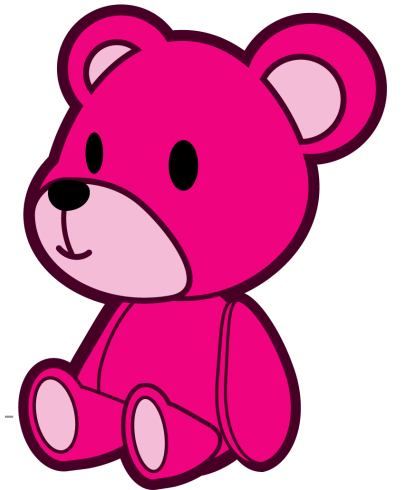


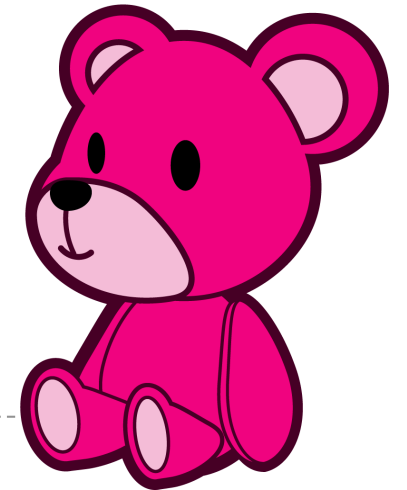
**Qcon Tokyo 2010**  
**Scala + Lift** による  
超実用開発

**Takafumi Miki @ PatentBureau.Co.,Ltd**

ご来場頂き  
ありがとうございます。

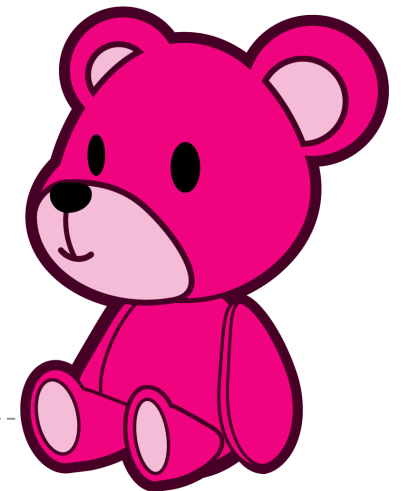


# まずは、自己紹介

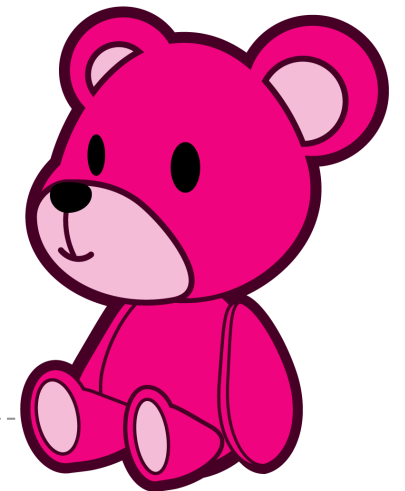


# 自己紹介

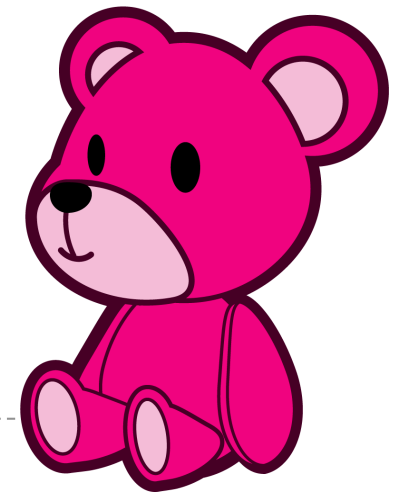
- ・ 三木 隆史
- ・ 株式会社パテントビューロ所属
- **twitter: kuma3\_neko3**
- ・ インフラ、サーバーエンジニア
- ・ データベースエンジニア
- ・ アプリケーションエンジニア
- ・ 研究・開発部門マネージャー



「何でも屋さん」だね

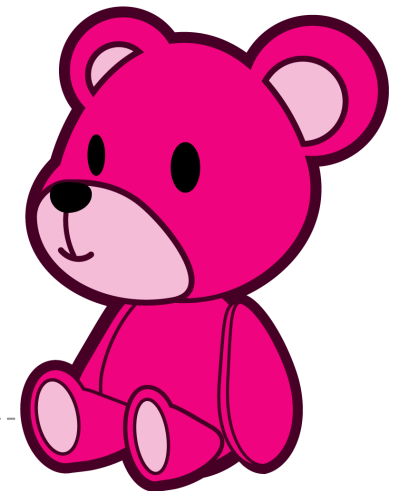


・・・という事で



PatentBureauは、  
2008年10月より

メイン言語 = Scala + Lift



世界Scala化の流れ

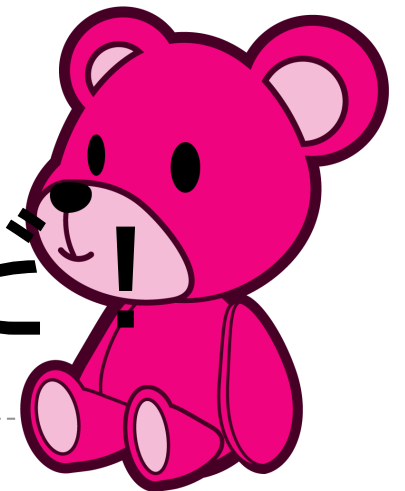
+

Scala + Liftを

仕事で使ってきた経験

=

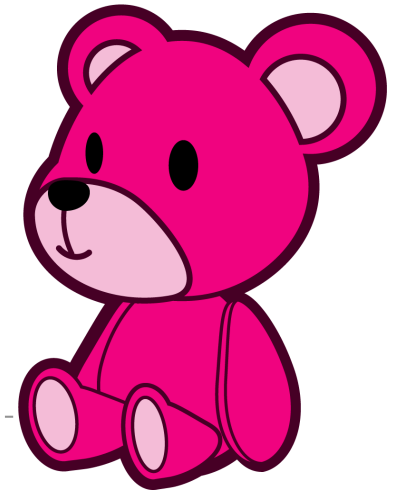
Scalaは、超実用言語だ！



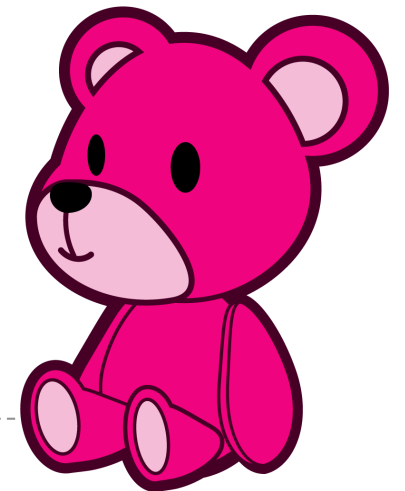
・・・と感じて欲しい



宜しくお願ひ致します。

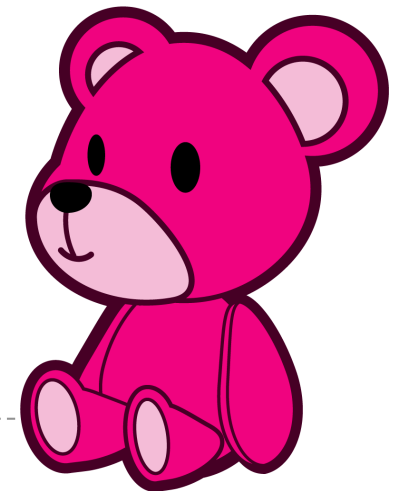


# 世界Scala化の流れ



# Scalaでの開発事例

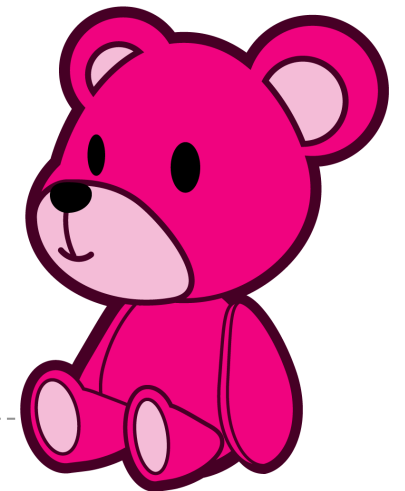
- Twitter
- Électricité de France Trading
- Xebia
- XMPie(Xerox)
- FourSquare
- Sony Pictures Imageworks
- Siemens
- GridGain
- AppJet



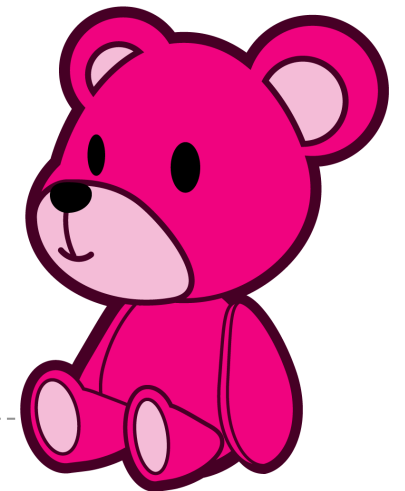
# Scalaでの開発事例

- Reaktor
- Triental
- Sygneca
- HD Holdings
- SAIC Mimesis Republic
- WattsOn

そして……Patentbureau



彼らの発言から、  
**Why Scala?** を  
解き明かしていきます



# Twitter

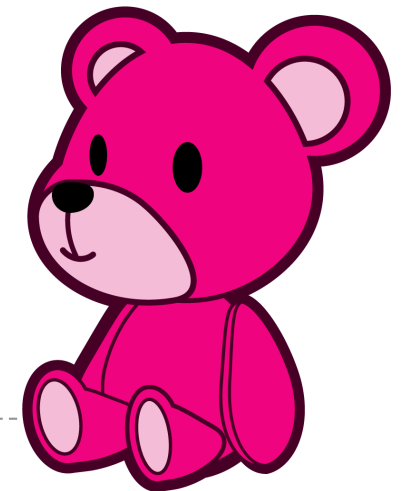
- ・ メインとなるメッセージキューの処理を  
RubyからScalaに置き換えた  
(Twitter Kestrel Project)
- ・ 最近、Scalaによる分散フレームワーク「Gizzard」を公開

速くて、関数型で、表現豊かで、  
静的型付で、かつオブジェクト指向で、  
並列実行できて、そして美しい！

パワフルな言語仕様！

並列処理に強い！

十分な実行速度！



# Électricité de France Trading

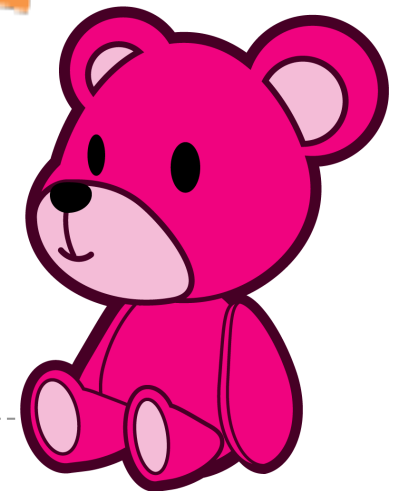
- ・ フランス最大のエネルギー関連企業
- ・ 30万行のJavaコード  
→ Scalaに置き換えた

我々は、ミスによって多額の損害がでるような、**クリティカルなビジネス領域**でScalaを使用している。

リスクはたくさんある。

だが、Scalaを使ってからは問題ないよ。

**高い堅牢性！**



# Xebia

- ・ 年間粗利益2千万ドルの  
ITコンサルタント/プロジェクト開発企業
- ・ Javaテクノロジーを活かしたアジャイル開発  
手法や、アウトソーシングサービスを提供。

多くの企業がシンプルで簡潔なソリューションを求めている。

Scalaは生産性を向上すると共に、  
既存のミドルウェアを使い続けたい企業  
に、費用対効果が高いソリューション。

シンプル！

費用対効果が高い！



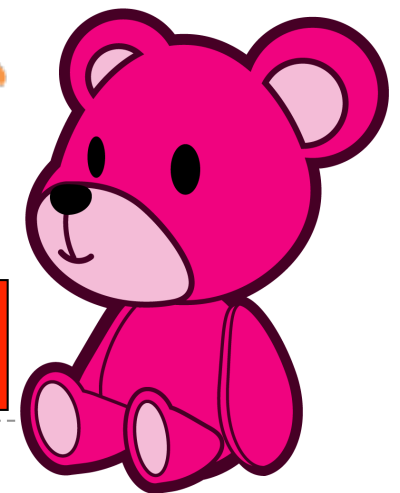
## XMPie(Xerox)

- ・ Xerox(UK)の顧客サービス改善のためのプロジェクトにScalaを使用
- ・ 受賞歴のあるPersonalEffectというソフトウェアを開発した。
- ・ 開発の中心人物にLiftのコミッターがいる

Scalaの学習コストは、それによって得られる生産性の向上を考えると**十分に元がとれる**

学習する価値がある！

高い生産性！



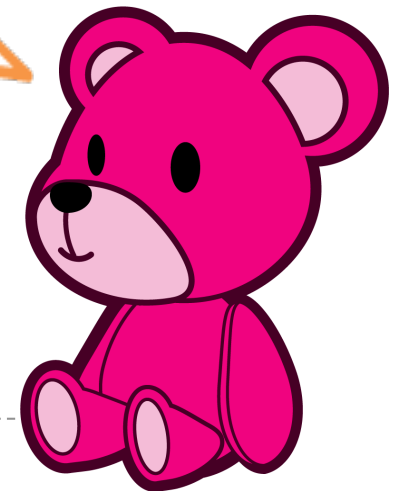
# FourSquare

- ・ ワシントン・ポストが「次のTwitter」といっている、ロケーションを活かしたSNS。
- ・ LAMP環境の稼働中アプリケーションをわずか3ヶ月でLiftに置き換えた。
- ・ Liftのscalaコードは14,000行ほど

Liftはすばらしいフレームワークだ。

これからも使い続けるだろうし、Liftが今後も成長し続けることを期待している。

魅力的で実用的なフレームワークの存在



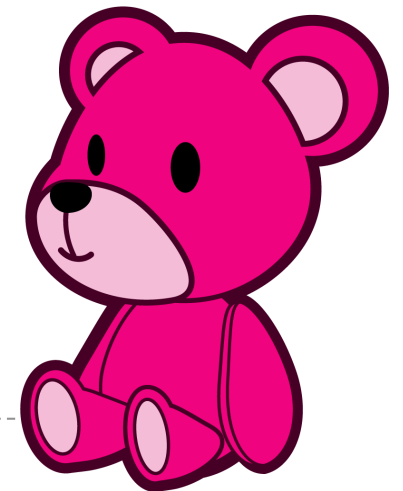
# Sony Pictures Imageworks

- ・ キャラクターアニメーションとビジュアルエフェクトのリーディングカンパニー。
- ・ 提供している5つオープンソースのうちの1つにDBのスキーマ管理ツールであるScala Migrationsがある。

