B-011

Worker Thread デザインパターンの分散化
Distributed Extension of Worker Thread design pattern

武笠 寛幸 † 吉田 紀彦 †
Hiroyuki Mukasa Norihiko Yoshida

1. はじめに

並行処理用デザインパターンの一つである Worker Thread パターン [1,2] は，マルチスレッドを利用したクライアントサーバシステムの構築法を提供するデザインパターンであり，スレッドを再利用することでパフォーマンスの低下を抑え，且つ高い応答性を備えている．しかし，Worker Thread パターンは単一計算機での動作を前提としており，実際にサーバとして使用する場合は高性能な (高価な) 計算機を用意する必要がある．

そこで，本研究では既存の Worker Thread パターンを拡張し，複数の計算機を利用したクライアントサーバシステムを構築するためのデザインパターンを提案する [3]．複数の計算機で並列処理を行うことにより，高性能なシステムを構築することを目指す．

2. Worker Thread パターン

まず，既存の Worker Thread パターンについて説明する．図 1 に Worker Thread パターンのクラス図を示す．

Channel は Client-Worker 間で Request の受け渡しを安全に行うための仕組みを提供するクラスである．Channel は Client からの Request を保持するための requestQueue フィールド，requestQueue を操作するための putRequest，takeRequest メソッドを持つ．これら 2 つのメソッドは synchronized で宣言されており，スレッドの排他制御を行うことで requestQueue の安全性を確保している．

Client は Channel に仕事を依頼するクラスである．Client は Request を生成し，Channel の putRequest メソッドを呼び出し，requestQueue に Request を格納する．仕事の依頼は以上で終了し，Client は即座に自分の仕事に戻ることが出来るため，応答性に優れたシステムになっている．

Worker は Client からの Request を処理するクラスである．まず，Worker は Channel の takeRequest メソッドを呼び出し，Request を取り出す．次に，取り出した Request の execute メソッドを呼び出し，Request を処理する．処理を終えた Worker は再び takeRequest メソッドを呼び出し，以降は上記の動作を繰り返す．Worker Thread パターンでは Worker を再利用しているため，Request 毎に Worker を生成するシステムと比べて，パフォーマンスに優れている．

3. Worker Sender パターン (提案パターン)

Worker Thread パターンは単純化のために，Client に処理結果を返信しない．また，Client-Channel 間のネットワーク通信に関しても考慮されていない．そこで，以下の 3 つの機能を追加する．

†埼玉大学 Saitama University

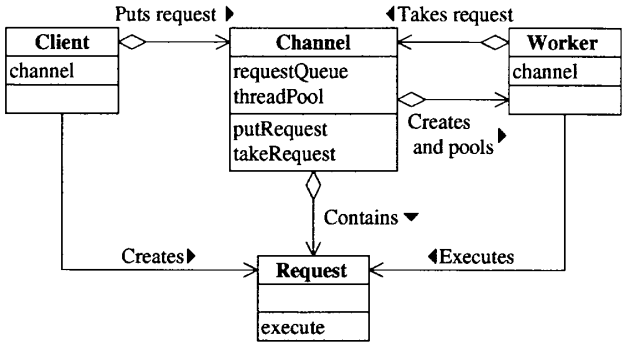

図 1: Worker Thread パターンのクラス図

- 処理結果を返信する機能
- ネットワーク通信機能
- 負荷分散機能

上記の機能を追加した提案パターンを Worker Sender パターンと呼ぶことにする．図 2 に Worker Sender パターンのクラス図を示す．

3.1 処理結果を返信する機能

処理結果を返信するための準備として，Request，Channel，Worker の 3 つのクラスを拡張する．

まず，Request クラスに処理結果を格納するための result フィールド，Client のアドレス (IP アドレス+ポート番号) を格納するための clientData フィールドを追加する．次に，Channel クラスに resultQueue フィールド，putResult，takeResult メソッドを追加する．これらは requestQueue，putRequest，takeRequest と同様の構造になっている．最後に，Worker の動作内容に「Request の処理後に putResult メソッドを呼び出し，処理結果を Channel に格納する動作」を追加する．

実際に Client に処理結果を返信する機能については，次節で説明する．

3.2 ネットワーク通信機能

Worker Thread パターンでは，Client は putRequest メソッドを呼び出すことで，Request の処理を依頼していた．しかし，Client は別の計算機上に存在する Channel の putRequest メソッドを直接呼び出すことが出来ない．そのため，Socket を用いたネットワーク通信によって Request の受け渡しを行うようにする．

まず，Client からの Socket 接続要求を受け付ける Accepter クラスを追加する．Accepter は Client から Socket 接続要求を受け取り，Client との間にコネクションを生成する．Client から Request を受け取ると Channel の putRequest メソッドを呼び出し，Request を格納する．

