［3-3.］シリアル化と情報漏洩

Javaのオブジェクトをファイルに保存したりネットワーク経由で送受信するときには「シリアル化」という手法が用いられる。メモリ上のオブジェクトは無関係なクラスからのぞき見ることは出来ないが，シリアル化されたデータストリームを読むことは難しくない。シリアル化されたデータから重要な情報が漏れないよう注意が必要である。

● ● ●

シリアル化

オブジェクトの「シリアル化」（serialize,「直列化」と訳されることもある）とは，オブジェクトの内部状態をバイトストリームに変換し，そのバイトストリームから再び元と同じ内容を持ったオブジェクトを再現することができるJavaの機能のことである。たとえば，あるマシン上のオブジェクトをシリアル化してネットワーク経由で転送し，別のマシン上にオブジェクトの複製を作ることができる。また，オブジェクトをシリアル化したバイトストリームはファイルに保存可能である。

ただし，どのようなクラスのオブジェクトでもシリアル化が可能なわけではない。シリアル化可能なクラスとは，次のインタフェースのいずれかを実装しているクラスである。

```
java.io.Serializable
java.io.Externalizable
```

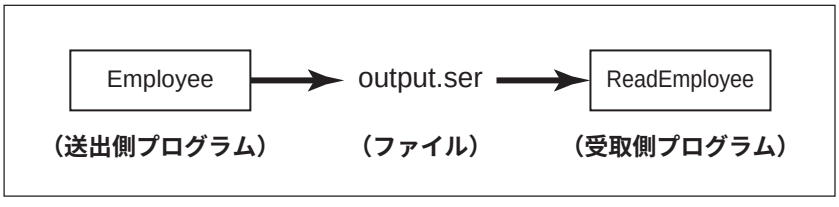
Java のコアクラスの多くはシリアル化可能である。

では，実際にシリアル化機能を利用してみよう。シリアル化機能を利用する場合，オブジェクトの書き出し，読み込みには，それぞれ次のメソッドを使うことになる。

書き出し　`ObjectOutputStream` クラスの `writeObject()`メソッド
読み込み　`ObjectInputStream` クラスの `readObject()`メソッド

図１のようにオブジェクトをシリアル化しファイルで受け渡す例を示そう。**リスト１**，**リスト２**をご覧いただきたい。**リスト１**は，社員情報をシリアル化してファイルに書き込むためのクラスである。3 行目でSerializableを実装してシリアル化可能なクラスとして宣言し，12～14行目でEmployeeクラスのオブジェクトをファイル「output.ser」に書き込んでいる。また，**リスト２**は，シリアル化されたオブジェクトを読み込むためのクラスである。5～7 行目でファイル「output.ser」より Employee クラスのオブジェクトを読み込んでオブジェクトの複製を作成し，8～13行目で社員情報を出力している。**リスト１**，**リスト２**を実行した結果を**画面１**に示す。Employeeクラスのオブジェクトがファイルを通して読み書きされていることがお分かりいただけると思う。

図１　オブジェクトをシリアル化してファイルで受け渡す



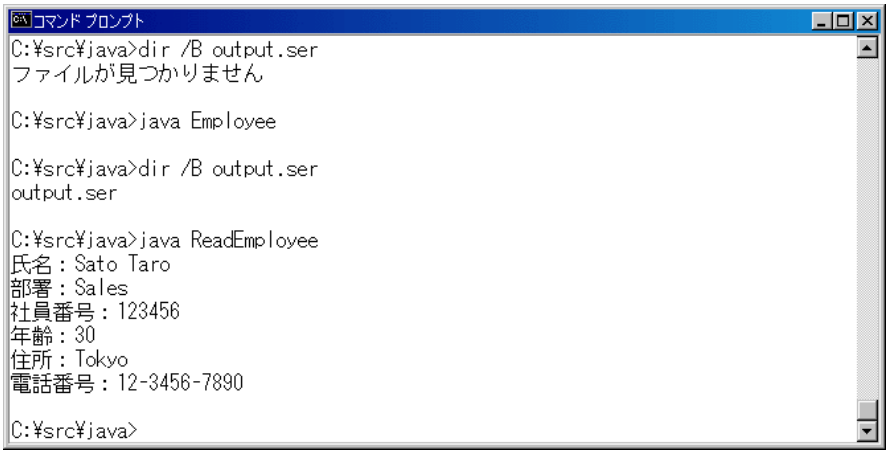
リスト１　自分自身をシリアル化して書き出すクラス Employee

```
 1 import java.io.*;
 2
 3 public class Employee implements Serializable{
 4   protected String name;          //氏名
 5   protected String dept;          //部署
 6   protected long employee_number; //社員番号
 7   protected long age;             //年齢
 8   protected String address;       //自宅住所
 9   protected String tel;           //自宅電話番号
10
11   public static void main(String[] args) throws Exception{
12     FileOutputStream fos = new FileOutputStream("output.ser");
13     ObjectOutputStream oos = new ObjectOutputStream(fos);
14     oos.writeObject(new Employee("Sato Taro","Sales",123456,30,"Tokyo","12-3456-7890"));
15     oos.close();
16   }
17
18   public Employee(String name, String dept, long empnum, long age, String address, String tel){
19     this.name = name;
20     this.dept = dept;
21     this.employee_number = empnum;
22     this.age = age;
23     this.address = address;
24     this.tel = tel;
25   }
26 }
```

