

**Grails** で  
システムを  
作る  
感覚

日本 Grails/Groovy ユーザーグループ  
yamadamasaki@jggug.org  
masaki@metabolics.co.jp

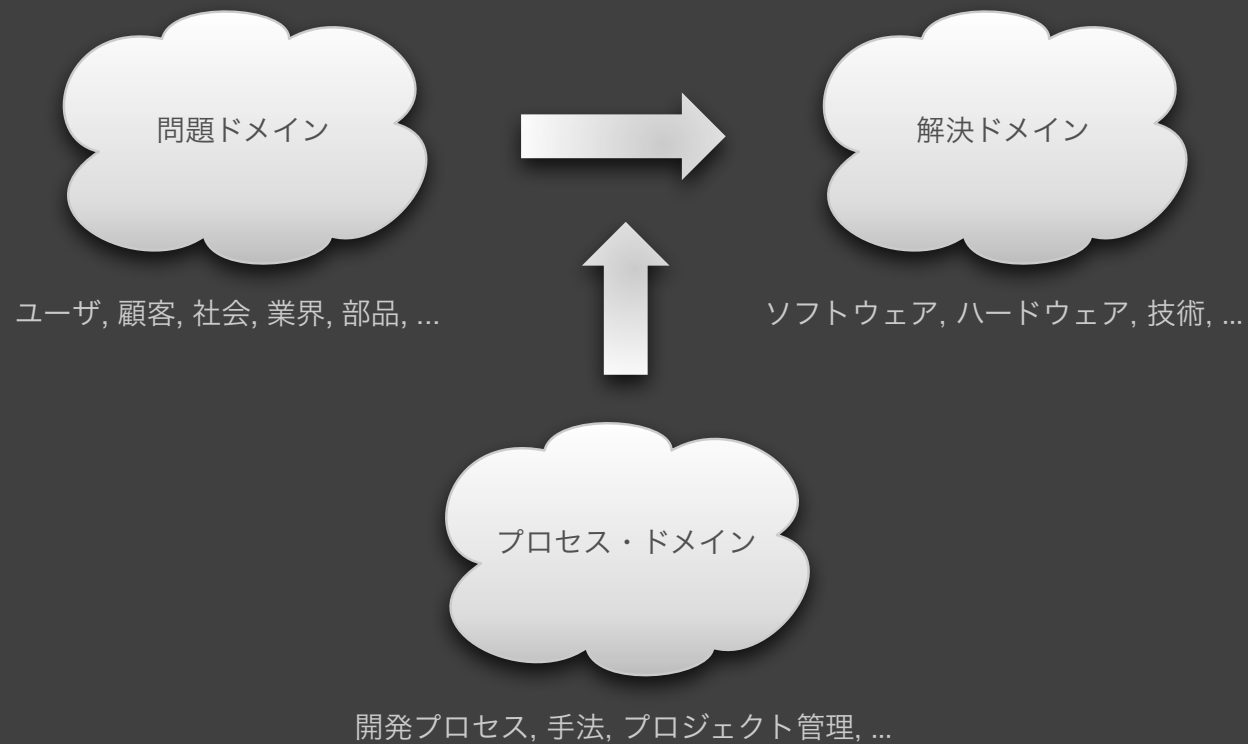


1. ドメインを**見つけ**よう
2. ドメインを表す**言語**を作ろう
3. ドメインの**実現**方法を考えよう
4. ドメインを**表現**しよう
5. **プラグイン**にしよう
6. ドメインを**組み合わせ**よう
7. ドメインを**再利用**しよう

