Spring2.0

～もっと有効活用

本日の話の流れ

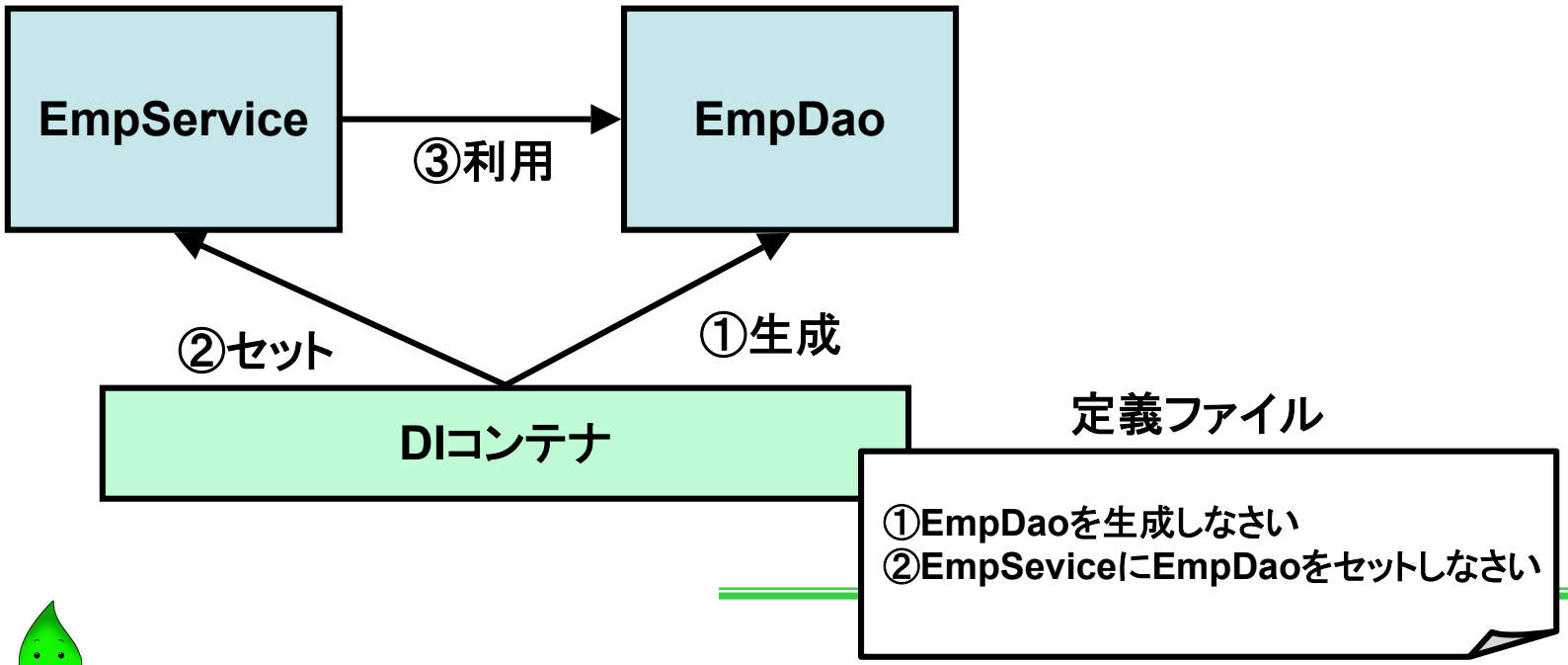
- DIとAOPの基本的な話
- Springの概要
  - 本当の概要
- Spring2.0のいいところ
  - Bean定義ファイルが簡単になった
  - 動的言語がサポートされた
  - JPAと連携できる
  - Spring JDBCが簡単になった
  - Struts連携もいい感じになった
  - その他

Spring, Seasar, EJB3.0, Guice･･･

- エンタープライズ･アプリケーション開発の核(メタフレームワーク)
  - DI
    - コンポーネントを疎結合にする技術
    - 様々な技術を結合する基盤
  - AOP
    - ソースコードに手を入れることなく処理を追加する技術
      - トランザクション管理
      - 例外処理、ログ
- 機能を提供するだけのフレームワークではない
  - フレームワークの細かい使い方ではなく、設計上どのように利用するか

