Java API to ArkTS API mapping rules

# Introduction

During conversion of a Java code into equivalent ArkTS code special care should be taken about use in the source code Java standard library entities (packages, classes, methods, fields, constants). All these entities will have to be somehow mapped to its ArkTS standard library equivalents or substituted with ArkTS code fragments. It is possible to hard code the mapping rules directly into the code of the Migration Tool. But such approach is not flexible and will be hard to support and evaluate in future. It is better:

* to prepare set of mapping rules in a separate file
* and to create special code which will apply the rules during the migration process.

# General approaches

1. The rules are specified in XML format.
2. The rule file should be named ‘java-api-mapper.xml’.
3. The root element is <**JavaApiMappingRules**>.
4. The child elements of the root element will represent list of mapping rules.
5. There are five types of mapping. The rule names (and its child element names) are inspired by the names of corresponding ArkTS grammar productions ([StaticTSParser.g4](https://gitee.com/openharmony-sig/arkcompiler_ets_frontend/blob/master/migrator/src/com/ohos/migrator/staticTS/parser/StaticTSParser.g4)): <**ImportDeclarationRule**>, <**CallExpressionRule**>, <**MemberAccessExpressionRule**>, <**NewClassInstanceExpressionRule**> and <**TypeReferenceRule**>.  
   NOTE: The child elements whose name starts with <**Arkts** or **<Src** are the mapper specific ones and they have no relation to the names used in ArkTS grammar ([StaticTSParser.g4](https://gitee.com/openharmony-sig/arkcompiler_ets_frontend/blob/master/migrator/src/com/ohos/migrator/staticTS/parser/StaticTSParser.g4)).
6. The rules will be used by special visitor class to perform transformation of **ArkTS** AST nodes.
7. Each rule element has the set of attributes used to perform matching of a visited **ArkTS**AST node. If the **ArkTS**AST node matches the criteria then the rule will be applied to it.
8. The list of the match attributes varies and depends on the rule type.
9. Rule elements may have attribute ‘**action**’ which specifies what kind of action should be performed to the matched **ArkTS**AST node. The possible values of the action are:
   * **remove** – the **ArkTS**AST node will be just removed from the final **ArkTS**AST tree.
   * **replace** – the whole **ArkTS**AST node will be replaced with the node(s) the structure of which is specified in the rule body.
   * **rebuild** – the content of the **ArkTS**AST node (its chilren) will be rebuilt in according to the rule body.
10. For the case of **replace** action the child elements of the rule element specify the **complete** structure of the new ArkTS AST node.
11. For the case of **rebuild** action the child elements of the rule element specify new structure/values of child elements of the matched ArkTS AST node. If some children of the matched node are not defined in the child elements of the rule then they left in the resulting ArkTS AST node as is without any modification.

# **<ImportDeclarationRule>** rule

Match attributes:

* **javaImport** – a string representing the package imported in the original java code.

Actions:

* If the rule doesn’t contain any <**ImportDeclaration**> children then the matched ArkTS import declaration node has to be just removed. Optionally (to explicitly notify) it maybe specified: **action=**“**remove**”. Otherwise:
* the matched ArkTS node import declaration node has to be replaced with the set of **ImportDeclarations** which are specified by the child nodes of the rule. Optionally (to explicitly notify) it maybe specified: **action=**“**replace**”.

Child elements:

* <**ImportDeclaration**> – new import declaration. The element must have attributes:
  + **arktsItem –** the name of item to import.
  + **arktsAlias –** (**optional**) the alias for the imported item.
  + **arktsFrom –** the path from which the import has to be done.

Examples:

1. <ImportDeclarationRule javaImport="java.lang.Byte" action="remove"/>  
   Completely remove the specified import.
2. <ImportDeclarationRule javaImport="java.util.Set" action="replace"> <ImportDeclaration arktsItem="Set" arktsFrom="std/containers"/> </ImportDeclarationRule>  
   Replace  
    import java.util.Set  
   with  
    import Set from “std/containers”
3. <ImportDeclarationRule javaImport="java.util.\*" action="replace">  
   <ImportDeclaration arktsItem="\*" arktsFrom="std/containers"/>  
    <ImportDeclaration arktsItem="\*" arktsFrom="std/time"/></ImportDeclarationRule>  
   Replace  
    import java.util.\*  
   with  
    import \* from “std/containers”  
    import \* from “std/time”

