

サーチ

Cancel



- プロダクト

- Platform

- 概要
 - [AppExchange](#)
 - [Force.com](#)
 - [Heroku](#)
 - [Mobile](#)

Salesforce Apps

- [Analytics Cloud](#)
 - [Community Cloud](#)
 - [Marketing Cloud](#)
 - [Pardot](#)
 - [Quip](#)

feedk

Build Faster, Smarter & Together

- [Lightning](#)
 - [Einstein](#)
 - [Salesforce DX](#)



- リソース

- こちらをご覧ください

- [入門](#)
 - [ドキュメント](#)
 - [Trailhead](#)
 - [書籍&チートシート](#)
 - [オンデマンドWebセミナー](#)

- [認定資格](#)
- [ブログ](#)

ツール

- [Force.com IDE](#)
- [Lightning 設計システム](#)
- [データベース](#)
- [その他のツールやツールキット](#)

トピック別

- [アプリケーションの配布](#)
- [アプリケーションロジック](#)
- [アーキテクト](#)
- [データベース](#)
- [Lightning](#)
- [モバイル](#)
- [インテグレーション](#)
- [セキュリティ](#)
- [ユーザーインターフェース](#)
- [ウェブサイト](#)

feedk

- [コミュニティ](#)
 - [開発者フォーラム](#)
 - [イベントカレンダー](#)
 - [開発者MVP](#)
 - [開発者グループ](#)
 - [Developer Success Stories](#)
 - [ブログ](#)
- [ブログ](#)
 - [日本語のブログ](#)
 - [デベロッパーリレーションズ](#)
 - [エンジニアリング](#)
 - [Force.com Labs](#)
 - [Tech Docs](#)
- [Trailhead](#)
 - [Trailhead](#)
 - [- トレイル](#)
 - [- モジュール](#)
 - [- プロジェクト](#)
 - [- Superbadges](#)
 - [- Trailblazers](#)
 - [ヘルプ](#)
 - [コミュニティに参加](#)

ログイン > サインアップ >



私の開発者アカウント > アカウントを作成 > 私の設定 > Admin Site > CMS >

ログアウト

サーチ



ログイン > サインアップ >



私の開発者アカウント > アカウントを作成 > 私の設定 > Admin Site > CMS >

ログアウト

- プロダクト

- Platform

- 概要
- [AppExchange](#)
- [Force.com](#)
- [Heroku](#)
- [Mobile](#)

Salesforce Apps

- [Analytics Cloud](#)
- [Community Cloud](#)
- [Marketing Cloud](#)
- [Pardot](#)
- [Quip](#)

Build Faster, Smarter & Together

- [Lightning](#)
- [Einstein](#)
- [Salesforce DX](#)

feedk



- リソース

- こちらをご覧ください

- [入門](#)
- [ドキュメント](#)
- [Trailhead](#)
- [書籍&チートシート](#)
- [オンデマンドWebセミナー](#)
- [認定資格](#)
- [ブログ](#)

feedk

ツール

- [Force.com IDE](#)
- [Lightning 設計システム](#)
- [データベース](#)
- [その他のツールやツールキット](#)

トピック別

- [アプリケーションの配布](#)
- [アプリケーションロジック](#)
- [アーキテクト](#)
- [データベース](#)
- [Lightning](#)
- [モバイル](#)
- [インテグレーション](#)
- [セキュリティ](#)
- [ユーザーインターフェース](#)
- [ウェブサイト](#)

- コミュニティ

- [開発者フォーラム](#)
 - [イベントカレンダー](#)
 - [開発者MVP](#)
 - [開発者グループ](#)

- [Developer Success Stories](#)
- [ブログ](#)
- [ブログ](#)
 - [日本語のブログ](#)
 - [デベロッパーリレーションズ](#)
 - [エンジニアリング](#)
 - [Force.com Labs](#)
 - [Tech Docs](#)
- [Trailhead](#)
 - [Trailhead](#)
 - [- トレイル](#)
 - [- モジュール](#)
 - [- プロジェクト](#)
 - [- Superbadges](#)
 - [- Trailblazers](#)
 - [ヘルプ](#)
 - [コミュニティに参加](#)



[Home](#)»[Salesforce Developers Blog](#)»Post entry

Salesforce Developers Blog



feedk

Queueable Apex: More Than an @future

The Salesforce Winter '15 platform release brought us the new Queueable Apex interface. This interface is a cool new way to execute asynchronous computations on the Force.com platform, given you already know @future, Scheduled Apex Jobs, and Batch Jobs. Here's a practical use case.