リスト２　シリアル化ストリームを読み込んでオブジェクトを復元するクラス ReadEmployee

```
 1 import java.io.*;
 2
 3 public class ReadEmployee{
 4   public static void main(String[] args) throws Exception{
 5     FileInputStream fis = new FileInputStream("output.ser");
 6     ObjectInputStream ois = new ObjectInputStream(fis);
 7     Employee emp = (Employee)(ois.readObject());
 8     System.out.println("氏名：" + emp.name);
 9     System.out.println("部署：" + emp.dept);
10     System.out.println("社員番号：" + emp.employee_number);
11     System.out.println("年齢：" +  emp.age);
12     System.out.println("住所：" + emp.address);
13     System.out.println("電話番号：" + emp.tel);
14   }
15 }
```

画面１



シリアル化ストリームの内容

オブジェクトをシリアル化したバイトストリームは容易に解読できる。内容が暗号化されていない上に，そのフォーマットの仕様は公開されているのである（注1）。

リスト1のプログラムを実行して作られるファイル「output.ser」のシリアル化ストリームを16進ダンプしたものを**図2**に示す。最初の2バイト「AC ED」はシリアル化ストリームに固有のマジックナンバー，次の2バイト「00 05」はシリアル化仕様のバージョン番号である。そしてその後ろにEmployeeオブジェクトの内容が置かれている。

オブジェクトの内容の前半はEmployeeクラスに関する記述だ。クラス名「Employee」や，「age」をはじめとする6つのフィールド名が見て取れる（注2）。これらの後にオブジェクトの具体的なデータが続く。ここにはlong型フィールドの値2つとString型フィールドの値4つが前半のクラス情報のフィールド名に対応した順序で並んでいる。longの値は8バイトのバイナリデータがそのまま置かれているが，Stringの値は「74」という目印のバイト，内容のバイト数を表すshort型の数値，そして文字列本体から構成されている。

