- ***String index signatures***, specified using index type `string`, define type constraints for all properties and numeric index signatures in the containing type. Specifically, in a type with a string index signature of type *T*, all properties and numeric index signatures must have types that are subtypes of *T*.
- ***Numeric index signatures***, specified using index type `number`, define type constraints for all numerically named properties in the containing type. Specifically, in a type with a numeric index signature of type *T*, all numerically named properties must have types that are subtypes of *T*.

A ***numerically named property*** is a property whose name is a valid numeric literal. Specifically, a property with a name *N* for which ToNumber(*N*) is not NaN, where ToNumber is the abstract operation defined in ECMAScript specification.

An object type can contain at most one string index signature and one numeric index signature.

Index signatures affect the determination of the type that results from applying a bracket notation property access to an instance of the containing type, as described in section 4.10.

3.7.5 Method Signatures

A method signature is shorthand for declaring a property of a function type.

MethodSignature:
PropertyName `?`$_{opt}$ *CallSignature*

If the identifier is followed by a question mark, the property is optional. Otherwise, the property is required. Only object type literals and interfaces can declare optional properties.

A method signature of the form

PropName `<` *TypeParamList* `>` `(` *ParamList* `)` `:` *ReturnType*

is equivalent to the property declaration

PropName `:` `{` `<` *TypeParamList* `>` `(` *ParamList* `)` `:` *ReturnType* `}`

A literal type may ***overload*** a method by declaring multiple method signatures with the same name but differing parameter lists. Overloads must either all be required (question mark omitted) or all be optional (question mark included). A set of overloaded method signatures correspond to a declaration of a single property with a type composed from an equivalent set of call signatures. Specifically

PropName `<` $TypeParamList_1$ `>` `(` $ParamList_1$ `)` `:` $ReturnType_1$ `;`
PropName `<` $TypeParamList_2$ `>` `(` $ParamList_2$ `)` `:` $ReturnType_2$ `;`
...
PropName `<` $TypeParamList_n$ `>` `(` $ParamList_n$ `)` `:` $ReturnType_n$ `;`

is equivalent to