Spring Framework 2.0
デファクトスタンダードDIコンテナの現在と未来

zuisener@gmail.com

自己紹介

- Java系Webプログラム開発者
- Struts/Spring/iBATISコンビネーション
  - ✓特にiBATIS最高。内部のコード読んでると死ねますが…
  - ✓でも今日はSpringの話です。

本日のメニュー

- Spring 2.0:
  - ✓ **何が変わったのか**
  - ✓ **何ができるようになったのか**
- The Spring Experience 2006:
  - ✓ **何を目指しているのか**

おさらい

- 日本ではSeasar2と並んで
  DIxAOPコンテナの二大巨頭
- POJOベース。テストしやすい。
- XML地獄

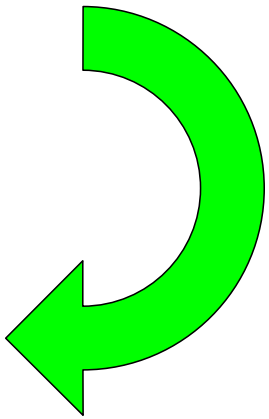
2.0

- **シンプル**になったXMLファイル
- AspectJと統合して**強力**になったAOP
- **その他諸々機能拡張**
  - ✓ **それでいて**1.x**系と下位互換**

DI

Beanのライフサイクル

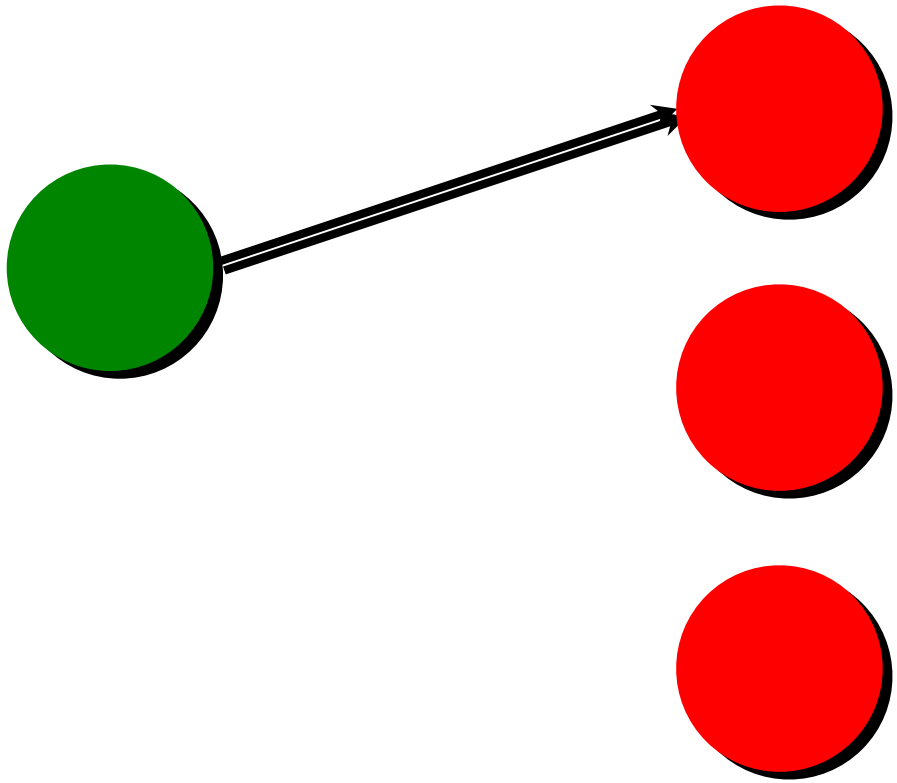
1. **ライフサイクル（スコープ）の種類が増えた**
   - ✓ singleton
   - ✓ prototype
   - ✓ request
   - ✓ session
   - ✓ globalSession
   - ✓ myScope



