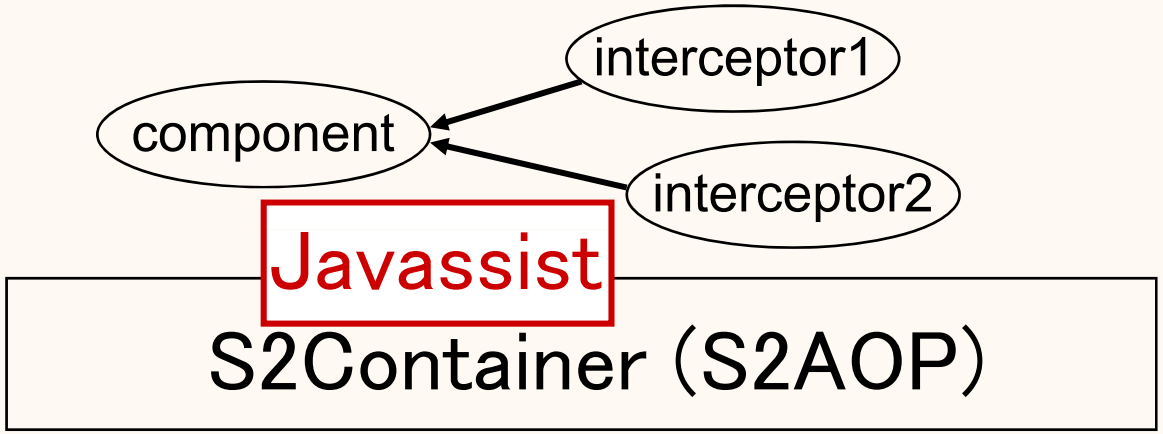
Seasar の中の中

楽天株式会社 楽天技術研究所
西澤 無我

Seasar の中の中

- Javassist (Java バイトコード変換器) の説明
  - S2Container (特に S2AOP) は
    静的に、動的にコンポーネントを拡張可能
  - 実行時に Java バイトコードを生成・編集
  - Javassist を利用



自己紹介

- 名前: 西澤 無我
- 所属: 楽天株式会社 楽天技術研究所

- 現在:
  - ROMA: 分散ハッシュテーブル
  - fairy: 並列分散データ処理フレームワーク
- 昔:
  - 研究で Javassist を利用
  - Javassist, Hibernate の開発

目次

- 概要
- バイトコード変換とは？
- S2Container による Javassist の利用方法
- Javassist の内部の挙動
- まとめ

目次

- **概要**
- バイトコード変換とは？
- S2Container による Javassist の利用方法
- Javassist の内部の挙動
- まとめ

背景

- バイトコード変換を行うソフトウェアが増加
  - 例: S2Container, Hibernate, JBoss, JRuby, …
- 利用者の開発効率の向上
  - 利用者に隠れて、面倒な処理を肩代わり
  - 例: 設定ファイル、注釈を書くだけで…

問題

- 進行するブラックボックス化
  - 便利なツールはブラックボックスになりやすい
  - S2Container などは特に

- バイトコード変換部分は、
  ブラックボックスになりやすい

Javassist を理解しよう！

- Javassist はバイトコード変換器の 1 つ
- Javassist を理解すれば、
  - ブラックボックスがなくなる
  - トラブルシューティングや
    パフォーマンスチューニングに役立つ
  - 面白そう
- S2Container も利用
  - 特に、S2AOP の interceptor や interType

目次

- 概要
- **バイトコード変換とは？**
- S2Container による Javassist の利用方法
- Javassist の内部の挙動
- まとめ

バイトコード変換とは？

- Java バイトコードとは？
  - クラスファイル内のクラス情報のバイナリ
- バイトコード変換とは？
  - バイナリを直接編集し、クラスの定義を生成・変更
- なぜバイトコード変換が必要なのか？
  - クラスを動的に拡張 (AOP) したいから

S2AOP のサンプル (1/2)

- Interceptor を用いたログ出力の例

Person.java

```
public class Person { // Person.java
    public Person() {}
    public void sayHello() {
        System.out.println("Hello, World!“);
} }
```

PersonTest.java

```
public class PersonTest { // PersonTest.java
    public static void main(String[] args) {
        S2Container s2c = S2ContainerFactory.create();
        s2c.init();
        Person p = (Person) s2c.getComponent(Person.class);
        p.sayHello();
} }
```

S2AOP のサンプル (2/2)

- Interceptor とその dicon ファイル

LogInterceptor.java

```
public class LogIeInterceptor extends AbstractInterceptor {
    public Object invoke(MethodInvocation invocation) {
        try {
            System.out.println("before");
            return invocation.proceed();
        } finally {
            System.out.println("after");
} } }
```

app.dicon の一部

```
<component class="Person">
    <aspect pointcut="sayHello">
        <component class="LogInterceptor"/>
    </aspect>
</component>
```

サンプルの実行結果

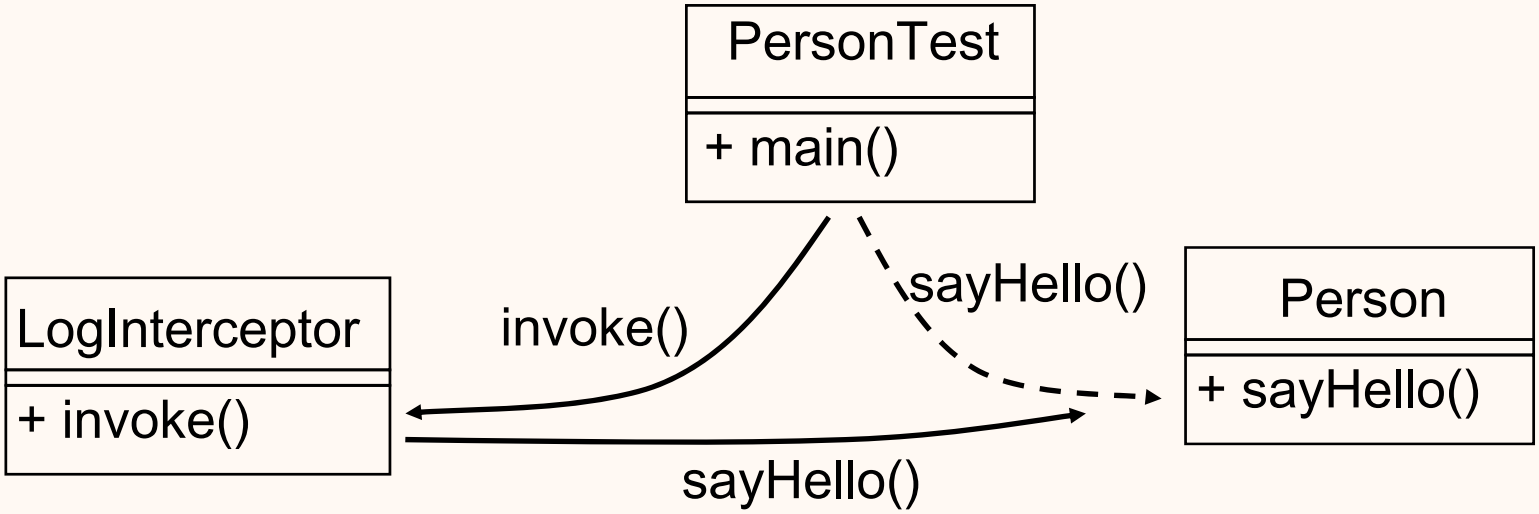
- “Hello, World!” だけではなく、
  LogInterceptor 内の出力命令も実行されている

```
> java PersonTest

before
Hello, World!
after
```

