閉じ込め型を用いたリージョンマネージャに基づくセキュア Java/LR

石井 祐平　伊藤 貴康
東北大学大学院情報科学研究科

1 はじめに

近年のネットワークの発達に伴い複数の計算機上に分散する資源にアクセスしたり, 異なる環境を結び付ける分散システム技術は非常に重要なものとなっている. Java 言語 [1] はインターネット環境で広く用いられているコンパクトなオブジェクト指向言語であり, 分散処理のための言語として優れた機能を備えている.

セキュアな分散システムを実現するため, ロケーションとリージョンという場所を表す概念を Java に導入した mJava/LR[2] が設計, 試作された. mJava/LR は, C 言語を用いて実装されているため JVM に基づく既存の Java システムとの互換性に欠ける問題がある.

JVM 上で動作する場所の概念を備えた Java 言語 Java/LR が筆者等の研究室で作成されてきた [5]. しかし場所の概念に伴う構成要素を Java のクラスライブラリとして実装しただけでは, 既存の Java システムと同様のセキュリティに関する問題を抱える. このため,Java/LR によるセキュリティ確保の根幹となるリージョンマネージャの保護を閉じ込め型 (Confined Types[3]) によって実現することが課題となっていた.

Java/LR のセキュリティを制御する単位であるリージョンにおけるセキュリティの方針を決定するのがリージョンマネージャである. 外部からリージョンマネージャを変更されることのないよう保護するために, 指定した型の値をパッケージ外に漏らさないことを静的に保証するのに閉じ込め型を用いたセキュア Java/LR システムを作成したので報告する.

2 Java/LR の概要

Java/LR には次のような概念や機能が導入されている.

- **ロケーション**

ロケーションはネットワーク上の場所を表すものであり,Java/LR では JVM 毎に 1 つのロケーションが与えられる. ロケーションを用いてリモートオペレーションやマイグレーションが実現されている.

- **リージョン**

リージョンはリージョンマネージャによって規定, 管理されるロケーションの集合であり,Java/LR におけるセキュリティ設定の単位となるものである. リージョンマネージャには認証機構やアクセス制限が備えられ, それらによってセキュリティ機能が実現される.

- **場所の概念を持つオブジェクト**

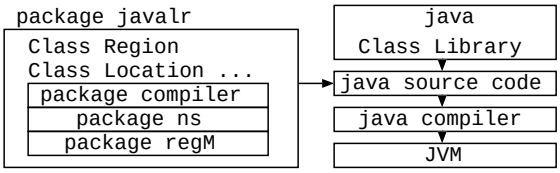

図 1: Java/LR の用法

Java/LR ではロケーション上で生成したオブジェクトに場所の概念を持たせることが可能である. このオブジェクトへの参照は (Java/LR コンンパイラによって生成される)Proxy オブジェクトを用いて実現される.

ロケーションとリージョンという場所の概念を Java に導入した Java/LR システムには次の特徴がある.

- **オブジェクトのアクセス制限**

ロケーション上のオブジェクトは, どのロケーションからもアクセス可能な public クラス,1 つのリージョン内のみでアクセス可能な region クラス,1 つのロケーション (JVM) 内のみでアクセス可能な location クラス, オブジェクトを生成したロケーションのみでアクセス可能な only クラスからなる. それらを用いて他のロケーションとのオブジェクトの共有とアクセス制限が行われる.

- **非同期メソッド 呼び出し**

通常の Java におけるメソッド呼び出しは, メソッドを呼び出してリターンするまでブロックする同期通信であるが,Java/LR ではメソッド呼び出しと戻り値の受け取りを分割する非同期通信を導入している. メソッドが委託した計算結果を必要とした際, ブロックして戻り値を取得することを可能にするために Future オブジェクトが導入されている. これを用いると非同期呼び出しを行った際には直ちに値が返さる. 計算途中に Future 値を他のメソッドに渡すことなども可能である.

