Map Class

Contains methods for the Map collection type.

Namespace

System

Usage

The Map methods are all instance methods, that is, they operate on a particular instance of a map. The following are the instance methods for maps.

Note

- Map keys and values can be of any data type–primitive types, collections, sObjects, user-defined types, and built-in Apex types.
- Uniqueness of map keys of user-defined types is determined by the `equals` and `hashCode` methods, which you provide in your classes. Uniqueness of keys of all other non-primitive types, such as sObject keys, is determined by comparing the objects' field values. Use caution when you use an sObject as a map key because when the sObject is changed, it no longer maps to the same value. For information and examples, see https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_map_sobject_considerations.htm
- Map keys of type String are case-sensitive. Two keys that differ only by the case are considered unique and have corresponding distinct Map entries. Subsequently, the Map methods, including `put`, `get`, `containsKey`, and `remove` treat these keys as distinct.
- With the `keySet()` method, the returned keySet is backed by the map and reflects any changes made to the map, and vice versa.

For more information on maps, see Maps.

- **Map Constructors**
- **Map Methods**

Map Constructors

The following are constructors for `Map`.

- **Map<T1,T2>()**
  Creates a new instance of the `Map` class. T1 is the data type of the keys and T2 is the data type of the values.
- **Map<T1,T2>(mapToCopy)**
  Creates a new instance of the `Map` class and initializes it by copying the entries from the specified map. T1 is the data type of the keys and T2 is the data type of the values.
- **Map<ID,sObject>(recordList)**
  Creates a new instance of the `Map` class and populates it with the passed-in list of sObject records. The keys are populated with the sObject IDs and the values are the sObjects.

Signature

```
public Map<T1,T2>()
```

Example

```
Map<Integer, String> m1 = new Map<Integer, String>();
m1.put(1, 'First item');
m1.put(2, 'Second item');
```

Map<T1,T2>(mapToCopy)

Creates a new instance of the `Map` class and initializes it by copying the entries from the specified map. T1 is the data type of the keys and T2 is the data type of the values.

Signature

```
public Map<T1,T2>(Map<T1,T2> mapToCopy)
```

Parameters

mapToCopy

Type: Map<T1, T2>

The map to initialize this map with. T1 is the data type of the keys and T2 is the data type of the values. All map keys and values are copied to this map.

Example

```
Map<Integer, String> m1 = new Map<Integer, String>();
m1.put(1, 'First item');
m1.put(2, 'Second item');
Map<Integer, String> m2 = new Map<Integer, String>(m1);
// The map elements of m2 are copied from m1
System.debug(m2);
```

Map<ID,sObject>(recordList)

Creates a new instance of the `Map` class and populates it with the passed-in list of sObject records. The keys are populated with the sObject IDs and the values are the sObjects.

Signature

```
public Map<ID,sObject>(List<sObject> recordList)
```

Parameters

recordList

Type: List<sObject>

The list of sObjects to populate the map with.

Example

```
List<Account> ls = [select Id,Name from Account];
Map<Id, Account> m = new Map<Id, Account>(ls);
```

Removes all of the key-value mappings from the map.

- clone()
  Makes a duplicate copy of the map.
- containsKey(key)
  Returns `true` if the map contains a mapping for the specified key.
- deepClone()
  Makes a duplicate copy of a map, including sObject records if this is a map with sObject record values.
- equals(map2)
  Compares this map with the specified map and returns `true` if both maps are equal; otherwise, returns `false`.
- get(key)
  Returns the value to which the specified key is mapped, or `null` if the map contains no value for this key.
- getSObjectType()
  Returns the token of the sObject type that makes up the map values.
- hashCode()
  Returns the hashcode corresponding to this map.
- isEmpty()
  Returns true if the map has zero key-value pairs.
- keySet()
  Returns a set that contains all of the keys in the map.
- put(key, value)
  Associates the specified value with the specified key in the map.
- putAll(fromMap)
  Copies all of the mappings from the specified map to the original map.
- putAll(sobjectArray)
  Adds the list of sObject records to a map declared as Map<ID, sObject> or Map<String, sObject>.
- remove(key)
  Removes the mapping for the specified key from the map, if present, and returns the corresponding value.
- size()
  Returns the number of key-value pairs in the map.
- toString()
  Returns the string representation of the map.
- values()
  Returns a list that contains all the values in the map.

clear()

Removes all of the key-value mappings from the map.

Signature

```
public Void clear()
```

Return Value

Type: Void

clone()

Makes a duplicate copy of the map.

Type: Map (of same type)

Usage

If this is a map with sObject record values, the duplicate map will only be a shallow copy of the map. That is, the duplicate will have references to each sObject record, but the records themselves are not duplicated. For example:

To also copy the sObject records, you must use the `deepClone` method.

Example

```
Account a = new Account(
   Name='Acme',
   BillingCity='New York');

Map<Integer, Account> map1 = new Map<Integer, Account> {};
map1.put(1, a);

Map<Integer, Account> map2 = map1.clone();
map1.get(1).BillingCity =
'San Francisco';

System.assertEquals(
   'San Francisco',
   map1.get(1).BillingCity);

System.assertEquals(
   'San Francisco',
   map2.get(1).BillingCity);
```

containsKey(key)

Returns `true` if the map contains a mapping for the specified key.