オープンソースはいつでもうちの会社と共にあるんだ。

いままではオープンソースコミュニティにあまり貢献できてなかったけど、これから変えていくつもりだよ。

オープンソース！



# Siemens

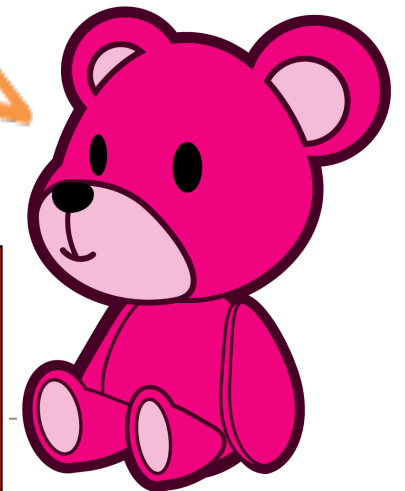
- ・ ソーシャルネットワークの力を利用して企業の生産性を向上させるサービスを提供。
- ・ Enterprise Social Messaging Experiment (ESME)というオープンソースツールをScalaとLiftで開発。

RubyとRuby on Railsも好きだけど、ScalaとLiftの方がもっと好きだね。

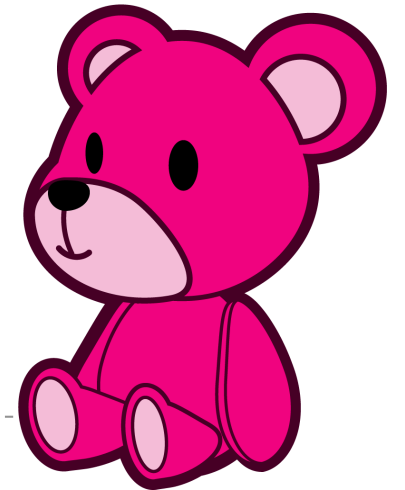
2年ほどかけて、RoRのソフトウェアをすべてLiftに置き換えたよ。後戻りするつもりはないよ。

JavaからRuby?

いや、JavaからScalaでしょ！

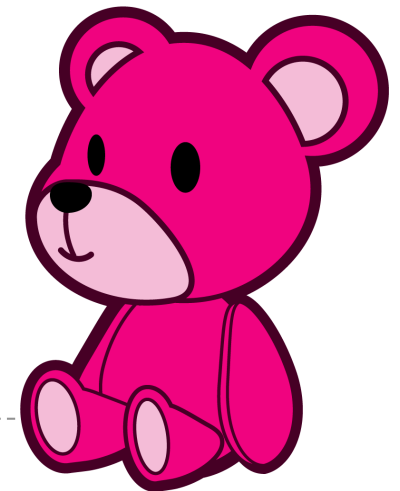


そろそろ、まとめます。



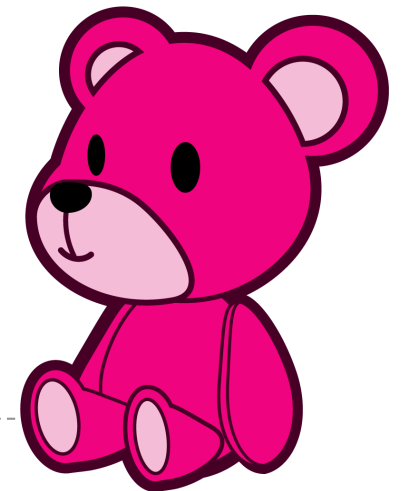
# Why Scala (& Lift)?

- ・ **パワフルな言語仕様**
  - ・ オブジェクト指向 + 関数型 の良い所取り
  - ・ Javaの「おまじない」を極限までカット
  - ・ 便利な機能を多数搭載
- ・ **並列処理との親和性**
  - ・ 並列ライブラリ標準装備
- ・ **十分な実行速度**
  - ・ コンパイルすれば、  
Javaのバイトコードに変換される



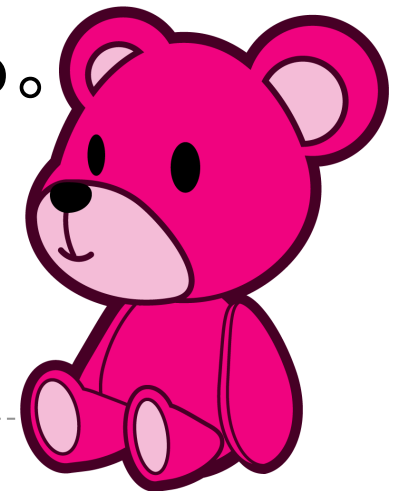
# Why Scala (& Lift)?

- ・ **高い堅牢性**
  - ・ コンパイルによる静的チェック
  - ・ Nullを極力使用しない体系
- ・ **シンプルさ**
  - ・ おまじないを書か回数が激減
  - ・ 自然に、Collectionクラスを多用出来る



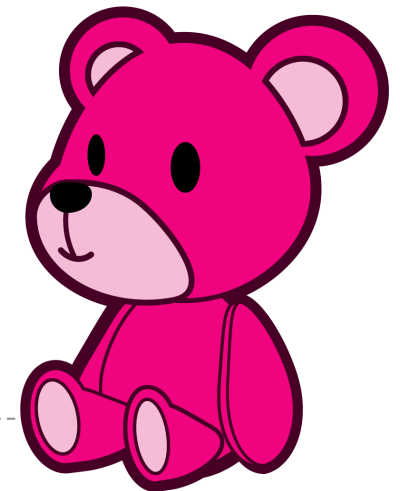
# Why Scala (& Lift)?

- ・ **費用対効果の高さ、高い生産性**
  - ・ ロジックを簡単に実現するサポート機能が充実
    - ・ 暗黙の型変換
    - ・ 型推論
    - ・ 強力なパターンマッチ
    - ・ Collectionクラスの整備
      - ・ 多くのデータを、同じ流れで処理
  - ・ ☆ 純粹にロジックを考える事に注力できる。

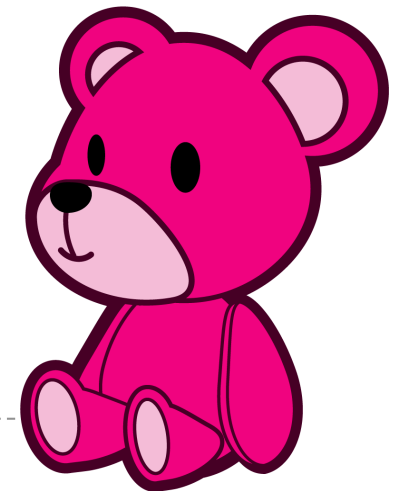


# Why Scala (& Lift)?

- ・ 魅力的で実用的なフレームワークの存在
  - ・ Liftの存在
- ・ オープンソース
- ・ Scalaを支える多くの優秀な企業
  - ・ 中国でも流行ってきている！？

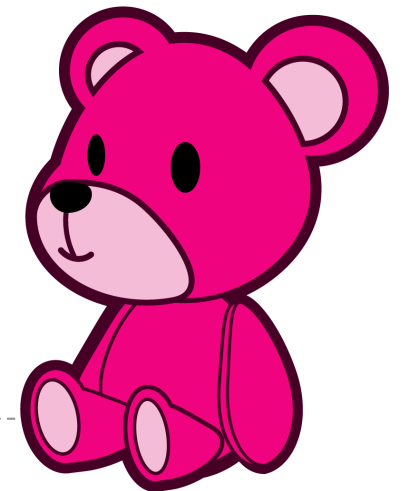


彼らが、  
こう感じる理由は？



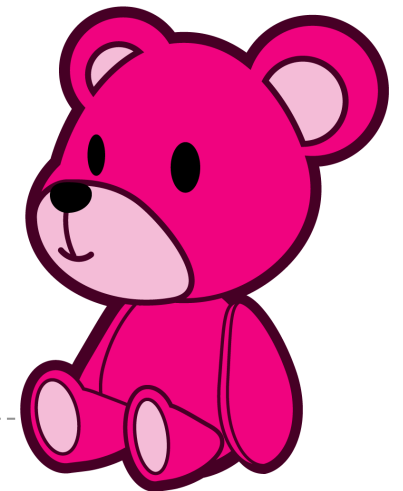
# Scalaの特徴

- Like Java
  - オープンソース
  - 強い静的型付言語
  - JVM上で動作
  - Java <-> Scala の相互呼び出し。
    - Javaの資産を流用可能



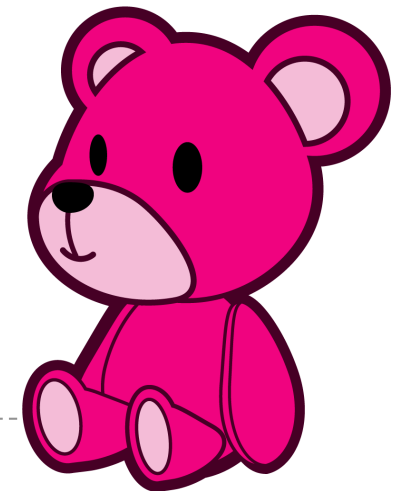
# Scalaの特徴

- **Scala オリジナル**
  - オブジェクト指向と関数型のハイブリッド
  - 型推論、暗黙の型変換
  - 強力なパターンマッチング
  - Collectionクラスの整備
  - 生産性を高めるシンタックスシュガー
  - Mix-inによる多重継承が可能
  - 並列処理ライブラリ付属



# Liftの特徴

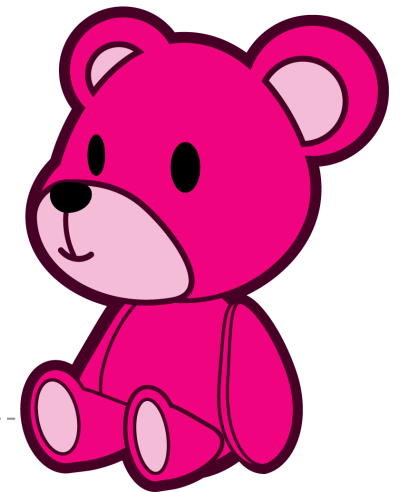
- Railsのように**簡素**で使い易い
- 型安全なのにStrutsのように**冗長**すぎない
- SeasideやWicketのように**セキュア**でステートフル
- **高速**
- 自由度が高い
- **デザイナーに優しい**テンプレートエンジン
- Ajaxサポート
- どのフレームワークよりも手厚いCometサポート



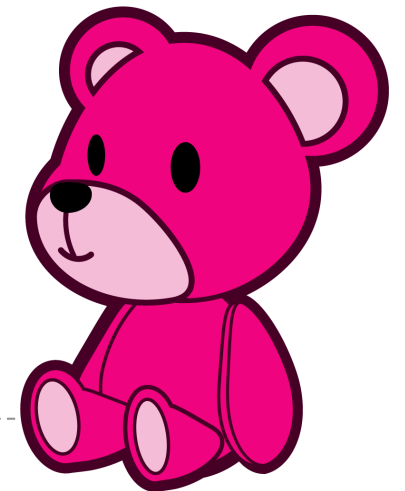
現状に満足していますか？  
Javaを使い続けますか？  
それともRubyに移行？



次は・・・



# パテントビューロが Scalaを選んだ理由



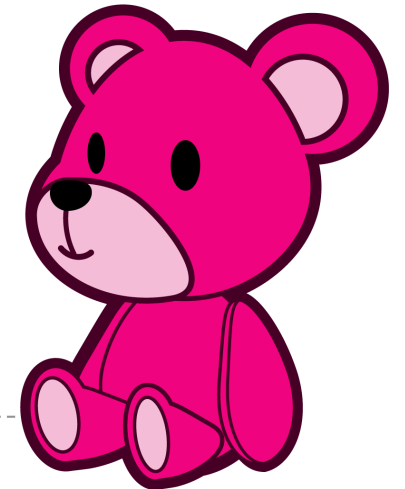
# ■ 背景

—新規サービス作成

→ <http://astamuse.com>

「俺は、これを作る為に起業したんだ！」

by 社長



Webベース

ログ解析

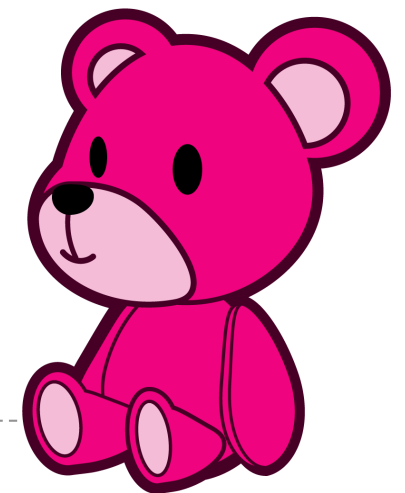
大量文書(1億以上)

データマイニング

全文検索

リコメンドエンジン


自然言語解析



検索する

 アスタミューゼHOME

産業別ランキング

 ソフトウェア・通信業

設定産業の変更

- 1  [リース料率算出システム、方法及びプログラム](#)  
中国電力株式会社
- 2  [全国合成レーダ雨量を用いた分布型流出予測シ…](#)  
財団法人河川情報センター
- 3  [RFIDタグ](#)  
三菱電機株式会社
- 4  [予測制御装置、予測制御方法、予測制御プログ…](#)  
高木産業株式会社
- 5  [ウォータディスプレイ装置](#)  
株式会社光栄
- 6  [アミロイドの凝集及び／又は沈着に起因する疾…](#)  
富士フイルムRIファーマ株式…
- 7  [円偏波平面機能アンテナ](#)  
国立大学法人佐賀大学
- 8  [デマンド制御装置、デマンド制御方法およびデ…](#)  
ダイキン工業株式会社
- 9  [エレベータの清掃運転システム](#)  
東芝エレベータ株式会社
- 10  [地中熱利用ヒートポンプサイクル装置](#)  
株式会社前川製作所

