NTTソフトウェア

Grails

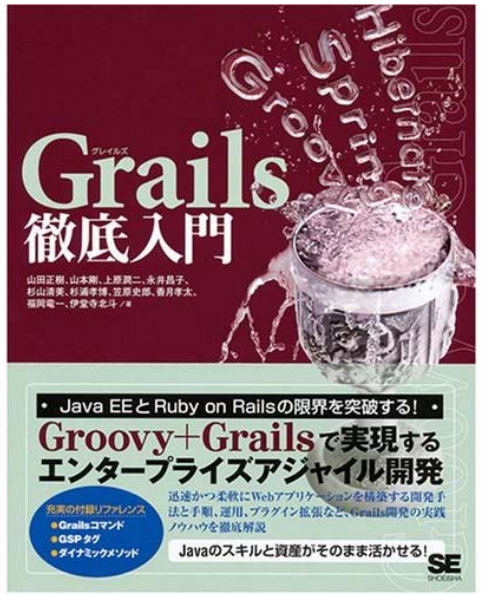
Groovyベースの高生産性 Webアプリフレームワーク

Seasar Conference 2008 Autumn 2008/9/6

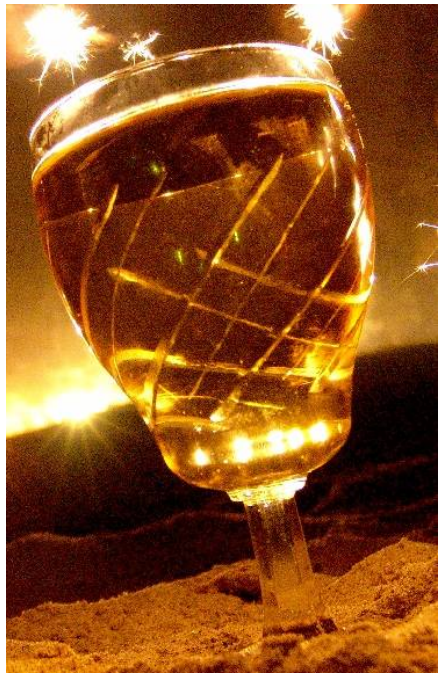
NTTソフトウェア株式会社　上原潤二

発表者

- 上原潤二
  - NTTソフトウェア株式会社
  - 翔泳社「Grails徹底入門」執筆(共著)
  - ブログ「Grな日々」
  - Grails Code Reading メンバ



発表内容



- Grailsとは何か
- GrailsとGroovy
- Grailsの特徴
- まとめ

1. Grailsとは何か

簡単に言うと

- Java EEを基盤とするWebアプリフレームワーク
  - .war生成
  - Spring Framework、Hibernate、SiteMesh,etc,をベース
  - Groovy DSLで開発
  - プラグイン機構