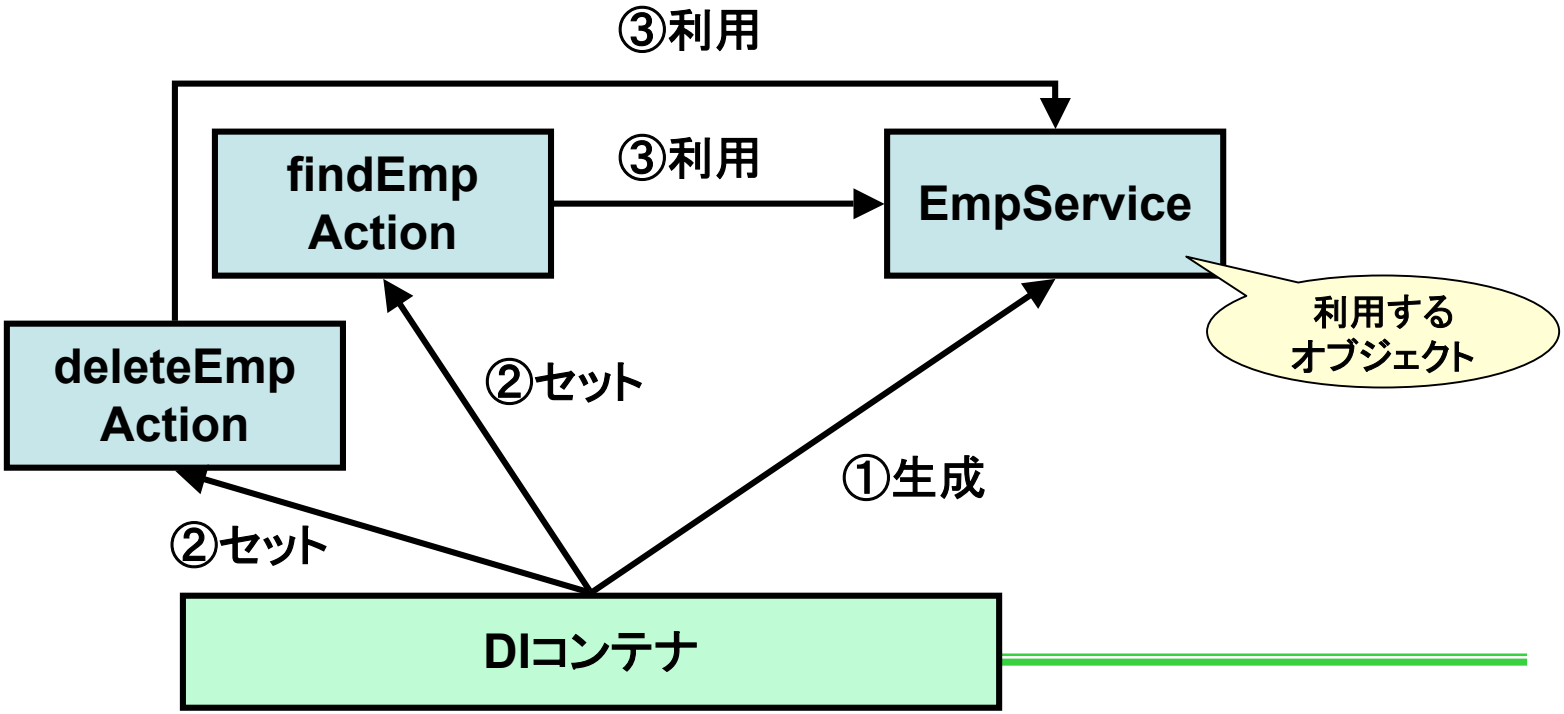
DIコンテナ〜基本

- 設計が終わった時点で、どのオブジェクトがどのオブジェクトを利用するのかは分かっている
- だったら利用するオブジェクトはセットしてあげよう
  - new演算子がコードからなくなる
  - しかもPOJO



DIコンテナ
～Webアプリはマルチスレッド

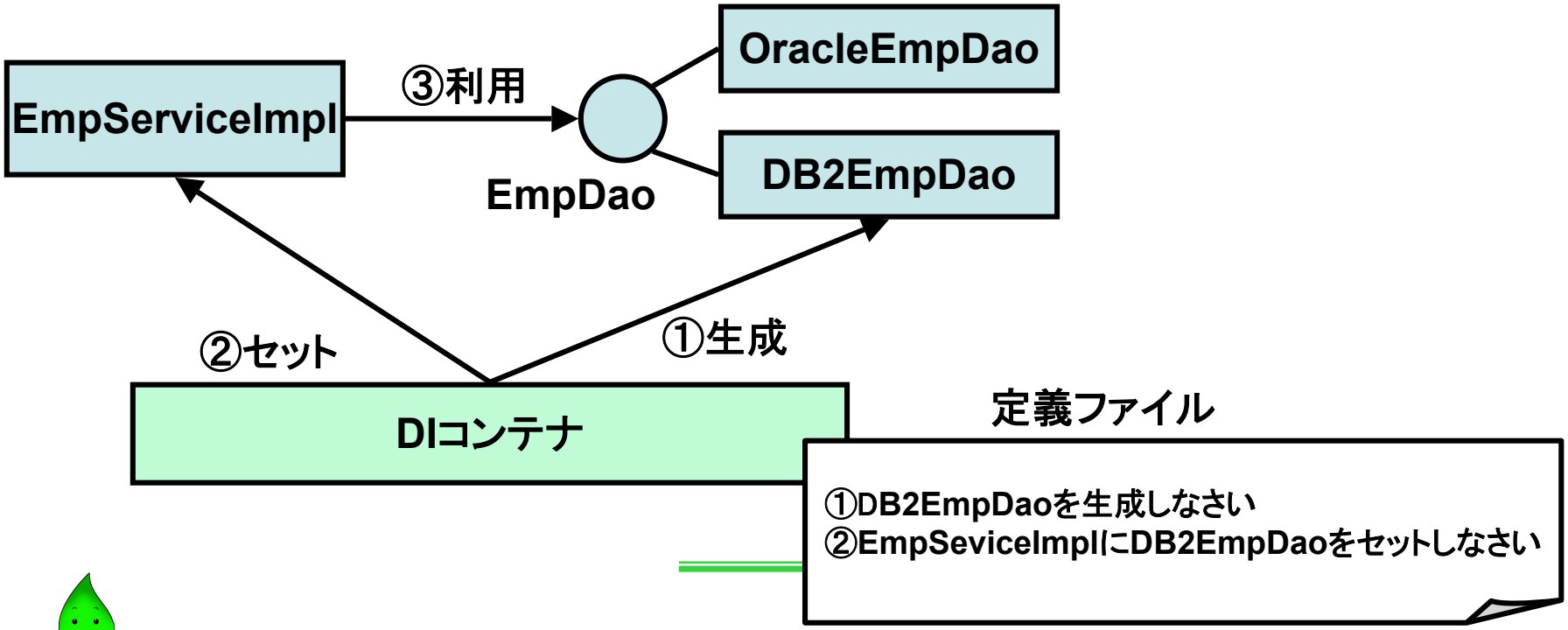
- Webアプリ(Servlet)はマルチスレッド
- 利用するオブジェクトを毎回生成してたらもったいない(Singletonにしよう)



