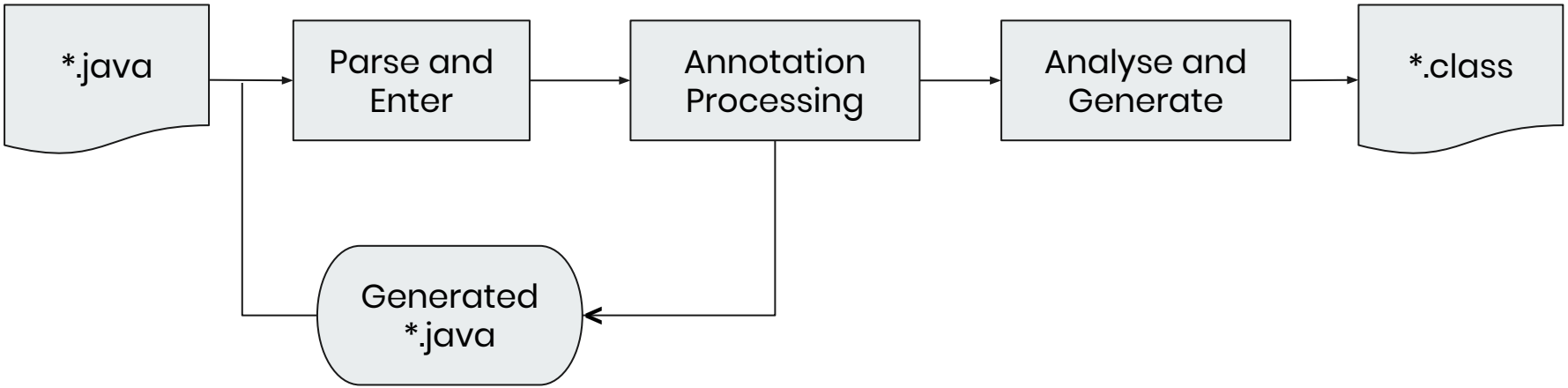
- A tool which is build in **javac** for scanning and processing annotations at compile time.

- Can’t manipulate already existing files

Annotation processing steps

1. Build starts in java compiler. (java compiler knows all processors, So If we want to create new one, we need to tell to compiler about that.)
2. Starts all Annotation Processors which is not executed.(Every processor has its own implementation)
3. Loop over annotated elements inside the processor
4. Finds annotated classes, methods, fields.
5. Generate a new class with metadata of founded classes, methods, fields. (This is the place where you generate code.)
6. Create new file and write your generated string as a class.
7. Compiler checks if all annotation processors are executed. If not, start to next round.

Annotation processing steps



Lets Go

- ❏ Leverage existing libraries
- ❏ Peek inside and how they work
- ❏ Write smart codes

References

https://youtu.be/Hzs6OBcvNQE
http://mcomella.xyz/blog/2016/thread-annotations.html
https://blog.mindorks.com/improve-your-android-coding-through-annotations-26b3273c137a
https://developer.android.com/studio/write/annotations
https://blog.mindorks.com/creating-custom-annotations-in-android-a855c5b43ed9

Thank you all

Slide available soon @Speakerdeck

Demo Codes available soon @Github

Github :

https://github.com/wyphyoe/