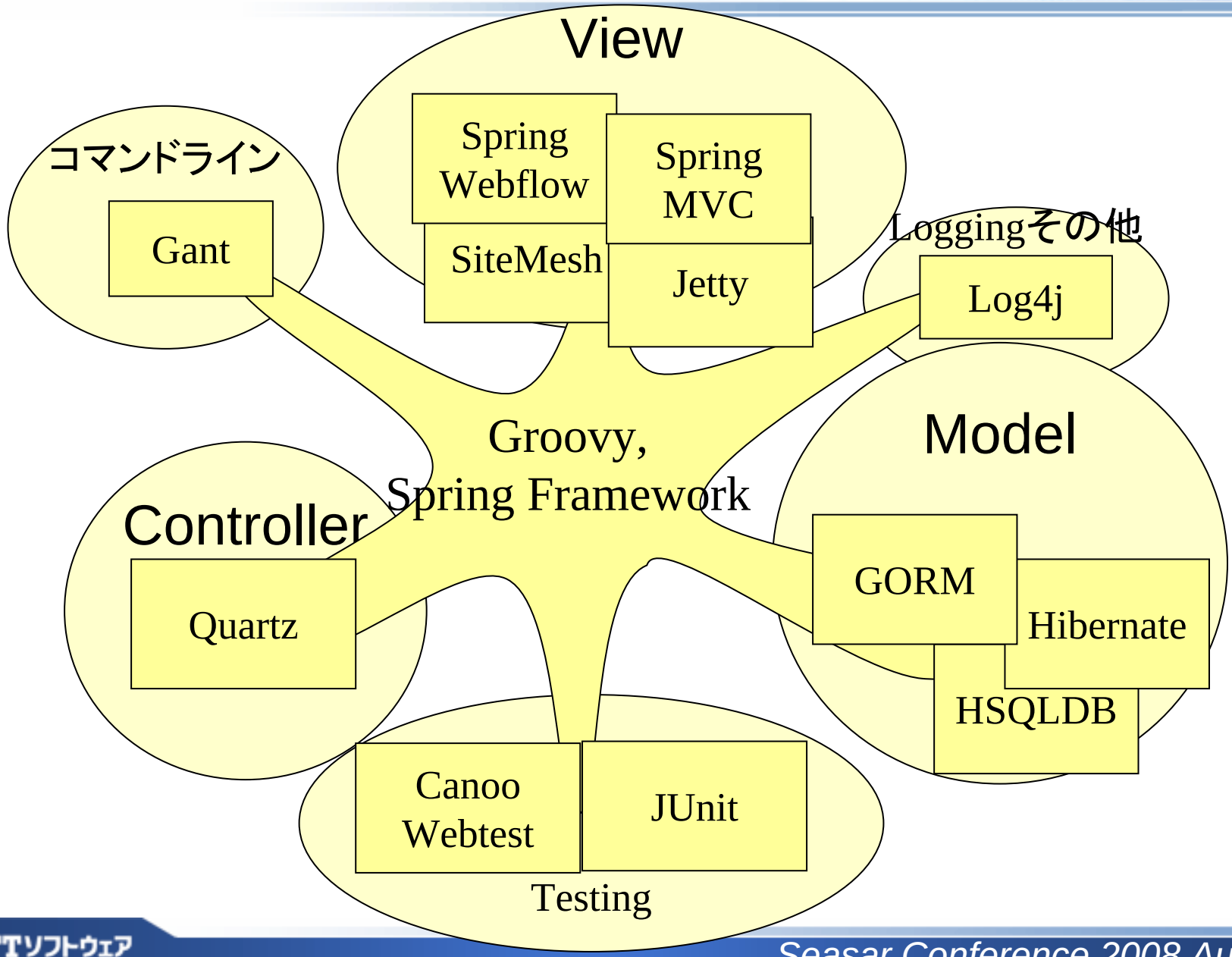
Grailsは,かつて何であったか

- 幼名「Groovy on Rails」
  - Ruby on RailsのGroovy版
  - 大きな影響
  - 多数のアイデア継承

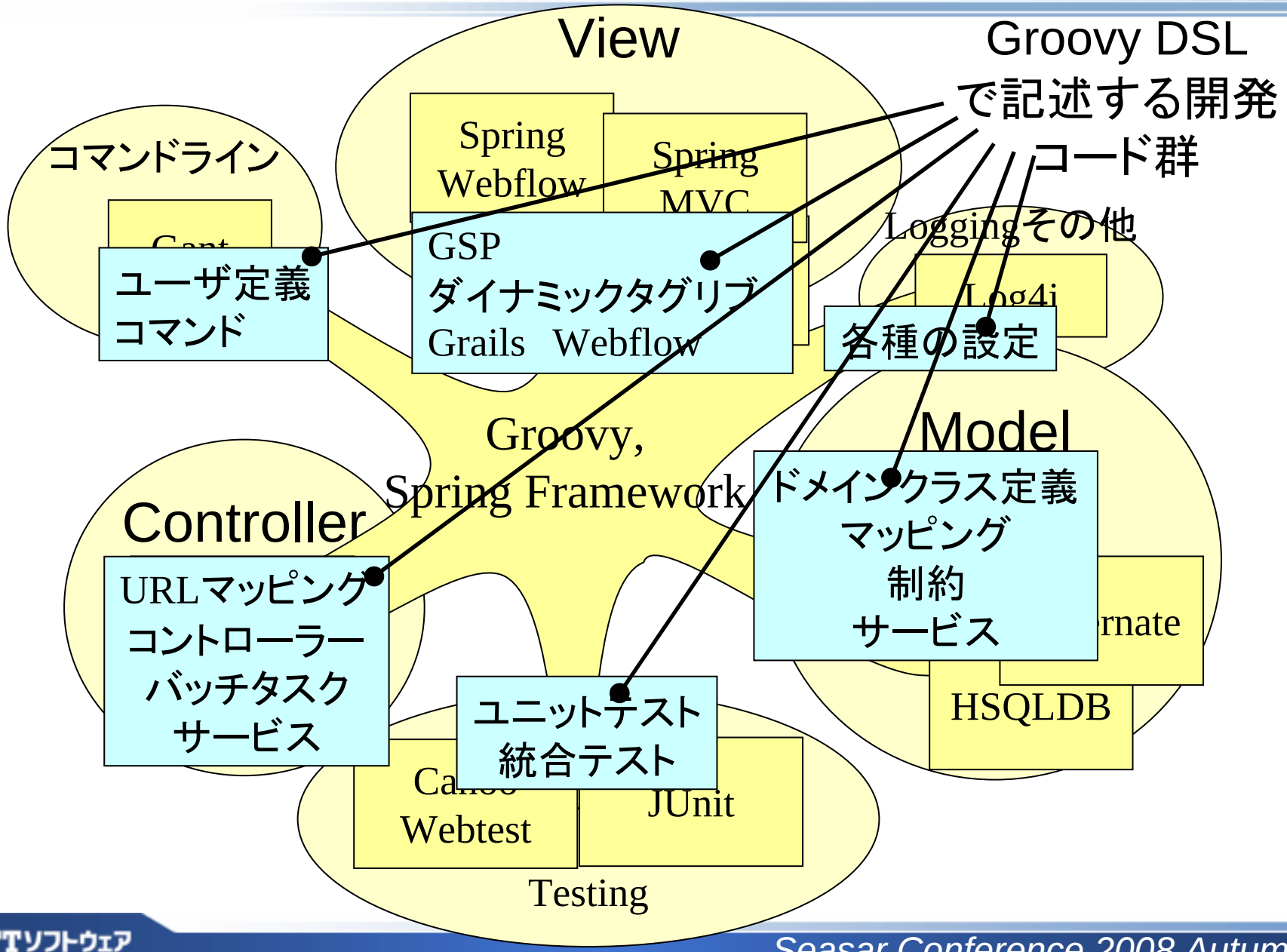
Grailsは，今や何であるか

- 2つの意味でメタフレームワーク
  - フレームワークの集合体(フルスタック)
  - フレームワークを作るためのフレームワーク
    - 「DSL指向フレームワーク作成キット」
    - Grailsは実現サンプルの一つ

Grailsの構成


View
Spring Webflow
Spring MVC
SiteMesh
Jetty
コマンドライン
Gant
Loggingその他
Log4j
Groovy,
Spring Framework
Model
Controller
Quartz
GORM
Hibernate
HSQLDB
Canoo Webtest
JUnit
Testing

Grailsの構成


View
Groovy DSL
で記述する開発
コード群
コマンドライン
Spring
Webflow
Spring
MVC
Loggingその他
GSP
ダイナミックタグリブ
Grails Webflow
ユーザ定義
コマンド
各種の設定
Log4j
Groovy,
Spring Framework
Model
Controller
ドメインクラス定義
マッピング
制約
サービス
URLマッピング
コントローラー
バッチタスク
サービス
HSQLDB
ユニットテスト
統合テスト
Webtest
JUnit
Testing

Grailsの歴史

- 2006年3月20日に0.1版リリース
- 2008年2月4日、1.0版リリース