- **リモート オペレーション**

Java/LR では, ロケーションおよびリージョンを指定して処理を行わせることが可能である. リモートオペレーションを行う際は, 処理を行わせるクラスにインターフェイス ROP をインプリメントさせ, 処理の内容を処理を行う場所とともに Do クラスの at メソッドに渡すことによって mJava/LR における $\underline{\text{do}}\{S\}\underline{\text{at}}(loc)$ 構文の機能を実現している.

- **リージョンマネージャによるセキュリティ設定**

各リージョンは必ず 1 つのリージョンマネージャを持

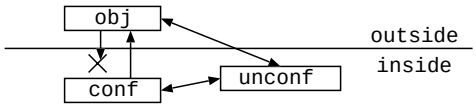

図 2: パッケージ間の参照

ち, これを用いてリージョン内外のロケーション間での通信を管理し, セキュリティの設定を行う. 異なるリージョンに属するロケーションの通信をマイグレーションやリモートオペレーションによって行う際はリージョンマネージャによるチェックを受け, オブジェクトの侵入, 退出を許可するかどうかの判定が行われる. リージョンマネージャはユーザが定義, 設定可能である. リージョンを階層的に構築することにより, セキュリティをより堅牢なものにすることができる.

Java/LR は mJava/LR の failure 機能以外の機能を全てクラスライブラリとして実現しており, 図 1 のように Java/LR のパッケージ群をインポートすることによって利用される.Java/LR のパッケージには次のようなものがある.

- `javalr`: Java/LR の基本的なパッケージ. ロケーションを表す Location クラス, リージョンを表す Region クラスなどを含む.
- `javalr.compiler`: オブジェクトに場所の概念を持たせるための Proxy オブジェクトを生成するコンパイラを表す Compiler クラスなどを含むパッケージ.
- `javalr.ns`: Java/LR におけるネームサーバ機能を実現するために用いるパッケージ. ロケーション, リージョン等の名前を管理するためのネームサーバを表す Server クラス, ネームサーバにアクセスするためのネームサーバオブジェクトを表す NameServer クラスなどを含む.
- `javalr.regM`: 各種のリージョンマネージャを定義するパッケージ. リージョンマネージャの基となる DefaultRegionManager などを含む.

3 リージョンマネージャの閉じ込め型による保護

Java/LR においては, 異なるリージョン間でのオブジェクトの侵入, 退出等の許可は全てリージョンマネージャによって行われる. このリージョンマネージャの値がもし悪意のあるユーザによってリージョン外部から参照され, 書き換えられると, 害のあるオブジェクトの送信, 実行などが各リージョン内で自由に行われることとなり,Java/LR のセキュリティは喪失してしまう. これを防ぐために, リージョンマネージャを閉じ込め型によって保護する.

3.1 閉じ込め型

もしある型のインスタンスへの全ての参照が 1 つのパッケージからのみであるとき, その型は参照が行われているパッケージ内における**閉じ込め型**であると言われる. 閉じ込め型のパッケージ間の参照関係を図 2 に示す. パッケージ inside の conf が閉じ込め型である. 図に表されているように, 外部のパッケージから conf への参照は不可であり, その他は全て参照が可能である.

閉じ込め型のプロパティを実現するために, 閉じ込め型として宣言されるクラスやインタフェースには以下のような制限が課せられる.

- 閉じ込め型のクラスやインタフェースは public や protected で宣言されてはならない. 属するパッケージを明示されなければならない.
- 閉じ込め型のサブクラスは閉じ込め型でなければならない. また, スーパークラスと同じパッケージに属さなければならない.
- 閉じ込め型から閉じ込め型でない型への, 代入, メソッド呼び出し引数, リターン命令文, 明示的なキャストによる参照の拡張は禁じられている.
- 閉じ込め型のオブジェクトにおいて呼び出されるメソッドは閉じ込め型クラスで定義されるか匿名メソッドでなければならない.
- 閉じ込め型クラスのコンストラクタによって呼び出されるコンストラクタは閉じ込め型クラスで定義されているか匿名コンストラクタでなければならない.