JSUG

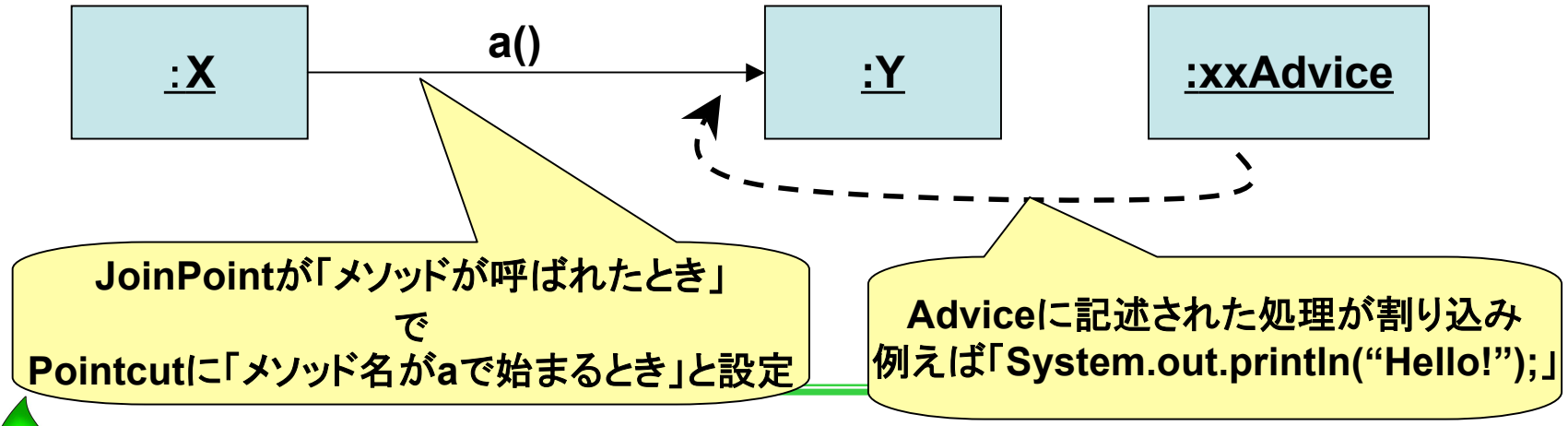
DI～インタフェースを利用

- 自分でnewしないのなら、オブジェクトはインタフェースをとおして扱うようにしよう
  - テストコードへの置き換えが用意
  - 利用する側に影響を与えずに実装クラスの変更が可能



AOP～基本

- 処理を後から追加できる
  - 追加できるところ（仕様）→JoinPoint
    - 例）メソッドが利用されたとき、プロパティが利用されたとき…
  - JoinPointを絞り込むフィルタ→PointCut
    - 例)メソッド名がaで始まる、クラスXの全てのメソッド・・・etc
  - 追加したい処理を書くところ→Advice
    - 例)種別としてBeforeAdvice，AfterAdvice，ThrowAdvice・・・etc



AOPへやって良いこと悪いこと

- 良いこと
  - トランザクション管理、ログ、例外
    - メソッドの前後に付加することができる、いわゆる共通処理
    - 個別処理(メソッドの中)に付属するようなユーティリティではない
  - 誰が提供するか
    - アーキテクトチームとか基盤チーム
- 悪いこと
  - 個別の処理(特定業務の処理、デバッグ)
    - プログラマが個別にAOPをいれるのは不可
      - そこで何をやっているのかが分からなくなる

DIxAOPのまとめ

- 再利用性
- テスト容易性
- 拡張性、変更容易性
- 分散開発
- 開発容易性(複雑さの隠蔽)

※現状の開発で、DIxAOPを使わない理由は
　見当たらない

Spring

- Spring Framework
  - ロッド・ジョンソン氏（Interface21）が中心に開発したDIxAOPコンテナ
  - Java/J2EEアプリケーション開発用フレームワーク
  - ASL（Apache Software Licence）に基づいたオープンソース

Springの主な足取り

- 2002年11月 Expert One-on-One J2EE
- 2003年8月 実践J2EE(Expert One-on-One J2EE翻訳)
- 2004年3月 Spring 1.0
  - 一ヶ月に10,000ダウンロードを記録
- 2004年7月 JavaWorldにSpring登場(国内)
- 2004年9月 Spring1.1
- 2005年5月 Spring1.2
- 2006年10月 Spring 2.0
- 2007年1月 JSUG発足
- 2007年10月 現在　Spring2.0.7，Spring2.1M4

Springのプロジェクト

- サブプロジェクト
  - Spring Security
  - Spring Web Flow
  - Spring Rich Client
  - Spring BeanDoc
  - Spring IDE for Eclipse
  - Spring Web Services
  - etc･･･
- 関連プロジェクト
  - Spring Batch
  - Spring Modules
  - Spring Framework .NET
  - JSF-Spring
  - etc･･･