産業別新着情報

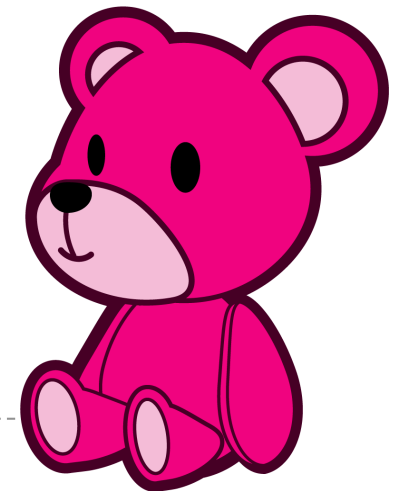
技術コンテンツの検索ワードランキング

- 1  [センスマージン](#)
- 2  [アルカリ蓄電池 特性 容量](#)
- 3  [アルカリ蓄電池 特性](#)
- 4  [鉛蓄電池 容量](#)
- 5  [陶磁器製品](#)
- 6  [アルカリ蓄電池](#)
- 7  [カメラキャリブレーション](#)
- 8  [蓄電池 容量率](#)
- 9  [4081744号](#)
- 10  [キャリブレーション](#)

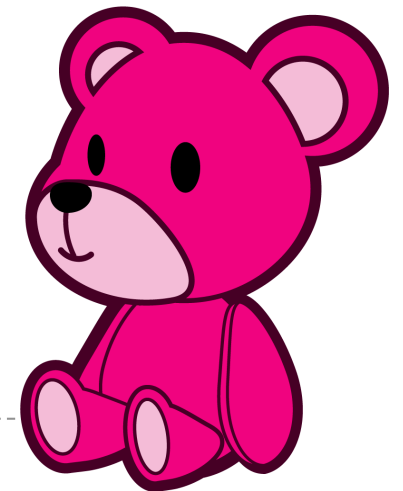
技術コンテンツのアクセス推移



- ・サーバーが何台あっても足りないなあ。。
- ・要素技術開発が多いなあ。。。
- ・バッチ処理が永遠に続く。。。
- ・Java以外に何か無いかな。。。

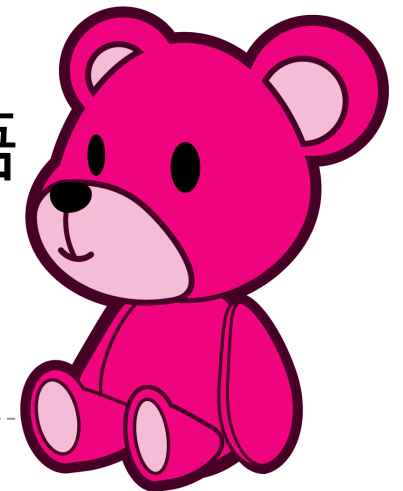


- 少数精鋭の開発
  - 生産効率重視
- 多くの要素技術開発の必要性
  - Webアプリはサクッと作って、ガリガリ研究
- 新サービス
  - リファクタリング前提
  - あらゆる未知な要求に柔軟に対応
- 莫大な量のバッチ処理
  - 高速性 + 分散・並列実行



# Patent Bureau Co., Ltd. パテントビューロで必要な技術要素

- ・ Webプログラミング
  - シンプルな記述ができる言語
  - 自由度が高いフレームワーク
- ・ 大規模データ処理
  - マルチコアと親和性が高い言語
  - スケーラブルな言語
  - 高速な言語
- ・ 自然言語解析、データマイニング
  - 高度な抽象化ができる言語
  - 簡潔な記述と、静的チェックが出来る言語
- ・ BtoB
  - 堅牢性の高い言語



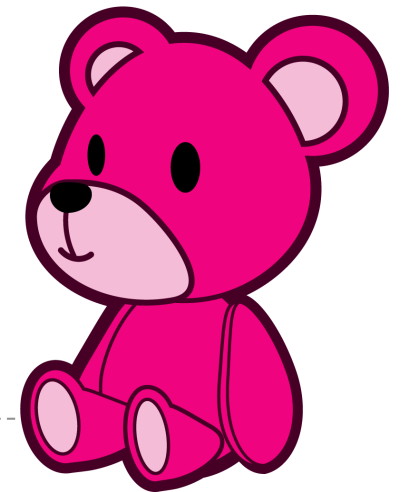
だから



scala



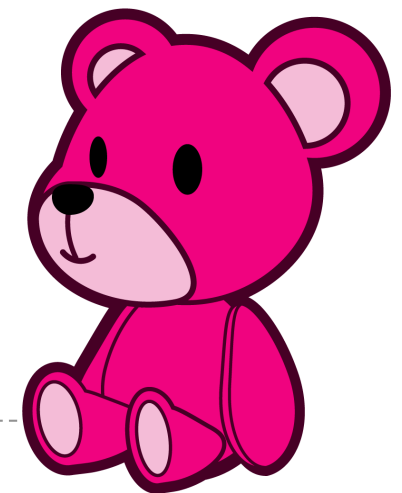
# 事例紹介



# 事例：JavaからScalaへのリプレース

- ・ 機能
  - 登録・編集・ログイン・ログアウト・SSO・WebAPI
- ・ Javaでの構成
  - Struts + Hibernate(Postgres) + JSP
  - 1ヶ月
- ・ Scalaでの構成
  - Lift + Lift-mapper(Postgres) + Lift-template
  - 制作期間2週間(Scala経験1、2ヶ月程度)

	java	.scala	jsp	.html	.xml
Java	8553	0	6689	690	1307
Scala	0	3281	0	1587	198



# 事例：転職サイト

- 概要

- 機能

- 求人票表示
      - CSVから自動インポート機能
      - 全文検索
    - 経歴等の登録機能

- 特徴

- サイト複数展開に対応した、フレームワーク機能

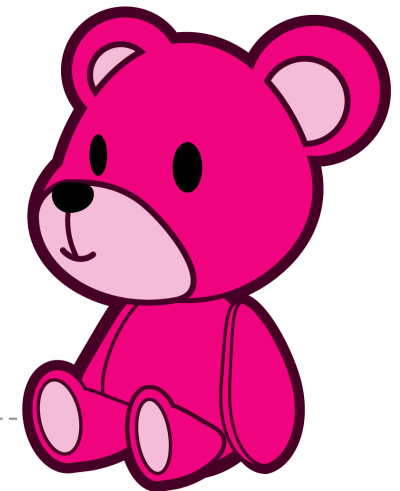
- 制作期間

- 2週間 + (サイト増加工数: 1日/1サイト)

- 構成

- Scala + Lift + Postgres

- .xml = 約500行、.html = 約8000行、.scala = 約3000行  
(htmlの共通化は、上手く行えていない)



## 半導体設計転職ナビとは？

「半導体設計転職ナビ」は、半導体設計エンジニア職(LSI設計、EDAなど)に特化した数多くの非公開求人情報と転職サポートを提供するサイトです。

※電気・電子の回路設計に特化した求人は、「[回路設計転職ナビ](#)」にてご紹介しています。

キーワード検索

検索

### ● 地域から探す

[北海道](#) [東北](#) [東京都](#) [関東\(東京都除く\)](#) [近畿・関西](#)  
[中部・東海](#) [北陸・甲信越](#) [中国](#) [四国](#) [九州・沖縄](#) [海外](#)

### ● その他の業種から探す

[ASIC設計](#) [FPGA設計](#) [PFO設計](#) [センサー設計](#) [システムLSI開発](#)  
[画像処理](#) [通信処理](#) [電池開発](#) [LED開発](#)

## 🔍 デジタルIC設計(ロジック) に該当する非公開求人一覧

### ハードウェアエンジニア兼部長 ※FPGAのロジック設計※

 転勤無し

ジョブNo.	NJB845111
ポジション名	ハードウェアエンジニア兼部長 ※FPGAのロジック設計※
勤務地	北海道
求める経験	<p>【必須条件】</p> <ul style="list-style-type: none"><li>◆ハード機器の設計経験5年以上</li><li>◆管理職経験(部長クラス)</li><li>◆プロジェクトリーダー経験</li></ul> <p>【歓迎項目】</p> <ul style="list-style-type: none"><li>◆FPGA設計経験者</li></ul>

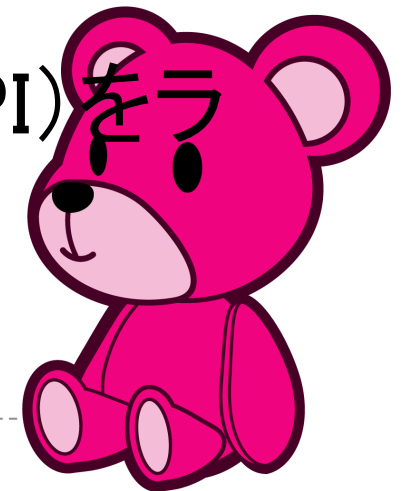
詳細を見る

, Ltd.



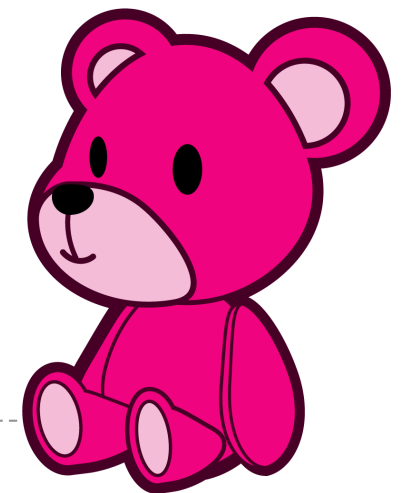
# 事例：その他

- XML解析
  - 100kb/ファイル
  - 1万2000ファイル/週
- Lucene全文検索サーバー  
Actor(並列実行ライブラリ)
  - 1Actor1ポートを受け持つ × N 個
    - » Lucene全文検索実行、レスポンスを返す
  - » Google Analytics API(アクセス解析API)をラップした、アクセス統計表示機能

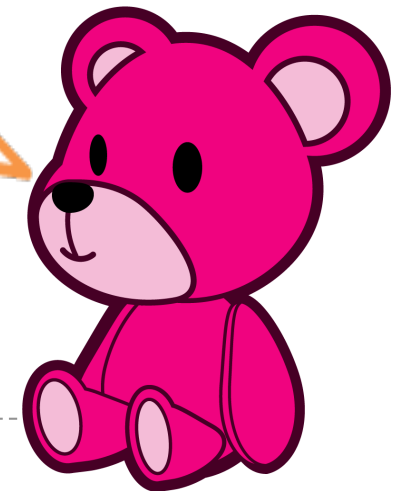


# 事例：その他

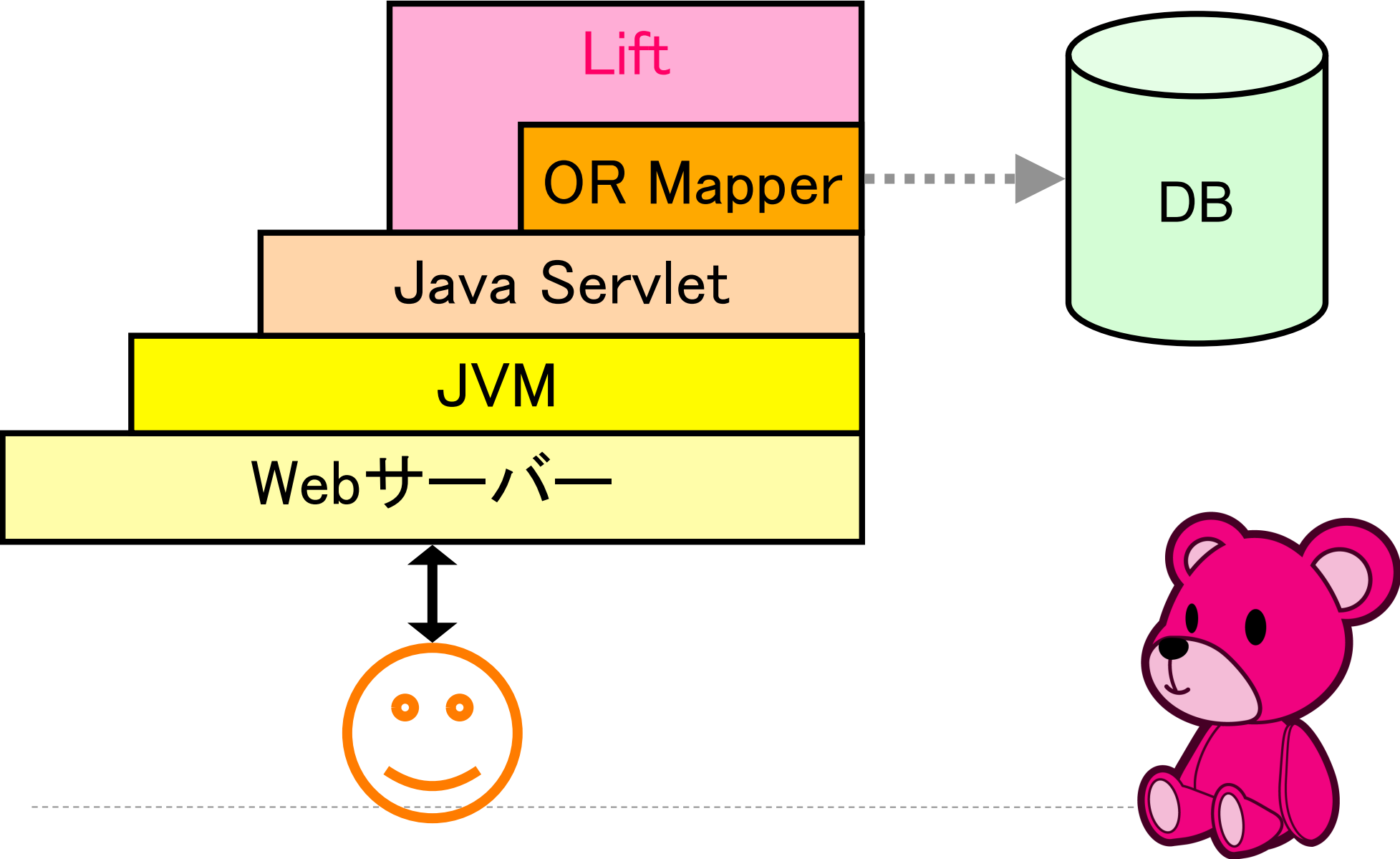
- ≫ 自然言語解析処理
  - ≫ 50G程度のテキストデータに対する各種処理
    - ≫ 特徴語抽出
    - ≫ 単語の説明抽出
    - ≫ オントロジー作成
    - ≫ などなど……
- ≫ リコメンドエンジン
- ≫ ログ解析処理