# **<CallExpressionRule>** rule

Match attribures:

* **javaType** – the fully qualified type of the java class method of which has to be matched.
* **javaTypeArgs** – the optional comma separated list of java type arguments. No spaces between the types should be present. The attribute is applicable only for the case of matching methods of **generic classes**.
* **javaMethodName** – the name of the method to be matched.
* **javaMethodTypeArgs** – the optional comma separated list of the method type arguments. No spaces between the types should be present. The attribute is applicable only for the case of matching **generic methods**.
* **javaMethodArgs** – the comma separated list of java types of the method arguments. No spaces between the types should be present. If the attribute is not specified then a method with zero arguments has to be matched.

Actions:

* **rebuild** – the child elements of the rule specify the replacements for corresponding child ArkTS nodes. At the moment it’s the only possible action. So this attribute may be skipped.

Child elements:

* <**ArktsTypeName**> – **Optional**. The name of ArkTS class or interface. It is applicable only for the case of static method, when the method should be qualified with the name of the class or interface. If this element specified then <**ArktsObject**> must NOT be present. The element must have the only attribute:
  + **value** – the ArkTS type name.
* <**ArktsObject**> – **Optional**. The object (in general case an expression) whose method has to be called. By default (the element is not specified at all) the same object as it is in the source code is used. If this element specified then <**ArktsTypeName**> must not be present. It has to have one child element which may be one of (the list of possible children may be expanded in feature):
  + <**NewClassInstanceExpression**> – the constructor call description of an ArkTS type. Its children should be:
    - <**TypeReference**>– Look below for the description of this element and its structure.
    - <**Arguments**> – Optional list of the constructor call arguments. Look below for the description of this element and its structure.
  + **<SrcArgument index=**”src.index.value”**/**> – The source argument with the specified index should be used as the object whose method is called.
* <**ArktsMethodName**> – **Optional**. The new ArkTS name of the method.If the element is not defined then original method name is left in the resulting tree. The element must have the only attribute:
  + **value** – the ArkTS method name.
* <**TypeArguments**> – **Optional**. The list of ArkTS type arguments. If it is not specified then the original type arguments remain. If this element is specified then its child elements **define the whole list of** the new type arguments. The order of child elements defines the order of the type arguments in the resulting function node. There could be any sequence of following child elements:
  + <**SrcTypeArgument index=**”srcArgIndex”**/**> – The source <**TypeArgument**> with the specified index should be used in this position.
  + <**TypeReference arktsName=**”arktsTypeNname”> – The name of ArkTS type. Optionally it may have type arguments specified with the following child elements:
    - <**TypeArguments**> which may contain any sequence of children of following types:
      * <**TypeReference**>
      * <**ArrayType**>
      * <**WildcardType**>
  + <**ArrayType**> – The ArkTS array type. The element type is specified with one of the two possible child elements:
    - <**PredefinedType name**="byte OR ubyte OR short OR ushort OR int OR uint OR long OR ulong OR float OR double OR boolean OR string OR char OR void"/>
    - <**TypeReference**>
  + <**WildcardType**> – with optional children:
    - <**WildcardBound type**="extends OR super">
      * <**TypeReference**>
* <**Arguments**> – Optional list of arguments. If it is not specified then the original arguments remain. If this element is present then its child elements **define the whole list of** the new arguments. There could be any sequence of following elements:
  + **<SrcObject/**> – The source object (whose method is called in the source code) should be used at this position. This element has neither attributes no child elements.
  + **<SrcArgument index=**”src.index.value”**/**> – The source argument with the specified index should be used at this position.
  + **<SrcArgumentsTail index=**”src.index.value”**/**> – The source arguments starting with the specified index and till the end should combined into an array which should be used at this position. It’s for the case of varargs function. For example *String.join(CharSequence delimiter, CharSequence... elements)* should be converted to *String.join(strings: String[], delim: String): String*. In this example the tail of source arguments should be combined into array of Strings and the array should be used as the first argument of the ArkTS function.
  + Any of possible literals  
    <**Literal type=**"null OR boolean OR string OR char OR decimal OR hexInteger OR octalInteger OR binaryInteger" **value=**"proper\_value"/>
  + Any of the suitable SingleExpressions (see the grammar [StaticTSParser.g4](https://gitee.com/openharmony-sig/arkcompiler_ets_frontend/blob/master/migrator/src/com/ohos/migrator/staticTS/parser/StaticTSParser.g4)) which may use source arguments to form the expression:
    - <**CallExpression**> - describes a method call result of which will be used as the argument at this position. The children of this element are the same as child elements of this rule (<**CallExpressionRule**>). And it doesn’t have any attributes.
    - <**UnaryExpression index**="src.index.value" **operation=**"- OR + OR ~ OR !"/> - Unary operation applied to the specified source argument.
    - <**BinaryExpression operation=**"\* OR + OR << OR >> OR >>> OR < OR <= OR == OR >= OR > OR & OR ^ OR | OR && OR ||"/> - There should be two child elements specify the arguments of the multiplication. It could be any of:
      * <**SrcArgument index=**”src.index.value”/> - The source argument with the specified index.
      * Any of possible literals.
      * Any of the suitable SingleExpressions which may use the source arguments to form the expression.
    - <**TernaryExpression**> - The children specify the three operands and could be any of the following elemenent types:
      * <**SrcArgument index=**”src.index.value”/>
      * Any of possible literals
      * Any of suitable SingleExpressions
    - <**ThisExpression**> - StsType.this. The child element of type <**TypeReference**> specifies the StsType.
    - <**SuperExpression**> - StsType.super. The child element of type <**typeReference**> specifies the StsType.
    - <**ArrayLiteralExpression**> - The child elements represent the list of elements which form the ArrayLiteral. Any of the following element types could be used as a child:
      * <**SrcArgument index=**”src.index.value”/>
      * Any of possible literals
      * Any of suitable SingleExpressions
    - <**CastExpression**> - One of its two child specifies the object/expression which has to be type cast. It could be any of the SingleExpressions of this list.  
      And its another child specifies the ArkTS type to which the cast has to be done. The child node could be of one of the two types:
      * <**IntersectionType**> - with a sequence of child elements of type <**TypeReference**>
      * <**PrimaryType**> - with one child of any of the <**PredefinedType**> or <**TypeReference**> or <**ArrayType**>