2. **広いライフサイクルから狭いライフサイクルが参照できるようになった**
3. **ライフサイクルの仕組みが拡張可能になった**

Beanのライフサイクル

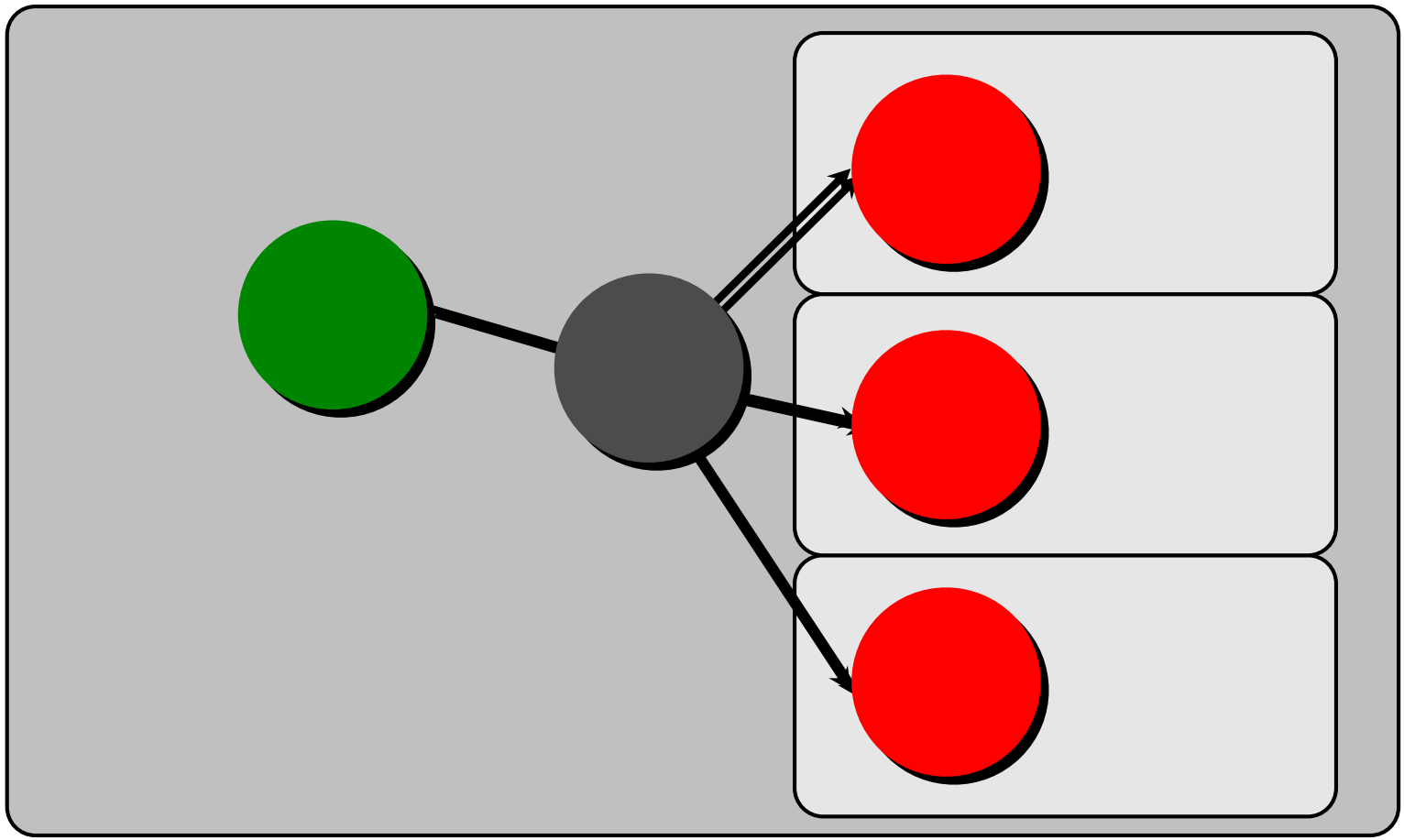
- 今まで



singleton　　prototype

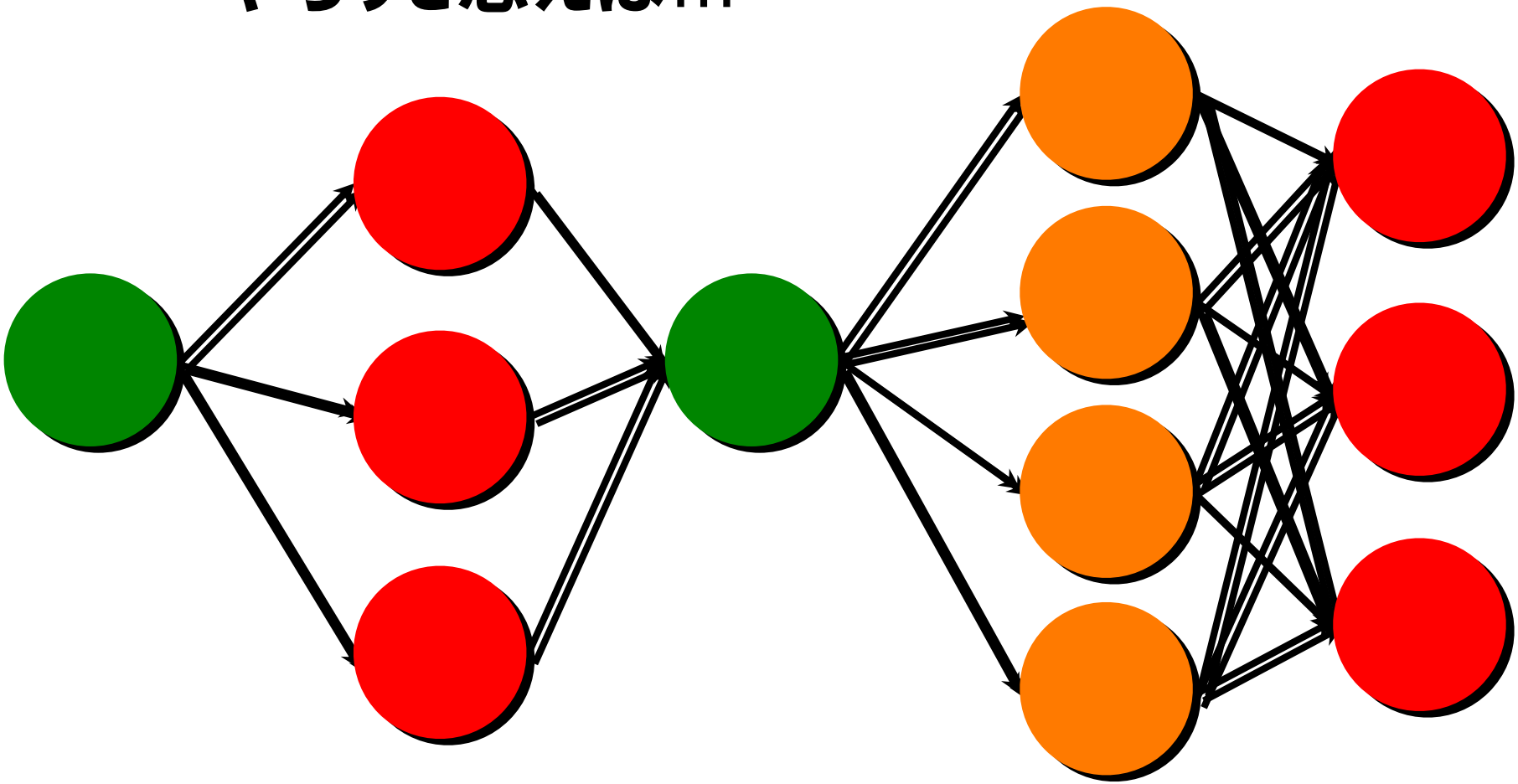
Beanのライフサイクル

- 2.0



Beanのライフサイクル

- やろうと思えば…



singleton session singleton request session

Beanのライフサイクル

- 設定

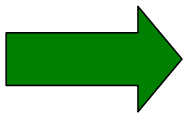
```
<bean id="scopeService" class="jp.co.example.ScopeService"
scope="request">
        session
      globalSession
        singleton
        prototype

  <aop:scoped-proxy/>
</bean>
```

※singleton,prototypeの場合は
あまり意味がないです

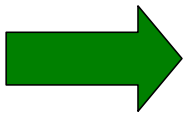
```
org.springframework.web.context.request.
RequestContextListener
```