- 2008年8月25現在、1.0.3最新版

- 欧州で開発が活発

2. GrailsとGroovy

Grails & Groovy

- 主な開発者は両方のコミッター
- Grailsの成果も逐次Groovyにフィードバック

- Groovy：Grailsの最重要技術
- Grails　：Groovyのキラー応用

Groovyとは何か(1)

- 「GroovyはJava言語の拡張である」

C → C++

Java → Groovy

Javaと共有

- JVM
- 動作原理/ライブラリ共通
- 開発スキル

Groovyとは何か(2)

C++ = C + オブジェクト指向

Groovy = Java + 

- 動的
- 高抽象・簡潔記述
- 拡張可(内部DSL)

クロージャ活躍

Groovyコード例

Java

```
import java.util.*;
public class HashMapTest {
   public static void main(String[] arg) {
      Map<String,Integer> map = new HashMap<String,Integer>();
      map.put("太郎", 35);
      map.put("次郎", 30);
      map.put("三郎", 20);
      for (Map.Entry<String,Integer> entry : map.entrySet()) {
         System.out.println(entry.getKey()+":"+entry.getValue());
      }
   }
}
```

Groovy

```
map = [太郎:35, 次郎:30, 三郎:20]

map.each { println "${it.key}:${it.value}" }
```

