Creating an instance of a generic type in DART

Asked 7 years, 8 months ago Active 8 months ago Viewed 36k times

53

16

I was wondering if is possible to create an instance of a generic type in Dart. In other languages like Java you could work around this using reflection, but I'm not sure if this is possible in Dart.

I have this class:

```
class GenericController <T extends RequestHandler> {

    void processRequest() {
        T t = new T();  // ERROR
    }
}
```

dart

Share Improve this question Follow

edited Apr 16 '14 at 14:46

Günter Zöchbauer
532k ● 177 ● 1848 ● 1474

asked Apr 16 '14 at 14:18

shuriquen
679 ● 1 ● 5 ● 9

1 Type parameters in Dart implements the interface `Type`. The `Type` interface does not declares any members. This means that the interface `Type` used only as the identity key of runtime types. The reflection procedures built into Dart SDK but they are not a part of Dart core. This means that if you want to introspect your program you should use reflection library. Bridge between your program (at runtime) and reflection library are the interface `Type`. You request (reflect) required information about the classes using this interface. – mezoni Apr 16 '14 at 15:53

1 See also github.com/dart-lang/sdk/issues/12921 – Günter Zöchbauer Jun 4 '15 at 14:29

6 Answers

Active Oldest Votes

88

I tried mezonis approach with the Activator and it works. But it is an expensive approach as it uses mirrors, which requires you to use "mirrorsUsed" if you don't want to have a 2-4MB js file.

This morning I had the idea to use a generic typedef as generator and thus get rid of reflection:

You define a method type like this: (Add params if necessary)

or even better:

```
typedef ItemCreator<S> = S Function();
```

Then in the class that needs to create the new instances:

```
class PagedListData<T>{
  ...
  ItemCreator<T> creator;
  PagedListData(ItemCreator<T> this.creator) {

  }

  void performMagic() {
      T item = creator();
      ...
  }
}
```

Then you can instantiate the `PagedList` like this:

```
PagedListData<UserListItem> users
        = new PagedListData<UserListItem>(()=> new UserListItem());
```

You don't lose the advantage of using generic because at declaration time you need to provide the target class anyway, so defining the creator method doesn't hurt.

Share Improve this answer Follow

edited May 7 '21 at 15:16

answered Feb 17 '15 at 6:39

Patrick Cornelissen
7,629 ● 3 ● 46 ● 68

6 Works with flutter – Soul_man Jun 10 '18 at 23:24

1 it totally worked for me. But I don't understand how `creator()` is creating an instance of `T` and not an instance of `ItemCreator<T>`. Can you explain that for me? – tufekoi Apr 26 '19 at 22:22

1 ItemCreator<T> is a function type of a function that returns something of type T, as you can see in the typedef. In the last code snippet you can see that the PagedListData instance is created and an instance of the creator function is provided. This is just a template and not a function call. This function is only called in the middle code snippet at "creator()". Does that help? – Patrick Cornelissen Apr 27 '19 at 20:27

2 There really is no magic here. It boils down to delegating the instantiation responsibility to the consumer, via a callback. It's not a bad solution, but it would have been nice to be able to dynamically instantiate a type inside `PagedListData` without it. But I don't think it's possible without mirrors, which we don't have in Flutter. – DarkNeuron May 13 '19 at 15:25

As a matter of taste you might want your PagedListData and creator() declared abstract so that they are implemented on the sub-class level rather than by client. – Anton Duzenko Jul 17 '19 at 15:29

You can use similar code:

14

```
import "dart:mirrors";

void main() {
  var controller = new GenericController<Foo>();
  controller.processRequest();
}

class GenericController<T extends RequestHandler> {
  void processRequest() {
    //T t = new T();
    T t = Activator.createInstance(T);
    t.tellAboutHimself();
  }
}

class Foo extends RequestHandler {
  void tellAboutHimself() {
    print("Hello, I am 'Foo'");
  }
}

abstract class RequestHandler {
  void tellAboutHimself();
}

class Activator {
  static createInstance(Type type, [Symbol constructor, List
      arguments, Map<Symbol, dynamic> namedArguments]) {
    if (type == null) {
      throw new ArgumentError("type: $type");
    }

    if (constructor == null) {
      constructor = const Symbol("");
    }

    if (arguments == null) {
      arguments = const [];
    }

    var typeMirror = reflectType(type);
    if (typeMirror is ClassMirror) {
      return typeMirror.newInstance(constructor, arguments,
        namedArguments).reflectee;
    } else {
      throw new ArgumentError("Cannot create the instance of the type '$type'.");
    }
  }
}
```

Share Improve this answer Follow

edited Apr 16 '14 at 15:39

answered Apr 16 '14 at 15:34

mezoni

8,950 ● 4 ● 29 ● 47

9

I don't know if this is still useful to anyone. But I have found an easy workaround. In the function you want to initialize the type T, pass an extra argument of type `T Function()`. This function should return an instance of T. Now whenever you want to create object of T, call the function.

```
class foo<T> {
    void foo(T Function() creator) {
        final t = creator();
        // use t
    }
}
```

P.S. inspired by Patrick's answer

Share Improve this answer Follow

answered Apr 30 '20 at 19:20

Hemil

784 ● 6 ● 24

7

Here's my work around for this sad limitation

```
class RequestHandler {
  static final _constructors = {
    RequestHandler: () => RequestHandler(),
    RequestHandler2: () => RequestHandler2(),
  };
  static RequestHandler create(Type type) {
    return _constructors[type]();
  }
}

class RequestHandler2 extends RequestHandler {}

class GenericController<T extends RequestHandler> {
  void processRequest() {
    //T t = new T(); // ERROR
    T t = RequestHandler.create(T);
  }
}

test() {
  final controller = GenericController<RequestHandler2>();
  controller.processRequest();
}
```

Share Improve this answer Follow

edited Aug 27 '19 at 9:39

answered Aug 27 '19 at 8:13

Anton Duzenko

1,912 ● 1 ● 19 ● 22

4

Sorry but as far as I know, a type parameter cannot be used to name a constructor in an instance creation expression in Dart.

Share Improve this answer Follow

answered Apr 16 '14 at 14:55

yorodm
3,892 ● 17 ● 29

2

Working with FLutter

```
typedef S ItemCreator<S>();

mixin SharedExtension<T> {

    T getSPData(ItemCreator<T> creator) async {
        return creator();
    }
}

Abc a = sharedObj.getSPData(()=> Abc());
```

P.S. inspired by Patrick

Share Improve this answer Follow

edited Aug 27 '20 at 11:25

Hemil
784 ● 6 ● 24

answered Jul 22 '20 at 1:55

Teenenggr
93 ● 8

Why is the function async btw? – Hemil Aug 26 '20 at 11:45