Examples:

1. <CallExpressionRule javaType="java.lang.Math" javaMethodName="getExponent" javaMethodArgs="float">  
    <ArktsTypeName value=""/> *<!-- The empty value means the function in ArkTS is a global one. -->  
    <!-- This rule specifies the type argument for the resulting generic method. -->* <ArktsMethodTypeArgs>  
    <TypeArguments>  
    <TypeReference arktsName="Float"/>  
    </TypeArguments>  
    </ArktsMethodTypeArgs>  
   </CallExpressionRule>  
   Replace  
    Math.getExponent(x)  
   with  
    getExponent<Float>(x)
2. <CallExpressionRule javaType="java.lang.Math" javaMethodName="cbrt">  
    *<!-- rename java.lang.Math.cbrt() ==> std.std.math.math.cuberoot() -->* <ArktsTypeName value=""/> *<!-- The empty value means the function in ArkTS is a global one. -->* <ArktsMethodName value="cuberoot"/>  
   </CallExpressionRule>  
   Replace  
    Math.cbrt(x)  
   with  
    cuberoot(x)
3. <CallExpressionRule javaType="java.util.Arrays" javaMethodName="fill" javaMethodArgs="int[],int,int,int">  
    *<!--  
    static void fill(byte[] a, int fromIndex, int toIndex, byte val)  
    ==>  
    function fill(arr: int[], value: int, startIndex: int, endIndex: int): void  
    -->* <ArktsTypeName value=""/> *<!-- The empty value means the function in ArkTS is a global one. -->  
    <!-- The method name is the same ==> there is no ArktsMethodName element here. -->* <Arguments> *<!-- Reorder the arguments. -->* <SrcArgument index="0"/>  
    <SrcArgument index="3"/>  
    <SrcArgument index="1"/>  
    <SrcArgument index="2"/>  
    </Arguments>  
   </CallExpressionRule>  
   Replace  
    Arrays.fill(array, from, to, value)  
   with  
    fill(array, value, from, to)
4. <CallExpressionRule javaType="java.lang.Double" javaMethodName="compare" javaMethodArgs="double,double">  
    *<!--  
    static int compare(double d1, double d2)  
    ==>  
    new Double(d1).compareTo(new Dobule(d2))  
    -->* <ArktsObject>   
    <NewClassInstanceExpression> *<!-- new Double(d1) -->* <TypeReference arktsName="Double"/>  
    <Arguments>  
    <SrcArgument index="0"/>  
    </Arguments>  
    </NewClassInstanceExpression>  
    </ArktsObject>  
     