「簡潔記述」を強調するGroovyのコード例。
動的性質・拡張性など、他の性質については表現できていない。

3. Grailsの特徴

Grailsの特徴

1. Grails流儀化されたフレームワークの集合体
2. GroovyベースのDSLでロジックと設定を書く。
3. 非グラフィカル統合環境
4. CoC, DRY＋
5. Scaffold
6. プラグイン

Grailsの特徴

1.Grails流儀化されたフレームワークの集合体

2.GroovyベースのDSLでロジックと設定を書く。

3.非グラフィカル統合環境

4.CoC, DRY+

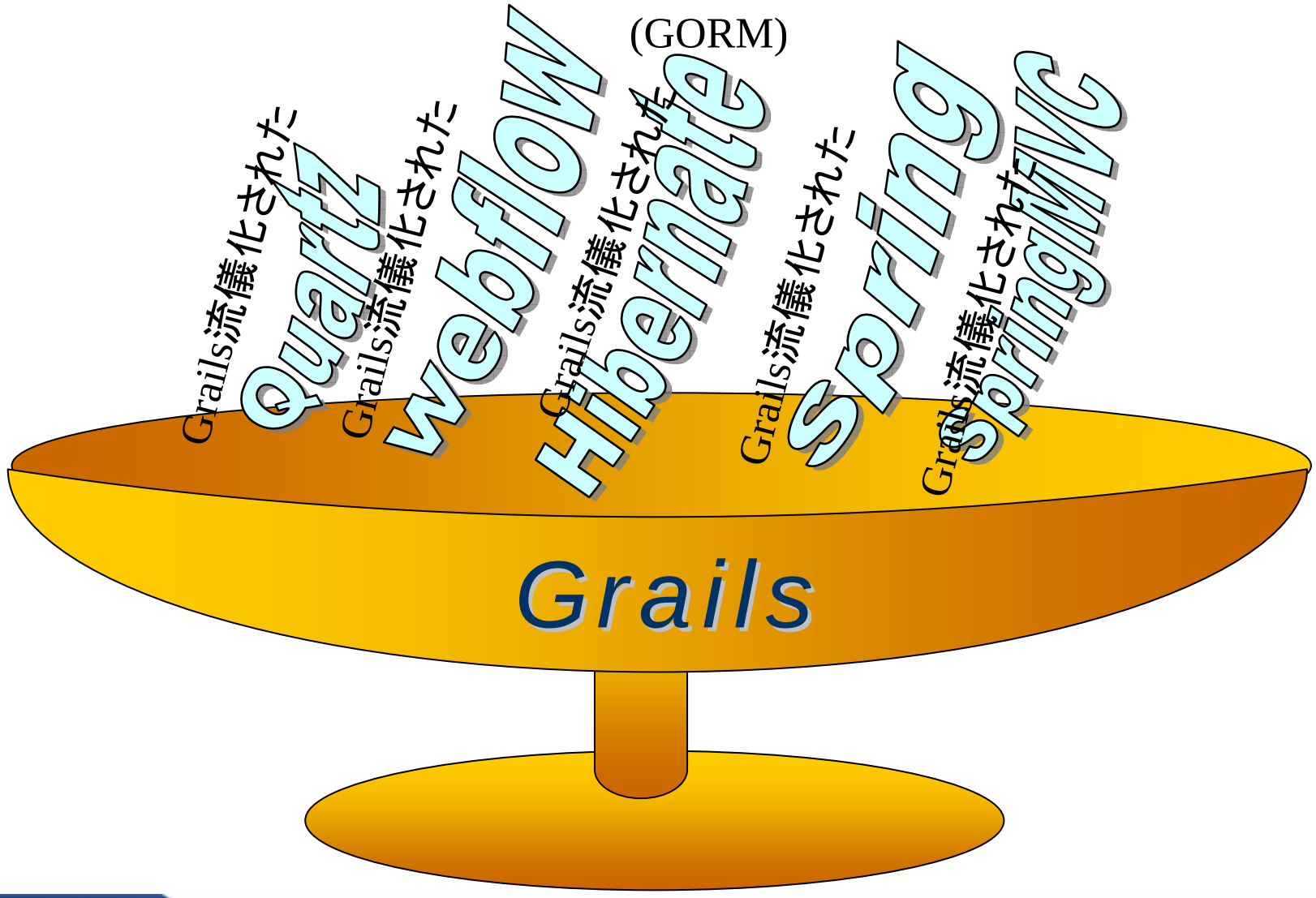
5.Scaffold

6.プラグイン

Grails流儀化

- 既存Javaフレーワークを、下記が可能になるようにWrap
  - 記述・設定をDSLで表現
  - 適度なCoC
  - より動的に
  - プラグイン化

Grails流儀化


Grails流儀化された
Quartz
Grails流儀化された
webflow
(GORM)
Grails流儀化された
Hibernate
Grails流儀化された
Spring
Grails流儀化された
SpringMVC
Grails

Grailsの特徴

1.Grails流儀化されたフレームワークの集合体

2.GroovyベースのDSLでロジックと設定を書く。

3.非グラフィカル統合環境

4.CoC, DRY+

5.Scaffold

6.プラグイン

Groovy DSLで記述(1)

- ドメインクラス

```
class Person {
  int age = 18
  String name
  static constraints = {
    age min:18, nullable:false
    name size:10..30
  }
  static belongsTo = Group
}
```

テーブル定義に該当

制約記述用DSL

関連の表現

- “grails-app/domain”配下に置く

Groovy DSLで記述(2)

ドメインクラス

```
class Person {
  int age = 18
  String name
  static constraints = {
    age min:18, nullable:false
    name size:10..30
  }
  static belongsTo =
}
```

スキャフォルド生成されたビュー

```
        :
<input type="text“
   maxlength="30" id="name“
   name="name" value=""/>
```

テーブル定義

```
create table person (
  id bigint generated by default
    as identity (start with 1),
  version bigint not null,
  age integer not null,
  name varchar(30),
  primary key (id)
)
```

Groovy DSLで記述(3)