# ドメインって**何**だ

- 支配
- 王国, 領土, 領地
  - 言語, 宗教, 税法, 慣習などが共通
- 共通の性質を共有する領域

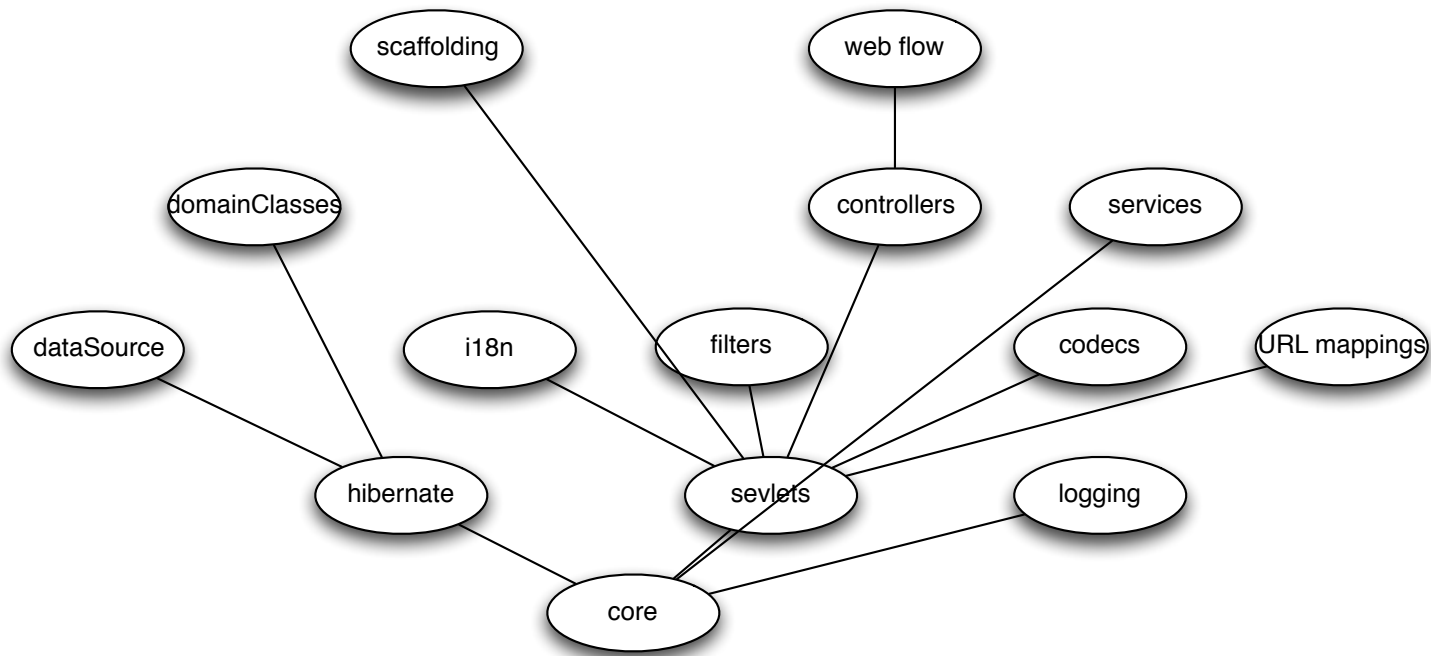
# 典型的なドメイン



# ドメインを見つけよう

- 問題ドメインでは, 例えば
  - ワークフロー
  - 組織構造
- 解決ドメインでは, 例えば
  - データ構造 (ドメイン・クラス, 概念クラス)
  - 画面遷移
- プロセス・ドメインでは, 例えば
  - テスト
  - 構成管理
- アプリケーション固有のドメインも有り

# 組み込みドメインの例



当然だけど、ほとんどが技術ドメインに属する

# ドメインを表す言語を 作るう

- 例えば概念クラスの場合
  - これは標準的なドメインなので, 組込みで用意されている
- 例えばワークフローの場合
  - これはまだないので, 作らなければならない
- 例えばマスタ画面テストの場合
  - これは概念クラスの言語を流用できるのではないか

# 概念クラスの例

```
class Product implements Comparable, Serializable {  
  
    String name, description  
  
    static constraints = {  
        name(blank:false, unique:true)  
        description(blank:false, validator: {  
            it?.indexOf("<script") < 0 &&  
            it?.indexOf("<link") < 0  
        })  
    }  
  
    static mapping = {  
        cache usage:"transactional"  
    }  
  
    static belongsTo = [category:Category]  
  
    static searchable = true  
}
```

← 制約記述DSL

プラットフォーム非依存  
scaffoldなどでも利用

← ORM DSL

Hibernate依存

← 単なるプロパティだが  
解釈される

← これはまた別のドメインで解釈

# ワークフローの例

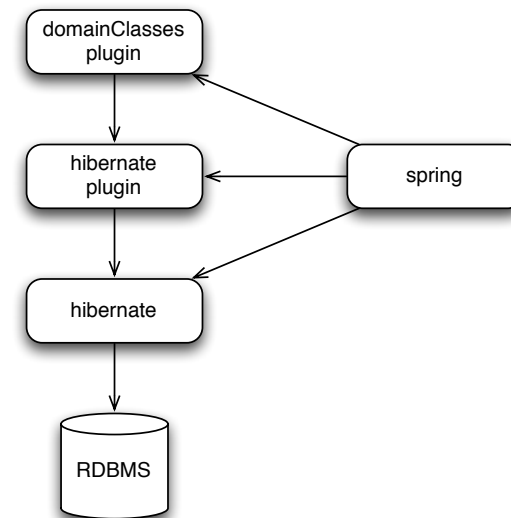
```
def OrderProductTestProcess = {                                     ←プロセスを定義
    OrderFormWait(isStart:true, dueDate:0,                        ←ノードを定義
nodeInfo:'Product Order') {
    variable(productId:READ_ONLY, catalogId:READ_ONLY,
              price:READ_ONLY, quantity:WRITE_READ,
              commentary:WRITE_READ)                             ←変数を定義
    action {                                                     ←アクションを定義
        Log(logMessage:'Starting ordering product...')
        return Order()
    }
    on('Order').to(['CheckingOrder'])                             ←遷移を定義
}
// ...
}
```