sayHello() 実行の出力

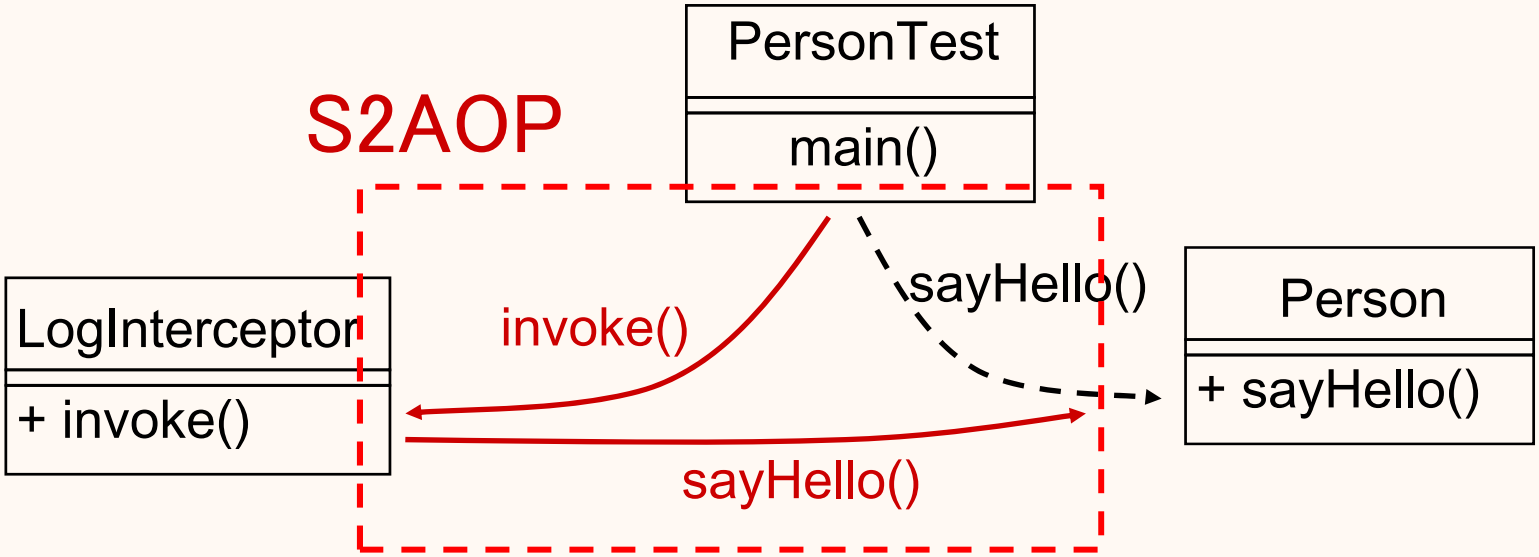
サンプルの挙動

- Interceptor はメソッド実行を拡張できる
  - Person オブジェクトの sayHello() の代わりに、LogInterceptor に実装した invoke() が実行



S2AOP の役割

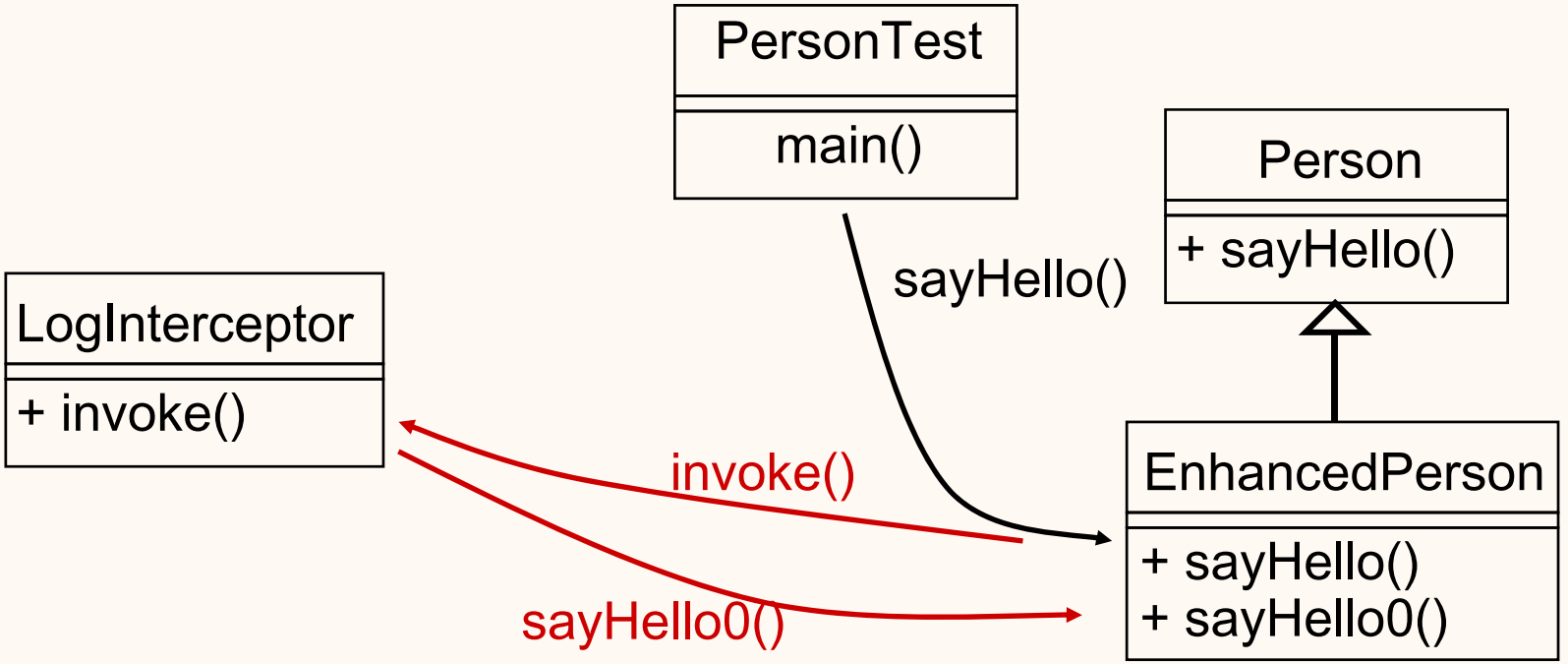
- S2AOP はコンポーネントを動的に拡張
  - S2AOP は dicon ファイルを読み込んで、Person に対し、LogInterceptor という機能を追加