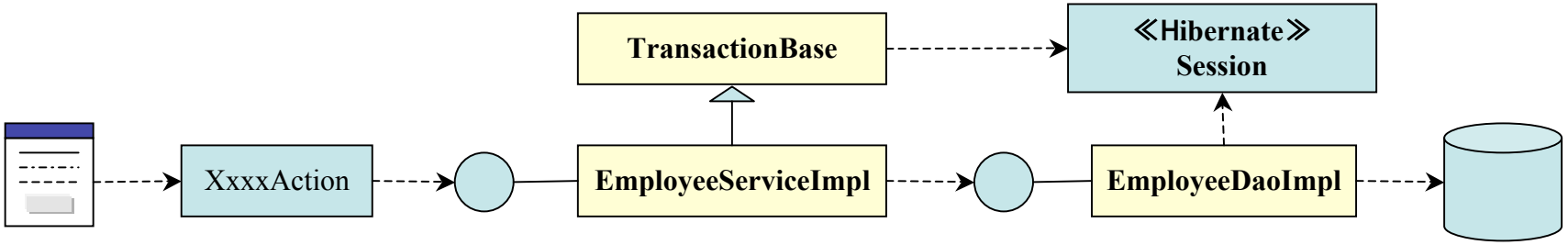
国内でのSpring利用実績

- 公開可能な利用実績のダウンロード(PDF)
  - http://modern.sourceforge.jp/
  - 公開できないものも実は一杯あります

Spring～何が嬉しいか

- POJO
  - 再利用
  - テスト
- トランザクション管理
- ログや例外
- フレームワークとのインテグレーション

Hibernateだけのコード例



Hibernateの制御

```
public class TransactionBase {
    private static SessionFactory sessionFactory = null;

    private static synchronized SessionFactory
                                    getSessionFactory()
        throws HibernateException {
        if(sessionFactory == null){
            Configuration cfg = new Configuration();
            cfg.configure();
            sessionFactory = cfg.buildSessionFactory();
        }
        return sessionFactory;
    }

    public Session getSession() throws HibernateException {
        SessionFactory sf = getSessionFactory();
        return sf.openSession();
    }
}
```

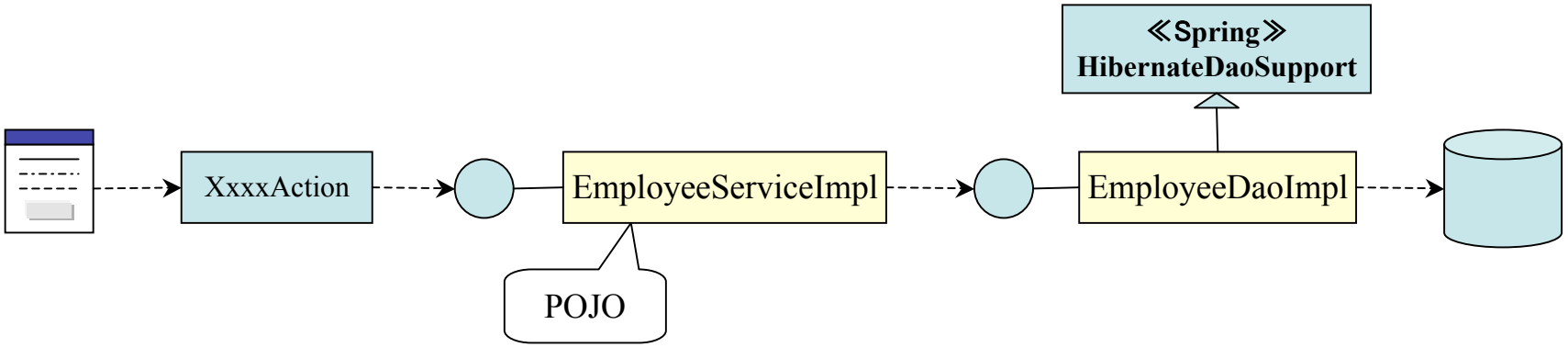
```
public class EmployeeServiceImpl
    extends BaseDao implements EmployeeService {
  public void addEmployee(Employee employee) {
      Session session = null;
      Transaction transaction = null;
      try {
          session = getSession();
          transaction = session.beginTransaction();
          new HibernateEmployeeDao().addEmployee(session, employee);
          transaction.commit();
      } catch (Exception e) {
          if(transaction != null) {
              try {
                  transaction.rollback();
              }catch (HibernateException ex) {}
          }
      }finally {
          if(session != null) {
              try {
                  session.close();
              } catch (HibernateException e) {}
          }
      }
  }
…
```

トランザクションの開始・終了

テーブルへのアクセス

```
public class Employe...DaoImpl
    implements EmployeeDao{
  public void addEmployee
          (Session session, Employee employee) {
  try {
    session.save(employee);
  } catch (Exception ex) {
    throw new RuntimeException(ex.getMessage());
  }
}
```

Spring+Hibernateコード例




```
public class EmployeeServiceImpl implements EmployeeService {
  private EmployeeDao employeeDao = null;
  public void setEmployeeDao(EmployeeDao employeeDao) {
    this.employeeDao = employeeDao;
  }
  public void addEmployee(Employee employee) {
    employeeDao.addEmployee(employee);
  }
...
}
```

