Type Class

Contains methods for getting the Apex type that corresponds to an Apex class and for instantiating new types.

Namespace

System

Usage

Use the `forName` methods to retrieve the type of an Apex class, which can be a built-in or a user-defined class. You can use these methods to retrieve the type of public and global classes, and not private classes even if the context user has access. Also, use the `newInstance` method if you want to instantiate a Type that implements an interface and call its methods while letting someone else, such as a subscriber of your package, provide the methods' implementations.

Note

A call to `Type.forName()` can cause the class to be compiled.

Example: Instantiating a Type Based on Its Name

The following sample shows how to use the Type methods to instantiate a Type based on its name. A typical application of this scenario is when a package subscriber provides a custom implementation of an interface that is part of an installed package. The package can get the name of the class that implements the interface through a custom setting in the subscriber's org. The package can then instantiate the type that corresponds to this class name and invoke the methods that the subscriber implemented.

In this sample, `Vehicle` represents the interface that the `VehicleImpl` class implements. The last class contains the code sample that invokes the methods implemented in `VehicleImpl`.

This is the `Vehicle` interface.

```
global interface Vehicle {
    Long getMaxSpeed();
    String getType();
}
```

This is the implementation of the `Vehicle` interface.

```
global class VehicleImpl implements Vehicle {
    global Long getMaxSpeed() { return 100; }
    global String getType() { return 'Sedan'; }
}
```

The method in this class gets the name of the class that implements the `Vehicle` interface through a custom setting value. It then instantiates this class by getting the corresponding type and calling

```
public class CustomerImplInvocationClass {

    public static void invokeCustomImpl() {
        // Get the class name from a custom setting.
        // This class implements the Vehicle interface.
        CustomImplementation__c cs = CustomImplementation__c.getInstance('Vehicle');

        // Get the Type corresponding to the class name
        Type t = Type.forName(cs.className__c);

        // Instantiate the type.
        // The type of the instantiated object
        //   is the interface.
        Vehicle v = (Vehicle)t.newInstance();

        // Call the methods that have a custom implementation
        System.debug('Max speed: ' + v.getMaxSpeed());
        System.debug('Vehicle type: ' + v.getType());
    }
}
```

Class Property

The `class` property returns the `System.Type` of the type it is called on. It's exposed on all Apex built-in types including primitive data types and collections, sObject types, and user-defined classes. This property can be used instead of `forName` methods.

Call this property on the type name. For example:

```
System.Type t = Integer.class;
```

You can use this property for the second argument of `JSON.deserialize`, `deserializeStrict`, `JSONParser.readValueAs`, and `readValueAsStrict` methods to get the type of the object to deserialize. For example:

```
Decimal n = (Decimal)JSON.deserialize('100.1', Decimal.class);
```

Type Methods

The following are methods for `Type`.

- **equals(typeToCompare)**
  Returns `true` if the specified type is equal to the current type; otherwise, returns `false`.
- **forName(fullyQualifiedName)**
  Returns the type that corresponds to the specified fully qualified class name.
- **forName(namespace, name)**
  Returns the type that corresponds to the specified namespace and class name.
- **getName()**
  Returns the name of the current type.
- **hashCode()**
  Returns a hash code value for the current type.
- **isAssignableFrom(sourceType)**
  Returns `true` if an object reference of the specified type can be assigned from the child type; otherwise, returns `false`.

equals(typeToCompare)

Returns `true` if the specified type is equal to the current type; otherwise, returns `false`.

Signature

```
public Boolean equals(Object typeToCompare)
```

Parameters

typeToCompare

Type: Object

The type to compare with the current type.

Return Value

Type: Boolean

Example

```
Type t1 = Account.class;
Type t2 = Type.forName('Account');
System.assert(t1.equals(t2));
```

forName(fullyQualifiedName)

Returns the type that corresponds to the specified fully qualified class name.

Signature

```
public static System.Type forName(String fullyQualifiedName)
```

Parameters

fullyQualifiedName

Type: String

The fully qualified name of the class to get the type of. The fully qualified class name contains the namespace name, for example, `MyNamespace.ClassName`.

Return Value

Type: `System.Type`

Usage

Note

- This method returns `null` if called outside a managed package to get the type of a non-global class in a managed package. This is because the non-global class isn't visible outside the managed package. For Apex saved using Salesforce API version 27.0 and earlier, this method does return the corresponding class type for the non global managed package class.
- When called from an installed managed package to get the name of a local type in an organization with no defined namespace, the `forName(fullyQualifiedName)` method returns `null`. Instead, use the `forName(namespace, name)` method and specify an empty string or `null` for the namespace argument.
- A call to `Type.forName()` can cause the class to be compiled.

Returns the type that corresponds to the specified namespace and class name.

Signature

```
public static System.Type forName(String namespace, String name)
```

Parameters

namespace

Type: String

The namespace of the class. If the class doesn't have a namespace, set the *namespace* argument to `null` or an empty string.

name

Type: String

The name of the class.

Return Value

Type: `System.Type`

Usage

Note

- This method returns `null` if called outside a managed package to get the type of a non-global class in a managed package. This is because the non-global class isn't visible outside the managed package. For Apex saved using Salesforce API version 27.0 and earlier, this method does return the corresponding class type for the non-global managed package class.
- Use this method instead of `forName(fullyQualifiedName)` if it's called from a managed package installed in an organization with no defined namespace. To get the name of a local type, set the namespace argument to an empty string or `null`. For example, `Type t = Type.forName('', 'ClassName');`.
- A call to `Type.forName()` can cause the class to be compiled.