- URLマッピング

```
class UrlMappings {
  static mappings = {
    "/$blog/$year?/$month?/$day?/$id?" {
      controller = "blog"
      action = "show"
      constraints {
        year(matches:/d{4}/)
        month(matches:/d{2}/)
        day(matches:/d{2}/)
      }
    }
    "500"(view:'/error')
}
```

URLパターンにマッチングさせると同時に部分文字列を変数に代入

追加的マッチング条件

Groovy DSLで記述(4)

- Quartz(プラグイン)

```
class MyJob {
   def cronExpression = “0 0 24 * * ?”
   def execute() {
      // 処理実行
   }
}
```

- “grails-app/jobs”配下に書くだけでバッチ実行

Groovy DSLで記述(5)

- ダイナミックTaglib

```
class HogeTaglib {
  def mytag = {
        out << "HELLO"
   }
}
```

- grails-app/taglibに置くだけ
- GSP中の<g:mytag/>が"HELLO"に展開
  - ~~.tld XML、コンパイル、jar~~ ←不要
  - 単体試験しやすい

その他DSL大活躍

- 設定ファイル一般(ConfigSlurper)
- フィルタ記述
- GSP(just like JSP, but not sucks!)
- Spring Bean設定(BeanBuilder)
- Webflowの画面遷移フロー定義
- Hibernateマッピング定義
- Hibernateのクエリ定義
- Json記述(JSonBuilder)

:

Groovyをとことん使うための

- Groovyは簡単。しかし以下は困難。
  - DSLを作る
  - 動的性質を最大限引き出す(リロード、更新伝播)
  - CoC、DRYを駆使
  - 既存Javaライブラリ・フレームワーク類を、一定の流儀で統合
- Grails:上記の定式化された仕組み