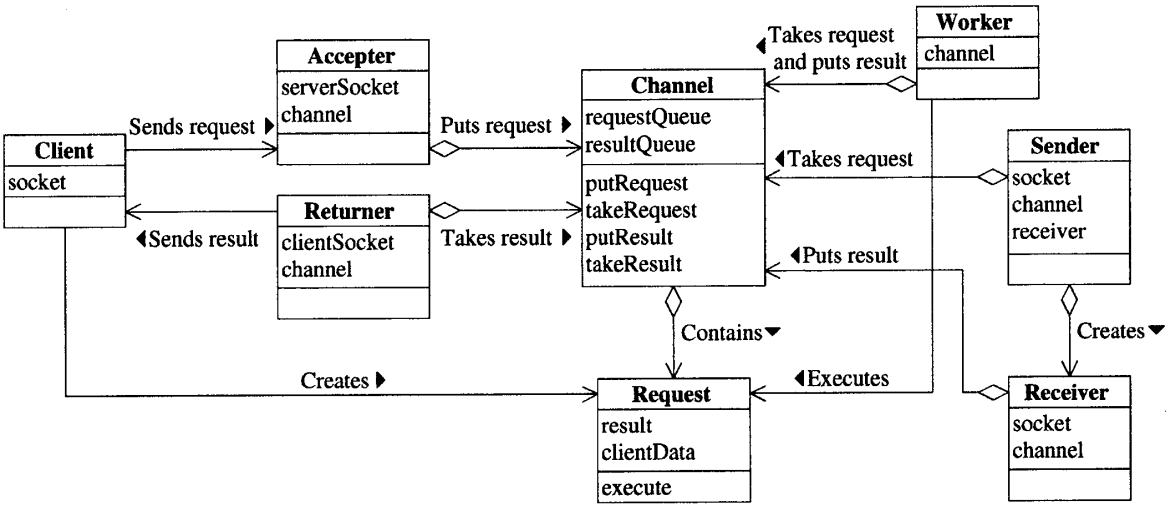

図 2: Worker Sender パターンのクラス図

さらに，Client に処理結果を返信する Returner クラスを追加する．Returner は Channel の takeRequest メソッドを呼び出し，Request を取り出す．Request の clientData フィールドから Client のアドレスを取り出し，Socket 通信により処理結果を返信する．

3.3 負荷分散機能

複数の計算機を利用して高性能なシステムを構築するために，負荷分散機能を追加する．負荷分散を行うために，他計算機 (サーバ) に Request を転送する Sender クラスを追加する．

まず，Sender は Socket を生成し，他サーバの Accepter に接続する．次に，Channel の takeRequest メソッドを呼び出し，Request を取り出す．そして，他サーバに Request を転送する．この時，Sender は

$$\frac{\text{Accepter が受け取った Request(未処理) の数}}{\text{Sender の数}+1}$$

以上の Request を送信しない．これは，各サーバで Request を等分させるために課している制限である．

また，処理結果を受け取るための Receiver クラスを追加する．Receiver は他サーバから処理結果を受け取り，Channel の putResult メソッドを呼び出し，Request を格納する．

4. シミュレーション

計算機シミュレーションにより，提案するデザインパターンの検証を行った．シミュレーションには性能の等しい 4 台の PC を使用した．

4.1 実験方法

まず，Client に 10,000 個の Request を生成させる．Client はある 1 台の PC(サーバ) に Request を送信する．Request を受け取った PC では，3 つの Sender が残りの 3 台の PC に対してそれぞれ Request を転送する．全ての Request が処理された後，各 PC が処理した Request の数を比較した．

4.2 実験結果

各 PC で処理された Request 数には多少の差があるが，全体的に理想的な結果となった．

表 1: 実験結果

PC 番号	処理した Request の数
PC1	2582
PC2	2511
PC3	2452
PC4	2455

5. まとめ

本研究では，Worker Thread パターンを拡張した Worker Sender パターンを提案した．Worker Sender パターンは，Sender が Request を複数の計算機に分散させ並列処理を行うことで，高性能なクライアントサーバシステムを構築することが可能となっている．

今後の課題としては，分割可能な Request の作成が挙げられる．現在のシステムでは，多数の軽い処理の Request についてはパフォーマンスが優れているが，少数 (例えば 1 つ) の重い処理の Request については計算機 1 台の場合とパフォーマンスが変わらない．そこで，予め分割可能な Request を用意しておくことで重い処理の Request を分割し，負荷分散を行うことによりパフォーマンスの向上を図る予定である．

参考文献

[1] Doug Lea：Concurrent Programming in Java, Addison-Wesley，1999

[2] 結城浩：Java 言語で学ぶデザインパターン入門【マルチスレッド編】，ソフトバンク，2002

[3] 武笠寛幸：Worker Thread デザインパターンの分散化，埼玉大学卒業論文，2005