S2AOP がやっていること (1/2)

- 元クラスのサブクラスを自動生成
  - Person クラスのサブクラスを自動生成

※EnhancedPerson と sayHello0 の名前は適当、実際には
Person$$EnhancedByS2AOP$$e3b895 と
$$sayHello$$invokeSuperMethod$$()



S2AOP がやっていること (2/2)

- S2Container 内でサブクラスを初期化
  - 利用者はサブクラスの存在を意識しない

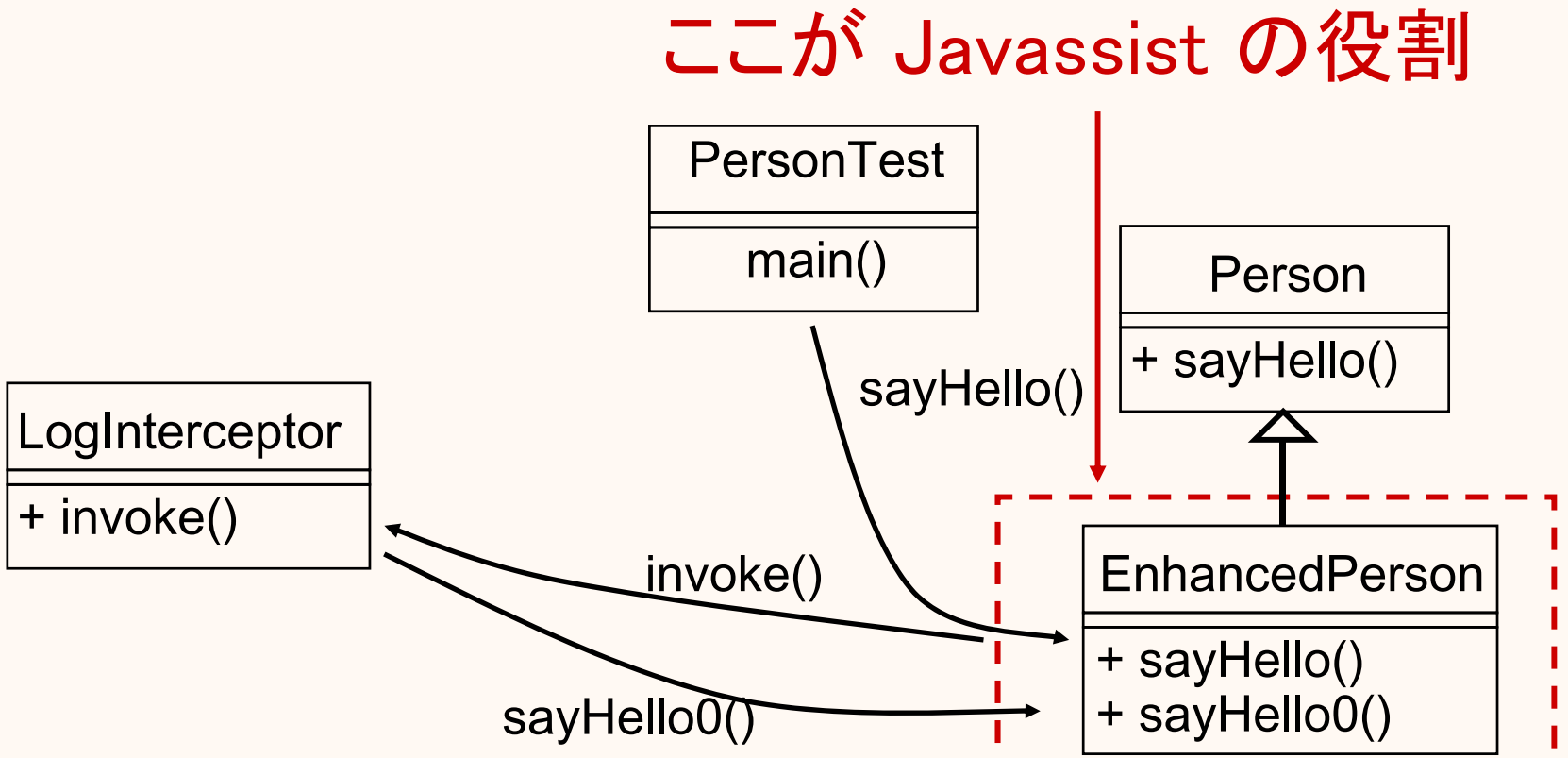
PersonTest.java

```
public class PersonTest {
    public static void main(String[] args) {
        S2Container s2c = S2ContainerFactory.create();
        s2c.init();
        Person p = (Person) s2c.getComponent(Person.class);
        p.sayHello();
} }
```

new EnhancedPerson();

Javassist の役割

- 元クラスのバイトコードの情報を利用し、
  そのサブクラスのバイトコードを生成



Javassist が生成したバイトコード (1/2)