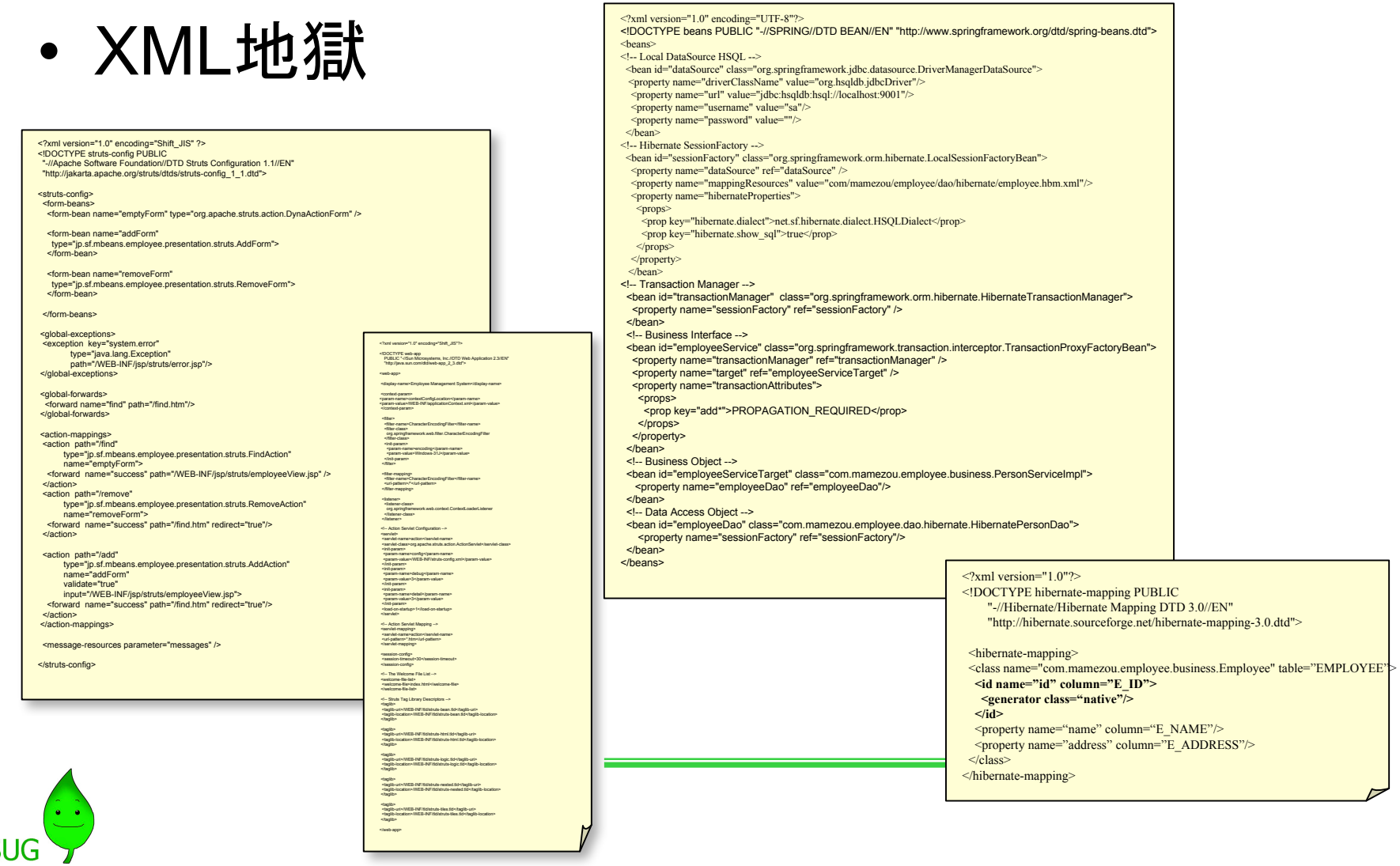
```
public class EmployeeDaoImpl
      extends  HibernateDaoSupport implements EmployeeDao {
   public void addEmployee(Employee employee) {
      getHibernateTemplate().save(employee);
   }
   ...
}
```