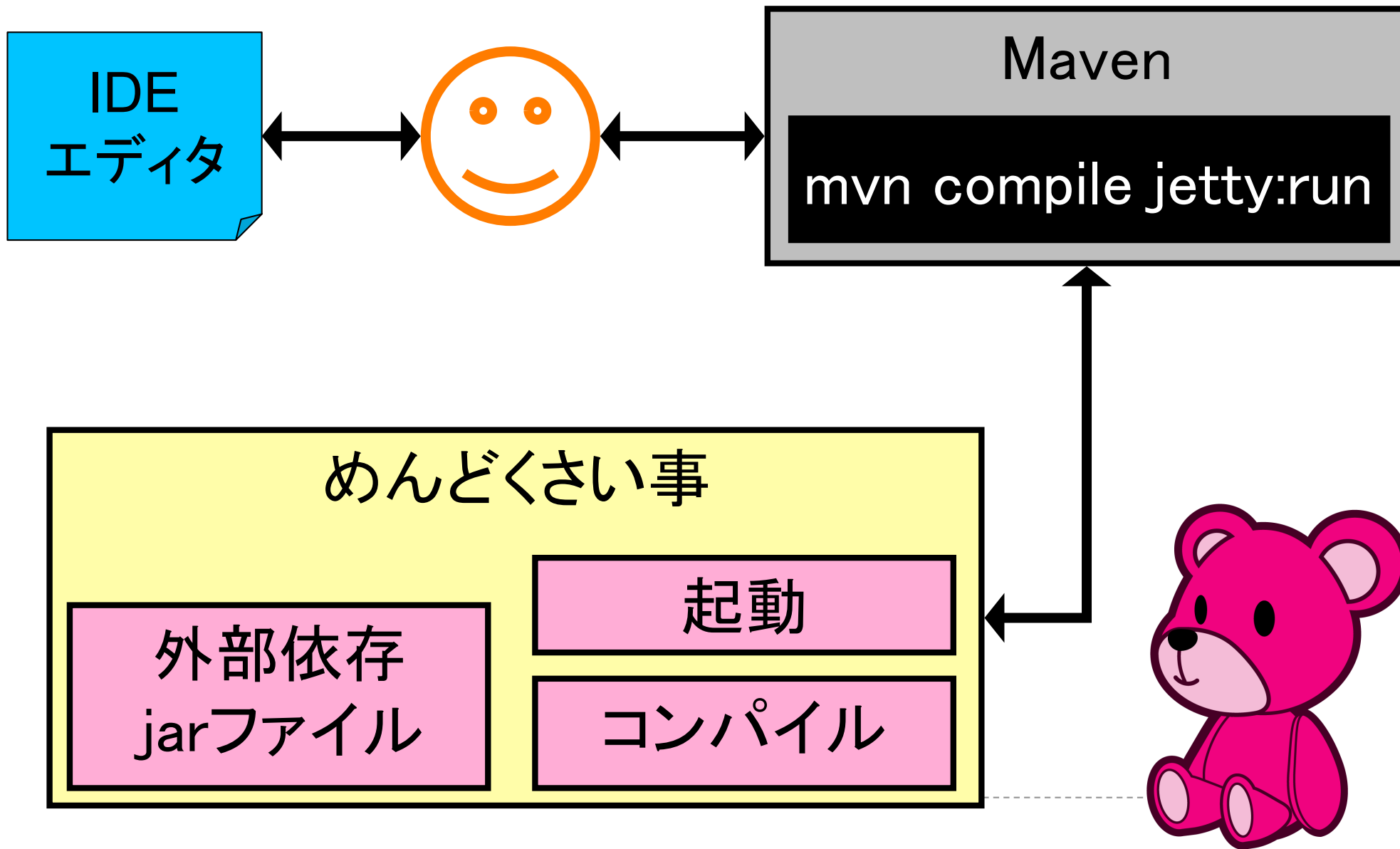
ここからは、  
Lift概要だよ。



# Lift 実行イメージ

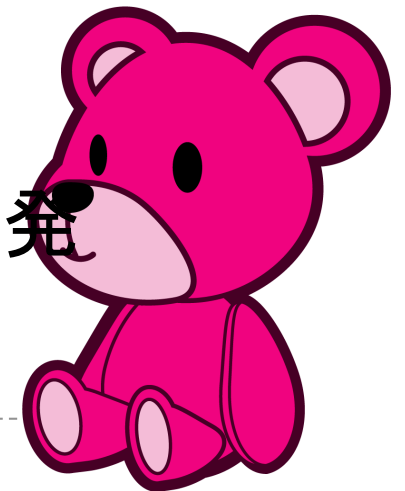


# Lift 開発イメージ

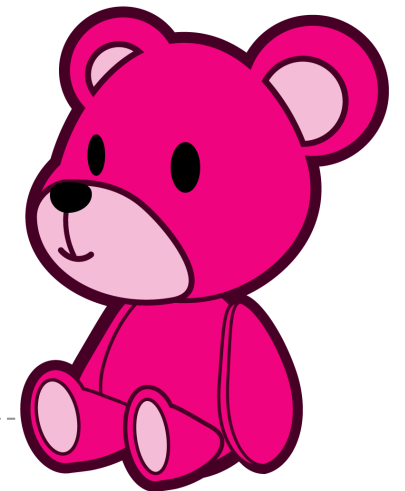


# Lift概要

- ・ 実行イメージ
  - Tomcat/jetty等のAPPサーバー
  - Lift
  - (DB)
- ・ 開発イメージ
  - Editor + Maven 又は IDEのインストール (InteliJ、eclipse、NetBeans)
  - プログラミング
  - mavenコマンドでコンパイル～起動まで一発



# Liftの便利な部分を紹介

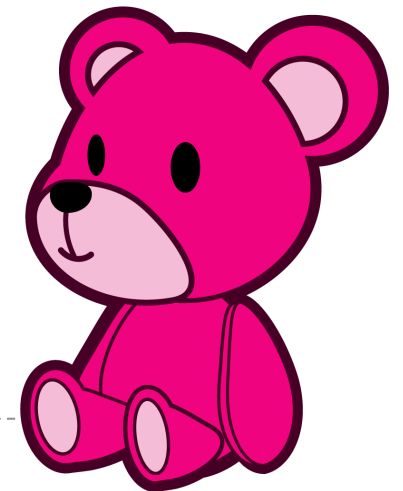


# デザイナーに優しいLiftテンプレート

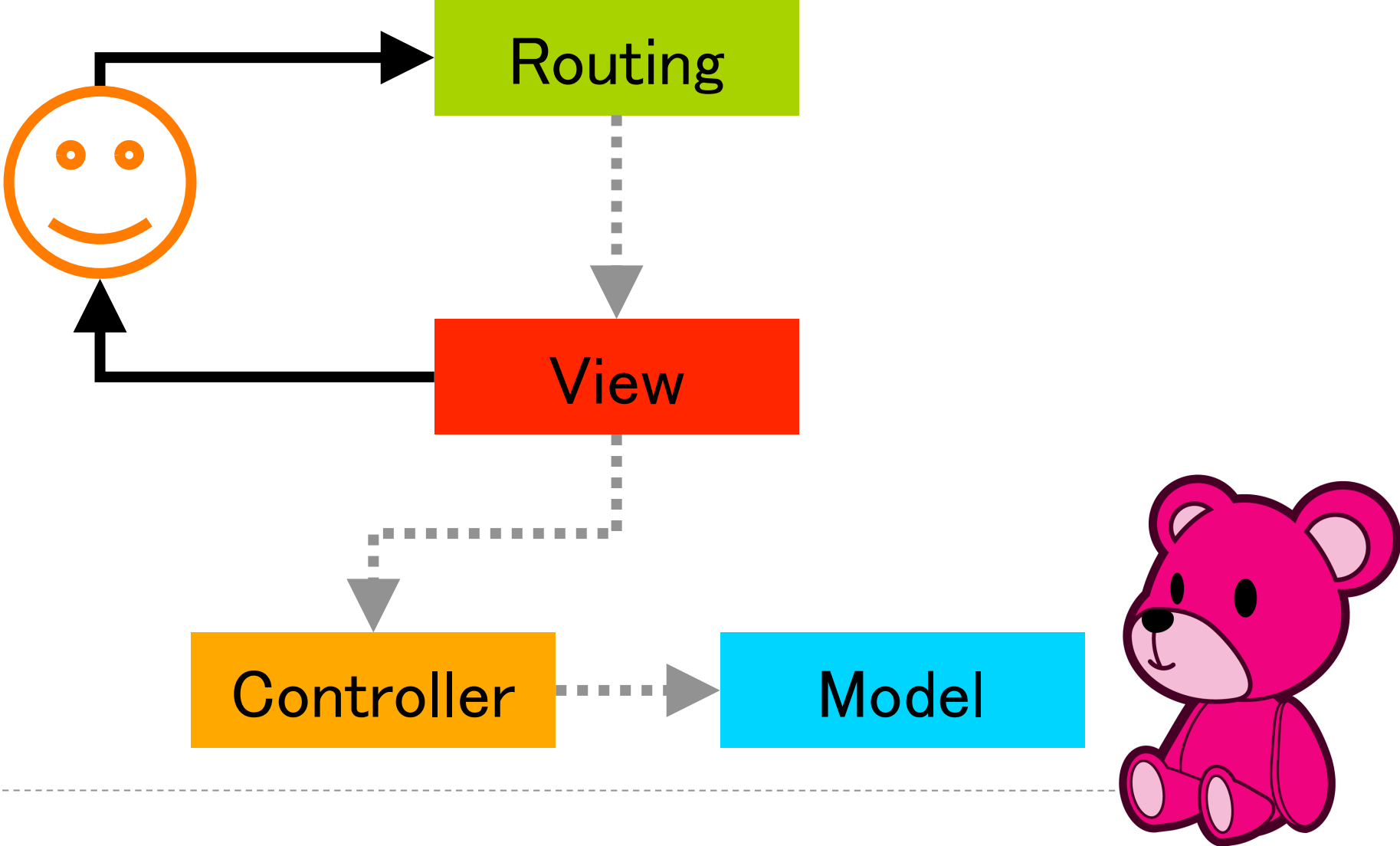
Lift の 哲学

## View First

Liftのテンプレートには  
プログラミングロジックや記号がありません  
※ちなみにRuby on RailsなどはController Firstと呼びます。