- EnhancedPerson のバイトコード

EnhancedPerson.class

※これはバイトコードを javap でダンプした結果

```
public class Person$$EnhancedByS2AOP$$e3b895
    extends Person{
public Person$$EnhancedByS2AOP$$e3b895();
  Code:
   0:   aload_0
   1:   invokespecial   #10; //Method Person."<init>":()V
   4:   return

public void $$sayHello$$invokeSuperMethod$$();
  Code:
   0:   aload_0
   1:   invokespecial   #15; //Method Person.sayHello:()V
   4:   aconst_null
   5:   pop
   6:   return
public void sayHello();
  Code:
   0:   new     #17; //class
Person$$EnhancedByS2AOP$$e3b895$$MethodInvocation$$s
ayHello0
   3:   dup
   4:   aload_0
   5:   iconst_0
   6:   anewarray       #19; //class java/lang/Object
   9:   invokespecial   #22; //Method
Person$$EnhancedByS2AOP$$e3b895$$MethodInvocation$$sayHello0.
nit>":(Ljava/lang/Object;[Ljava/lang/Object;)V
   12:  invokevirtual   #26; //Method
Person$$EnhancedByS2AOP$$e3b895$$MethodInvocation$$sayHello0.pr
oceed:()Ljava/lang/Object;
   15:  astore_1
   16:  return
   17:  astore_2
   18:  aload_2
   19:  athrow
   20:  astore_2
   21:  aload_2
   22:  athrow
   23:  astore_2
   24:  new     #34; //class java/lang/reflect/UndeclaredThrowableException
   27:  dup
   28:  aload_2
   29:  invokespecial   #37; //Method
java/lang/reflect/UndeclaredThrowableException."<init>":(Ljava/lang/Throwa
ble;)V
   32:  athrow
  Exception table:
   from   to  target type
     0    17    17   Class java/lang/RuntimeException
     0    17    20   Class java/lang/Error
     0    17    23   Class java/lang/Throwable
}
```

Javassist が生成したバイトコード (2/2)

- EnhancedPerson のソースコード表現

EnhancedPerson.class

```
public class EnhancedPerson extends Person {
    public EnhancedPerson() { super(); }
    public void sayHello0() {
        super.sayHello();
    }
    public void sayHello() {
        try {
            LogInterceptor li = new LogInterceptor(new Object[0]);
            li.invoke();
        } catch (Throwable t) {
            throw new UndefinedThrowableException(t);
} } }
```

※正確には LogInterceptor が new されるわけではない

バイトコード変換のまとめ

- S2AOP を実現するため、
  実行時にバイトコードの生成が必要
- それを行うのが Javassist の役割

  – これだけだったら、他の選択肢もありますが…
    - java.lang.reflect.Proxy (Dynamic Proxy) の利用

余談

- Subclassing によるバイトコード変換
  - 利点
    - 呼び出し元 (PersonTest) に手を加える必要なし
  - 欠点
    - ライブラリ経由でのオブジェクト生成 (new 演算子は不可)
- 元クラス自体を直接変更する方法だったら？？
  - 利点
    - new 演算子の使用
  - 欠点
    - System クラスは変換不可

目次

- 概要
- バイトコード変換とは？
- **S2Container による Javassist の利用方法**
- Javassist の内部の挙動
- まとめ

Javassist が生成したバイトコード

- Javassist をどのように利用すれば、
  このようなバイトコードが生成されるのか？

EnhancedPerson.class

```
public class EnhancedPerson extends Person {
    public EnhancedPerson() { super(); }
    public void sayHello0() {
        super.sayHello();
    }
    public void sayHello() {
        try {
            LogInterceptor li = new LogInterceptor(new Object[0]);
            li.invoke();
        } catch (Throwable t) {
            throw new UndefinedThrowableException(t);
} } }
```

おおまかな S2AOP での利用方法

- EnhancedPerson クラスのバイトコードを生成

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

makeClass メソッド

- 指定された名前のクラスのバイトコードを新規生成

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

ソースコードで表現すると

- 指定された名前のクラスを生成

EnhancedPerson.class

```
public class EnhancedPerson extends Person {
    public EnhancedPerson() { super(); }
    public void sayHello0() {
        super.sayHello();
    }
    public void sayHello() {
        try {
            LogInterceptor li = new LogInterceptor(new Object[0]);
            li.invoke();
        } catch (Throwable t) {
            throw new UndefinedThrowableException(t);
} } }
```

CtNewConstructor.make メソッド

- コンストラクタのバイトコード生成し、クラスに追加

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

ソースコードで表現すると

- コンストラクタを生成し、クラスに追加

EnhancedPerson.class

```
public class EnhancedPerson extends Person {
    public EnhancedPerson() { super(); }
    public void sayHello0() {
        super.sayHello();
    }
    public void sayHello() {
        try {
            LogInterceptor li = new LogInterceptor(new Object[0]);
            li.invoke();
        } catch (Throwable t) {
            throw new UndefinedThrowableException(t);
} } }
```

CtNewMethod.make メソッド

- メソッドのバイトコードを生成し、クラスに追加

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body,targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

ソースコードで表現すると

- メソッドを生成し、クラスに追加

EnhancedPerson.class

```
public class EnhancedPerson extends Person {
    public EnhancedPerson() { super(); }
    public void sayHello0() {
        super.sayHello();
    }
    public void sayHello() {
        try {
            LogInterceptor li = new LogInterceptor(new Object[0]);
            li.invoke();
        } catch (Throwable t) {
            throw new UndefinedThrowableException(t);
} } }
```

toBytecode メソッド

- クラスのバイトコードをバイト列に変換

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body,targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

ソースコードで表現すると

- 出来上がり

EnhancedPerson.class

```
public class EnhancedPerson extends Person {
    public EnhancedPerson() { super(); }
    public void sayHello0() {
        super.sayHello();
    }
    public void sayHello() {
        try {
            LogInterceptor li = new LogInterceptor(new Object[0]);
            li.invoke();
        } catch (Throwable t) {
            throw new UndefinedThrowableException(t);
} } }
```

本当はバイトコード

- Javassist が生成したバイトコード

EnhancedPerson.class

```
public class Person$$EnhancedByS2AOP$$e3b895
    extends Person{
public Person$$EnhancedByS2AOP$$e3b895();
  Code:
   0:   aload_0
   1:   invokespecial   #10; //Method Person."<init>":()V
   4:   return

public void $$sayHello$$invokeSuperMethod$$();
  Code:
   0:   aload_0
   1:   invokespecial   #15; //Method Person.sayHello:()V
   4:   aconst_null
   5:   pop
   6:   return
public void sayHello();
  Code:
   0:   new     #17; //class
Person$$EnhancedByS2AOP$$e3b895$$MethodInvocation$$s
ayHello0
   3:   dup
   4:   aload_0
   5:   iconst_0
   6:   anewarray       #19; //class java/lang/Object
   9:   invokespecial   #22; //Method
Person$$EnhancedByS2AOP$$e3b895$$MethodInvocation$$sayHello0."<i
nit>":(Ljava/lang/Object;[Ljava/lang/Object;)V
   12:  invokevirtual   #26; //Method
Person$$EnhancedByS2AOP$$e3b895$$MethodInvocation$$sayHello0.pr
oceed:()Ljava/lang/Object;
   15:  astore_1
   16:  return
   17:  astore_2
   18:  aload_2
   19:  athrow
   20:  astore_2
   21:  aload_2
   22:  athrow
   23:  astore_2
   24:  new     #34; //class java/lang/reflect/UndeclaredThrowableException
   27:  dup
   28:  aload_2
   29:  invokespecial   #37; //Method
java/lang/reflect/UndeclaredThrowableException."<init>":(Ljava/lang/Throwa
ble;)V
   32:  athrow
  Exception table:
   from    to  target type
     0    17    17   Class java/lang/RuntimeException
     0    17    20   Class java/lang/Error
     0    17    23   Class java/lang/Throwable
}
```