# ドメインの**実現**方法を 考えよう

- 実現方法の選択肢
  - 既に Java で使える技術はないか
  - JVM 上で使える技術はないか
  - それ以外で使える技術はないか
  - 自前で作ることができるか
    - Groovy は簡単!
- 技術を言語で “wrap” する
- 仕様を実装に “compile” する

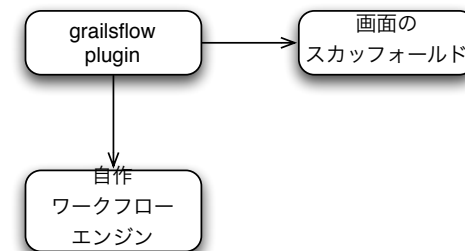
# 概念クラスの例

- hibernate を抽象化
  - \*.cfg.xml などを書く必要はない
  - ドメイン言語から生成
  - 詳細も設定可能
- ORM を抽象化
  - 問い合わせ言語を高レベル化
  - プログラミング言語と自然に統合
- 概念クラスの記述から自動的に永続化



# ワークフローの例

- ワークフロー記述を抽象化
- 実際には自前のワークフロー・エンジンを搭載
- ワークフローに対応する画面も生成
- エンジンを交換することもあり得るだろう



# 実現方法

- メタプログラミングを活用する
- クロージャを活用する
- Map をあたかもオブジェクトのように使う
- builder や slurper を活用する
- Spring Bean を活用する
- AST変換を活用する (コンパイル時メタプログラミング)
- .....

# ドメインを表現しよう

- 言語を用いて、個々のドメインを実際に表現してみる
  - うまく表現できる？
  - ユーザでも理解できる？
  - 固すぎない？
- 必要ならば拡張しよう
  - 拡張しやすい？
  - 実現可能？

# プラグインにしよう

- 言語, 実装, アダプタをセットにする
- 必要ならば
  - 実行独立プラグイン (CIP)
  - プラットフォーム独立プラグイン (PIP)
  - プラットフォーム特化プラグイン (PSP)
  - などに分ける