    <ArktsMethodName value="compareTo"/>  
    <Arguments>  
    <NewClassInstanceExpression> *<!-- new Double(d2) -->* <TypeReference arktsName="Double"/>  
    <Arguments>  
    <SrcArgument index="0"/>  
    </Arguments>  
    </NewClassInstanceExpression>  
    </Arguments>  
   </CallExpressionRule>  
   Replace  
    Dobule.compare(x, y)  
   with  
    new Double(x).compareTo(new Double(y))

# **<MemberAccessRule>**rule

1. Match attribures:

* **javaType** – the fully qualified type of java class to which member access is performed.
* **javaName** – the Java name of the accessed item.
* **arktsName** – **Optional**. the ArkTS name of the accessed item. **If this attribute is not specified** then:
  + the field access is supposed to be replace with the method call defined in the child <**CallExpression**> element.
  + The child <**CallExpression**> element **must** be specified in this case.

1. Actions:
2. Defined implisitly.
3. Child elements:

* <**TypeReference**> – **Optional**. If present then it specifies ArkTS type name **static field** of which is to be accessed.
* <**CallExpression**> – **Optional**. This element **must** be present if the **arktsName** attribute of the rule element is not specified. In this case the input field access will be substituted with the method call defined with this element. The children of this element are described in the <**CallExpressionRule**> rule description with one correction. The input field access doesn’t have any arguments. So this <**CallExpression**> element may not have any child which in any way refers to source arguments. In other words the method call either has to have no parameters or all parameters have to be presented by literal values. Usually this is used to replace input field access with corresponding getter.

1. Examples:
2. 1. <MemberAccessExpressionRule javaType="java.lang.Character" javaMemberName="SIZE" arktsName="CHAR\_BIT\_SIZE">  
    <TypeReference arktsTypeName="Char"/>  
   </MemberAccessExpressionRule>

2. <MemberAccessExpressionRule javaType="java.lang.Byte" javaMemberName="SIZE" arktsName="BIT\_SIZE"/>

3. <MemberAccessExpressionRule javaType="java.lang.String" javaMemberName="length">  
 <CallExpression arktsMethodName="length">  
 *<!-- int length() -->* <Arguments/> *<!-- Denotes an empty list of arguments. -->* </CallExpression>  
</MemberAccessExpressionRule>

# **<NewClassInstanceExpressionRule>** rule

1. Match attribures:

* **javaType** – the fully qualified java type of the class.
* **javaTypeArgs** – the optional comma separated list of java type arguments. No spaces between the types should be present. The attribute is applicable only for the case of matching **generic classes**.
* **javaMethodTypeArgs** – the optional comma separated list of the method type arguments. No spaces between the types should be present. The attribute is applicable only for the case of matching **generic constructors**.
* **javaMethodArgs** – the comma separated list of java types of the constructor arguments. No spaces between the types should be present. If the attribute is not specified then a method with zero arguments has to be matched.

1. Actions:
2. No. Always rebuild will be done.
3. Child elements:

* <**TypeReference**> – see the <**CallExpressionRule**> description for details.
* <**Arguments**> – see the <**CallExpressionRule**> description for details.

# **<TypeReferenceRule>** rule

Match attribures:

* **javaType** – the fully qualified java type.
* **javaTypeArgsSignature** – the optional comma separated list of java type arguments. No spaces between the types should be present. The attribute is applicable only for the case of matching generic types.

Actions:

No. Always replace will be done.

Child elements:

* <**TypeReference arktsTypeName=**”arktsTypeName”> – The name of ArkTS type. Optionally it may have type arguments specified with the following child elements:
  + <**TypeArguments**>
    - <**TypeReference**>
    - <**ArrayType**>
    - <**WildcardType**>

1. Examples:
2. <TypeReferenceRule javaType="java.lang.Integer">  
    <TypeReference arktsName="Int"/>  
   </TypeReferenceRule>

<TypeReferenceRule javaType="java.lang.Character">  
 <TypeReference arktsName="Char"/>  
</TypeReferenceRule>