Javassist の利点

- バイトコードを生成・編集するライブラリだが、それをしていると意識させない API を提供
  - まるでソースコードを書いている感覚で
    - 他のバイトコード変換器では提供していない
- Reflection API に似せている

目次

- 概要
- バイトコード変換とは？
- S2Container による Javassist の利用方法
- Javassist の内部の挙動
- まとめ

どこを説明するのか？

- S2AOP で利用されている箇所

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

まずは CtClass について

- Javassist を利用する上で、一番重要なクラス

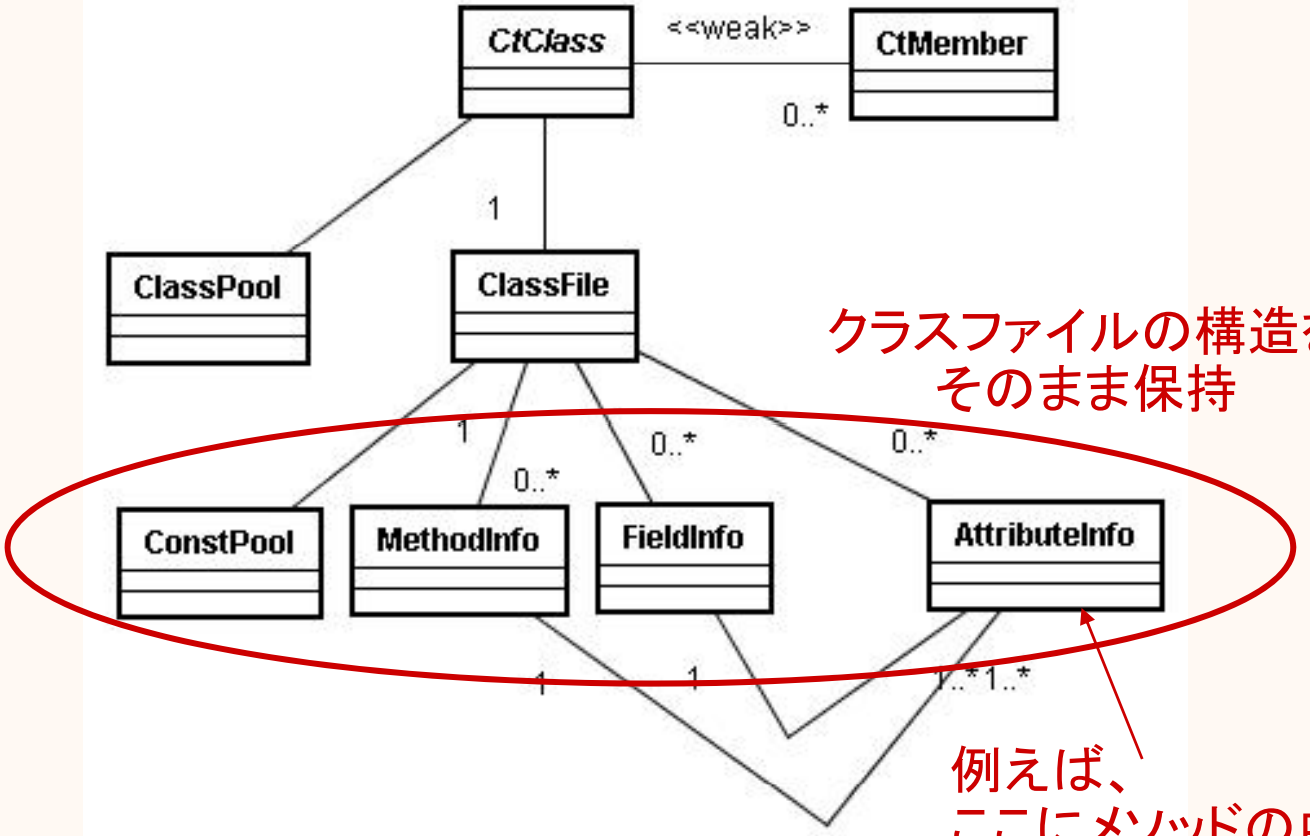
```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

CtClass (Compile-time Class) とは

- 1 つの Java クラス (.class) を表現するクラス
  - バイトコード操作を隠蔽している API
  - 例: setSuperclass, addMethod 等
- 利用者は CtXXX というオブジェクトを使用して、バイトコードを生成、編集
  - CtMethod: メソッドを表現するクラス
  - CtConstructor: コンストラクタを表現するクラス
  - CtField: フィールドを表現するクラス

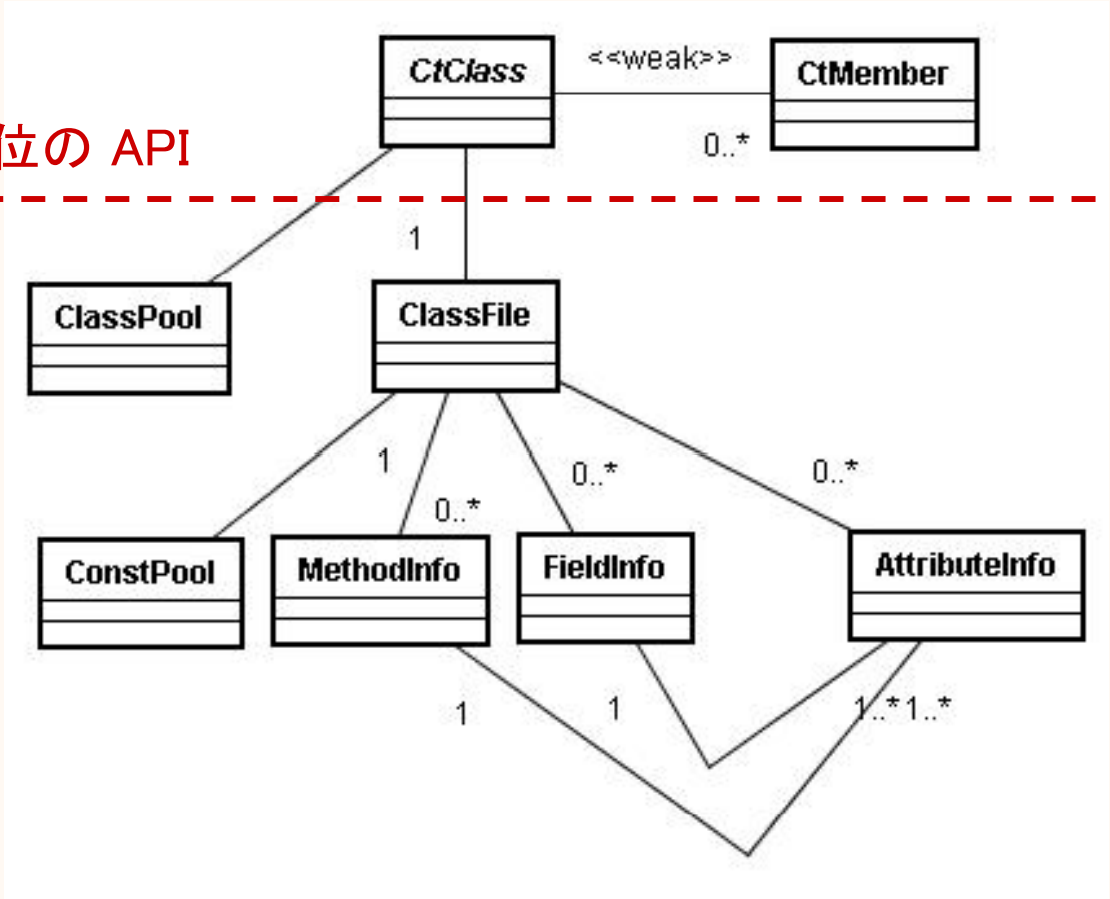
CtClass クラスのおおまかな構造

- もちろん、バイトコードの情報も保持



Javassist の上位の API

- 上位の API だけで、だいたいの事が可能



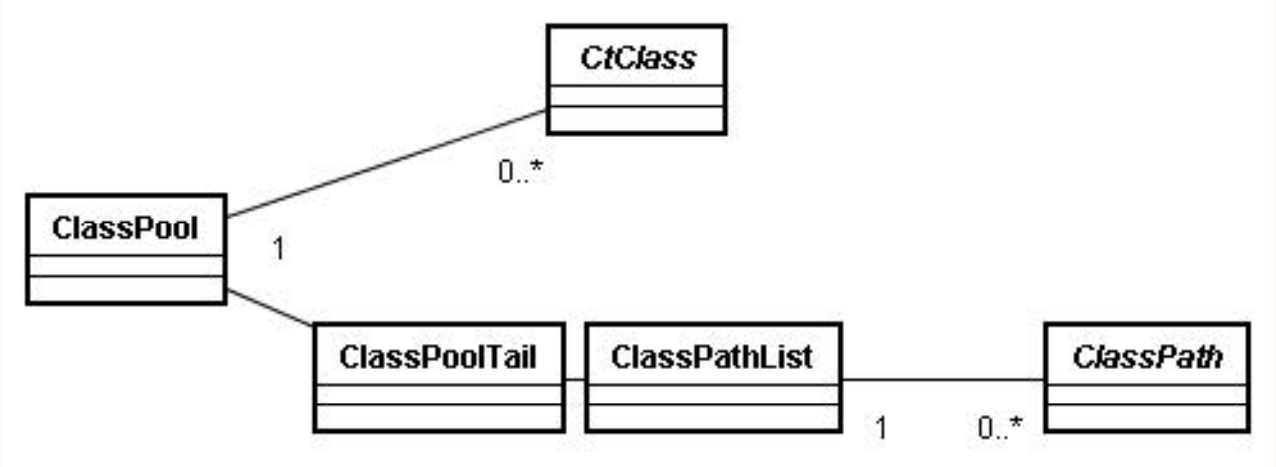
つぎに ClassPool

- CtClass オブジェクトの生成には欠かせない

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

ClassPool とは

- CtClass オブジェクトの入れ物
- CtClass オブジェクト生成の API
  - makeClass(): 新規クラスの CtClass オブジェクトを
  - get(): 既存クラスの CtClass オブジェクトを
    - 既存クラスのバイトコード (.class) を検索＆読み込み



ClassPath とは

- 検索パスを表すのが ClassPath オブジェクト