Grailsの特徴

1. Grails流儀化されたフレームワークの集合体
2. GroovyベースのDSLでロジックと設定を書く。
3. 非グラフィカル統合環境
4. CoC, DRY原則
5. Scaffold
6. プラグイン

grailsコマンド群

- 定型処理コマンド群

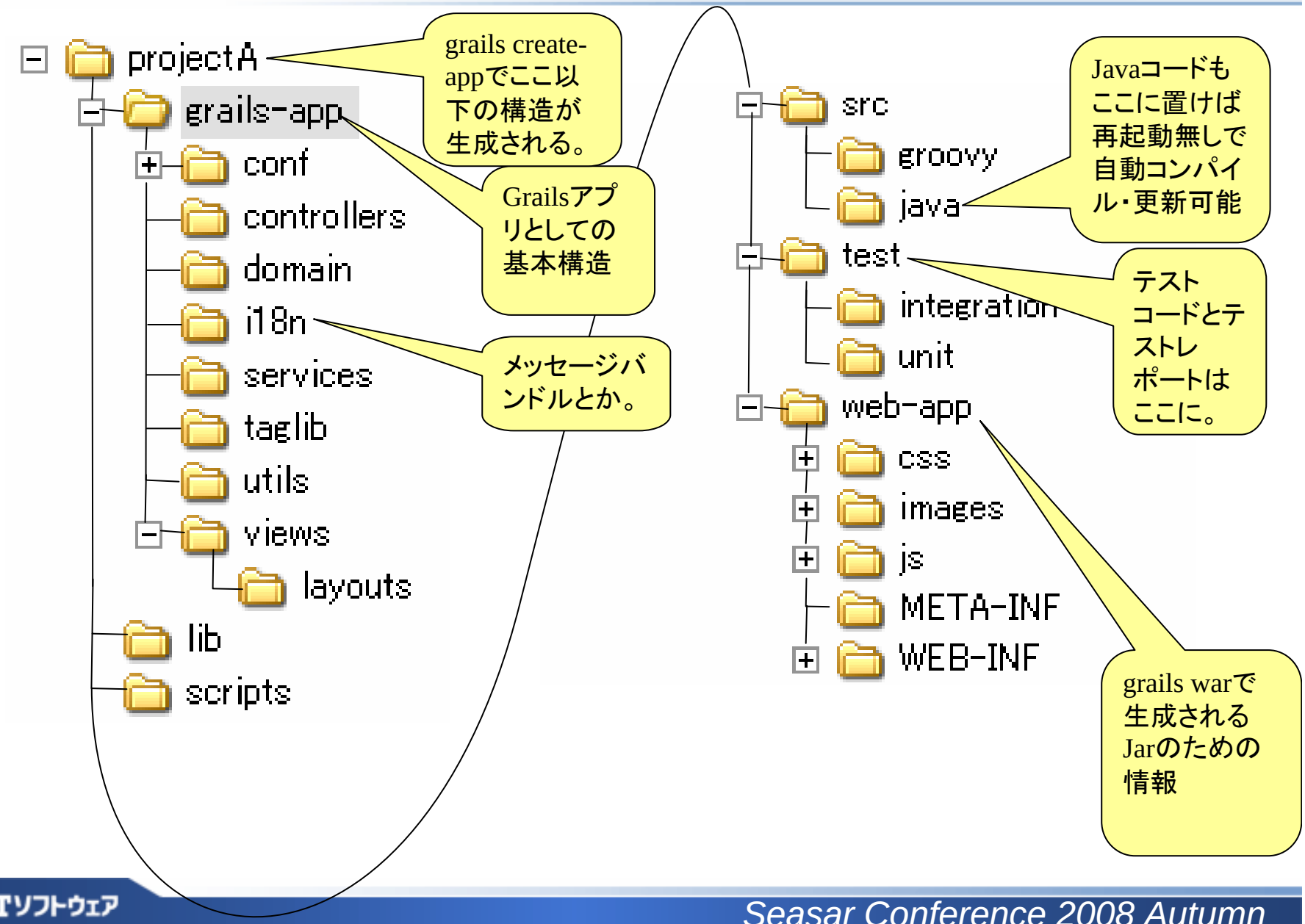
grails <command>

Command:

```
bootstrap                bug-report               clean
compile                  console                  create-app
create-controller        create-domain-class      create-integration-test
create-plugin            create-script            create-service
create-tag-lib           create-unit-test         doc
generate-all             generate-controller      generate-views
help                     install-plugin           install-templates
list-plugins             package-plugin           package
plugin-info              release-plugin           run-app-https
run-app                  set-proxy                set-version
shell                    stats                    test-app
upgrade                  scrip                    war
```

- それぞれの実体はGant(Groovy版rake)スクリプト
  GRAILS-APP/scripts/XxxYyy.groovy

プロジェクトフォルダ構成



Grailsの特徴

1. Grails流儀化されたフレームワークの集合体
2. GroovyベースのDSLでロジックと設定を書く。
3. 非グラフィカル統合環境

4. CoC, DRY＋
5. Scaffold
6. プラグイン

CoCの例

- クラス名が“Service”で終わるクラスは自動的にインスタンス化されて依存性注入

HogeController.groovy

```
class HogeController {
  def hogeService
        :
   hogeService.sayHello()

}
```

HogeService.groovy

```
class HogeService {
     void sayHello() {
     }

}
```

※ちなみにServiceクラスは(断らない限り)トランザクション境界

CoC with DSL

- XML設定ファイルは書かない
  - 適度なCoC
  - 規約の動作を変える場合、DSLで明示指定

DRYの先にあるもの

- リピートしないのは当然
  - 記述を極小化・最適化
- 「必要最低限の最適な記法」を考えるところから入る
  - 最適な記述>>>実装都合・制約

Grailsの開発手順(1回目)

1. 最適なDSLを考える
2. そのDSLを実装する
3. そのDSLで業務ロジックを書きくだす
4. 実行する

Grailsの開発手順(2回目以降)

1. 最適なDSLを考える
2. そのDSLを実装する

SKIP

1. そのDSLで業務ロジックを書きくだす
2. 実行する

Webアプリ開発に関していまここ(Grails)。

Grailsの開発手順(2回目以降)

1. 最適なDSLを考える
2. そのDSLを実装する

SKIP

1. そのDSLで業務ロジックを書きくだす
2. 実行する

Webアプリ開発に関していまここ(Grails)。

1,2は必要に応じプラグイン化

従来の開発手順(例)

1. UMLで論理モデルを書き下す
   - 簡潔に書き下すための特別な記法、業界慣例の記法を活かす、などの希望はかなえられにくい。
2. (必要に応じ)UMLで実装モデルに書き直す
3. フレームワークなどの都合に合わせ、JavaやXMLなど既存言語で、実装都合に合わせ全面的に書き直す。
   - 最小記述とか抽象性とかの希望よりも、実装技術制約が優先される。
4. 実行する

