QtJenny

Roadmap

Overview reminder

- QtJenny generates C++ wrappers for Java APIs
- ..by reading the Java metainfo from classes
- ..and generating C++ code that uses the Java classes via JNI (QJniObject)
- QJniObject is good, generated wrappers are good, generated wrappers that use QJniObject are even better

Status Quo

```
@NativeProxyForClasses(namespace = "android::os", classes = [Context::class, PowerManager::class, BatteryManager::class,
PowerManager.WakeLock::class, TelephonyManager::class, VibratorManager::class])

@NativeProxyForClasses(namespace = "android::view", classes = [View::class, Window::class, WindowManager::class,
WindowManager.LayoutParams::class])

@NativeProxyForClasses(namespace = "java::lang", classes = [String::class, StringBuffer::class])

class GenerateCppCode {

}
```

Status quo

- Uses an AnnotationProcessor
- ..which requires a dummy source file on the previous slide

Next

```
GenerateProxies {
    JarFile {
        it.namespace = "android::os"
        it.path = "/home/ville/Android/Sdk/platforms/android-35/android.jar"
        it.fullClassNames = "android.content.Context, android.os.BatteryManager, android.os.PowerManager, android.os.VibratorManager, android.telephony.TelephonyManager, android.view.View, android.view.Window, android.view.WindowManager"
    }
    JarFile {
        it.namespace = "java::lang"
        it.path = "/home/ville/Android/Sdk/platforms/android-35/android.jar"
        it.fullClassNames = "java.lang.String,java.lang.StringBuffer"
    }
    outputDirectory = project.file('src/main/cpp/gen').getAbsolutePath()
    headerOnly = true
}
```

Next

- Uses a classloader and reflection, doesn’t require a dummy build
- Single-target generator build done with a Gradle plugin

Status quo

```
// construct: public String(java.lang.StringBuffer buffer)
 static StringProxy newInstance__Ljava_lang_StringBuffer_2(jobject buffer) {
     StringProxy ret;
     ret.m_jniObject = QJniObject(FULL_CLASS_NAME, "(Ljava/lang/StringBuffer;)V", buffer);
     return ret;
 }
```

Next

```
// construct: public String(java.lang.StringBuffer buffer)
 static StringProxy newInstance__Ljava_lang_StringBuffer_2(StringBufferProxy buffer) {
     StringProxy ret;
     ret.m_jniObject = QJniObject(FULL_CLASS_NAME, "(Ljava/lang/StringBuffer;)V", buffer.object<jobject>());
     return ret;
 }
```

Summary

- Go from annotations to reflection; no dummy source code needed for code generation
- Type-safe APIs, mis-ordered arguments will give a compiler error, not a JNI runtime error
- Support writing Java interface implementations (i.e. callbacks) in C++ (again with type-safe function signatures)