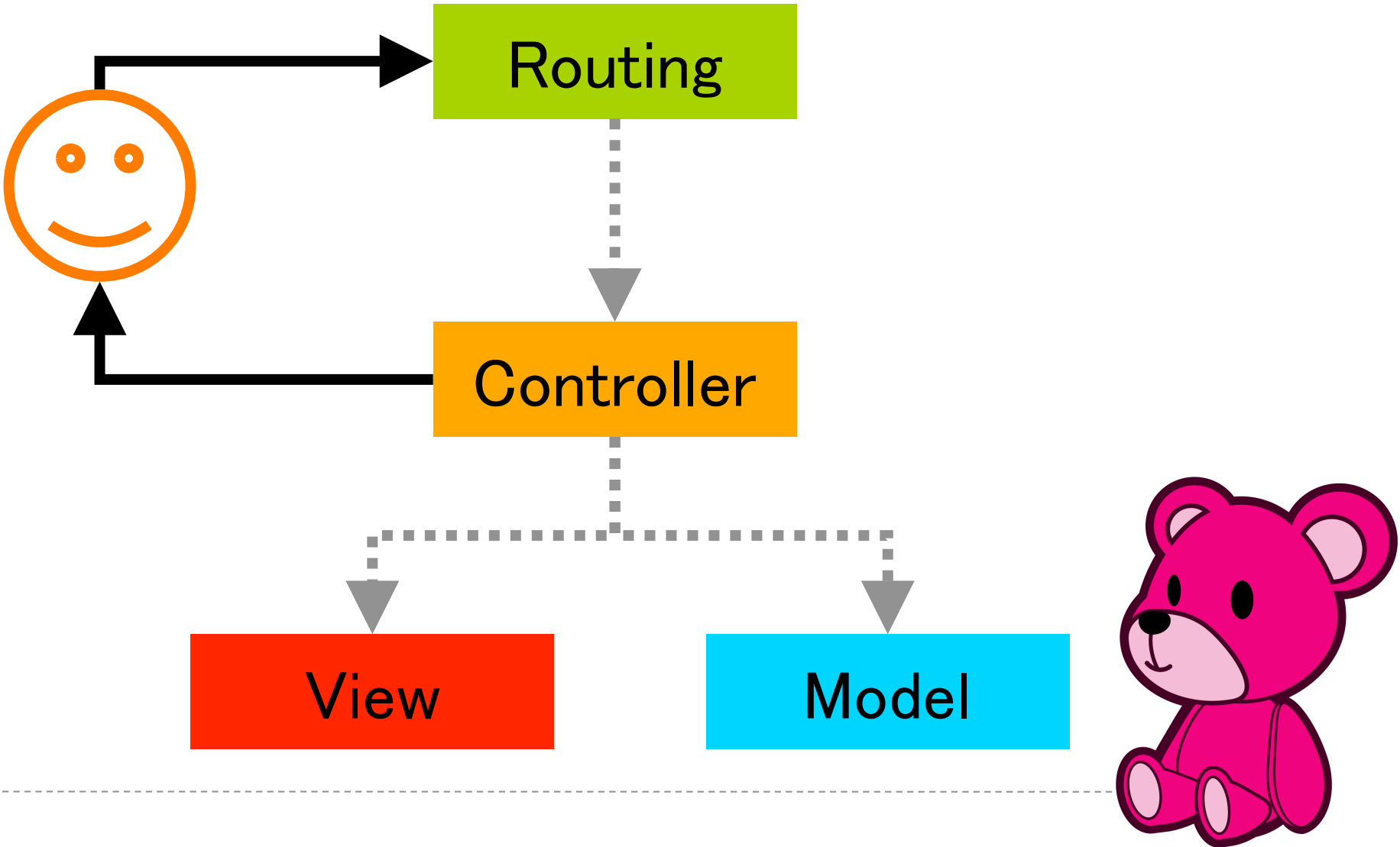


# Liftの View First のイメージ

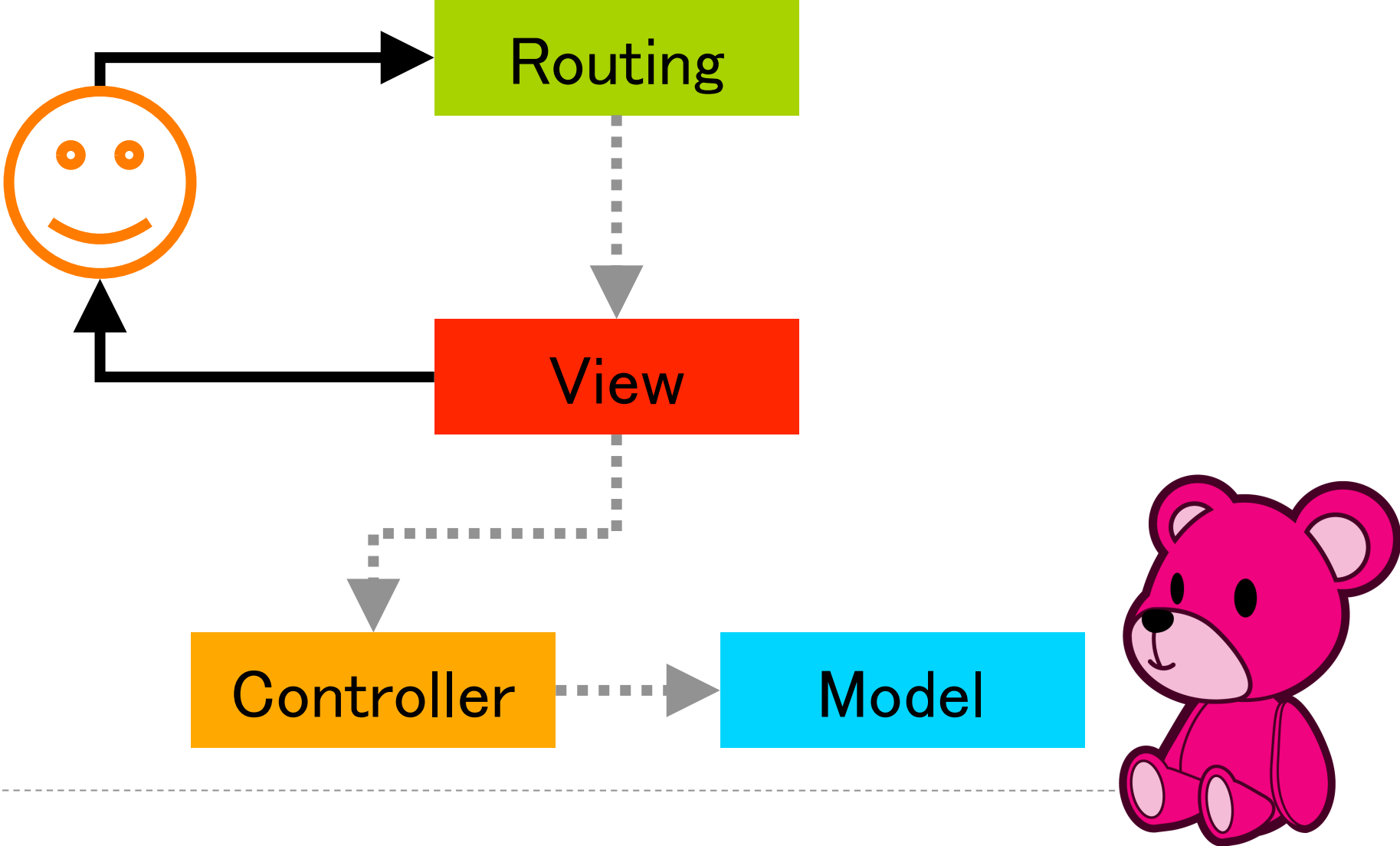


# Ruby on Rails

## Controller First のイメージ



# Liftの View First のイメージ



# テンプレートにプログラミングコードが入り込むと何故悪いの？

- 多くのビジネスロジックがテンプレートに紛れ込んでくる。
- テンプレートの間違いは、Scalaのすばらしい型システムも対応できません。
  - デザイナーに馴染みが無い「記号」がレイアウトに紛れ込んでくる。
  - デザイナーが使用するHTMLエディタが、プログラミングコードのシンタックスに対応していない編集が難しい。



# Lift-template の例

テンプレート

出力

```
<lift:userSnippet.showUser>
```

```
<p>氏:<user:lastName /></p>
```

```
<p>名:<user:firstName /></p>
```

```
</lift:userSnippet.showUser>
```

氏： 三木

名： 隆史

完全なXHTMLだよ



# Ruby on Rails のERBや JSPのようなテンプレートエンジンもありますよ

## Scalate

<http://scalate.fusesource.org/>

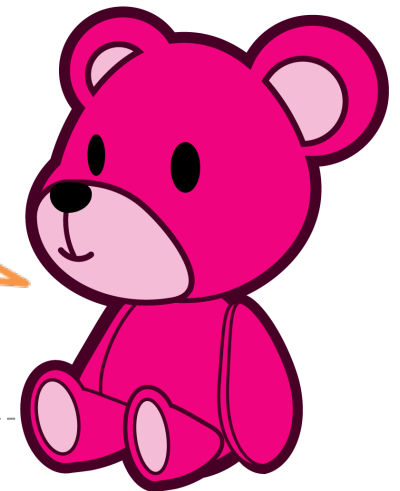
### テンプレート

```
<%@ var user: User %>  
<p>Hi ${user.name},</p>  
#for (i <- 1 to 3)  
<p>${i}</p>  
#end  
<p>See, I can count!</p>
```

### 出力

```
<p>Hi James,</p>  
<p>1</p>  
<p>2</p>  
<p>3</p>  
<p>See, I can count!</p>
```

とは言うものの、デザイナーの僕には  
解読不能・・・？



# 超実用Ajax

- ・ 様々なレスポンスに簡単対応
  - ・ Snippet（通常処理）
  - ・ Xml response（XMLに変換（WEBAPI））
  - ・ Json response（Jsonに変換（WEBAPI））
  - ・ Lift Ajax（LiftのみでAjax）

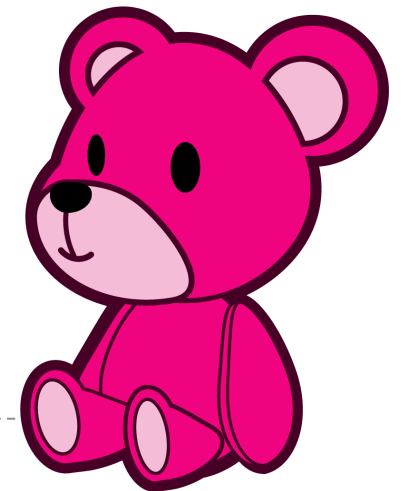
僕でも簡単に変換できるよ



# Snippet (通常の処理)

scalaコード

```
class UserSnippet {  
  def showName: NodeSeq = {  
    S.param("id").flatMap(id => User.find(id)) match  
    {  
      case Full(user) => {  
        <p>firstName:{user.firstName}</p>  
        <p>lastName:{user.lastName}</p>  
      }  
      case _ => <p>User not found!</p>  
    }  
  }  
}
```

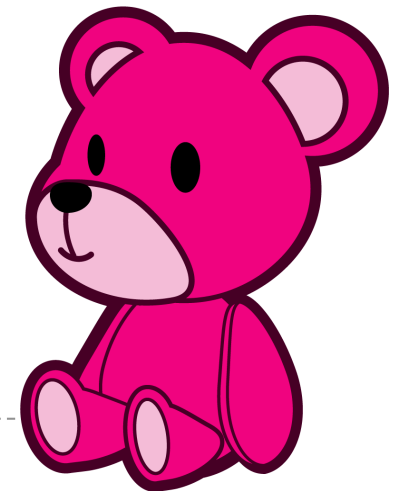


# Snippetを普通に呼び出す

HTML

```
<html>  
<body>  
  <lift:userSnippet.showName />  
</body>  
</html>
```

ブラウザ /user.html?id=y-miki  
firstName: Takafumi  
lastName: Miki



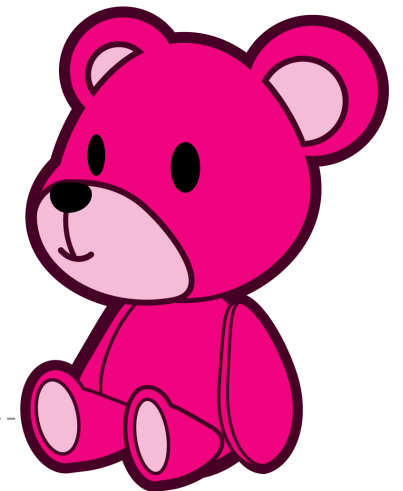
# Lift Ajax(Lift 2.0～)

## lazy loading に変換した例

HTML

```
<html>
<body>
<lift:lazy-load>
  <lift:userSnippet.showName />
  <p></p>
</lift:lazy-load>
</body>
</html>
```

スニペットも、普通のコードも  
**勝手に lazy loading**される！



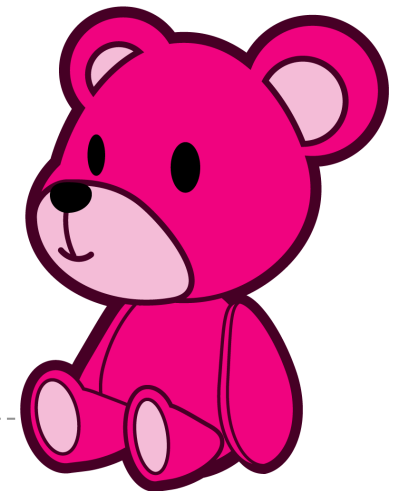
# XML Response に変換した例

scalaコード

```
LiftRules.dispatch.append {  
  case Req("user" :: id :: Nil, "xml", _) => () => {  
    User.find(id) match {  
      case Full(user) => Full(XmlResponse(user.toXML))  
      case _ => Empty  
    }  
  }  
}
```

ブラウザ /user/t-miki.xml

```
<user>  
  <firstName>takafumi</firstName>  
  <lastName>miki</lastName>  
</user>
```



# Json response (Lift 2.0~) に変換した例

scalaコード

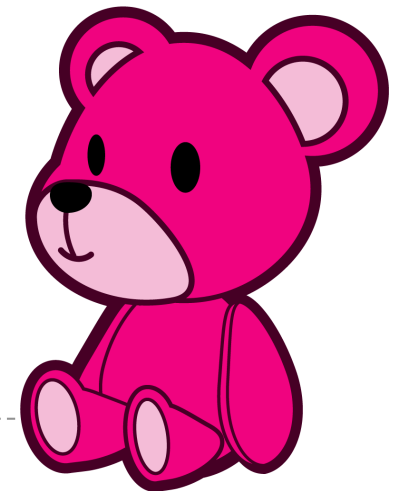
```
import net.liftweb.json.JsonAST
import net.liftweb.json.JsonDSL._
```

```
LiftRules.dispatch.append {
  case Req("user" :: id :: Nil, "json", _) => () => {
    User.find(id) match {
      case Full(user) => {
        val json = ("user" -> ("firstName" -> user.firstName) ~ ("lastName" -
        > user.lastName))
        Full(JsonResponse(JsonAST.render(json)))
      }
      case _ => Empty
    }
  }
}
```

ブラウザ /user/y-miki.json

```
{"user":{"firstName":"Takafumi","lastName":"Miki"}}
```

---



# Lift Ajax(2)

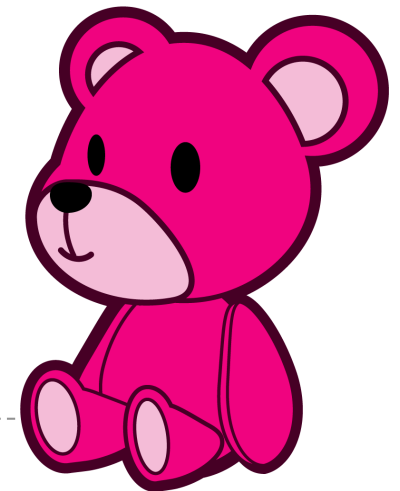
## ユーザ登録フォーム

Scalaコード

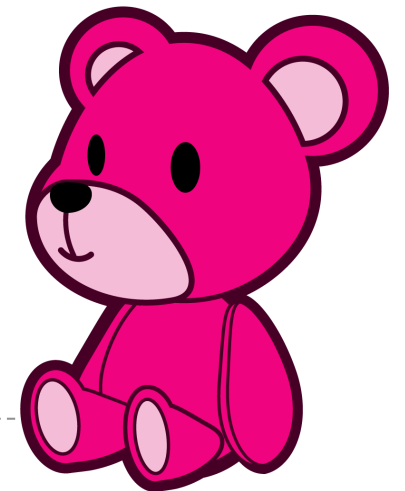
```
class UserSnippet {  
  lazy val user: User = User.create  
  
  def showForm(xhtml: NodeSeq) = {  
    ajaxForm(  
      bind("entry", xhtml,  
        "firstName" -> text(user.firstName, user.firstName(_)),  
        "lastName" -> text(user.lastName, user.lastName(_)),  
        "submit" -> submit("送信", () => user.save)  
      ) ++ hidden(() => {user.save} )  
    )  
  }  
}
```

html

```
<lift:userSnippet.showForm>  
  <entry:firstName /><entry:lastName />  
</lift:userSnippet.showForm>
```

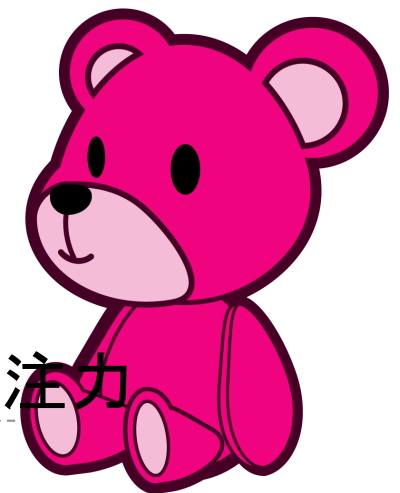


# Scalaを使ってみた感想



## – Scala単体

- ・ 弊社では、バッチ処理メイン
- ・ C、C++言語と比較して
  - ・ VM上で動作する安心感。
  - ・ こちら辺は、C、C++ 対 Java の感覚と同じ
- ・ Javaと比較して
  - 整備されたCollectionクラスにより、データの違いをあまり意識せずに、いつも、同じ感覚でプログラミング可能
  - ちょっとした変更が容易
  - 「おまじない」を極力回避出来る事により、プログラミングの本質、ロジカルシンキングに注力



