Retry Pattern

Πρόβλημα:

Το Retry Pattern χρησιμοποιείται σε εφαρμογές οι οποίες τρέχουν σε υπηρεσίες Cloud και επομένως είναι επιρρεπείς σε παροδικά σφάλματα.

Αυτά τα σφάλματα είναι συνήθως αυτοδιορθούμενα και επιλύονται με την επανάληψη του αποτυχημένου αιτήματος.

Ορισμένα από αυτά είναι:

- Προσωρινή διακοπή σύνδεσης δικτύου.
- Προσωρινή έλλειψη διαθεσιμότητας μιας υπηρεσίας.
- Timeouts εξαιτίας πλήρως απασχολημένης υπηρεσίας.

Λύση (1/2)

- **Μη παροδικά σφάλματα:** Η εφαρμογή προτείνεται να ακυρώσει τη δραστηριότητα και να αναφέρει κατάλληλη εξαίρεση.
- **Ασυνήθιστα ή σπάνια σφάλματα:** Προκαλούνται υπό απρόσμενες συνθήκες και η εφαρμογή προτείνεται να αποστείλει αίτημα στην υπηρεσία ξανά άμεσα.
- **Συνηθισμένα σφάλματα σύνδεσης ή "busy" σφάλματα:** Η υπηρεσία ή το δίκτυο ίσως χρειάζονται ορισμένο χρόνο για την αποκατάσταση τους και επομένως η εφαρμογή θα πρέπει να περιμένει για κατάλληλο χρονικό διάστημα πριν αποστείλει ξανά αίτημα.

Λύση (2/2)

Για τα πιο συνηθισμένα παροδικά σφάλματα η χρονική περίοδος ανάμεσα στις επαναλήψεις των αιτημάτων θα πρέπει να επιλέγεται έτσι ώστε αιτήματα από διάφορα στιγμιότυπα του προγράμματος να ισομοιράζονται χρονικά.

Αυτό βοηθάει την αποφυγή βομβαρδισμού της υπηρεσίας με αιτήματα, κάτι το οποίο πιθανώς θα αύξανε τον χρόνο ανάκαμψης της.

Αν το αίτημα αποτύχει πάλι η περίοδος αναμονής ανάμεσα σε επαναληπτικά αιτήματα θα πρέπει να αυξηθεί (π.χ. αθροιστικά, exponential back-off).

Βήμα 1: Η εφαρμογή αποστέλλει αίτημα προς την υπηρεσία. Το αίτημα αποτυγχάνει και η υπηρεσία επιστρέφει HTTP σφάλμα 500.
Βήμα 2: Η εφαρμογή περιμένει για μικρό χρονικό διάστημα και ξαναπροσπαθεί. Το αίτημα αποτυγχάνει πάλι και η υπηρεσία επιστρέφει HTTP σφάλμα 500.
Βήμα 3: Η εφαρμογή περιμένει για αυξημένο χρονικό διάστημα και ξαναπροσπαθεί. Το αίτημα επιτυγχάνει και επιστρέφεται HTTP κωδικός 200 (OK).

Πρόσθετα Χαρακτηριστικά (1/2)

- Η πολιτική επανάληψης πρέπει να ρυθμίζεται ανάλογα με τις απαιτήσεις της επιχείρησης και την φύση των σφαλμάτων.
- Μια εξαιρετικά επιθετική πολιτική επανάληψης μπορεί να οδηγήσει σε υποβάθμιση μιας υπηρεσίας η οποία είναι κοντά στο ή έχει φτάσει το όριο της. Πρόσθετα, αυτό θα μπορούσε να επηρεάσει την απόκριση της εφαρμογής.
- Εάν το αίτημα αποτυγχάνει μετά από μεγάλο αριθμό επαναλήψεων ίσως είναι καλύτερος ο ορισμός μιας χρονικής περιόδου κατά την οποία δεν θα υπάρξουν επαναλήψεις.
- Η αποτυχία ενός αιτήματος μπορεί να οφείλεται σε ποικίλους λόγους και να οδηγεί σε διαφορετικές εξαιρέσεις. Πιθανώς είναι ωφέλιμο η πολιτική επανάληψης να προσαρμόζει την χρονική περίοδο μεταξύ επαναλήψεων ανάλογα με τον τύπο της εξαίρεσης.

Πρόσθετα Χαρακτηριστικά (2/2)

- Εάν ένα αίτημα αποτελεί μέρος μιας ευρύτερης συναλλαγής πρέπει να ληφθεί υπόψιν ο βαθμός στον οποίο επηρεάζεται αυτή στο σύνολο της. Η πολιτική επανάληψης θα πρέπει να μεγιστοποιεί την πιθανότητα επιτυχίας της συναλλαγής και να μειώνει την πιθανότητα αναίρεσης όλων των προηγούμενων βημάτων αυτής.
- Ο κώδικας επανάληψης πρέπει να είναι πλήρως ελεγμένος για ποικιλία συνθηκών αποτυχίας. Δεν πρέπει να επηρεάζει σε μεγάλο βαθμό την απόδοση ή την αξιοπιστία της εφαρμογής, να παράγει μεγάλο φόρτο για τις υπηρεσίες ή να οδηγεί σε bottlenecks.
- Η εφαρμογή της λογικής των επαναλήψεων πρέπει να γίνεται όταν είναι πλήρως κατανοητό το περιβάλλον του αποτυχημένου αιτήματος.
- Είναι σημαντικό να καταγράφονται όλες οι αποτυχίες σύνδεσης που προκαλούν επανάληψη αιτήματος ώστε να ανιχνεύονται τα υποβόσκοντα προβλήματα.
- Είναι σημαντική η ερεύνα των αιτιών ενός σφάλματος για την εξαγωγή συμπεράσματος σχετικά με το αν είναι πιθανό το σφάλμα αυτό να είναι μακράς διαρκείας ή τερματικό.