Beanのライフサイクル

- ライフサイクルの種類が増えた
- ライフサイクルの広いほうから狭いほうを参照できるようになった



Beanに指定したライフサイクルそのままに動作する



セッションごとに状態を分けて管理できる

Beanのライフサイクル

- **注意点**
  - ✓ sessionスコープに入れたBeanは
    セッションタイムアウト/セッション破壊まで残る
  - ✓ **同一sessionから同時多重リクエストが**
    あった場合、同期しないと状態が壊れる

普通のsessionと同じ。
意識しないで使えてしまうので注意

Beanのライフサイクル

- ライフサイクルの仕組みが拡張可能
  - ✓ 独自のライフサイクルを作ることができる
- 例：threadスコープ

XMLファイルの簡易化

- XML設定ファイルが
  DTDベースからXMLスキーマベースに対応した
- 汎用的過ぎるタグから
  機能に応じた書式で書けるようになった

XMLファイルの簡易化

- Springは**XML地獄**

XMLファイルの簡易化

- 2.0

クラス名は
覚えなくていい

```
<jee:jndi-lookup id="dataSource"

                 jndiName          ="jdbc/dataSource"/>
 < jee:          jndiEnvironment >

            username =user
            password= password

 </ jee:jndiEnvironment>
</ jee:jndi-lookup>
```

機能に応じた要素・属性が
定義されている

XMLファイルの簡易化

- 記法が簡単に
  - ✓ 機能に応じた要素・属性
  - ✓ クラス名の隠蔽化
- 拡張可能に
  - ✓ スキーマ定義ファイルを書けば
    オリジナルの記法が追加できる
  - ✓ 社内向けにフレームワーク拡張したときや
    プロジェクトに特化したBean定義が簡単に

XMLファイルの簡易化

- スキーマを作ればオリジナルの要素が作れる
  - ✓ 完全修飾クラス名を
    利用者に覚えてもらわなくてもよい
  - ✓ 決まりきった組み合わせでしか使わないなら
    複数の定義がひとつになり、簡潔になる
- 注意
  - ✓ 一部クラスがDIコンテナの管理から
    外れるので拡張性が落ちる