```
> java –classpath “.:../seasar.jar” PersonTest
```

ここを表現

- ClassPool は ClassPath のリストを持つ
  - ClassPool.get() が呼ばれると、順番にリストを検索

さらに CtMethod, CtNewMethod

- Javassist が普及したのはこのおかげ

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

CtNewMethod.make()

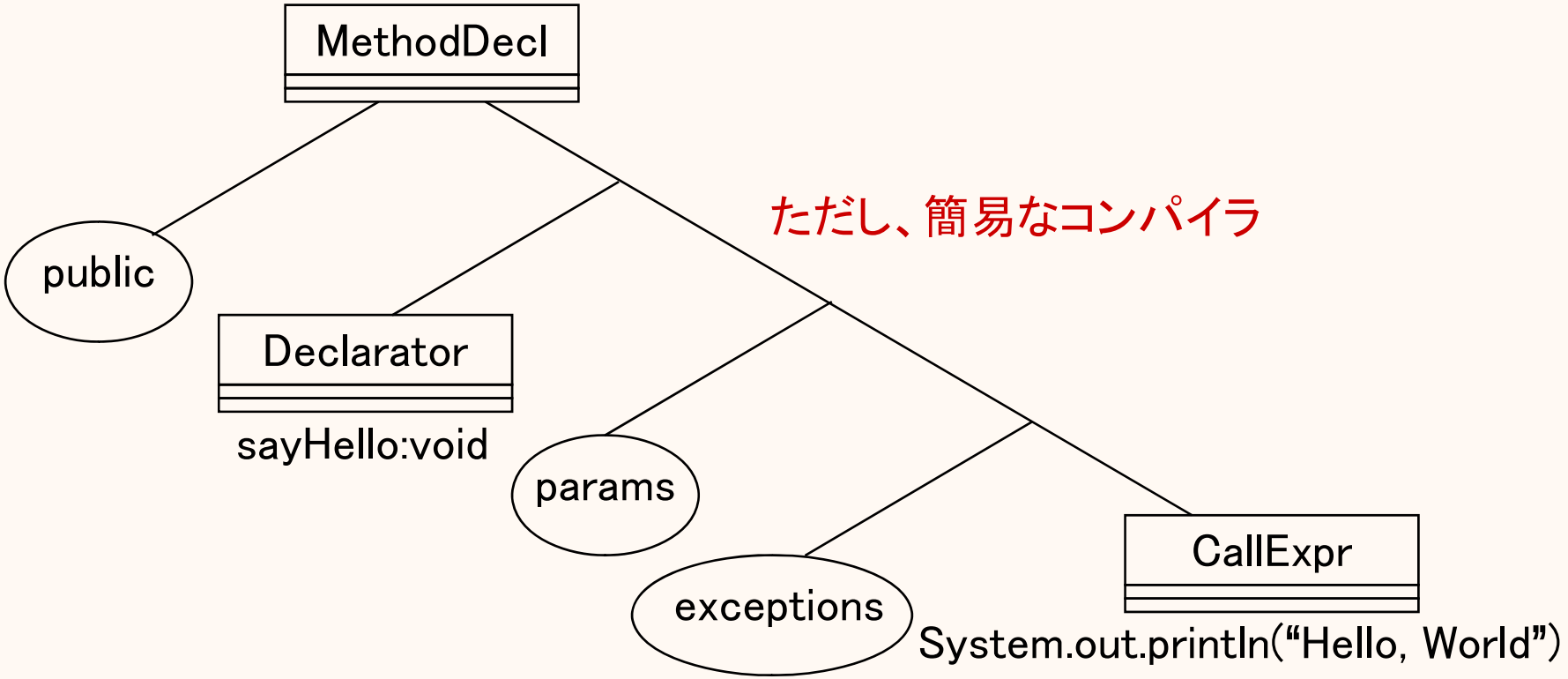
- 新規メソッドをクラスに追加
  - sayHello メソッドのバイトコードをクラスに追加

