Module java.base
Package java.lang

Interface StringTemplate.Processor<R,E extends Throwable>

Type Parameters:

R - Processor's process result type

E - Exception thrown type

All Known Implementing Classes:

`FormatProcessor`[PREVIEW]

Enclosing interface:

`StringTemplate`[PREVIEW]

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface
public static interface StringTemplate.Processor<R,E extends Throwable>
```

> **`Processor` is a preview API of the Java platform.**
> *Programs can only use `Processor` when preview features are enabled.*
> *Preview features may be removed in a future release, or upgraded to permanent features of the Java platform.*

This interface describes the methods provided by a generalized string template processor. The primary method `process(StringTemplate)` is used to validate and compose a result using a `StringTemplate's`[PREVIEW] fragments and values lists.

For example:

```
class MyProcessor implements Processor<String, IllegalArgumentException> {
    @Override
```

```
    public String process(StringTemplate st) throws IllegalArgumentException {
         StringBuilder sb = new StringBuilder();
         Iterator<String> fragmentsIter = st.fragments().iterator();

         for (Object value : st.values()) {
             sb.append(fragmentsIter.next());

             if (value instanceof Boolean) {
                 throw new IllegalArgumentException("I don't like Booleans");
             }

             sb.append(value);
         }

         sb.append(fragmentsIter.next());

         return sb.toString();
    }
}

MyProcessor myProcessor = new MyProcessor();
try {
    int x = 10;
    int y = 20;
    String result = myProcessor."\{x} + \{y} = \{x + y}";
    ...
} catch (IllegalArgumentException ex) {
    ...
}
```

Implementations of this interface may provide, but are not limited to, validating inputs, composing inputs into a result, and transforming an intermediate string result to a non-string value before delivering the final result.

The user has the option of validating inputs used in composition. For example an SQL processor could prevent injection vulnerabilities by sanitizing inputs or throwing an exception of type `E` if an SQL statement is a potential vulnerability.

Composing allows user control over how the result is assembled. Most often, a user will construct a new string from the string template, with placeholders replaced by string representations of value list elements. These string representations are created as if invoking `String.valueOf(java.lang.Object)`.

Transforming allows the processor to return something other than a string. For instance, a JSON processor could return a JSON object, by parsing the string created by composition, instead of the composed string.

`StringTemplate.Processor`[PREVIEW] is a `FunctionalInterface`. This permits declaration of a processor using lambda expressions;

```
Processor<String, RuntimeException> processor = st -> {
    List<String> fragments = st.fragments();
    List<Object> values = st.values();
    // check or manipulate the fragments and/or values
    ...
    return StringTemplate.interpolate(fragments, values);
};
```

The `StringTemplate.interpolate()`[PREVIEW] method is available for those processors that just need to work with the string interpolation;

```
Processor<String, RuntimeException> processor = StringTemplate::interpolate;
```

or simply transform the string interpolation into something other than `String`;

```
Processor<JSONObject, RuntimeException> jsonProcessor = st -> new JSONObject(st.interpolate());
```

Implementation Note:

The Java compiler automatically imports `StringTemplate.STR`[PREVIEW]

See *Java Language Specification*:

15.8.6 Process Template Expressions

Since:

21

See Also:

`StringTemplate`[PREVIEW], `FormatProcessor`[PREVIEW]

Nested Class Summary

Nested Classes

Modifier and Type	Interface	Description
static interface	`StringTemplate.Processor.Linkage`[PREVIEW]	**Preview.** Built-in policies using this additional interface have the flexibility to specialize the composition of the templated string by returning a customized `MethodHandle` from `linkage`[PREVIEW].

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method	Description
static <T> `StringTemplate.Processor`[PREVIEW] <T,`RuntimeException`>	`of(Function<? super StringTemplate`[PREVIEW]`, ? extends T> process)`	This factory method can be used to create a `StringTemplate.Processor`[PREVIEW] containing a `process(java.lang.StringTemplate)` method derived from a lambda expression.
`R`	`process (StringTemplate`[PREVIEW] `stringTemplate)`	Constructs a result based on the template fragments and values in the supplied `stringTemplate`[PREVIEW] object.

Method Details

process

```
R process(StringTemplate[PREVIEW] stringTemplate)
   throws E
```

Constructs a result based on the template fragments and values in the supplied `stringTemplate`[PREVIEW] object.

API Note:

Processing of a `StringTemplate`[PREVIEW] may include validation according to the particular facts relating to each situation. The `E` type parameter indicates the type of checked exception that is thrown by `process(java.lang.StringTemplate)` if validation fails, ex. `java.sql.SQLException`. If no checked exception is expected then `RuntimeException` may be used. Note that unchecked exceptions, such as `RuntimeException`, `NullPointerException` or `IllegalArgumentException` may be thrown as part of the normal method arguments processing. Details of which exceptions are thrown will be found in the documentation of the specific implementation.

Parameters:

`stringTemplate` - a `StringTemplate`[PREVIEW] instance

Returns:

constructed object of type R

Throws:

`E` - exception thrown by the template processor when validation fails

of

```
static <T>
StringTemplate.Processor[PREVIEW]<T,RuntimeException> of(Function<? super StringTemplate[PREVIEW],? extends T> process)
```

This factory method can be used to create a `StringTemplate.Processor`[PREVIEW] containing a `process(java.lang.StringTemplate)` method derived from a lambda expression. As an example;

```
Processor<String, RuntimeException> mySTR = Processor.of(StringTemplate::interpolate);
int x = 10;
int y = 20;
String str = mySTR."\{x} + \{y} = \{x + y}";
```

The result type of the constructed `StringTemplate.Processor`[PREVIEW] may be derived from the lambda expression, thus this method may be used in a var statement. For example, `mySTR` from above can also be declared using;

```
var mySTR = Processor.of(StringTemplate::interpolate);
```

`RuntimeException` is the assumed exception thrown type.

Type Parameters:

T - Processor's process result type

Parameters:

`process` - a function that takes a `StringTemplate`[PREVIEW] as an argument and returns the inferred result type

Returns:

a `StringTemplate.Processor`[PREVIEW]