Περιπτώσεις Χρήσης

Σε περίπτωση που η εφαρμογή είναι πιθανό να αντιμετωπίσει παροδικά σφάλματα ενώ προσπαθεί να επικοινωνήσει με μια απομακρυσμένη υπηρεσία ή έναν απομακρυσμένο πόρο. Τέτοια σφάλματα έχουν συνήθως μικρή διάρκεια ζωής και επιδιορθώνονται σε επόμενη προσπάθεια αιτήματος.

Περιπτώσεις Πιθανής Αποφυγής Χρήσης

- Για σφάλματα με πιθανόν μεγάλη διάρκεια ζωής.
- Για αποτυχίες οφειλόμενες σε μη παροδικά σφάλματα (π.χ. λογικά σφάλματα του προγράμματος)
- Ως εναλλακτική για την αντιμετώπιση μη επαρκών πόρων (π.χ. χωρητικότητα server)

Παράδειγμα (1/2)

Στο συγκεκριμένο παράδειγμα, βλέπουμε μια υλοποίηση του Retry Pattern σε C#, από την οποία έχουνε παραληφθεί οι λεπτομέρειες της TransientOperationAsync().

Η κλήση της TransientOperationAsync() γίνεται μέσα σε ένα try block το οποίο με την σειρά του βρίσκεται μέσα σε ένα infinite loop. Έξοδος από αυτή την επανάληψη πραγματοποιείται εφόσον η TransientOperationAsync() δεν παρουσιάσει exception.

Εάν η TransientOperationAsync() παρουσιάσει exception τότε μέσα στο catch block ελέγχεται ο αριθμός επαναλήψεων που έχει γίνει ήδη και το αν το exception αυτό είναι παροδικό.

Σε περίπτωση που ο αριθμός των επαναλήψεων είναι μεγαλύτερος από αυτόν που έχει οριστεί ή το exception δεν είναι παροδικό η OperationWithBasicRetryAsync() παράγει exception το οποίο πρέπει να χειριστεί ο κώδικας που την καλεί.

Διαφορετικά, υπάρχει μια ορισμένη από τον προγραμματιστή αναμονή πριν να γίνει καινούρια επανάληψη.

```C#
private int retryCount = 3;
...

public async Task OperationWithBasicRetryAsync()
{
  int currentRetry = 0;

  for (; ;)
  {
    try
    {
      // Calling external service.
      await TransientOperationAsync();

      // Return or break.
      break;
    }
    catch (Exception ex)
    {
      Trace.TraceError("Operation Exception");

      currentRetry++;

      // Check if the exception thrown was a transient exception
      // based on the logic in the error detection strategy.
      // Determine whether to retry the operation, as well as how
      // long to wait, based on the retry strategy.
      if (currentRetry > this.retryCount || !IsTransient(ex))
      {
        // If this is not a transient error
        // or we should not retry re-throw the exception.
        throw;
      }
    }


    // Wait to retry the operation.
    // Consider calculating an exponential delay here and
    // using a strategy best suited for the operation and fault.
    Await.Task.Delay();
  }
}

// Async method that wraps a call to a remote service (details not shown).
private async Task TransientOperationAsync()
{
  ...
}
```

Παράδειγμα (2/2)

Η μέθοδος IsTransient() είναι αυτή που ελέγχει αν κάποιο exception μπορεί να θεωρηθεί παροδικό. Είναι τύπου bool και δέχεται ως όρισμα αντικείμενο τύπου Exception.

Αρχικά γίνεται ένας έλεγχος για το αν η εξαίρεση είναι τύπου OperationTransientException οπότε και επιστρέφεται true.

Στη συνέχεια, γίνεται προσπάθεια μετατροπής (cast) της εξαίρεσης ως WebException. Σε περίπτωση αποτυχίας επιστρέφεται false.

Αν η μετατροπή είναι επιτυχής ελέγχεται το αν η εξαίρεση έχει προκληθεί από: ConnectionClosed, Timeout, RequestCanceled και επιστρέφεται αντίστοιχα true ή false αν πρόκειται για διαφορετική αιτία.

```C#
private bool IsTransient(Exception ex)
{
  // Determine if the exception is transient.
  // In some cases this may be as simple as checking the exception type, in other
  // cases it may be necessary to inspect other properties of the exception.
  if (ex is OperationTransientException)
    return true;

  var webException = ex as WebException;
  if (webException != null)
  {
    // If the web exception contains one of the following status values
    // it may be transient.
    return new[] {WebExceptionStatus.ConnectionClosed,
                  WebExceptionStatus.Timeout,
                  WebExceptionStatus.RequestCanceled }.
            Contains(webException.Status);
  }

  // Additional exception checking logic goes here.
  return false;
}
```

Πηγές

Cloud Design Patterns, Prescriptive Architecture Guidance For Cloud Applications, Microsoft (εικόνες και πληροφορίες από το κεφάλαιο “Retry Pattern”.