トランザクションの開始・終了が
ソースコードから消えている。
Hibernateの制御もない

JSUG

Spring～何が嬉しくなかったか

- XML地獄

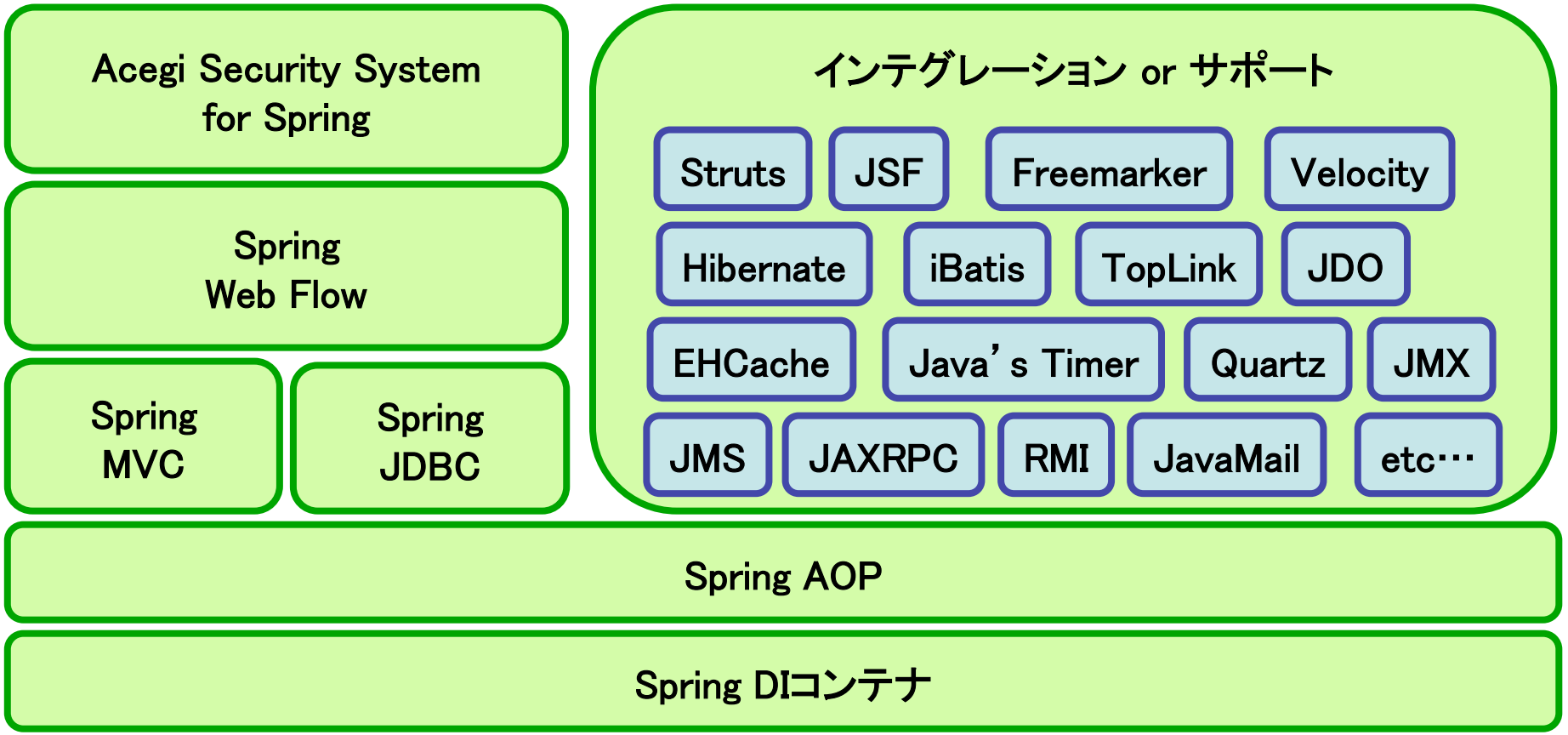


JSUG

SpringからSpring 2.0へ

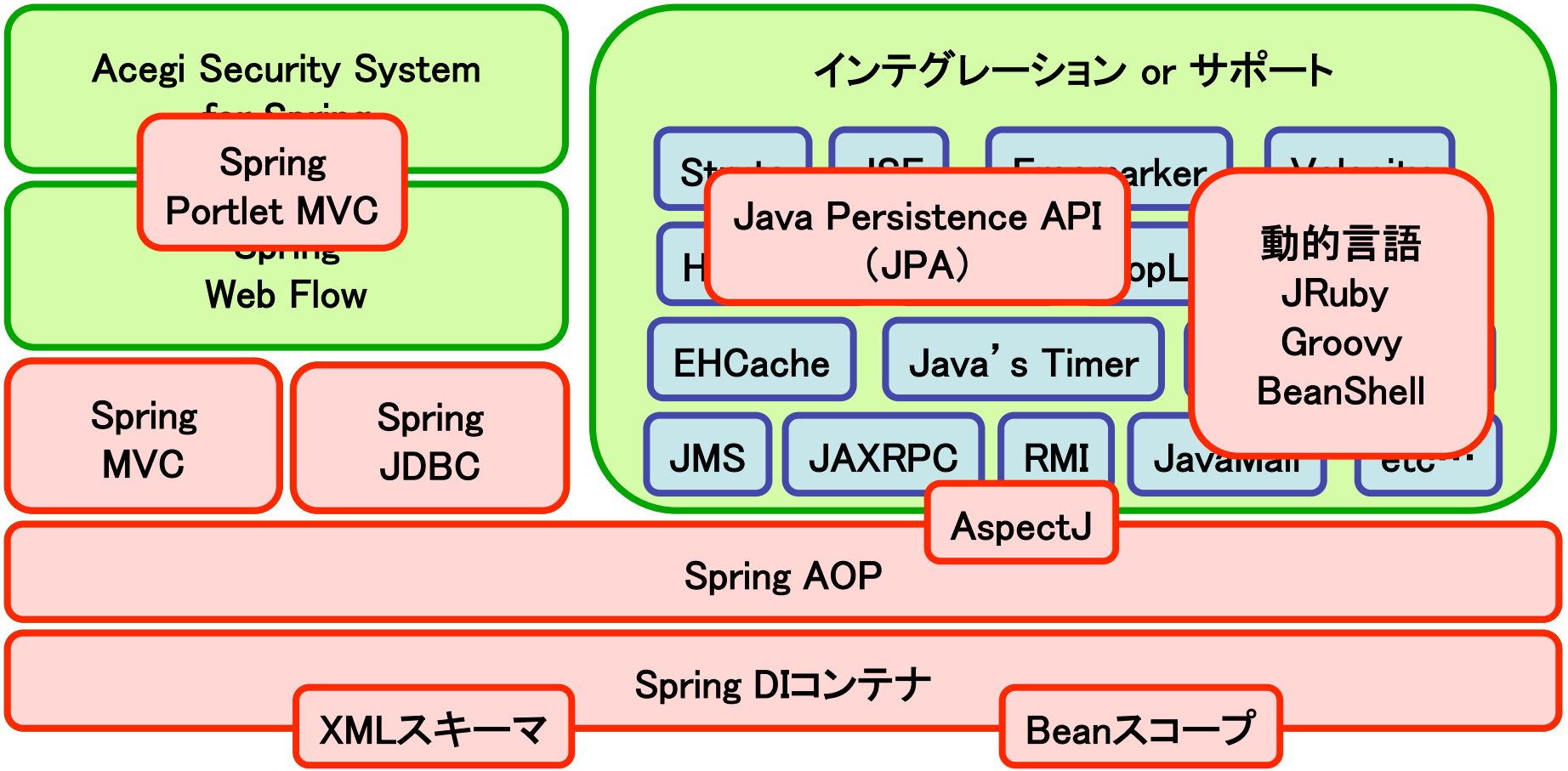
- 「よりシンプルにパワフルに」を目的
- 主な機能強化と新規機能
  - 定義ファイル記述の簡潔化と強化
  - JPA(Java Persistence API)のサポート
  - SpringJDBCの強化
  - Beanのスコープ追加
  - Portlet MVC フレームワーク
  - スクリプト言語で書かれた Bean を利用する
  - Message Driven POJOの導入

Spring1.x



JSUG

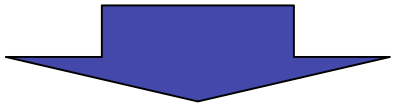
Spring1.x → Spring2.x



JSUG

定義ファイル 今、昔

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  …
</beans>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
   http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util.xsd">
<beans>
  …
</beans>
```

Bean定義の簡略化

- XMLスキーマで定義されたタグ、つまり簡略化されたタグを利用できる
- スキーマ一覧
  - utilスキーマ･･･ユーティリティ機能の設定
  - langスキーマ･･･動的言語の設定
  - aopスキーマ･･･AOPの設定
  - txスキーマ･･･トランザクションの設定
  - jeeスキーマ･･･JNDI/EJBのlookupの設定

Spring1.xのAOPの設定

- めんどくさい