DI

- まとめ
  - ✓ より**柔軟**に
  - ✓ より**簡潔**に

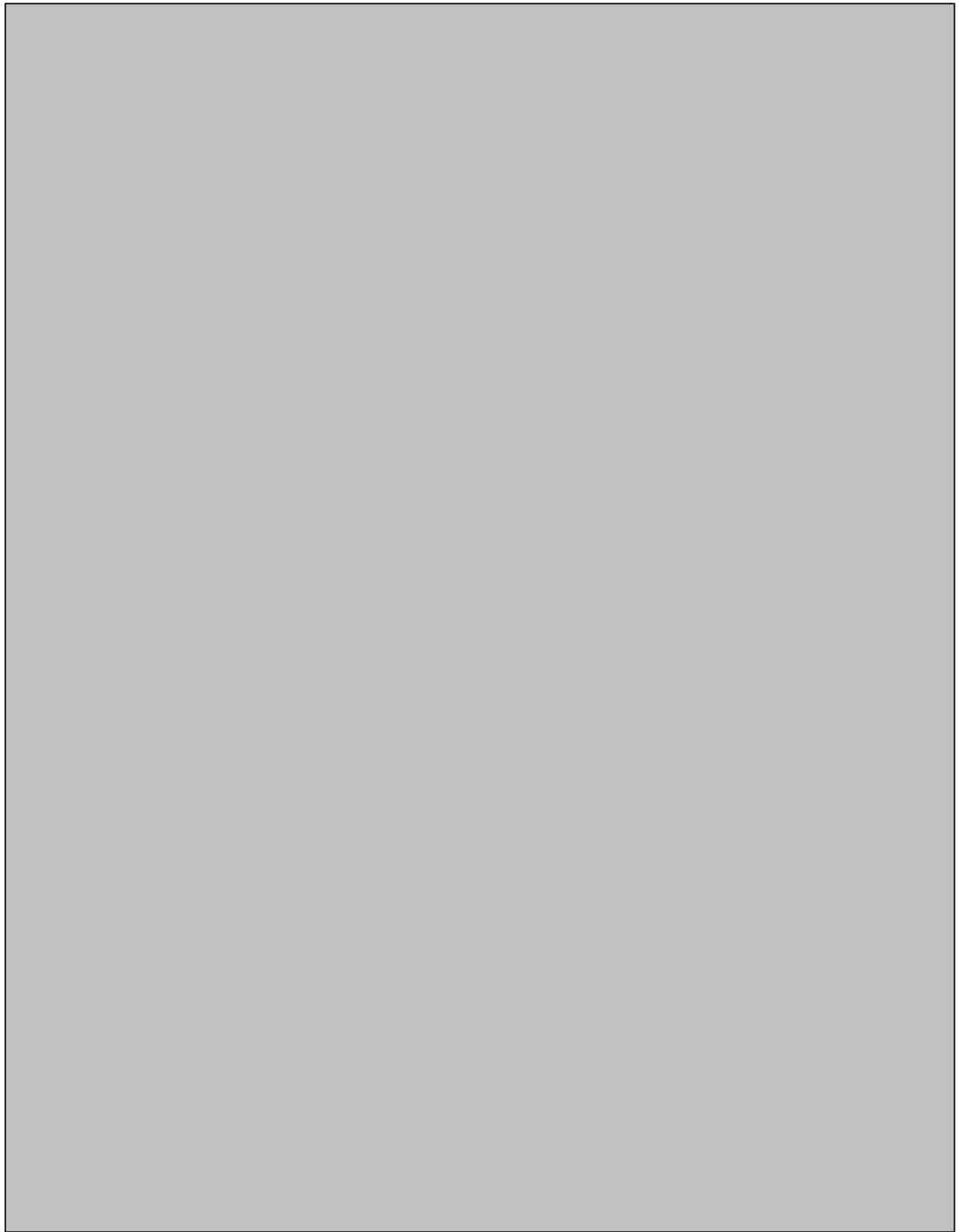
AOP

- Answer Or Problem? -

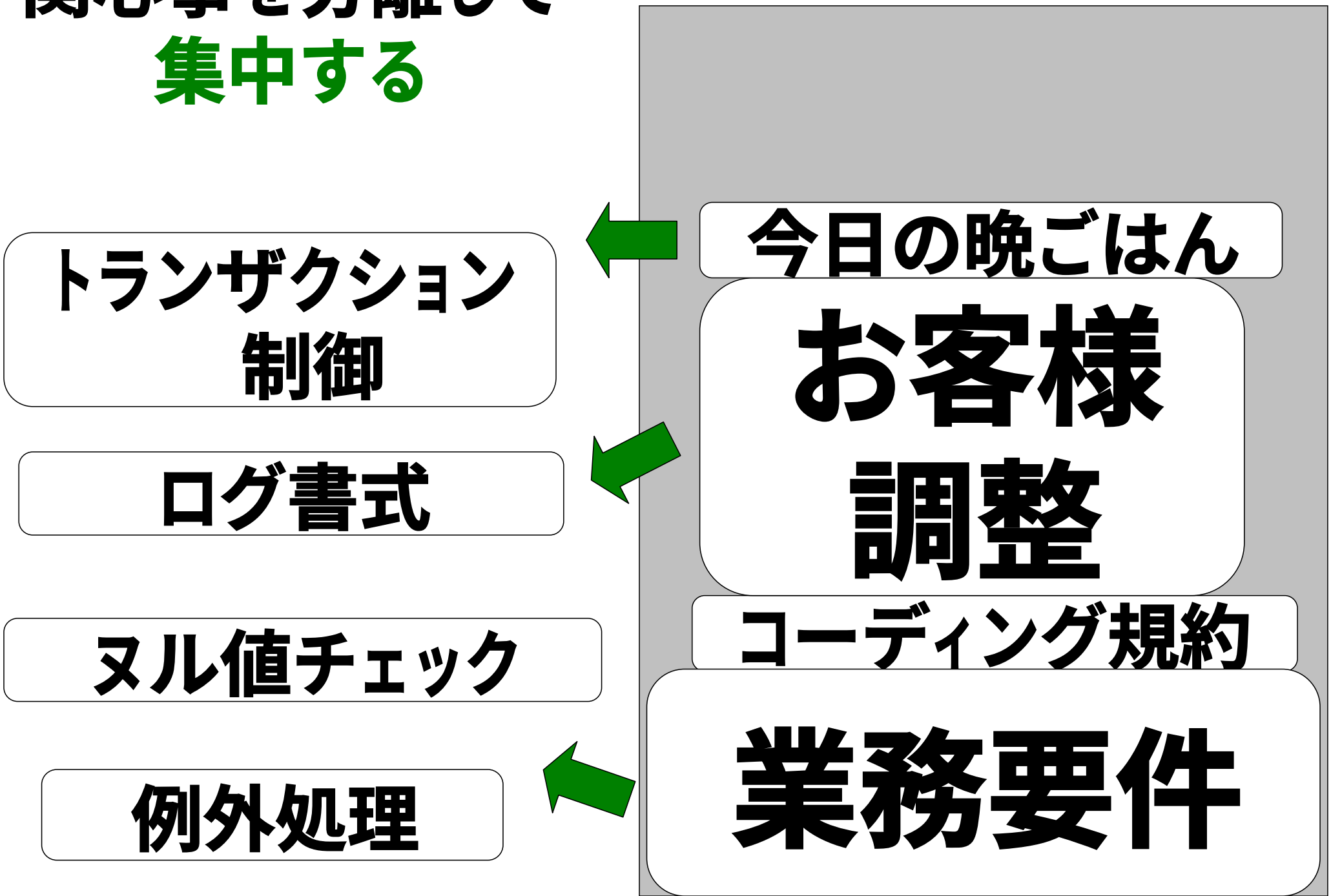
そもそもAOPってなんで必要なんだっけ？

- AOPを使ってみたが予測不能な動きで
  バグ頻出
- AOPを埋め込んだら
  単体テストの意味がないのでは？
- **また新しいことを覚えるのか！！**

AOPはプログラマの
脳力をフル活用する



関心事を分離して
集中する



AOPは銀の弾丸ではない

- すべてをAOPで解決しようとしない
  - ✓ AOPで分離できないこともある
  - ✓ 使い方を間違えれば自打球（自撃弾？）
- 銀の弾丸ではないが鋼鉄の弾丸
  - ✓ うまく使えば有効な武器になる

そもそもAOPってなんで必要なんだっけ？

- AOPを使ってみたが予測不能な動きで
  バグ頻出
- AOPを埋め込んだら
  単体テストの意味がないのでは？

  ➡ 使いどころを絞ろう
- また新しいことを覚えるのか！！

  ➡ 開発するときに特定のAPI / 規約を
  忘れられる薬

もちろんそれ以外にも後付け可能とか色々

AspectJ統合

- AOPの実現方法が増えた
  - ✓ @AspectJスタイル
  - ✓ XMLスタイル
- AOPのかけ方が簡単になった
  - ✓ アノテーション
  - ✓ aopスキーマ

AspectJ統合

- @AspectJスタイル
  - ✓ AspectJ5で登場した「アノテーションを活用した記法」
  - ✓ 完全にJava言語に則っている
  - ✓ AspectJを使ってバイトコードに織り込むこともSpringのプロキシで動的に織り込むこともできる