Example

This example shows how to get the type that corresponds to the `ClassName` class and the `MyNamespace` namespace.

```
Type myType =
   Type.forName('MyNamespace', 'ClassName');
```

Versioned Behavior Changes

In API version 60.0 and later, using an invalid namespace while calling this method returns null. Previously, Apex allowed you to specify an invalid namespace such as `Type.forName('InvalidNamespace', 'OuterClass.InnerClass')` or use an outer class as a namespace such as `Type.forName('OuterClass', 'InnerClass')` with indeterminate results.

getName()

Returns the name of the current type.

Signature

Example

This example shows how to get a Type's name. It first obtains a Type by calling `forName`, then calls `getName` on the Type object.

```
Type t =
   Type.forName('MyClassName');

String typeName =
   t.getName();
System.assertEquals('MyClassName',
   typeName);
```

hashCode()

Returns a hash code value for the current type.

Signature

`public Integer hashCode()`

Return Value

Type: Integer

Usage

The returned hash code value corresponds to the type name hash code that `String.hashCode` returns.

isAssignableFrom(sourceType)

Returns `true` if an object reference of the specified type can be assigned from the child type; otherwise, returns `false`.

Signature

`public Boolean isAssignableFrom(Type sourceType)`

Parameters

sourceType

The type of the object with which you are checking compatibility.

Return Value

Type: Boolean

The method returns `true` when the method is invoked as parentType.isAssignableFrom(childType). When invoked in any of the following ways, the method returns `false`:

- childType.isAssignableFrom(parentType)
- typeA.isAssignableFrom(TypeB) where TypeB is a sibling of TypeA
- typeA.isAssignableFrom(TypeB) where TypeB and TypeA are unrelated

Note

A childType is the child of a parentType when it implements an interface, extends a virtual or abstract class, or is the same `System.Type` as the parentType.

The following code demonstrates how a typical ISV customer can use `isAssignableFrom()` to check compatibility between a customer-defined type (customerProvidedPluginType) and a valid plugin type.

```
//Scenario: Managed package code loading a “plugin” class that implements a managed interf
String pluginNameStr = Config__c.getInstance().PluginApexType__c;
Type customerProvidedPluginType = Type.forName(pluginNameStr);
Type pluginInterface = ManagedPluginInterface.class;

// Constructors may have side-effects, including potentially unsafe DML/callouts.
// We want to make sure the class is really designed to be a valid plugin before we instar
Boolean validPlugin = pluginInterface.isAssignableFrom(customerProvidedPluginType); // val

if(!validPlugin){
    throw new SecurityException('Cannot create instance of '+customerProvidedPluginType+'
}else{
    return Type.newInstance(validPlugin);
}
```

Example

The following code snippet first defines sibling classes A and B that both implement the Callable interface and an unrelated class C. Then, it explores several type comparisons using `isAssignableFrom()`.

```
//Define classes A, B, and C

global class A implements Database.Batchable<String>, Callable {
    global Iterable<String> start(Database.BatchableContext context) { return null; }
    global void execute(Database.BatchableContext context, String[] scope) { }
    global void finish(Database.BatchableContext context) { }
    global Object call(String action, Map<String, Object> args) { return null; }
}
```

```
global class B implements Callable {
    global Object call(String action, Map<String, Object> args) { return null; }
}
```

```
global class C { }
```

```
Type listOfStrings = Type.forName('List<String>');
Type listOfIntegers = Type.forName('List<Integer>');
boolean flagListTypes = listOfIntegers.isAssignableFrom(listOfStrings); // false
```

```
//Examples with stringType and idType
Type stringType = Type.forName('String');
Type idType = Type.forName('Id');
boolean isId_assignableFromString = idType.isAssignableFrom(stringType); // true
//isAssignableFrom respects that String can be assigned to Id without an explicit cast
```

```
    boolean isTypeA_ofTypeC = typeA.isAssignableFrom( typeC ); // false - unrelated types
    boolean isTypeA_ofTypeA = typeA.isAssignableFrom(typeA); // true - identity
```

```
    //Examples with callableType and batchableType
    Type callableType = Type.forName('Callable');
    Type batchableType = Type.forName('Database.Batchable');
    boolean isTypeA_Callable = callableType.isAssignableFrom( typeA ); // true - type A is a c
    boolean isTypeA_Batchable = batchableType.isAssignableFrom( typeA ); // true - type A is a
    boolean isCallableOfTypeA = typeA.isAssignableFrom( callableType ); // false - Callable ty
    boolean isBatchableOfTypeA = typeA.isAssignableFrom( batchableType ); // false - Batchable
```

newInstance()

Creates an instance of the current type and returns this new instance.

Signature

`public Object newInstance()`

Return Value

Type: Object

Usage

Because `newInstance` returns the generic object type, you should cast the return value to the type of the variable that will hold this value.

This method enables you to instantiate a Type that implements an interface and call its methods while letting someone else provide the methods' implementation. For example, a package developer can provide an interface that a subscriber who installs the package can implement. The code in the package calls the subscriber's implementation of the interface methods by instantiating the subscriber's Type.

Example

This example shows how to create an instance of a Type. It first gets a Type by calling `forName` with the name of a class (`ShapeImpl`), then calls `newInstance` on this Type object. The `newObj` instance is declared with the interface type (`Shape`) that the `ShapeImpl` class implements. The return value of the `newInstance` method is cast to the `Shape` type.

```
    Type t =
       Type.forName('ShapeImpl');

    Shape newObj =
       (Shape)t.newInstance();
```

toString()

Returns a string representation of the current type, which is the type name.

Signature

`public String toString()`

Return Value

Type: String

Example

This example calls `toString` on the Type corresponding to a list of Integers.

```
Type t = List<Integer>.class;
String s = t.toString();
System.assertEquals('List<Integer>', s);
```

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Share your feedback