```
<bean id="sampleService"
      class="…….aop.framework.ProxyFactoryBean">
  <property name="target" ref="sampleServiceTarget" />
  <property name="interceptorNames">
    <list><value>aroundAdvisor</value></list>
  </property>
</bean>

<bean id="aroundAdvisor"
      class="…….aop.support.NameMatchMethodPointcutAdvisor">
  <property name="advice" ref="aroundAdvice" />
  <property name="mappedNames">
    <list><value>doSomething</value></list>
  </property>
</bean>
```

JSUG

Spring2.xのAOPの設定

- とっても簡単

```
<aop:config>
  <aop:pointcut id="doSomethingPC"
                expression="execution(* doSomething())"/>
  <aop:advisor pointcut-ref="doSomethingPC"
               advice-ref="aroundAdvice"/>
</aop:config>
```

AOPの拡張

- POJOをAdviceとして利用できる
- AspectJのPointcutの設定を利用できる

aopスキーマこんな使い方

- <aop:advisor
  pointcut="execution(* *..ProductManager.*(..))
  || execution(* *..ProductDao.*(..)) "
  advice-ref="logging"/>

- <aop:advisor
  pointcut= "!execution(* ..ProductManager.*(..)) "
  advice-ref="logging"/>

Spring1.xのトランザクションの設定

- めんどくさい

```
<bean id="sampleDao"
      class="…interceptor.TransactionProxyFactoryBean">
  <property name="transactionManager"
            ref="transactionManager" />
  <property name="target" ref="sampleDaoTarget" />
  <property name="transactionAttributes">
    <props>
      <prop key="get*">PROPAGATION_REQUIRED, readOnly</prop>
      <prop key="add*">PROPAGATION_REQUIRED</prop>
    </props>
  </property>
</bean>
```

Spring2.xのトランザクションの設定

- やっぱり簡単

```
<aop:config>
  <aop:advisor pointcut="execution(* *..*Dao*.*(..))"
               advice-ref="txAdvice"/>
</aop:config>

<tx:advice id="txAdvice">
  <tx:attributes>
    <tx:method name="get*" propagation="REQUIRED"
               read-only="true"/>
    <tx:method name="add*" propagation="REQUIRED"/>
  </tx:attributes>
</tx:advice>
```

txスキーマに色々設定

```
<tx:advice id="txAdvice">
    <tx:attributes>
        <tx:method name="find*”
                propagation="REQUIRES_NEW”
                read-only="true”
                no-rollback-for=”….DataAccessException"/>
        <tx:method name="add*”
                isolation="DEFAULT”
                rollback-for=”....DataAccessException"/>
          …
    </tx:attributes>
</tx:advice>
```

@Transactional+txスキーマ

```
@Transactional
public class SampleBeanImpl
        implements SamplBean {
    public void method() {…}
```

```
<tx:annotation-driven/>
<bean name=”sampleBean"
   class="example.SampleBeanImpl">
</bean>
```

追加されたBeanスコープ

- Spring1.x
  - singleton・・・シングルトン
  - prototype・・・その都度インスタンス生成
- Spring2.x
  - request・・・Servlet APIのrequestスコープ
  - session・・・Servlet APIのsessionスコープ
  - globalSession・・・Portlet APIのグローバルsessionスコープ

動的言語のサポート

- 簡単にJavaから動的言語を利用できる！
- サポートしている動的言語
  - JRuby
    - JavaVM上で動くRuby処理系
    - Rubyと同じ文法
  - Groovy
    - Javaエンジニアのための動的言語
    - JSR-241
    - Javaと文法が似ている
  - BeanShell
    - JavaVM上で動くスクリプト処理系
    - Javaと文法が似ている

動的言語の利用

- Javaのインタフェースを定義する
- 動的言語でインタフェースを実装する
- langスキーマでBean定義を行う


```
package jp.springframework.script;

public interface SampleService {
    void doSomething();
}
```

動的言語でインタフェースを実装する

- JRubyを使いました

```
require 'java'

include_class 'jp.springframework.script.SampleService'

class SampleServiceImpl < SampleService
  def doSomething
    printf @message + "\n"
  end
  def setMessage(message)
    @message = message
  end
end
```

langスキーマでBean定義を行う

- プロパティにインジェクションできる！
- スクリプトの処理を変更して動的にシステムに反映できる！

```
<lang:jruby
    id="sampleService"
    script-source="file:res/SampleService.rb"
    script-interfaces="jp.springframework.script.SampleService"
    refresh-check-delay="500">
  <lang:property name="message" value="日本Springユーザー会"/>
</lang:jruby>
```

JPA連携

- EJBコンテナなしで宣言トランザクションが利用できる
- Hibernate連携と同じような感じで、簡単にDaoが作成できる
  - EntityManagerの取得/クローズ処理が不要
  - トランザクション呼び出しが不要

JPA連携の利用

- JpaDaoSupportを継承する

```
public class SampleDaoImpl extends JpaDaoSupport
                           implements SampleDao {

    public Integer addSample(Sample sample) {
        getJpaTemplate().persist(sample);
        getJpaTemplate().flush();
        return sample.getId();
    }

    public List<Sample> getSamples() {
        return getJpaTemplate()
                    .find("select s from Sample s");
    }
}
```

使いやすくなったSpring JDBC

- J2SE5のジェネリックを利用できる！
  - SimpleJdbcTemplate
- SQL文に名前で値をバインドできる！
  - NamedParameterJdbcTemplate