AspectJ統合

- XMLスタイル
  - ✓ XMLで書く方法
  - ✓ aopスキーマの登場で簡潔に書ける
  - ✓ ソースコードを@でいっぱいにしない

AOP

- まとめ
  - ✓ より**柔軟**に
  - ✓ より**簡潔**に

その他

- Just Plain Cool -

動的言語

- Groovy
  - ✓Java**オフィシャルのスクリプト言語**
- JRuby
  - ✓JVM**上で動作する**Ruby**実装**
- BeanShell
  - ✓**シェルスクリプトのような**Java**系スクリプト言語**

使ってる方いるのでしょうか…

動的言語

- 言葉の壁を越えよう
  - ✓ Beanを動的言語で書けるようになった
  - ✓ Javaのインタフェース＋動的言語の実装

- 動的言語で書かれた部分は稼働中に変更可能

動的言語

- **実装**

Java
インタフェース

```
public interface AccountService {
  int getAmout(String accountId);
}
```

JRuby
実装

```
require ‘java’
include_class ‘AccountService’
class AccountServiceMock < AccountService
  def getAmount
    return 1000
  end
end
```

動的言語

- 実装

```
<lang:jruby id="accountServiceMock"
  script-source="classpath:accountServiceMock.rb"
  script-interfaces="AccountService"
  refresh-check-delay="1000" />
```

