◇目的

Iteratorは、意味として「繰り返す」という意味があり、日本語にすると「反復子」などと呼ばれます。
Iteratorパターンは、GoFで、
集約オブジェクトが基にある内部表現を公開せずに、その要素に順にアクセスする方法を提供する。
と定義されています。
Iteratorパターンは、
「順次アクセスする方法をがうぶクラスに委譲すること」
です。

◇効果

Iteratorを利用する側は、オブジェクトの内部を意識することなく要素にアクセスできます。
異なる内部構造を持つリストの要素にもアクセスできる。

走査する処理はすべて外部クラス内にあるので、走査したい要素が有るクラス内で処理を書く必要がなくなり、肥大化、柔軟性の消失を防ぐことができます。

処理の変更も、極めて限定的とすることができます。

集合要素に複数の順次アクセス方法を提供することが可能で、走査処理の実装を変更することで、昇順、逆順、直接指定などをすることができます。

◇背景

Iteratorパターンは、
・要素の集まったオブジェクト(配列など)にアクセスする。
・集合の要素に順にアクセスする必要がある。
という場面で使用に使用します。

◇Iterator(イテレータ)パターンの実際のコードと考え方

Iteratorパターン使い集合オブジェクトにアクセスする考え方は次のようになる。

```
Iteratorクラス
{
                集合オブジェクトを操作する処理。
                next()・・・次の要素が有るか確認
                has next()・・・要素の取得
}
```

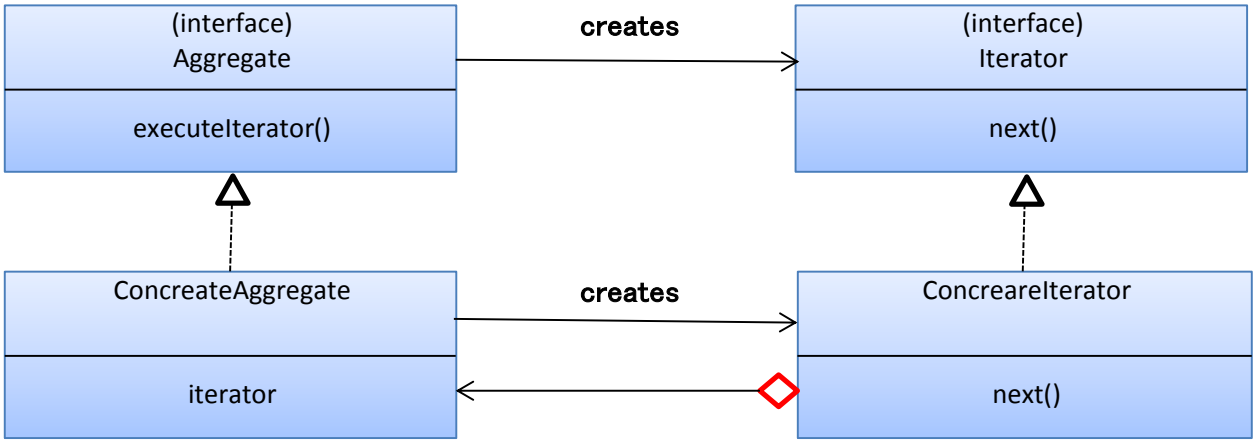
実際に実装していくと次のようになる

```
class Iterator

 @index = 0
 @array = nil

 #集約オブジェクト
 def initialize(arrayclass)
    #集約オブジェクト代入
    @array = arrayclass
    #要素の始まり0を代入
    @index = 0
 end

 #次の要素が有るかの確認
 def hasNext()
      #要素番号が要素の長さより大きいか小さいか
      if @index < @arrayname.getLength()
```

```
#次の要素の取得
def next()
    #次の要素が有った場合、その要素を取得する。
    contents = self.has_next ? ? @array.getarrayAt(@index); nill
    #要素に1を足す
    @index += 1

    return contents
    end
end
```

集約オブジェクトを取得し、要素数を0に設定する。次に、集約オブジェクトに次の要素の有無を確認し、有った場合は、要素の内容を取得し要素数をインクリメントする。

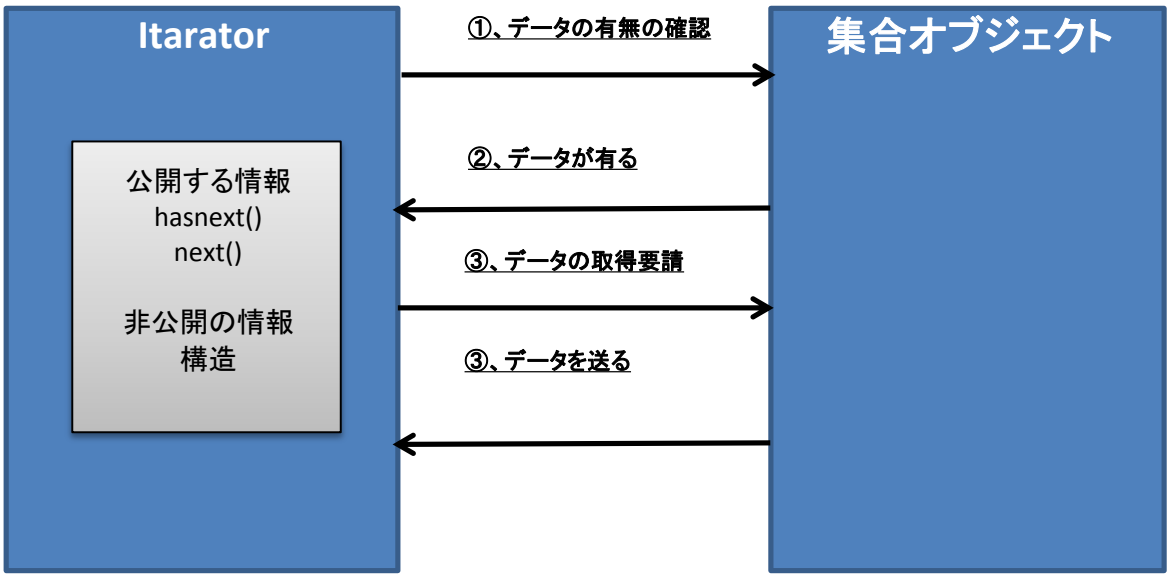
◇Itaretorパターンクラス図



◇問題点の改善

Iteratorに渡すデータを条件に合わせてソートしておくと、様々な所でつかうことができます。。

◇Iterator(イテレータ)パターンのまとめ



◇注意

Iteratorパターンを使う場合、getAtメソッド実装が簡単なものには使わない方よいです。
また、ランダムアクセスするようなものには使用できません。

◇総括

Iteratorパターンを使うと、内部構造を公開することなく相手に集約オブジェクトに順次アクセスすることができます。
大きな集合データだった場合は、後ろでIteratorを走らせておくことで利用者はデータに柔軟にアクセスすることができます。