```
String decl = "public void sayHello() { " +
                    "    System.out.println(¥"Hello, World!¥"); " +
                    "} ";
CtMethod targetMethod = CtNewMethod.make(decl, targetClass);
targetClass.addMethod(targetMethod);
```

- メソッド宣言をソースコードで渡せる

Javassist は Java コンパイラを持つ

- 文字列をコンパイルし、抽象構文木を生成
- さらに、バイトコードを生成



最後に detach

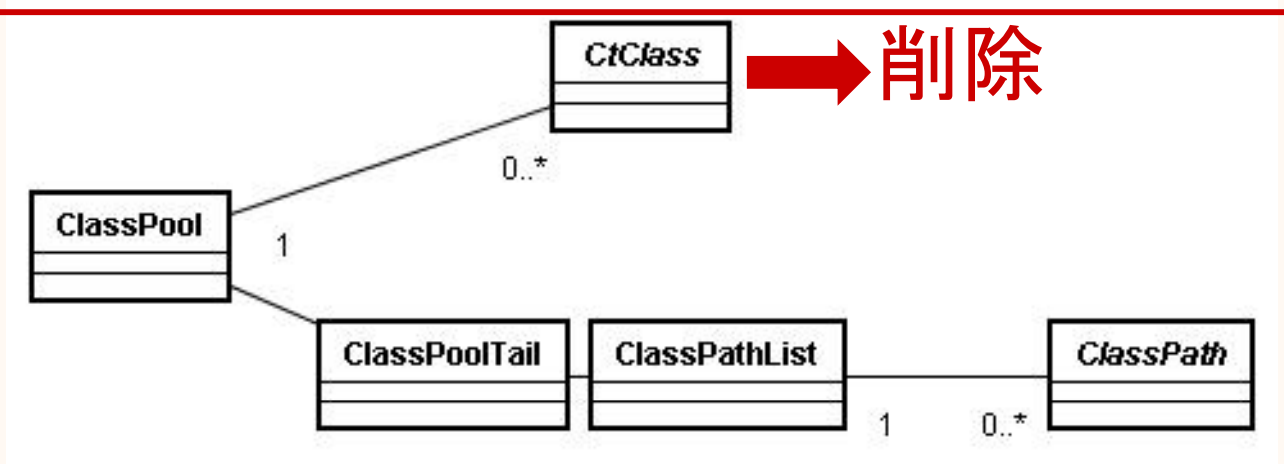
- メモリの有効活用

```
String targetName = “EnhancedPerson”;

ClassPool cp = new ClassPool();
cp.appendClassPath(new LoaderClassPath(classLoader));

CtClass targetCtClass = cp.makeClass(targetName, …);

CtConstructor targetCtCons = CtNewConstructor.make(…,targetCtClass);
targetCtClass.addConstructor(targetCtCons);

CtMethod targetCtMethod = CtNewMethod.make(…, body, targetCtClass);
targetCtClass.addMethod(targetCtMethod);

byte[] bytecode = targetCtClass.toBytecode();

