RestParameter:

`...` *Identifier* *TypeAnnotation*$_{opt}$

Parameter names must be unique. A compile-time error occurs if two or more parameters have the same name.

A parameter is permitted to include a `public` or `private` modifier only if it occurs in the parameter list of a *ConstructorImplementation* (section 8.3.1).

A parameter with a type annotation is considered to be of that type. A type annotation for a rest parameter must denote an array type.

A parameter with no type annotation or initializer is considered to be of type `any`, unless it is a rest parameter, in which case it is considered to be of type `any[]`.

When a parameter type annotation specifies a string literal type, the containing signature is a specialized signature (section 3.7.2.4). Specialized signatures are not permitted in conjunction with a function body, i.e. the *FunctionExpression*, *FunctionImplementation*, *MemberFunctionImplementation*, and *ConstructorImplementation* grammar productions do not permit parameters with string literal types.

A parameter can be marked optional by following its name with a question mark (`?`) or by including an initializer. The form that includes an initializer is permitted only in conjunction with a function body, i.e. only in a *FunctionExpression*, *FunctionImplementation*, *MemberFunctionImplementation*, or *ConstructorImplementation* grammar production.

3.7.2.3 Return Type

If present, a call signature's return type annotation specifies the type of the value computed and returned by a call operation. A `void` return type annotation is used to indicate that a function has no return value.

When a call signature with no return type annotation occurs in a context without a function body, the return type is assumed to be the Any type.

When a call signature with no return type annotation occurs in a context that has a function body (specifically, a function implementation, a member function implementation, or a member accessor declaration), the return type is inferred from the function body as described in section 6.3.

3.7.2.4 Specialized Signatures

When a parameter type annotation specifies a string literal type (section 3.2.8), the containing signature is considered a specialized signature. Specialized signatures are used to express patterns where specific string values for some parameters cause the types of other parameters or the function result to become further specialized. For example, the declaration