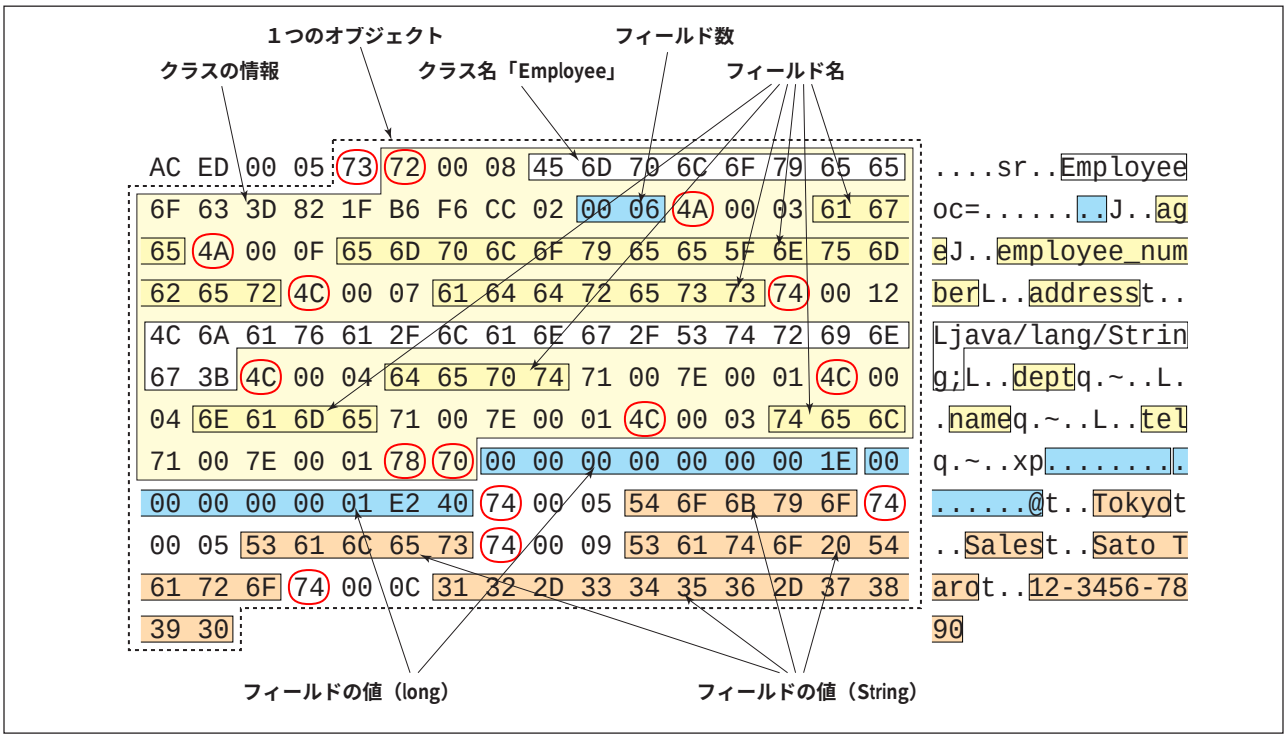
（**注1**・シリアル化ストリームの仕様については参考文献に挙げた『オブジェクト直列化仕様』（Webコンテンツ）をご覧いただきたい。）

（**注2**・Javaでは文字列データは通常Unicodeで扱われるが，シリアル化ストリーム上ではUnicodeの変形版であるUTFで記録される。）

シリアル化ストリームは無防備

Javaのオブジェクトの保護されたフィールドを他者が読み出すことは，通常できないようになっている。しかし，オブジェクトのシリアル化ストリームをファイルやネットワークから読み取ることができれば，そのオブジェクトの各フィールドの値を入手できてしまう。

図2　シリアル化ストリームのダンプ



このような情報漏洩の危険についてはprivate宣言されたフィールドも例外ではない。なぜなら，オブジェクトのシリアル化ではフィールドのpublic/protected/privateの別なく，同じようにバイトストリームへの書き出しが行われるからである。

ファイルやネットワークが使われていなくても用心は必要である。複数のプログラマがかかわる開発プロジェクトでは，メモリ上だけでシリアル化ストリームを作ってオブジェクトの内容を盗み出す処理をひそかに組み込む者がいることも考えられるからだ。