targetCtClass.detach();
```

メモリの再利用 (1/3)

- CtClass オブジェクトを ClassPool から削除
- メモリの有効活用
  - 不必要になった CtClass オブジェクトを、利用者は自主的に削除
  - ただし、変更した CtClass オブジェクトに依存したソースコードをコンパイルするには、それを残しておかなければならない

ご指摘を受け発表後に、修正いたしました。



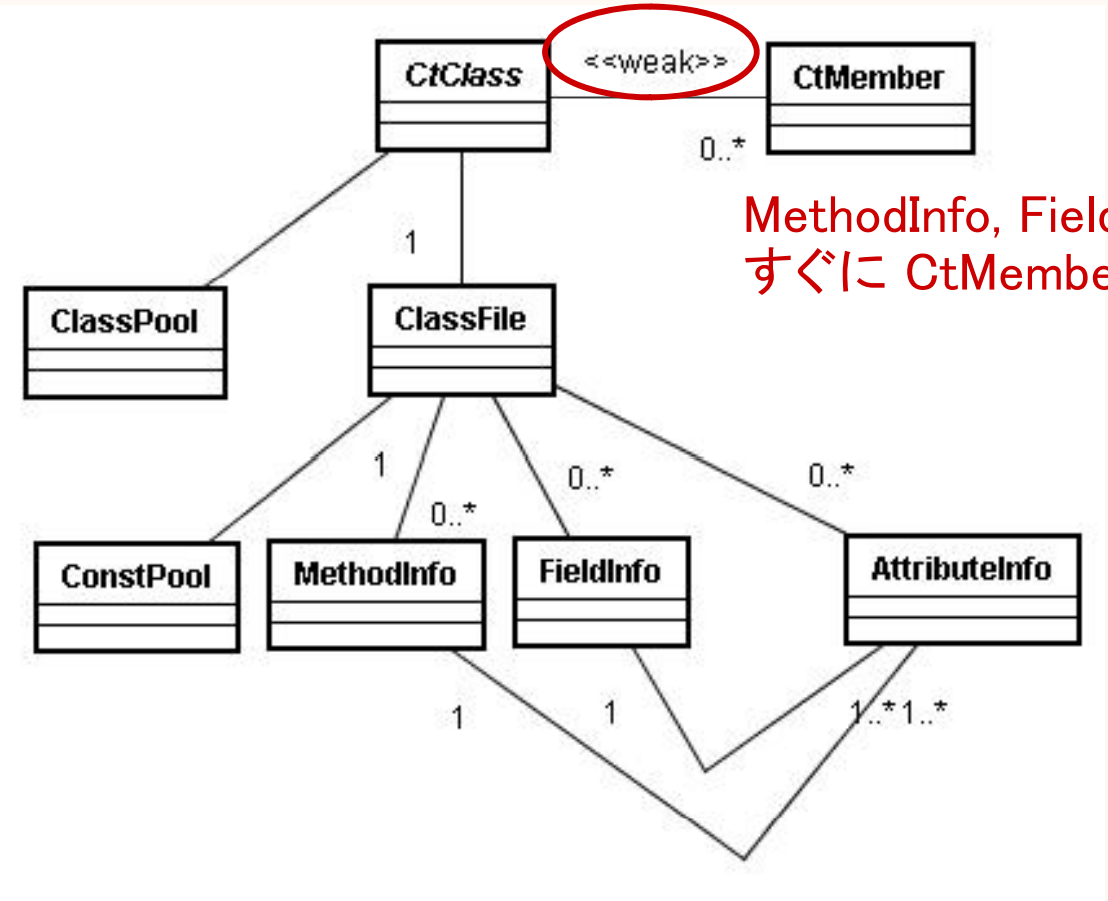
GC のコストとのトレードオフ

- 同じクラスのバイトコードを編集したくなったら？
  - ClassPool.get() 呼出
  - キャッシュがなくなったので、
    再度 CtClass オブジェクトを生成

- クラスファイルの再読み込み +
  CtClass オブジェクト生成にかかるコスト
  vs GC のコスト

メモリの再利用 (2/3)

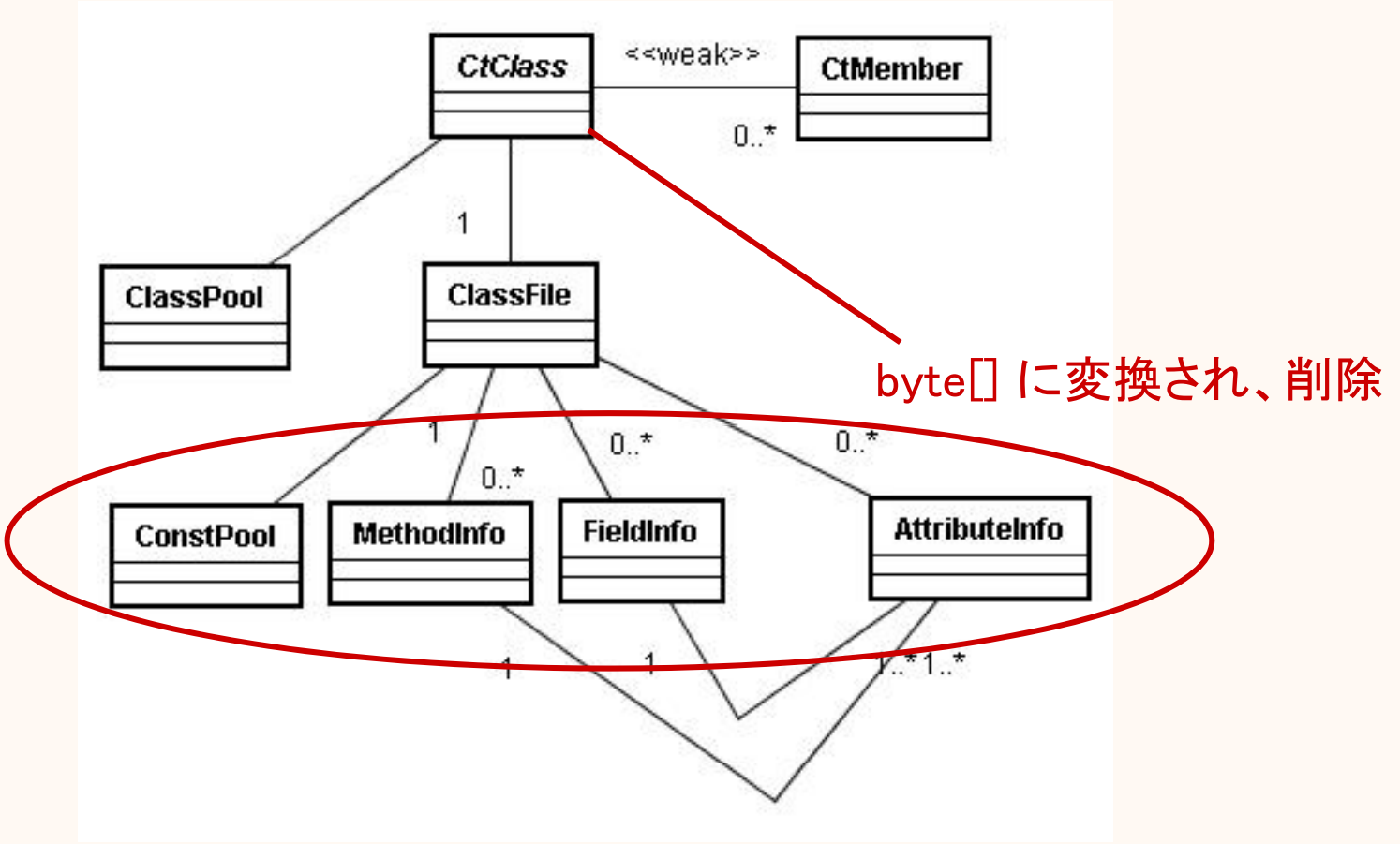
- CtMember のリストは WeakReference



MethodInfo, FieldInfo があれば
すぐに CtMember は生成可能

メモリの再利用 (3/3)

- 閾値を超えると、ClassPool の圧縮



まとめ

- 概要
- バイトコード変換とは？
- S2Container による Javassist の利用方法
- Javassist の内部の挙動

おわり

- ご清聴ありがとうございました