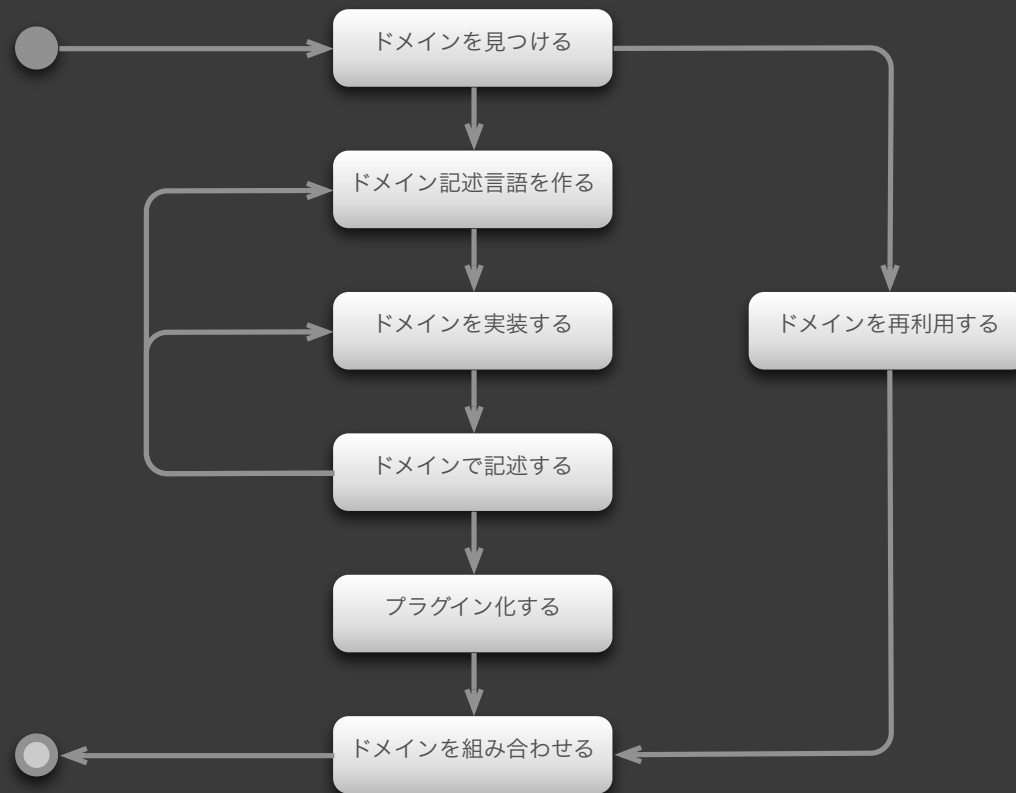
# ドメインを 組み合わせよう

- システムを作るとは, ドメインを組み合わせること
- 一部は従来のような“ベタな”コードでもいい

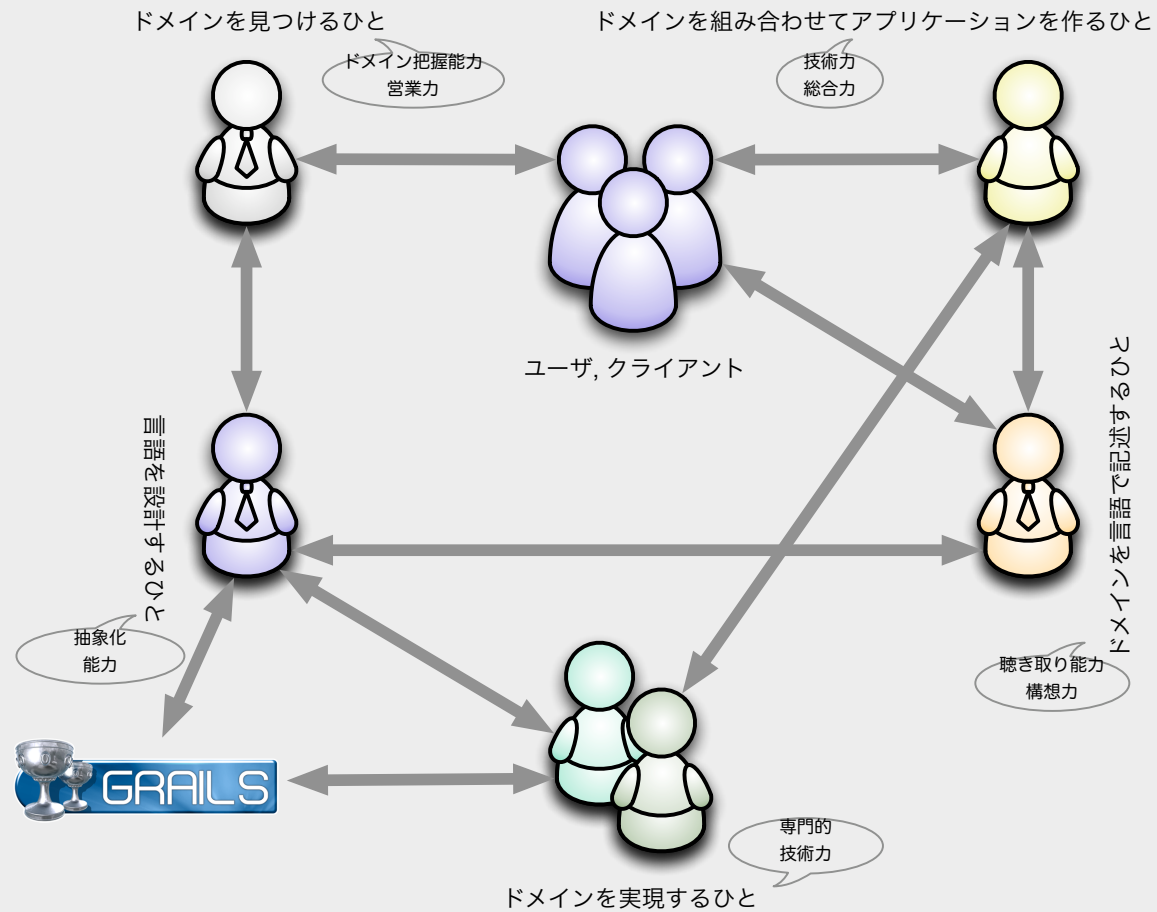
# ドメインを 再利用しよう

- 多くの場合, ドメインは再利用可能
  - コンポネントよりも再利用性が高いことが多い
- プラグインはドメインを実体化したもの
- 必要ならば拡張したり, 洗練したり, 汎用化したりする
- ドメインは資産である

# grails 開発プロセス



# 役割



- コーディング/テストの**工数**を大きく減らすことができる
- 仕様記述により, 開発や変更, 保守が**迅速, 容易**になる
- 技術や問題に関するノウハウをプラグインとして**資産化**できる
- チームのさまざまな**能力**を活かすことができる
- **労働集約型から知識集約型へ**

<http://www.jggug.org/>

日本 Grails/Groovy ユーザーグループは  
皆さんの参加をお待ちしています

毎月 g\* workshop を開催  
名古屋支部, 関西支部もあり

JAPAN GRAILS/GROOVY USER GROUP  
INFO@JGGUG.ORG  
HTTP://WWW.JGGUG.ORG/  
2009.04.21