- ・ Scala + Lift

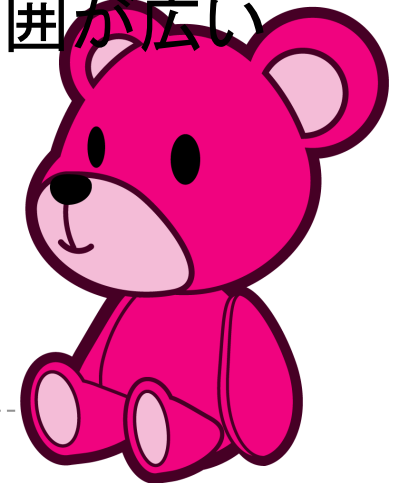
- テンプレート機能が強力

- 簡潔な構造の為、理解しやすい

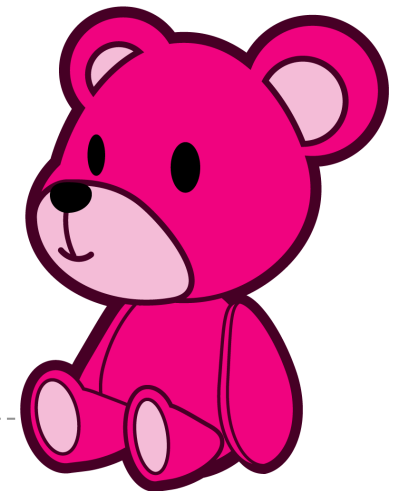
- 他フレームワークでのweb系開発と比較すると？

- ・ 毎回全く違う機能を作成しているので、フレームワーク間に差は無いのでは？

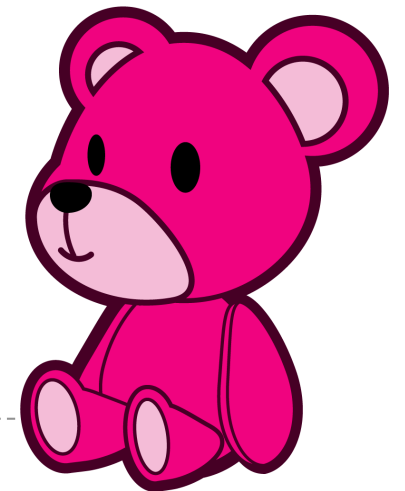
- ・ フレームワークの制限が少ないので、対応範囲が広い



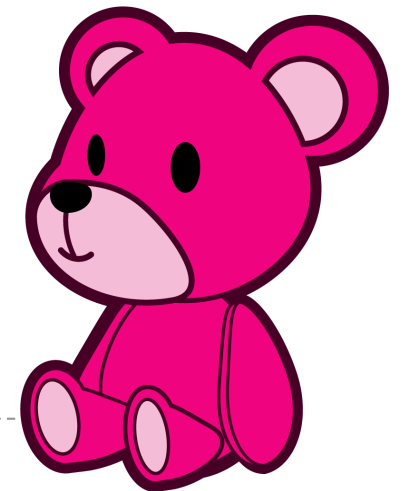
- ・ Scalaや、Liftのメリットは、十分伝わったと思います！？
- ・ 安心してScala・Liftを使っ  
て下さい！



つと…言っても…  
色々不安ですよね？



不安を取り除けるだけ、  
取り除いていきます。



# バグは無いの？

- ・ Scala本体のバグに当たった事はありません。
  - ちょっと凝った(変な)事をしようとした時に、Scalaの言語仕様の壁に当たる事があります……
    - ・ Case class の引数が30個位まで……とか
      - クラス宣言を、普通のclassにすればOK
- ・ Liftのバグ
  - 本体のバグに当たった事は？？？
    - ・ 細かい不具合には当たります。
      - Appサーバーを停止しようとしても、停止してくれない……
        - » Kill -9 jetty
        - » ※最近はずっとちゃんと停止してくれます
      - 活発に開発されているので、バージョンUPについて行かないといけない……
    - Mapperが未完成(しかも☆再実装中☆)
      - ・ 単純にテーブルとモデルを1対1で対応させる分には問題無し
      - ・ Hibernateっぽく使いたい人は、Lift2.0を待ちましょう。
        - でも。。。実際の所、SQLを覚えて書く方が良いと思います。。。

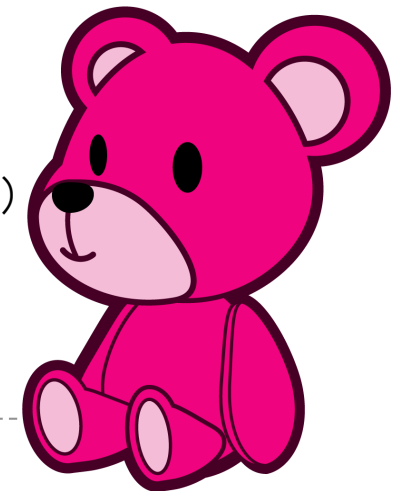


# IDEの対応状況はどうなの？

## IntelliJ Community Edition の場合

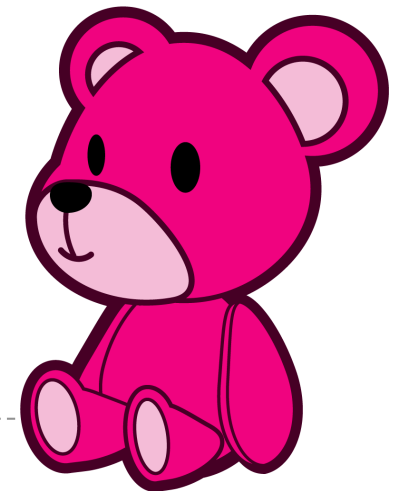
### ・ 無料

- ・ シンタックスハイライト
  - 当然できる。配色も決められる。
- ・ コード補完
  - importされているパッケージ内はコード補完が効く
  - importされていない場合も存在するクラス名を書けばimportするかどうか聞いてくる
  - 変数の補完はほぼ完璧にできる。
- ・ ビルド
  - Mavenとの親和性も高いので全く問題なし
  - JRevel Plugin を使えばさらに快適に
- ・ リファクタリング
  - かなり完成度は高い。
- ・ 安定性
  - たまに再起動しないとだめになる…。(Scalaが原因かどうかは分からない)



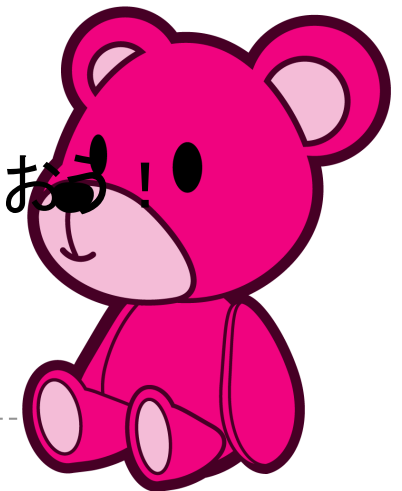
# メモリ使用量

- ・ メモリ使用量が多いと聞いたけど？
  - 確かに、javaで動作させるよりは必要です
    - オブジェクト生成数多め
  - オススメ解決法
    - ・ <http://kakaku.com/pc/pc-memory/>
    - ・ メモリ増設(価格コム最安 4G 8000円！?)
    - ・ 社長！お願いします！



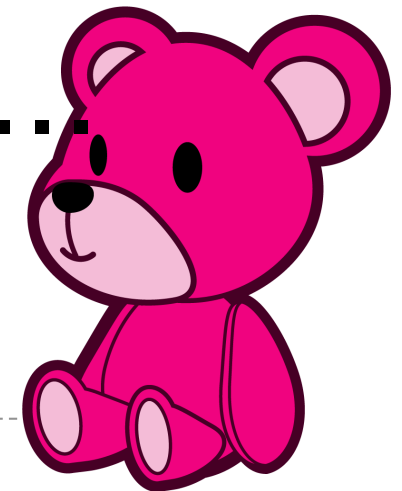
# ビルド時間

- 確かに、ビルド時間は長い
  - ・ 型推論の部分で、色々長くなっていきます
  - ・ (ビルド時間参考値) WinXP + Core2 + メモリ4G
    - ・ 3000行:1分
    - ・ 13000行:5分
- オススメ解決法
  - ・ 機能単位でjarファイル化してMavenで簡単管理
  - ・ Web系の場合、JRebelのホットデプロイ機能を使おう!
    - (今の所、Scalaは無料)



# 教育コスト

- ・ ☆Scalaは、**すべての機能を理解しなくても実務で使い始められる。**
- ・ 機能実装出来るようになるまで
  - 文法理解
    - ・ Scala本があります(コップ本)
    - ・ 1週間~2週間
  - 色々、APIの使い方に悩む
    - ・ 1ヶ月
  - ☆手続き型っぽく使えるようになる
  - ここから、関数型としてのScalaの始まり.....
    - ・ 関数型っぽく使えるようになるまで
      - 勉強中なので分かりません(汗)



# 教育コスト

- ・ **複雑なライブラリを設計出来るまで**
  - 全く未知です。
  - 設計が言語により縛られるという事が、ほぼ無い。
    - ・ どんな書き方でも出来てしまうので、ベストプラクティスを模索中。
- ・ **他人が書いたコードをデバッグ出来るまで**
  - 手続き型的、オブジェクト指向的なソースなら・・・
    - さほど難しく無い
  - 関数型全開のソースは・・・
    - 関数型に慣れるまでは大変



## Liftと一緒に歩むScala中級者への道

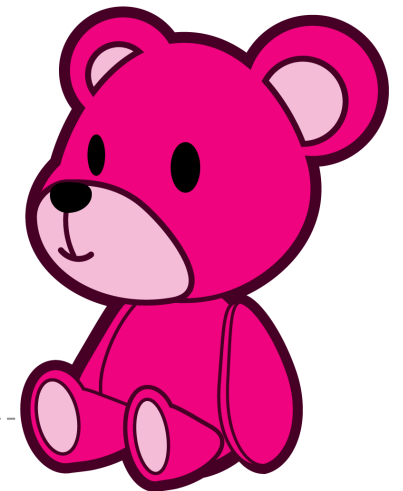
- Scalaは難しい！？と思っていませんか？
  - “関数型”のイメージが強い？
  - 関数型を使うかどうかは、皆さん次第です
- オブジェクト指向のプログラミング言語をやっていた人は、オブジェクト指向言語としてのScalaを使ってアプリケーションを書けば良いのです。
  - 当然、Liftでも同様です。



## Liftと一緒に歩むScala中級者への道

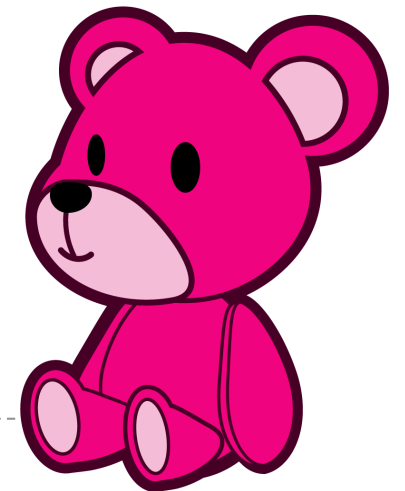
- Liftの開発をしながら、Scala初心者から中級者へスキルアップしていく過程をみていきます。

Scalaの知識がないと分からない部分があると思いますが、イメージは分かると思います。



## Scala Level 1 – Scala を Better Java として使う

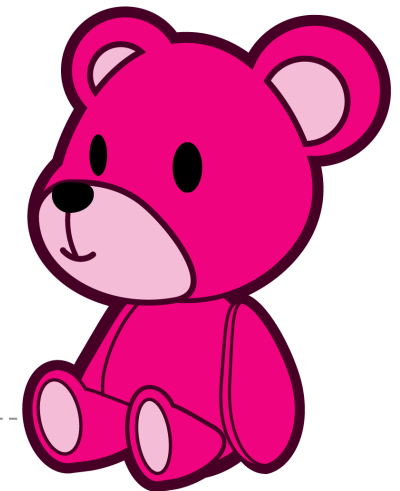
- 型宣言があまり必要のないJavaという感覚で使えるレベル
- Immutableなコレクションに戸惑うレベル
- Liftが提供するAPIを理解するには、まだ知識が足りず、チュートリアルを見ながら見よう見まねで作成できるレベル
- Javaでやるこれは、Scalaだとどれなんだー！と悩むレベル
- printlnって組み込み関数ですか？と思うレベル



# Scala Level 1 - Scala を Better Javaとして使う

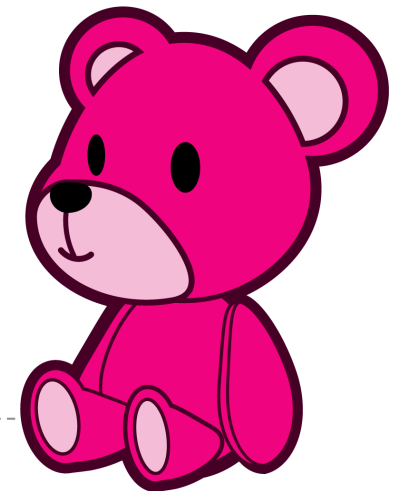
## 必要な知識

- ・ class、object、def、import、var、val などの宣言について基本的なことを理解する
- ・ 型の記述が必要な場所とそうでない場所を理解する
- ・ Javaのプリミティブ型は、scalaでは？
  - プリミティブ型に対応するリッチラッパーのメソッドを抑える
- ・ コレクションクラスについて簡単に理解する
  - Mutable と Immutable
  - ListBufferなどmutableなリストを生成するクラスについて知っておくと楽
- ・ 制御構文について理解する
  - if、while、for、try、catch
  - 2.7.xまではbreakが使えなかったので注意が必要だが、2.8.xからは breakable 関数の使い方を覚えれば大丈夫。
- ・ コレクションをループするプログラムが書けるようになる



## Scala Level 2 - Java + $\alpha$

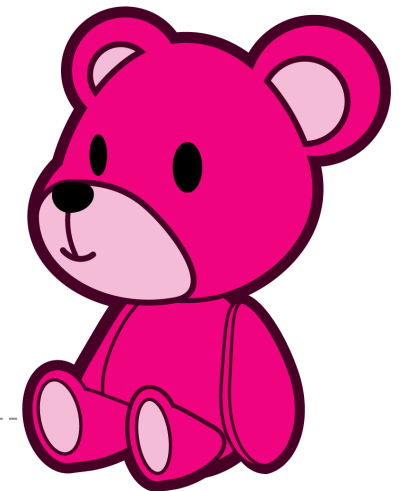
- Java +  $\alpha$  な知識を覚えていくレベル
- 覚えなければならないそれぞれのトピックについてなんとなく分かるが、利点や使いどころが分からないレベル
- まだimmutableよりmutableなコレクションクラスの方が使いやすレベル
- Optionとか邪魔なんだけど……。null返せよと思うレベル。
- PairとかTupleの v.\_1、v.\_2、v.\_3 とかもうカオス状態でしょと思うレベル
- パターンマッチングが使えるところでもif文を使うレベル。
- LiftのAPIを使ってはいるが、意図が読めず無駄に長いコードを書いてしまうレベル
- scalaを使っていて一番辛い時期かも……。なレベル