匿名メソッドとは, 呼ばれる型のアイデンティティを外部に漏らさないことを保証するものである. 閉じ込め型はこの匿名メソッドのみ呼び出すことができる.

また, 閉じ込め型のプロパティが満たされているかどうかの判定は CoffeeStrainer[4] というツールを用いて行う. まずソースコードにおいて閉じ込め型であると宣言するもの, 匿名メソッドであるとするものに定められたタグをつけ, それらのソースコードが含まれるパッケージに対してツールによる判定を実行する. 判定を実行すると, まず CoffeeStrainer はパッケージに含まれるクラスとパッケージに関連するクラスをコンパイルし, それを基に AST(Abstract Syntax Tree) を作成する. 例えば以下のようなプログラム

```
package p;
class A{
  B f;
}
class B {
  A g = new A();
}
```

からは、図 3 のような AST が作成される.

このような AST を用いてオブジェクトの参照関係を明らかにし, 閉じ込め型, 匿名メソッドであるとして宣言したクラスが上記の制限を違反していないかを CoffeeStrainer にインプリメントされている閉じ込め型に関する制限を用いてチェックする. そこでエラーが検出されなければそのパッケージ内で宣言されている閉じ込め型, 匿名メソッドはそのプロパティを満たし

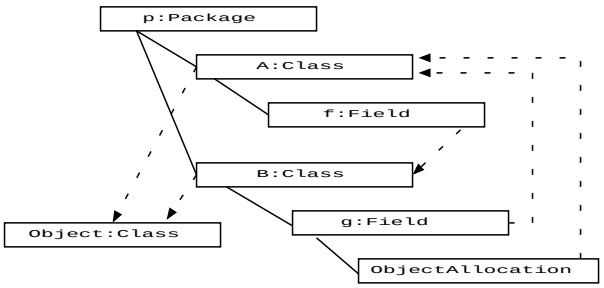

図 3: AST の例

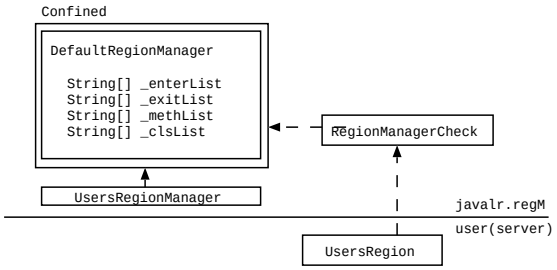

図 4: 閉じ込め型によるリージョンマネージャの保護

ていることになる. この判定は静的に行うのでランタイムのオーバーヘッドは起こらない. また, 閉じ込め型の違反に関する例外も考慮に入れなくてよいことになる. このツールを用いて Java/LR に閉じ込め型を導入する.

3.2 Java/LR への閉じ込め型の導入

リージョンマネージャに関するクラスはパッケージ regM 内に存在する. regM 内には DefaultRegionManager クラスがあり, このクラス全てのリージョンマネージャの基となるクラスである. リージョンマネージャは生成時に侵入, 退出, インスタンス生成, メソッドの実行の許可をするリストを与えることによって生成される. リージョンマネージャは動的に設定可能なものとなっている. あらたにリージョンマネージャを定義する際はこのクラスのサブクラスとして定義する.