「2回目以降」も同様

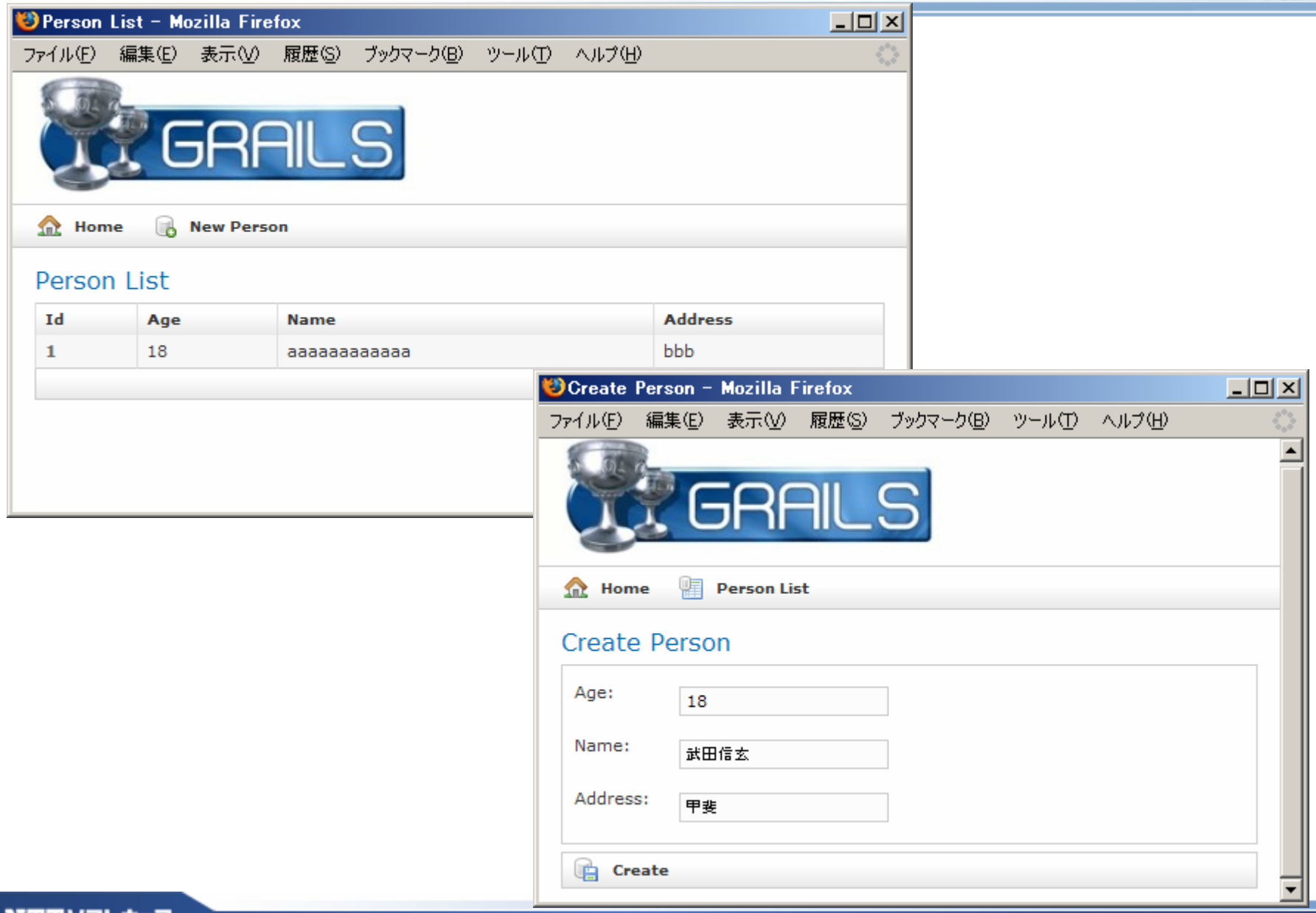
Grailsの特徴

1. Grails流儀化されたフレームワークの集合体
2. GroovyベースのDSLでロジックと設定を書く。
3. 非グラフィカル統合環境
4. CoC, DRY原則
5. Scaffold
6. プラグイン

Scaffold

- CRUDビューの自動生成
  - 静的スキャフォールド
  - 動的スキャフォールド
    - GSPページやコントローラクラスはファイルとしては存在しない
    - テンプレートから動的に生成
- マスター管理系は非常に楽