```
...
public int countAuthor() {
      return simpleTemplate.queryForObject(
                        "select count(*) from author", Integer.class);
}
public Integer addSample(Sample sample) {
    nameTemplate.update(
        "insert into sample (id, name) values(:id, :name)",
        new BeanPropertySqlParameterSource(sample));
    return getIdentity();
}
...
```

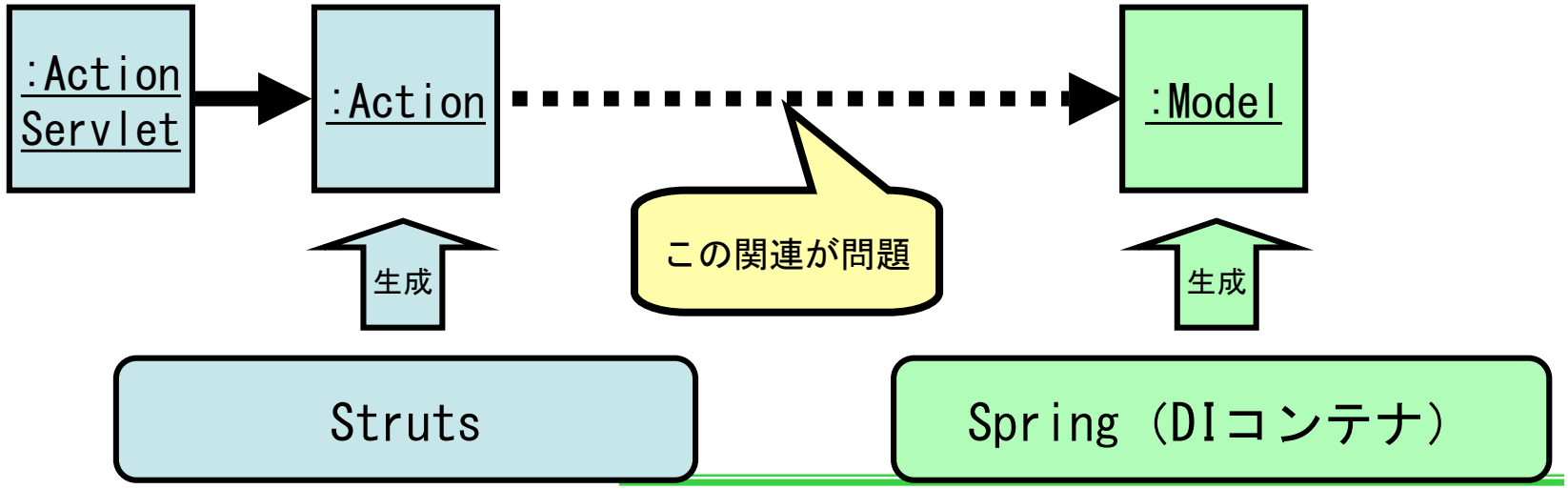
2.0じゃないけど～Struts連携

- 代理クラスを利用する
  - DelegatingActionProxy
- クラスを継承する
  - ActionSupport
- リクエストプロセッサを置き換える
  - AutowiringRequestProcessor

XMLが増えてイマイチ

今まではコレ

おすすめ！



JSUG

AutowiringRequestProcessor

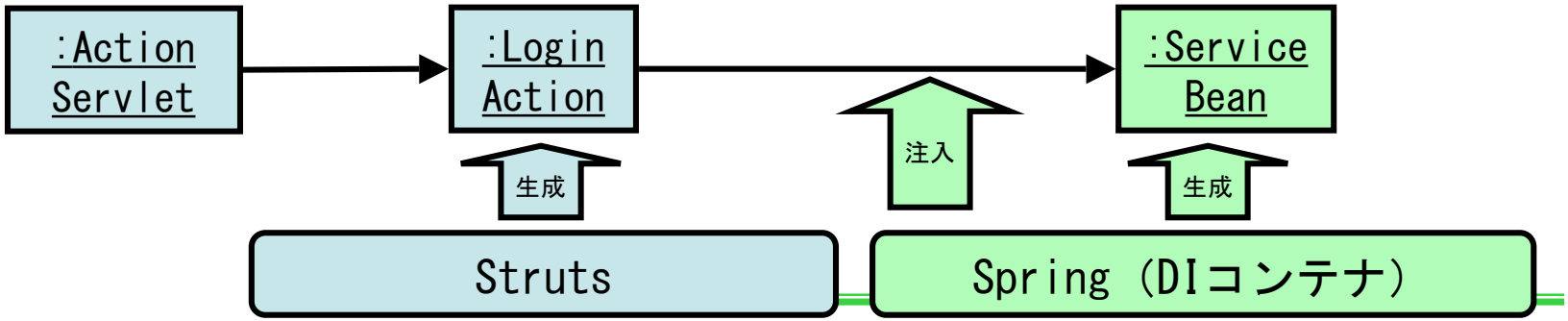
struts-config.xml

```
...
<controller
    processorClass="org.springframework.web.struts.AutowiringRequestProcessor" />
...
```

RequestProcessorの置き換え。

Actionクラス

```
public class LoginAction extends Action {
private ServiceBean serviceBean;
...
public void setServiceBean(ServiceBean serviceBean) {
  this. serviceBean = serviceBean;
}
...
```



JSUG

更にSpring

- Spring IDE
  - Bean定義ファイルの記述が簡単
- Spring Batch
  - 期待は大
  - でも今のところ使えない・・・乞うご期待

Springの情報を入手する

- JSUG
  - http://springframework.jp/
- JSUGメーリングリスト
  - http://groups.google.co.jp/group/jsug
- JSUG勉強会の資料
  - http://groups.google.co.jp/group/jsug/files
- Spring本家
  - http://www.springframework.org/

ありがとうございました

ライセンスについて

JSUGマスコットアイコン（本スライド左下）が残されている場合に限り、本作品（またそれを元にした派生作品）の複製・頒布・表示・上演を認めます。

非商用目的に限り、本作品（またそれを元にした派生作品）の複製・頒布・表示・上演を認めます。

本作品のライセンスを遵守する限り、派生作品を頒布することを許可します。