3.3 閉じ込め型を用いた保護

クラス DefaultRegionManager に保持されているセキュリティポリシーを決定する各種のリストの値がパッケージの外部に漏れ, 書き換えが行われると, リージョンのセキュリティのプロパティは損われてしまう. これを防ぐために閉じ込め型を導入する. 閉じ込め型を用いたクラスの構成を図 4 に示す. 既存のクラス DefaulRegionManager を閉じ込め型として宣言し, 次のように実装する.

```
class DefaultRegionManager implements
  confined.ConfinedType,
  confined.HasAnonymousMethods{
   String[] _enterList;
   String[] _exitList;
   .../* 各種 String 型配列 */
   /*:anon*/.../*各種コンストラクタ*/
   /*:anon*/.../*各種メソッド*/
}
```

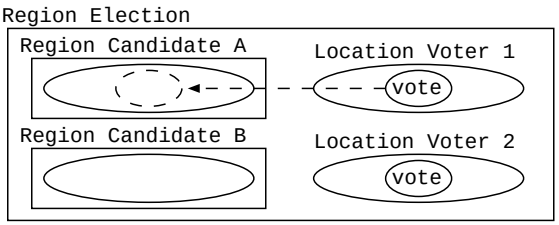

図 5: 選挙投票の例

ここで CoffeeStrainer による判定をパッケージ regM に対して行い,AST を作成して閉じ込め型として宣言した DefaultRegionManager クラスがパッケージ regM の外部に漏れるような命令がパッケージ内に含まれていないかをチェックすると閉じ込め型の違反は検出されないことが確かめられている. また,

- 各種 String 配列はインスタンス変数であり, オブジェクトを生成しないとアクセスすることはできない.
- DefaultRegionManager のオブジェクトへの参照がパッケージ外部に漏れることはない.

以上より,String 配列の値がパッケージの外部に漏れることはなく, 閉じ込め型を用いたリージョンマネージャの重要な値が外部に漏れないことが保証される.

4 システム構築の例

Java/LR の機能を用いて構築するシステムの例として, ネットワークを介した選挙の投票の例を図 5 に示す.

ネットワークを介した選挙投票において確保されるべきセキュリティのプロパティは次のようなものである.

- 関係のない第 3 者からいかなる要求も受けつけない.
- 投票者の 2 重投票を受けつけない.
- 関係者による妨害 (投票された票の他の候補者への移動等) を防ぐ.
- 投票期間外の投票は受けつけない.

これらの要求を満たすシステムとして, 図 5 のような 3 つのリージョンからなるシステムを閉じ込め型を備えた Java/LR を用いて構築する.

票を表すクラス Vote と各リージョンで用いるリージョンマネージャはあらかじめ作成する. 例えばリージョン Candidate のリージョンマネージャCandidateRegM は以下のように作成される.

```
DefaultRegionManager candRegM
  = new DefaultRegionManager();
Vote vote = new Vote();
candRegM.addEnter(vote);
Register r = new Register();
r.regist("CandidateRegM",candRegM);
```

リージョンマネージャを表すクラスとして閉じ込め型を用いて保護された DefaultRegionManager を用いる. addEnter() はリージョン内への侵入を許可するオブジェクトを追加するメソッドであり, ここでは票オブ

ジェクトを指定している. その後作成したリージョンマネージャを Register クラスを用いて登録する。

上記のように記述することで,Vote クラスの侵入のみを許可するリージョンマネージャを作成し登録する. 同様にリージョン Election のリージョンマネージャ ElectionRegM も閉じ込め型を用いて保護された DefauleRegionManager クラスを用いて作成, 登録を行う. それは, 外部からの全ての要求を受けつけないリージョンマネージャとして設定する. ここで, リージョンマネージャに関するクラスは閉じ込め型の制限により外部から参照することは不可であるため, それらが属するパッケージ regM 内に上記のコードを含んだプログラムを記述し実行する.

リージョンとロケーションの登録のため, パッケージ ns の Server クラスを実行することでネームサーバをスタートさせる. このホストのアドレスは host であるとする. 各リージョンをリージョンの名前と用いるリージョンマネージャの名前を与えて, 次のように作成する.

```
NameServer ns = new NameServer("host");
String elecRegM = "ElectionRegM";
String candRegM = "CandidateRegM";
Region Elec = new Region(elecRegM);
Region CandA
= Elec.makeSubRegion("CandA",candRegM);
Region CandB
 = Elec.makeSubRegion("CandB",candRegM);
Elec.startRegion();
ns.bind("Election", Elec);
ns.bind("CandA", CandA);
ns.bind("CandB", CandB);
```

makeSubRegion() はリージョン内にリージョンを作成するためのメソッドである. 作成したリージョンをスタートさせネームサーバに登録する.