The [Winter '15](#) platform release brought us the new [Queueable Apex interface](#). This interface is a cool new way to execute asynchronous computations on the Force.com platform, given you already know @future, Scheduled Apex Jobs, and Batch Jobs.

The main differences between [@future methods](#) and Queueable Apex jobs are:

1. When you enqueue a new job, you get a job ID that you can actually monitor, like batch jobs or scheduled jobs!
2. You can enqueue a queueable job inside a queueable job (no more “Future method cannot be called from a future or batch method” exceptions). As for Winter ’15 release, you can chain a maximum of two queueable jobs per call (so a job can fire another job and that’s it!). With [Spring ’15](#) release, this limit has been removed.
3. You can have complex Objects (such as SObjects or Apex Objects) in the job context (@future only supports primitive data types)

All I want to do in this article is show a practical use case for this interface (for those impatient out there, the complete code of this article can be found [here](#)).

Business requirement: You have to send a callout to an external service whenever a Case is closed.

Constraints: The callout will be a REST POST method that accepts a JSON body with all the non-null Case fields that are filled exactly when the Case is closed (the endpoint the service will be a simple [RequestBin](#)). feedk

The Queueable Apex Use Case

Using a future method, we will pass the case ID to the job and so make a subsequent SOQL query: this is against the requirement to pass the fields we have in the Case at the exact time of the update. This might seem an excessive constraint, but with big ORGs and hundreds of future methods in execution (due to system overload), future methods can actually be executed after minutes, so the Case state can be different from when the future was actually fired.

To store the attempts of callout (and the responses, this is only a helper method that allows for reportization of the attempts) we will use a new SObject called Callout__c with the given fields:

- Case__c: master/detail on Case
- Job_ID__c: external ID / unique / case sensitive, stores the queueable job ID
- Sent_on__c: date/time, when the callout took place
- Duration__c: integer, milliseconds for the callout to be completed (we can report timeouts easily)

- Status__c: picklist, valued are Queued (default), OK (response 200), KO (response != 200) or Failed (exception)
- Response__c: long text, stores the server response

To achieve the Business needs, we need a Case trigger:

```

1 trigger CaseQueueableTrigger on Case (after insert, after update) {
2
3     List<Callout__c> calloutsScheduled = new List<>();
4     for(Integer i = 0; i < 50){
5         insert calloutsScheduled;
6     }
7 }

```

The trigger iterates bulkily through the trigger's cases and if they are created as "Closed" or the Status field changes to "Closed," a new job is enqueued and a Callout__c object is added to the list that will be inserted outside the "for."

This way we always have evidence on the system that the callout has been fired.

feedk

Remember that you can add up to 50 jobs to the queue with System.enqueueJob in a single transaction, so you have to be sure that the trigger makes a maximum of 50 "System.enqueueJob" invocations (this is up to you!).

Let's have a look at the job class:

```

1 public class CaseQueueableJob implements Queueable, Database.AllowsCallouts {
2     . . .
3 }

```

The Queueable interface is the main reason of this article, while the Database.AllowsCallouts allow us to send a callout inside the job.