transient フィールド

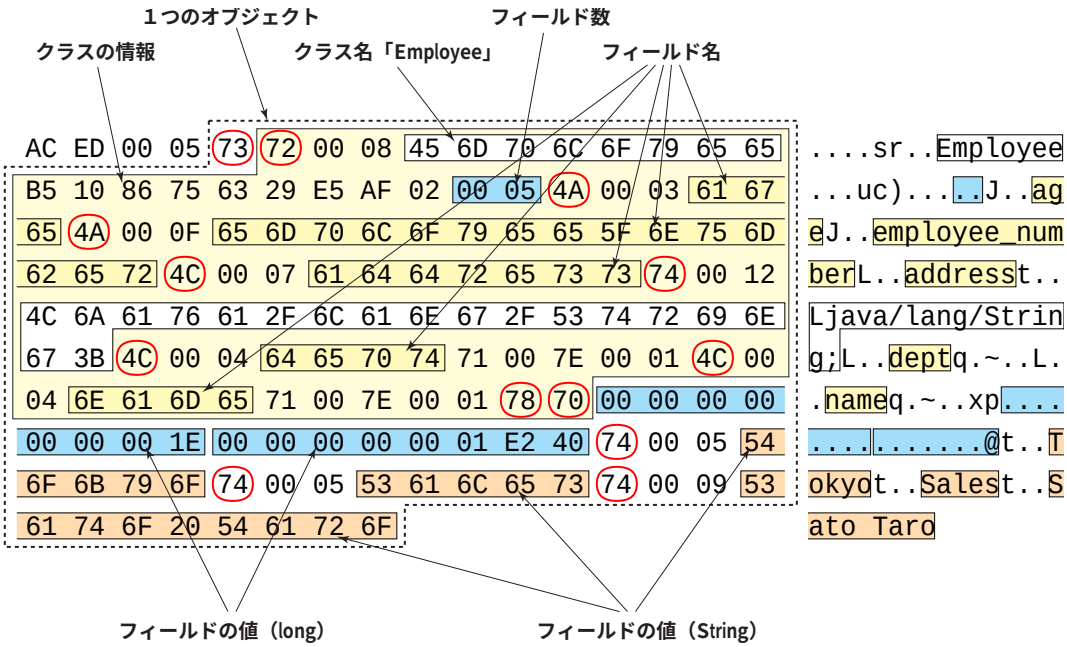
シリアル化された情報を保護するにはどうすればよいだろうか。

最も単純な対策として，外部に漏れてしまっては困るような機密情報のフィールドをシリアル化の対象から外す方法がある。対象から外すフィールドに「transient」を指定すればよい。**リスト３**は，**リスト１**の「tel」フィールドにtransient指定を加えたものである。修正したプログラムを実行して作られた「output.ser」ファイルのダンプを**図３**に示す。**図２**と見比べていただきたい。transient指定された「tel」フィールドの名前とその値「12-3456-7890」が姿を消していることに気づいていただけると思う。

リスト３　リスト１の tel フィールドに transient を指定

```
 1 import java.io.*;
 2
 3 public class Employee implements Serializable{
 4   protected String name;           //氏名
 5   protected String dept;           //部署
 6   protected long employee_number; //社員番号
 7   protected long age;              //年齢
 8   protected String address;        //自宅住所
 9   protected transient String tel; //自宅電話番号          ←transient を指定した
10
...以下略...
```

図３　transient フィールドを持つオブジェクトのシリアル化ストリーム



ただし，transientを指定する方法はフィールドの値を伝送することそのものをやめてしまうものであるため，伝送先に渡さなくてもいい二次的な情報にしか使えない。機密情報・重要な情報をシリアル化されたバイトストリームで伝送する場合には，通信路あるいはバイトストリームそのものを暗号化すべきである。

RMI，EJB とシリアル化

Java で分散型アプリケーションや大規模なビジネスアプリケーションを構築する際，RMI（Remote Method Invocation）や EJB（Enterprise Java Beans）といった技術を利用する機会が増えている（注3）。これらの仕組みの中では，異なるマシン間でオブジェクトをやり取りする際にシリアル化が利用されるため，取り扱う情報によってはネットワーク経路を暗号化する必要がある。

Sun から提供される JSSE（Java Secure Socket Extension）をはじめ，市販の Java 用の暗号化製品，アプリケーションサーバの暗号化オプション等の利用を考慮していただきたい。

（**注3**・ RMI や EJB の詳細については参考文献に挙げた資料を参照していただきたい。）

まとめ

シリアル化は，オブジェクトの永続的な保存やネットワークを経由した送受信に用いられる手法であるが，シリアル化ストリームの内容は解読が容易である。シリアル化ストリームがどのような箇所に露出するかを事前に想定したうえで，シリアル化ストリームへの出力を抑制したり暗号化するなどの対策を講じる必要がある。プログラマが明示的に使用しなくても RMI や EJB の実装にはシリアル化が使われているので注意が必要だ。

参考文献

『JAVA 2 SDK, Standard Edition ドキュメント』
http://java.sun.com/j2se/1.3/ja/docs/ja/index.html

『オブジェクト直列化仕様』
http://java.sun.com/j2se/1.3/ja/docs/ja/guide/serialization/spec/serialTOC.doc.html

『Java Remote Method Invocation (RMI)』
http://java.sun.com/j2se/1.3/ja/docs/ja/guide/rmi/index.html

『RMI と SSL について』
http://java.sun.com/j2se/1.3/ja/docs/ja/guide/rmi/SSLInfo.html

『Enterprise JavaBeans テクノロジー』
http://java.sun.com/j2ee/ja/ejb/index.html

『プロフェッショナル Java 下』，Brett Spell 著，アクロバイト監訳，インプレス