ロケーションは JVM を備えたホスト毎に, 例えば候補者 A の場合次のように登録する.

```
1: NameServer ns = new NameServer("host");
2: Location candA = Location.getInstance();
3: cand.start();
4: Region reg = ns.findReg("CandidateA");
5: candA.addRegion(reg);
6: ns.bind("candidateA", candA);
```

まず getInstance() メソッドによってロケーションオブジェクトを取得しロケーションをスタートさせる. 所属するリージョンを設定しネームサーバに登録する.

実際の投票は Vote クラスから Java/LR コンパイラを用いて Vote_Proxy クラスを生成し, この際 Vote_Proxy クラス内に作成される go メソッドを用いてオブジェクトをマイグレーションすることによって行う.

リージョン Election のリージョンマネージャElection-RegM は外部からの全ての要求を許可しないことにより第 3 者の介入を防ぐ. リージョン Candidate のリージョンマネージャCandidateRegM は票オブジェクト Vote の侵入のみ許可し流出は許可しないため, 投票された票の移動は不可能である. また, マイグレーションさせた票オブジェクトを再度参照しようとすると Java/LR の例外である ObjectHasMovedException がスローされる. よって 2 重投票を回避することができる. 投票期間が過ぎたら Candidate のリージョンマネージャCandidateRegM を, 上で登録した際と同様に外部からアクセスできないものに動的に変更し, 期間外の投票を受けつけないようにすることができる.

Java/LR における処理時間については, 同一リージョン内のリモートメソッド呼び出しの処理時間は通常の Java 処理系のものより 1.5〜2.0 倍程度遅くなっている.

リージョンマネージャの判定を伴う処理として上記のシステムのマイグレーションの処理時間を測定したところ, 閉じ込め型を用いない場合に比べて閉じ込め型導入後は 1.5 倍程度遅くなっている. リージョンマネージャの参照を間接的なものに変更したことが原因であると考えられる.

5 おわりに

閉じ込め型を導入することで静的にセキュリティ保護を行う場所の概念を備えたセキュア Java/LR システムについて報告した. 閉じ込め型によるリージョンマネージャの保護によってセキュアなネットワークの構築が可能になる. なお, ユーザが使用する際の利便性を考慮すると, リージョンマネージャへの閉じ込め型の導入において動的に拡張できるリストを使用できるようにすることが望まれる. また,mJava/LR で導入されている failure 機能を本稿のセキュア Java/LR システムに導入することも検討している.

謝辞: 本稿のセキュア Java/LR システムは有賀透氏[5] によって作成された Java/LR を基に, 閉じ込め型を導入して拡張したものである. また, 討論頂いた東北大学伊藤研究室の宮川氏, 森谷氏, 乗本氏に謝意を表する.

参考文献

1. James Gosling,Bill Joy,Guy Steele. *The Java LanguagesSpecification*. Addison-Wesley Publishing(1996).
2. 渡辺昌寛, 伊藤貴康. 「場所の概念を備えた Java 言語とその処理系」. 情報処理学会論文誌 Vol.40,No.S1G7(PRO4),pp.51-65(1999).
3. Boris Bokowski,Jan Vitek. *Confined Types*. Proc.OOPSLA,pp.82-96(1999).
4. Boris Bokowski. *CoffeeStrainer*. http://www.inf.fu-berlin.de/ bokowski/ CoffeeStrainer/
5. 有賀透, 伊藤貴康. 「場所概念に基づくセキュア Java システムの実現」. 東北大学電通談話会記録, 第 70 巻 1 号,pp477-478(2001).