Scaffold画面例



Grailsの特徴

1. Grails流儀化されたフレームワークの集合体
2. GroovyベースのDSLでロジックと設定を書く。
3. 非グラフィカル統合環境
4. CoC, DRY原則
5. Scaffold

6. プラグイン

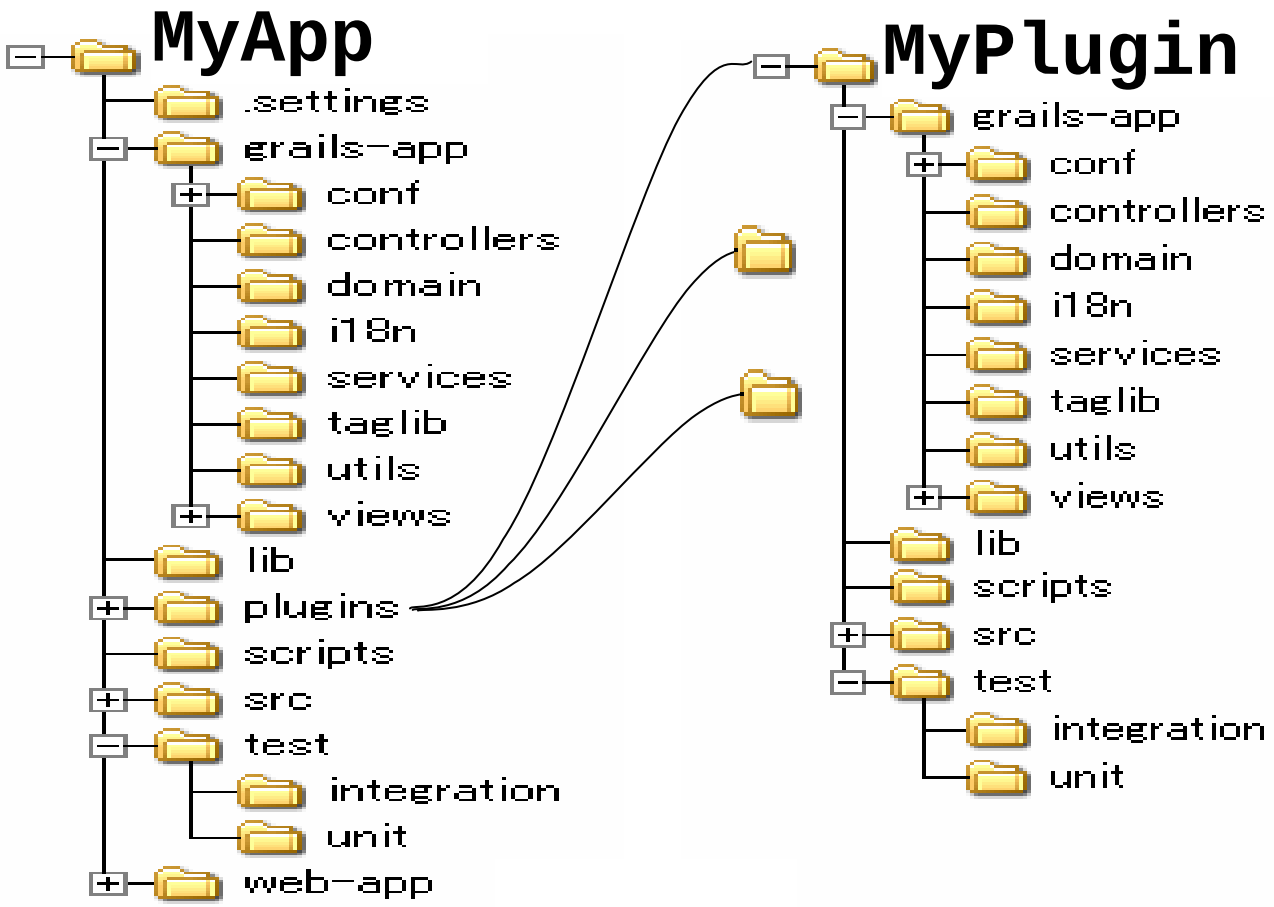
Grailsプラグイン

- 拡張モジュール
  - コマンド1発オンラインインストール
  - 更新検出/伝播、リロードの枠組み
  - 豊富なプラグイン(標準リポジトリで92個)
- Grails・・・プラグインの集合
  - `Codecs-plugin, Controllers-plugin, Hibernate-plugin, DomainClass-plugin, UrlMappings-plugin・・・`

- 再利用単位として秀逸

再利用単位としてのプラグイン

- プラグインはアプリケーションと相似形



アプリケーションと相似形なので

- プラグインは
  - ビューを含むことができる
  - テストを含むことができる
  - サポートスクリプトを含むことができる
  - 配布単位として独立性が高い
- クラス・コンポーネントより粗粒度

- アプリ開発と同列のサポート

プラグインが再利用しやすい理由

- DSLには、
  実装都合による依存・結合が
  ユーザコード上に現れない(or少ない)
  - DSL表現は「受身」の存在
  - データに近い

4. まとめ

Grailsは

- Javaプログラマー向けの、
- アジャイル開発を容易にする、
- 拡張を前提としたフレームワーク
  - プラグインによる再利用
  - DSL指向開発

Grailsの問題点

- Grailsによる隠蔽は十分?
  - Springは当面良く知らなくてもOK
  - Hibernateは会得必須
- 性能面
  - Javaより遅いのは確実
  - Groovy1.6で高速化期待
- デバッガ欲しい

参考情報・その他

日本語書籍

- 「Grails徹底入門 ～Groovy＋Grailsで実現するエンタープライズアジャイル開発」翔泳社2008年8月25発売
- 「Groovyイン・アクション」日本語版, 毎日コミュニケーションズ,2008年秋

コミュニティ

- Grails Code Reading
  - http://groups.google.com/group/grails-ja

第2回 07/08/24 groovy
第3回 07/09/19 grails実行ひとめぐり
第4回 07/10/31 プラグイン
第5回 07/11/29 acegiプラグイン
第6回 07/12/21 webflow
第7回 08/01/28 ビュー周り (GSP, taglib)
第8回 08/02/27 モデル周り
第9回 08/03/27 Service周りとCommand Object
第10回 08/04/24 リッチUIを見ました.
第11回 08/05/22 ビルダ
第12回 08/07/4 アーティファクト (+ BeanBuilder, Spring?)

ご清聴ありがとうございました