The constructor of the class consists on a single class member assignment:

```

01 /*
02  * Case passed on class creation (the actual ticket from the Trigger)
03  */
04 private Case ticket{get;set;}
05
06 /*
07  * Constructor
08  */
09 public CaseQueueableJob(Case ticket){
10     this.ticket = ticket;
11 }

```

Finally, let's watch the main execute method of the job (the one that stores all the asynchronous logic):

```

01 // Interface method.
02 // Creates the map of non-null Case fields, gets the Callout__c object
03 // depending on current context JobID.
04 // In case of failure, the job is queued again.
05
06 public void execute(QueueableContext context) {
07     //1 - creates the callout payload
08     String reqBody = JSON.serialize(createFromCase(this.ticket));
09
10     //2 - gets the already created Callout__c object
11     Callout__c currentCallout = [Select Id, Status__c, Sent_on__c, Response__c,
Case__c,
12                                     Job_ID__c From Callout__c Where Job_ID__c =
:context.getJobId()];
13
14     //3 - starting time (to get Duration__c)
15     Long start = System.now().getTime();
16
17     //4 - tries to make the REST call
18     try{
19         Http h = new Http();
20         HttpRequest request = new HttpRequest();
21         request.setMethod('POST');
22         //change this to another bin @ http://requestb.in
23         request.setEndpoint('http://requestb.in/nigam7ni');
24         request.setTimeout(60000);
25         request.setBody(reqBody);
26         HttpResponse response = h.send(request);
27
28         //4a - Response OK
29         if(response.getStatusCode() == 200){
30             currentCallout.status__c = 'OK';
31         //4b - Reponse KO
32         }else{
33             currentCallout.status__c = 'KO';
34         }
35         //4c - saves the response body
36         currentCallout.Response__c = response.getBody();
37     }catch(Exception e){
38         //5 - callout failed (e.g. timeout)
39         currentCallout.status__c = 'Failed';
40         currentCallout.Response__c = e.getStackTraceString().replace('\n', ' / ')+'
- '+e.getMessage();
41
42         //6 - it would have been cool to reschedule the job again 😞
43         /*
44         * Apprently this cannot be done due to "Maximum callout depth has been
reached." exception
45         ID jobID = System.enqueueJob(new CaseQueueableJob(this.ticket));
46         Callout__c retry = new Callout__c(Job_ID__c = jobID,
47                                     Case__c = this.ticket.Id,
48                                     Status__c = 'Queued');
49         insert retry;
50         */
51     }
52     //7 - sets various info about the job
53     currentCallout.Sent_on__c = System.now();
54     currentCallout.Duration__c = system.now().getTime()-start;
55     update currentCallout;
56     //8 - created an Attachment with the request sent (it could be used to manually
send it again with a bonification tool)
57     Attachment att = new Attachment(Name = 'request.json',
58                                     Body = Blob.valueOf(reqBody),
59                                     ContentType='application/json',
60                                     ParentId = currentCallout.Id);
61     insert att;
62 }

```

feedk

These are the steps of execution:

- 1) Creates the JSON payload to be sent though the POST request (watch the method in the provided [github repo](#)) for more details (nothing more than a describe and a map).
- 2) Gets the Callout__c SObject that was created by the Case trigger (and using the context's Job ID).
- 3) Gets the starting time of the callout being executed (to calculate the duration).
- 4) Tries to make the rest call
 - a. Server responded with a 200 OK
 - b. Server responded with a non OK status (e.g. 400, 500)
 - c. Saves the response body in the Response__c field
- 5) Callout failed, so the Respose__c field is filled with the stacktrace of the exception (believe me this is super usefull when trying to get what happened, expecially when you have other triggers / code in the "try" branch of the code).
- 6) Unfortunately, if you try to enqueue another job after a callout is done, you get the "Maximum callout depth has been reached." exception; this is because you can have only two jobs in the queue chain that makes callouts, so if you queue another job with the Database.AllowsCallouts interface, you get this error. This way the job would have tried to enqueue another equal job for future execution.
- 7) Sets time fields on the Callout__c object.
- 8) Finally, creates an Attachment object with the JSON request done: this way it can be expected, knowing the precise state of the Case object sent, and can be re-submitted using a re-submission tool that uses the same code (it could be a Batch job for instance).