Signature

```
public Boolean containsKey(Object key)
```

Parameters

key

Type: Object

Return Value

Type: Boolean

Usage

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Boolean contains = colorCodes.containsKey('Blue');
System.assertEquals(true, contains);
```

Signature

```
public Map<Object, Object> deepClone()
```

Return Value

Type: Map (of the same type)

Usage

To make a shallow copy of a map without duplicating the sObject records it contains, use the `clone()` method.

Example

```
Account a = new Account(
   Name='Acme',
   BillingCity='New York');

Map<Integer, Account> map1 = new Map<Integer, Account> {};

map1.put(1, a);

Map<Integer, Account> map2 = map1.deepClone();

// Update the first entry of map1
map1.get(1).BillingCity = 'San Francisco';
// Verify that the BillingCity is updated in map1 but not in map2
System.assertEquals('San Francisco', map1.get(1).BillingCity);
System.assertEquals('New York', map2.get(1).BillingCity);
```

equals(map2)

Compares this map with the specified map and returns `true` if both maps are equal; otherwise, returns `false`.

Signature

```
public Boolean equals(Map map2)
```

Parameters

map2

Type: Map

The *map2* argument is the map to compare this map with.

Return Value

Type: Boolean

Usage

Two maps are equal if their key/value pairs are identical, regardless of the order of those pairs. The `==` operator is used to compare the map keys and values.

The `==` operator is equivalent to calling the `equals` method, so you can call `map1.equals(map2);` instead of `map1 == map2;`.

get(key)

```
public Object get(Object key)
```

Parameters

key

Type: Object

Return Value

Type: Object

Usage

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

String code = colorCodes.get('Blue');

System.assertEquals('0000A0', code);

// The following is not a color in the map
String code2 = colorCodes.get('Magenta');
System.assertEquals(null, code2);
```

getSObjectType()

Returns the token of the sObject type that makes up the map values.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

Use this method with describe information, to determine if a map contains sObjects of a particular type.

Note that this method can only be used with maps that have sObject values.

For more information, see Understanding Apex Describe Information.

Example

```
// Create a generic sObject variable.
SObject sObj = Database.query('SELECT Id FROM Account LIMIT 1');

// Verify if that sObject variable is an Account token.
System.assertEquals(
   Account.sObjectType,
   sObj.getSObjectType());

// Create a map of generic sObjects
```

hashCode()

Returns the hashcode corresponding to this map.

Signature

```
public Integer hashCode()
```

Return Value

Type: Integer

isEmpty()

Returns true if the map has zero key-value pairs.

Signature

```
public Boolean isEmpty()
```

Return Value

Type: Boolean

Example

```
Map<String, String> colorCodes = new Map<String, String>();
Boolean empty = colorCodes.isEmpty();
System.assertEquals(true, empty);
```

keySet()

Returns a set that contains all of the keys in the map.

Signature

```
public Set<Object> keySet()
```

Return Value

Type: Set (of key type)

The returned keySet is backed by the map, so the keySet reflects any changes made to the map, and vice-versa.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Set <String> colorSet = new Set<String>();
colorSet = colorCodes.keySet();
```

put(key, value)

Parameters

key

Type: Object

value

Type: Object

Return Value

Type: Object

Usage

If the map previously contained a mapping for this key, the old value is returned by the method and then replaced.

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'ff0000');
colorCodes.put('Red', '#FF0000');
// Red is now #FF0000
```

putAll(fromMap)

Copies all of the mappings from the specified map to the original map.

Signature

```
public Void putAll(Map fromMap)
```

Parameters

fromMap

Type: Map

Return Value

Type: Void

Usage

The new mappings from *fromMap* are merged with any mappings that existed in the original map. If any of the keys match, the original map values are replaced by corresponding values in the new mapping.

Example

```
Map<String, String> map1 = new Map<String, String>();
map1.put('Red','FF0000');
Map<String, String> map2 = new Map<String, String>();
map2.put('Blue','0000FF');
// Add map1 entries to map2
map2.putAll(map1);
System.assertEquals(2, map2.size());
```

```
public Void putAll(sObject[] sobjectArray)
```

Parameters

sobjectArray

Type: sObject[]

Return Value

Type: Void

Usage

This method is similar to calling the Map constructor with the same input.

Example

```
List<Account> accts = new List<Account>();
accts.add(new Account(Name='Account1'));
accts.add(new Account(Name='Account2'));
// Insert accounts so their IDs are populated.
insert accts;
Map<Id, Account> m = new Map<Id, Account>();
// Add all the records to the map.
m.putAll(accts);
System.assertEquals(2, m.size());
```

remove(key)

Removes the mapping for the specified key from the map, if present, and returns the corresponding value.

Signature

```
public Object remove(Key key)
```

Parameters

key

Type: Key

Return Value

Type: Object

Usage

If the key is a string, the key value is case-sensitive.

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

String myColor = colorCodes.remove('Blue');
String code2 = colorCodes.get('Blue');
```

Returns the number of key-value pairs in the map.

Signature

```
public Integer size()
```

Return Value

Type: Integer

Example

```
Map<String, String> colorCodes = new Map<String, String>();

colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

Integer mSize = colorCodes.size();
system.assertEquals(2, mSize);
```

toString()

Returns the string representation of the map.

Signature

```
public String toString()
```

Return Value

Type: String

Usage

When used in cyclic references, the output is truncated to prevent infinite recursion. When used with large collections, the output is truncated to avoid exceeding total heap size and maximum CPU time.

- Up to 10 items per collection are included in the output, followed by an ellipsis (...).
- If the same object is included multiple times in a collection, it's shown in the output only once; subsequent references are shown as `(already output)`.

values()

Returns a list that contains all the values in the map.

Signature

```
public List<Object> values()
```

Return Value

Type: List<Object>

Usage

The order of map elements is deterministic. You can rely on the order being the same in each subsequent execution of the same code. For example, suppose the `values()` method returns a list containing `value1` and index 0 and `value2` and index 1. Subsequent runs of the same code result in those values being returned in the same order.

```
colorCodes.put('Red', 'FF0000');
colorCodes.put('Blue', '0000A0');

List<String> colors = new List<String>();
colors = colorCodes.values();
```

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

Share your feedback