設定

動的言語

- 使い道
  - ✓ テストドライバ・モック
  - ✓ フロー制御
  - ✓ プロトタイプ
  - ✓ ビジネスルール記述
  - ✓ GUIコンフィギュレーション
  - ✓ 条件分岐を含むような入力値検証

その他

- まとめ
  - ✓ より**柔軟**に
  - ✓ より**簡潔**に

The Spring Experience 2006

The Spring Experience 2006

- Interface21**主催のセミナー**
  - ✓ Spring**開発者のための技術トピック**
- 2006**年**12**月開催**
  - ✓ 2005**年から毎年開催**
- **約**700**名参加**
- **日本では知名度が低い（低かった）**

The Spring Experience 2006

- **世界中から技術者が参加**
  - ✓ **大体、米国・欧州＞インド＞アジアの感覚**
  - ✓ BEA・Oracle**などがスポンサー**
  - ✓ Sun**からもスピーカーが参加**

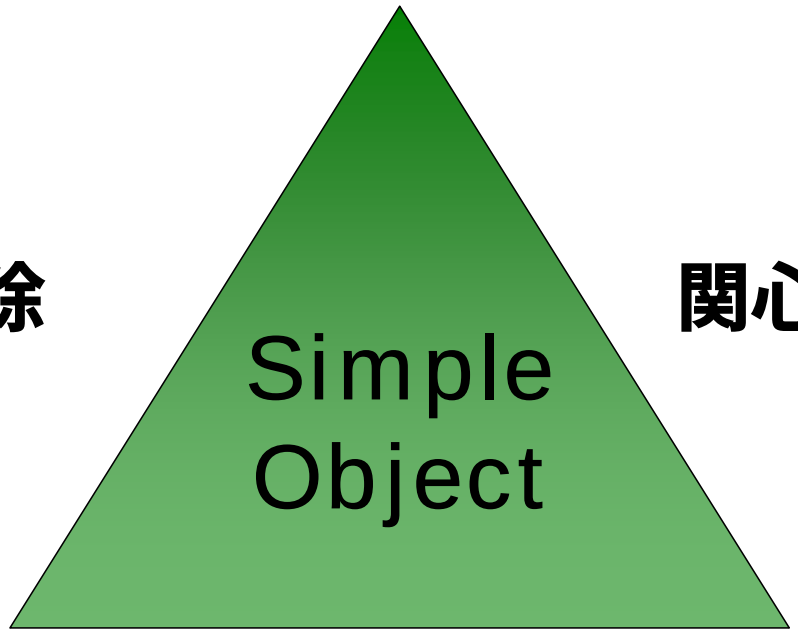
The Spring Experience 2006

- InfoQ**創立者によるトレンド予測**
  - ✓ **ドメインモデル**
    - DI, AOP…
  - ✓ **言語柔軟性**
    - **アノテーション**
    - **動的言語**
      - **特に**  Ruby *A Programmer's Best Friend*

※InfoQ**は**Java/.NET/Ruby**等の記事サイト**

Springとインタフェース21の方針

- **シンプルなオブジェクトを最重視する**



Springとインタフェース21の方針

- **シンプルなオブジェクトを実現するために**
  - ✓ **依存性の排除　⇒** DI
  - ✓ **関心事の分離　⇒** AOP
  - ✓ **環境依存部分の抽象化⇒** PSA

(Portable Service Abstractions)

Spring Remote（**プロトコルの抽象**）、
各種DB、AP**サーバの特化**クラス群、等

Spring Framework と Seasar

オブジェクトを
シンプルにする

ソフトウェア
工学的

抽象レイヤ
充実

システム
全範囲

開発者を楽にする

現場主義

開発方法論

サーバ
向け

同じDIxAOPコンテナ　しかし　方向性はちょっと異なる

それぞれ適した使い方をしよう

最後に ー Springの目指すもの

- **強力**な機能で、システムを**シンプル**にする
  - ✓ 変更に強い。ポータビリティが高い。
  - ✓ J2EEアーキテクチャに縛られない。
    オブジェクト指向設計を可能に。
  - ✓ あなたのシステムに最適な選択肢を。