This is an example request (if you are curious about what I'm sending):

```
01 And this is an example request:
02 {
03     "values": {
04         "lastmodifiedbyid": "005w0000003fj35AAA",
05         "businesshoursid": "01mw00000009wh7AAA",
06         "casenumber": "00001001",
07         "ownerid": "005w0000003fj35AAA",
08         "createddate": "2015-01-20T09:54:17.000Z",
09         "origin": "Phone",
10         "isescalated": false,
```

```
11     "status": "Closed",
12     "accountid": "001w0000019wqEIAAY",
13     "systemmodstamp": "2015-01-20T19:33:31.000Z",
14     "isdeleted": false,
15     "priority": "High",
16     "id": "500w000000fqNRaAM",
17     "lastmodifieddate": "2015-01-20T19:33:31.000Z",
18     "isclosedoncreate": true,
19     "createdbyid": "005w0000003fj35AAA",
20     "contactid": "003w000001EetwEAAR",
21     "type": "Electrical",
22     "closeddate": "2015-01-20T19:19:51.000Z",
23     "subject": "Test queueable interface",
24     "reason": "Performance",
25     "potentialliability": "Yes",
26     "isclosed": true
27 }
```

As already written, the full code for this Queueable Apex use case, with the related metadata, is available on this [GitHub repo](#).

About the author

Enrico Murru ([enree.co](#)) is a Solution Architect and Senior Developer at [WebResul](#) ([Engineering Group](#)).

feedk





He started working on the Force.com platform in 2009 and since then he grew a big experience following the growth of the platform; he is also a huge Javascript lover. In the last years he began to write technical blog posts for the dev community trying to document his fun with the Force.com platform and more.

His daydream is to become the first italian Force.com evalgelist, sharing his passion to all developers, and spend the rest of his professional life (and more) to learn new techs and experimenting.

Published

By [Enrico Murru](#)
May 13, 2015

Topics:

You may also find interesting


Sorry, no related posts.

Leave your comments...

Queueable Apex: More Than an @future

3 Comments developer-relations  Login ▾

 Recommend 1  Share Sort by Best ▾




Join the discussion...

feedk


LOG IN WITH OR SIGN UP WITH DISQUS 

Name

- 


crop1645 • 2 years ago

Am I missing something or is the example trigger code at the top of the post missing some lines? (and the for loop is invalid syntax)

1 ^ | ▾ • Reply • Share ›
- 

Atul Rajguru • 2 years ago

@ step 5 your you are calling 1st call out. So you should be able to give one more right ? we can call 2 call outs right?

^ | ▾ • Reply • Share ›
- 

Bijay S • 2 years ago

Excellent Post Enrico!! Nice way of doing callouts from Queueable Interface.

^ | ▾ • Reply • Share ›

始めよう

- [Salesforce Platform](#)

- [Force.com](#)
- [Heroku](#)

Salesforce 開発センター

- [Lightning Developer Center](#)
- [Mobile Developer Center](#)
- [Heroku 開発センター](#)
- [Desk.com </developers>](#)
- [Pardot 開発者サイト](#)

開発者リソース

- [モバイルサービス](#)
- [Force.com ドキュメント](#)
- [Force.com ダウンロード](#)
- [Heroku ドキュメント](#)
- [Heroku ダウンロード](#)
- [Learn Salesforce with Trailhead](#)

コミュニティ

- [開発者フォーラム](#)
- [Salesforce 開発者イベント](#)
- [Web セミナー](#)

詳細はこちら

- [Salesforce AppExchange](#)
- [Salesforce Administrators](#)
- [Salesforce.com ヘルプポータル](#)

© Copyright 2000-2017 salesforce.com, inc. All rights reserved. 本サイトに記載された商標は、各商標所有者に帰属します。

Salesforce.com, inc. The Landmark @ One Market, Suite 300, San Francisco, CA 94105, United States

- [個人情報の取り扱いについて](#)
- [セキュリティについて](#)
- [利用規約](#)
- [フィードバック](#)
- [お問い合わせ](#)
- [言語: 日本語](#)
- [言語を選択](#)

- [English](#)
- [日本語](#)

feedk