## scala level 2 – Java + $\alpha$

### 必要な知識

- ・ パターンマッチング
- ・ XMLリテラル
- ・ Trait
- ・ 暗黙の型変換
- ・ Pair、Tuple
- ・ Case class
- ・ Predef.scalaの中身を見ておく
- ・ lazy val
- ・ Option(Box)
- ・ ファーストクラスオブジェクトとしての関数
- ・ PartialFunction
- ・ コレクションクラスのある程度のメソッドの知識
- ・ 99 Scala problems の「Working with List」をこなす



## Scala Level 3 - Javaにはもどれない・・・かな？

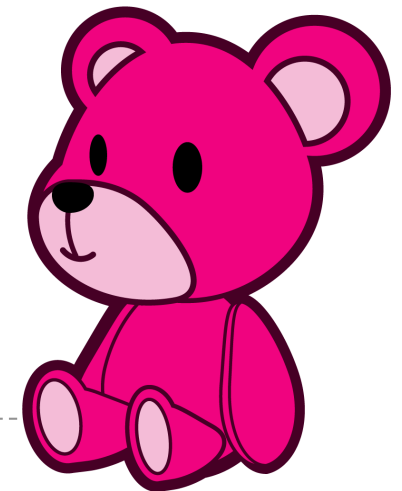
- 覚えた知識を有効な形で使えるようになってくるレベル
- Immutableなコーディングの仕方のコツが分かってくるレベル
- 再帰関数をよく書くようになるレベル
- OptionやBoxを返すメソッドを普通に定義しているレベル
- mapやfilterをつなげてワンライナーで書きすぎて、他の人から読みづらいと言われるレベル。
- LiftのAPIの意図が分かってきて、以前よりましなコードが書けるようになるレベル。
- モナド、高階関数、カーリー化、関数の部分適用などについて調べてみるが、どういうときに使えばいいのかよくわからずに悩むレベル。



## Scala Level 3 - Javaにはもどれない・・・かな？

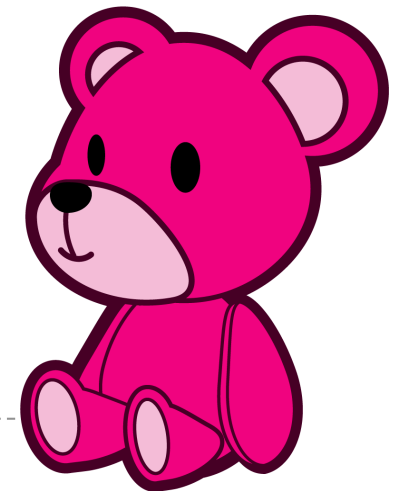
### 必要な知識レベル

- ・ Immutableであることの利点を理解する
- ・ varをなるべく使わずにプログラミングしてみる
- ・ ループ処理使わず、Listのメソッドや再帰関数を使ってコードを書く。
- ・ コレクションクラスが持つメソッドの使い方がほぼ分かっている
- ・ If文ではなくパターンマッチングを多用する
- ・ 引数に関数をとるメソッドを日常的に使う
- ・ 単項演算子、右結合関数、apply、update、などの特殊な関数の定義について理解する。



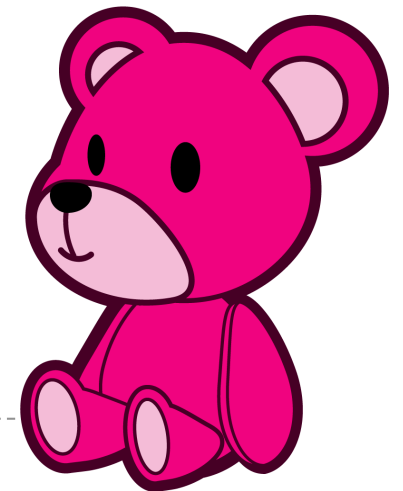
## Scala Level 4 - まだJava使ってるの？

- Javaにはもう戻れないレベル
- Scalaっぽくてかつきれいなコードが書けるようになるレベル
- Scala初心者のコードレビューがちょっとできるレベル
- LiftのAPIがどのように実装されているかだいたい理解できるレベル
- Liftで普通にアプリケーションが作れるレベル



## Scala Level 4 - まだJava使ってるの？ 必要な知識レベル

- for文を使ってワンライナーなコードを読みやすくする
- シンタックスシュガーが使える場所を知り、関数型のパワーを利用しながらも、可読性高いコードを目指す
- 覚えた知識を整理する(人に教えられるようにする)
- ScalaやLiftのソースコードを読んで、覚えた知識がどのように実装されているのかを確認していく



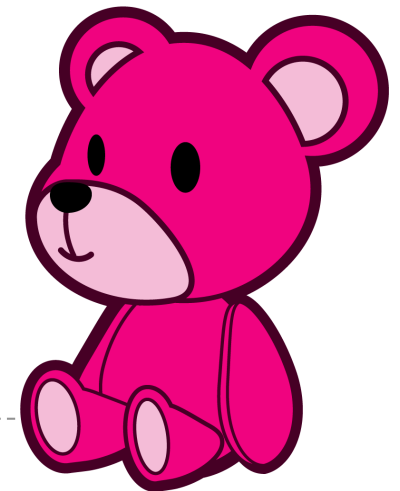
## Scala level 5 - 中級者から上級者へ

- モナド
- 高階関数
- カリー化
- 限定継続
- 関数の部分適用
- Actor
- Scalaの型システム

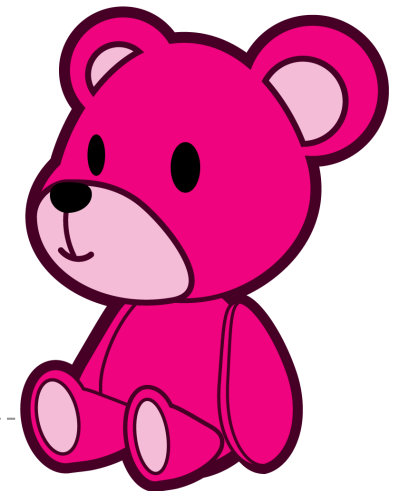
などについて理解を深める。

逆に言うと、Liftを使って一般的なアプリケーションを作るのに上記について深い理解はあまり必要ない。もちろん知っていて損はないけれど。。

これが必要な人は、ライブラリやフレームワークを作る人達。これらの機能を使いこなせれば、Java以上の物が作れる

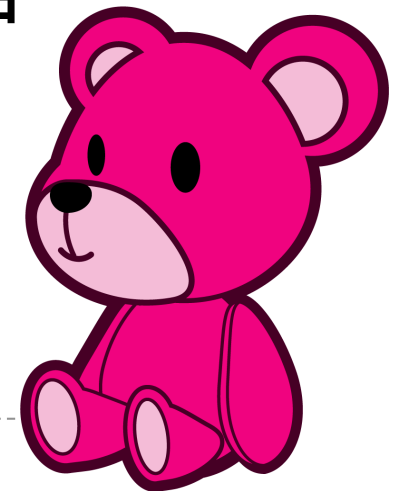


弊社メンバーに  
ヒアリングしてみました。



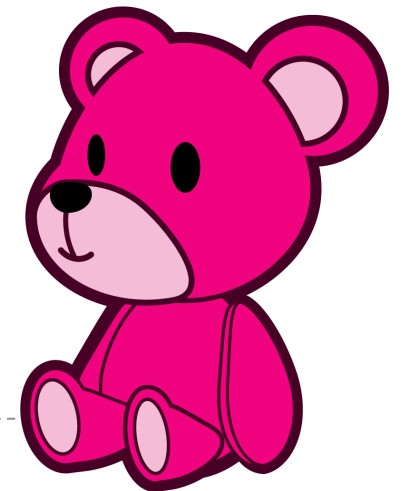
# 中国代表 王凱(1)

- ・ Scala歴: 9ヶ月
- ・ 業務でScalaを使いこなせるようになるまでにかかった期間: 6ヶ月
- ・ 他の言語歴
  - ・ JAVA 2.5年
- ・ ScalaとJava(既存アプリ。Struts)を担当



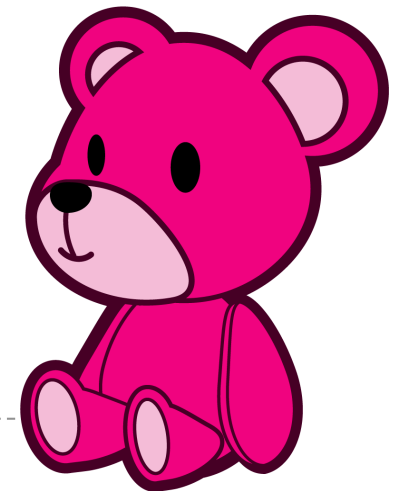
## 中国代表 王凱(2)

- ・ その他言いたいこと
  - ・ Scalaはまだ若い言語ですが、現在中国でも流行っていて、ScalaとLiftを使っている人がどんどん多くなっていくと思う。
  - ・ 中国でも、Scala + Liftの開発事例が出ている
  - ・ 中国で、Scalaの求人が出ていたのを見た事がある



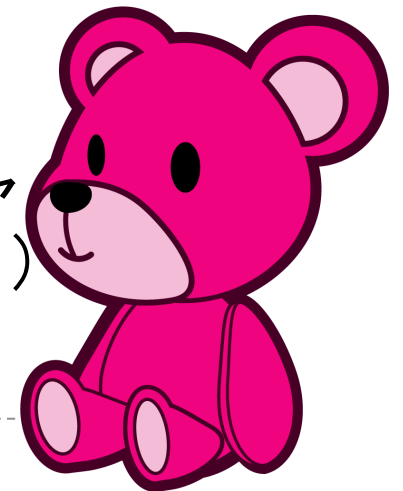
# 千葉県代表 石井(1)

- ・ Scala歴7ヶ月
- ・ Scalaを使いこなせるまでかかった期間
  - ・ まだまだかな。。
- ・ 他の言語歴
  - PHP 4年
  - Java 3年
  - Ruby 2年
  - ActionScriptとJavascriptをちょこっと



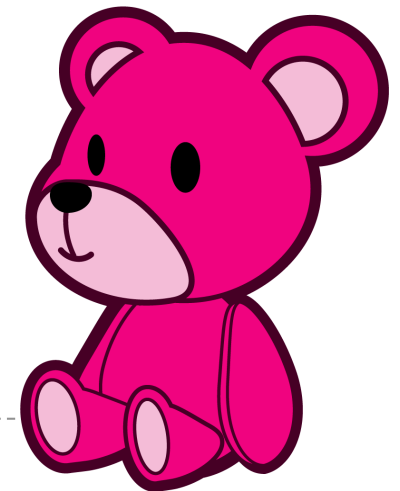
# 千葉県代表 石井(2)

- ・ もうJavaには戻れない。
- ・ コードの量は本当に3分の1くらいにはなる
- ・ Javaに+ $\alpha$ して覚えなければいけないことが増える分、問題解決の道具も増えて行く。
- ・ 型推論最高
- ・ Javaは設計が終わったらやること一緒。Scalaはメソッド書いてるときも楽しい。Javaだとただ面倒くさいだけの作業が、Scalaだとスマートに解決できる。逆にScalaでの設計はまだ未知なところが多い。
- ・ Lightweight感は他のLLと比べるとまだまだかな (スクリプト環境でgemのようなパッケージマネージャが呼び出せたら文句なしなのに……sbazは微妙。。)



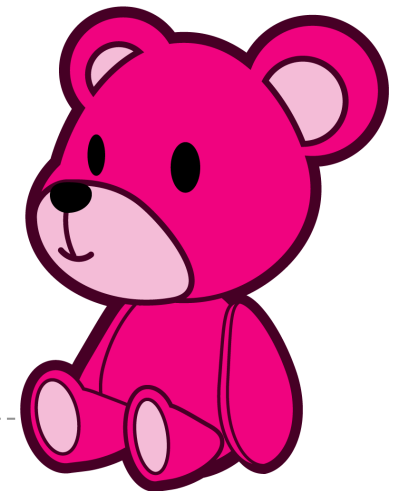
# 千葉県代表 石井(3)

- ・ ついついワンライナーで書いてしまうが、最近気をつけている。
  - Miki > なんで、ワンライナー気をつけてるの？
  - Ishii > 自分でも読みづらいからですw
- ・ var は最終手段。見やすく美しくimmutableに書く方法を模索している。
  - Miki > 関数型最高って感じ？
  - Ishii > 奥深い感じです・・・。
- ・ コンパイルが遅いのはJRebelで解決。
- ・ IDEはEclipseからIntelliJにした。
  - Miki > どうよ？
  - Ishii > Eclipseより気が効いてます。買っちゃおうかな。
- ・ Sbazの利点を誰か教えてください。  
ないならみんなMaven使ってください。



# 大阪府代表 三木（1）

- ・ Scala歴：1年3ヶ月
  - 最近は、Postgresを触ってる時間の方が・・・
- ・ 業務でScalaを使いこなせるようになるまでにかかった時間：まだまだかな。。
- ・ 他の言語歴
  - C言語：4年
  - C#、C++、Java、PHPなどをちょこちょこ



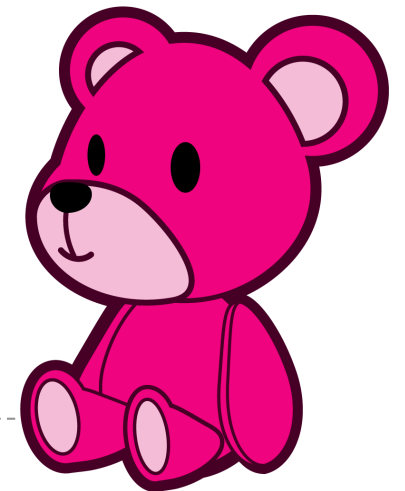
# 大阪府代表 三木 (2)

- ・ 型推論最高
- ・ 暗黙の型変換最高
- ・ Scalaで一番感動した事は？
  - 型推論
- ・ Scalaの良い部分
  - プログラマが自由に出来る部分が多い
    - ・ ルールで縛って押しつけるのはナンセンス。
    - ・ チーム内で何が最高の書き方か？模索していく
      - ワイワイ楽しくやれる。
    - ・ 結果、プログラマが自分のコードに責任と誇りを持つようになる



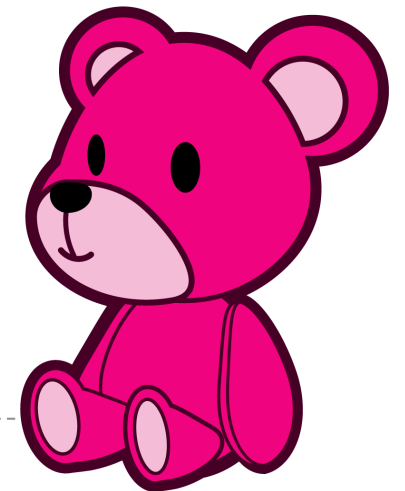
# 大阪府代表 三木（3）

- ・ REPL(対話型開発環境)は便利
  - その場で動作確認出来る
  - 関数型っぽく書いてあると、コピペするだけで動作確認が出来る
  - ちょっとしたバッチ処理を実行したり出来る



# 番外編 静岡県代表 白木(1)

- 職業 : webデザイナー 兼 UIエンジニア
- デザイナー歴 : 5年
- 使う言語 : JS, PHP, Python
- 昔はStrutsと闘いながらViewを担当して  
今はLiftと闘いながらViewを担当



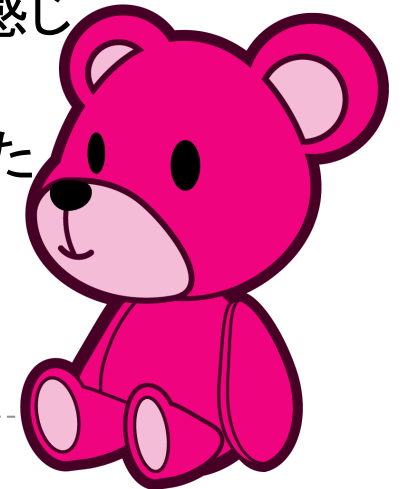
# 番外編 静岡県代表 白木(2)

## 【Liftとの邂逅】

- ・最初1ヶ月はXHTMLバリデーションでボコボコに…(自分が悪い)
- ・次の1ヶ月は「&」表記を「&amp;」にするなどのルールにめった打ちに…

## 【使ってみた印象】

- ・templateの機構がしっかりしているのでソースの管理コストが激減した  
(これは本当に良くできている！)
- ・ViewがControllerからしっかり独立しておりコードが綺麗で良い感じ
- ・その分、動的な部分はsnippet依存度が大きく、  
エンジニア-デザイナー間のコミュニケーションコストは上がった  
→仲良くなった



# 番外編 静岡県代表 白木(3)

- View上で条件分岐などが許容されないのは  
ちょっとモドカシイ(Strutsの<logic:notEmpty>が無い、というか)
- View上で急場しのぎ的なjavaを書けないのもモドカシイ  
(だがそれがいい)
- LiftAjaxなるものがあるらしいが、イマイチ使える感じがしない  
- MIKI > Lift2.0に期待☆

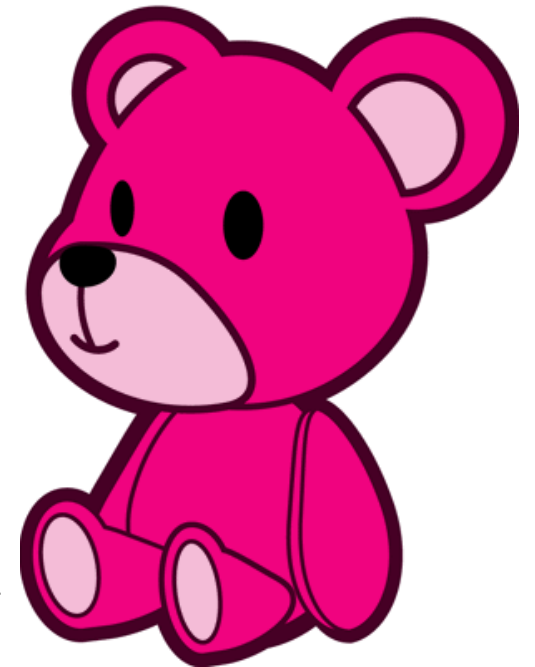
→独立性を高めた結果、Viewソースレベルでの柔軟性が落ちたが、それを差し引いてもメリットと思えるテンプレート機構があり複雑な仕様を持つサイトに向いている印象。

→Maven最高。



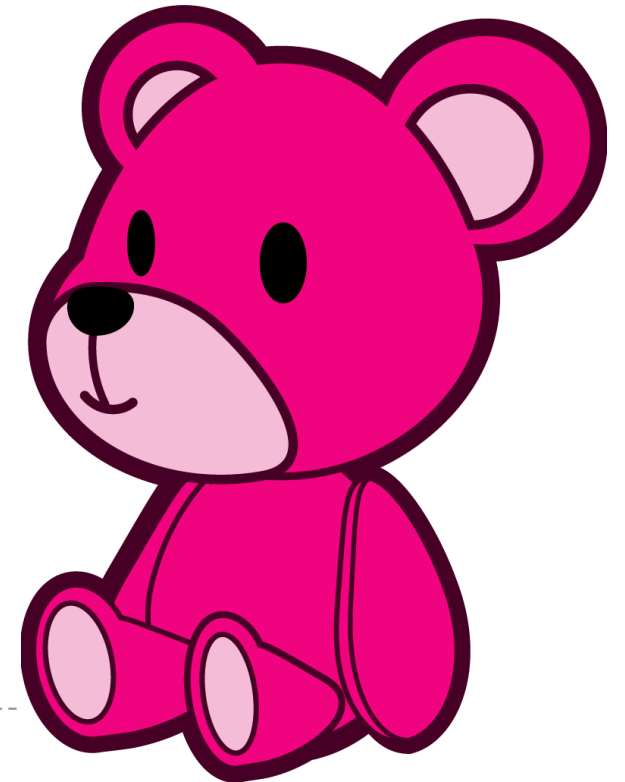
最後に

ご質問は  
御座いますでしょうか？



ご静聴

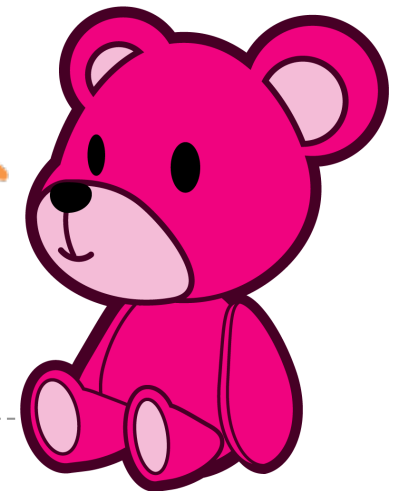
ありがとうございました。



# 終わり ～ the end ～

- 株式会社パテントビューロ 三木隆史
- **miki@patentbureau.co.jp**
- **twitter: kuma3\_neko3**

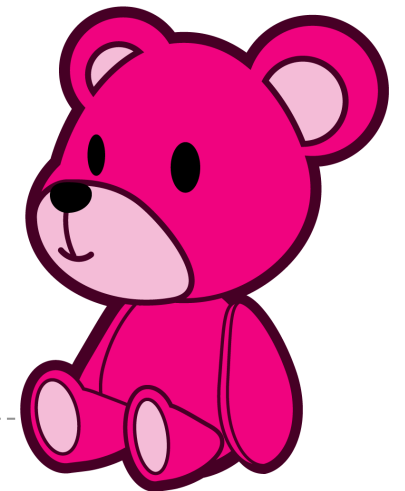
おまけ付きだよ



# Singleton パターン scala 4行

```
object MySingleton{  
  def getSomething(){//省略 }  
}  
  
val s = MySingleton.getSomething
```

おまけだよ

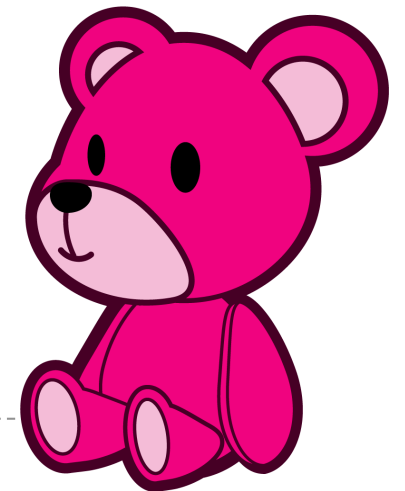


# Singleton パターン

## java 9行

```
class MySingleton{  
    private static MySingleton instance = null;  
    public static MySingleton getInstance() {  
        if (instance == null) { instance = new MySingleton() }  
        return instance;  
    }  
    public Something getSomething(){ //省略 }  
}
```

```
Something s = MySingleton.getInstance.getSomething
```



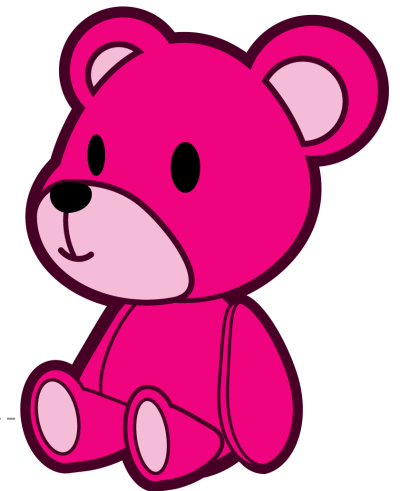
# Decorator パターン

## scala 15行

```
trait Instrument{
  def play:String
}
trait Drum extends Instrument{
  abstract override def play = super.play + "ドンドンドン!"
}
trait Guiter extends Instrument {
  abstract override def play = super.play + "ジャンジャン!"
}
class Vocal extends Instrument {
  def play = "せーの !"
}
val music = new Vocal with Drum with Guiter
println(music.play)
)
```

出力

> せーの !ドンドンドン!ジャンジャン!



# Decorator パターン

## java 28行

```
interface Instrument {
    String play();
}

public class Drum implements Instrument {
    private Instrument instruments = null;

    public Drum(Instrument instruments) {
        this.instruments = instruments;
    }

    public String play() {
        return instruments.play() + "ドンドンドン!";
    }
}

public class Guitar implements Instrument {
    private Instrument instruments = null;

    public Guitar(Instrument instruments) {
        this.instruments = instruments;
    }

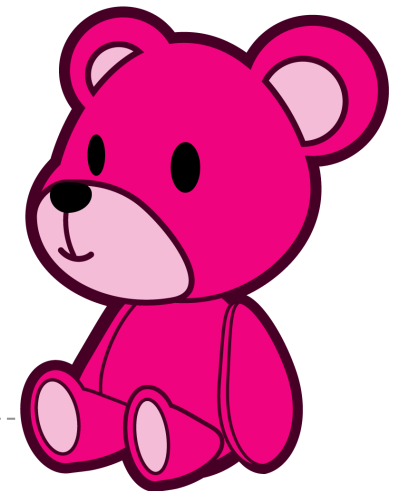
    public String play() {
        return instruments.play() + "ジャンジャンジャン!";
    }
}
```

```
public class Vocal implements Instrument {
    public String play() {
        return "せーの!";
    }
}

public static void main(String[] args) {
    Instrument music = new Drum(new Guitar(new Vocal
    ()));
    System.out.println(music.play());
}
```

出力

せーの！ジャンジャンジャン！ドンドンドン！



# Chain of Responsibility パターン

## scala 25行

```
object Logger extends Enumeration {
  val ERROR = Value(1)
  val NOTICE = Value(2)
  val DEBUG = Value(4)

  private var _loggers: List[Logger] = Nil
  private var initialized = false
  def init(loggers:List[Logger]) =
    if (!initialized) { _loggers = loggers; initialized = true }
    else throw new IllegalStateException("Logger is already initialized")
  def message(level:Logger.Value, msg:String) = _loggers.filter(!_isDefinedAt(level,msg)).foreach(l => l(level,msg))
}

abstract class Logger(val level:Logger.Value) extends PartialFunction[(Logger.Value, String), Unit] {
  override def isDefinedAt(in:(Logger.Value,String)) = in._1 == level
}

class StdoutLogger(level:Logger.Value) extends Logger(level) {
  override def apply(in:(Logger.Value,String)) = println("Writing to stdout: " + in._2)
}

class EmailLogger(level:Logger.Value) extends Logger(level) {
  override def apply(in:(Logger.Value,String)) = println("Sending via email: " + in._2)
}

class StderrLogger(level:Logger.Value) extends Logger(level) {
  override def apply(in:(Logger.Value,String)) = println("Writing to stderr: " + in._2)
}

Logger.init(new StdoutLogger(Logger.DEBUG) :: new EmailLogger(Logger.NOTICE) :: new StderrLogger(Logger.ERROR) :: Nil)
Logger.message(Logger.DEBUG,"It is debug message")
```



# Chain of Responsibility

Patent Bureau Co., Ltd.  
パターン

## java 56行

```
abstract class Logger
{
    public static int ERR = 3;
    public static int NOTICE = 5;
    public static int DEBUG = 7;
    protected int mask;

    // The next element in the chain of responsibility
    protected Logger next;
    public Logger setNext( Logger l)
    {
        next = l;
        return this;
    }
    public void message( String msg, int priority )
    {
        if ( priority <= mask )
        {
            writeMessage( msg );
            if ( next != null )
            {
                next.message( msg, priority );
            }
        }
    }
    abstract protected void writeMessage( String msg );
}

class StdoutLogger extends Logger
{
    public StdoutLogger( int mask ) { this.mask = mask; }

    protected void writeMessage( String msg )
    {
        System.out.println( "Writing to stdout: " + msg );
    }
}
```

```
class EmailLogger extends Logger
{
    public EmailLogger( int mask ) { this.mask = mask; }

    protected void writeMessage( String msg )
    {
        System.out.println( "Sending via email: " + msg );
    }
}

class StderrLogger extends Logger
{
    public StderrLogger( int mask ) { this.mask = mask; }

    protected void writeMessage( String msg )
    {
        System.out.println( "Sending to stderr: " + msg );
    }
}
```

```
public class ChainOfResponsibilityExample
{
    public static void main( String[] args )
    {
        // Build the chain of responsibility
        Logger l = new StdoutLogger( Logger.DEBUG ).setNext(
            new EmailLogger( Logger.NOTICE ).setNext(
                new StderrLogger( Logger.ERR ) );

        // Handled by StdoutLogger
        l.message( "Entering function y.", Logger.DEBUG );

        // Handled by StdoutLogger and EmailLogger
        l.message( "Step1 completed.", Logger.NOTICE );

        // Handled by all three loggers
        l.message( "An error has occurred.", Logger.ERR );
    }
}
```

