real world clojure

sean corfield
world singles

doing boring stuff
with
an exciting language

clojure is good for?

- big data & analytics
- logic programming & pattern matching
- heavy concurrency & massive scale
- hard problems!

clojure is also...

- "It endeavors to be a general-purpose language suitable in those areas where Java is suitable."
- -- http://clojure.org/rationale

world singles

- internet dating platform
- multi-lingual: half dozen languages
- multi-tenant: 50 sites, one code base

DESIKISS.COM™
Connecting Indians Worldwide

Already a member? **Sign In**

Find Your Indian Partner
For Life, Love and Marriage.

I am -Select-

From -Select-

Birthday -Month- -Day- -Year-

Username

Password

Email Address

By clicking Join Now, you agree to the Terms and Conditions

Join Now >

Arjun contacted me on DesiKiss and our lives haven't been the same since. We are now happily married and are awaiting our first child. After trying several other online dating sites, DesiKiss is where our fates collided. We are both eternally grateful for the opportunity you have afforded us.

- Sreeja (3jaLove)

LovingBBW.com™
Meet BBW Singles

Already a member? **Sign In**

Find Your BBW Partner For Life, Love and Marriage.

I am -Select-

From -Select-

Birthday -Month- -Day- -Year-

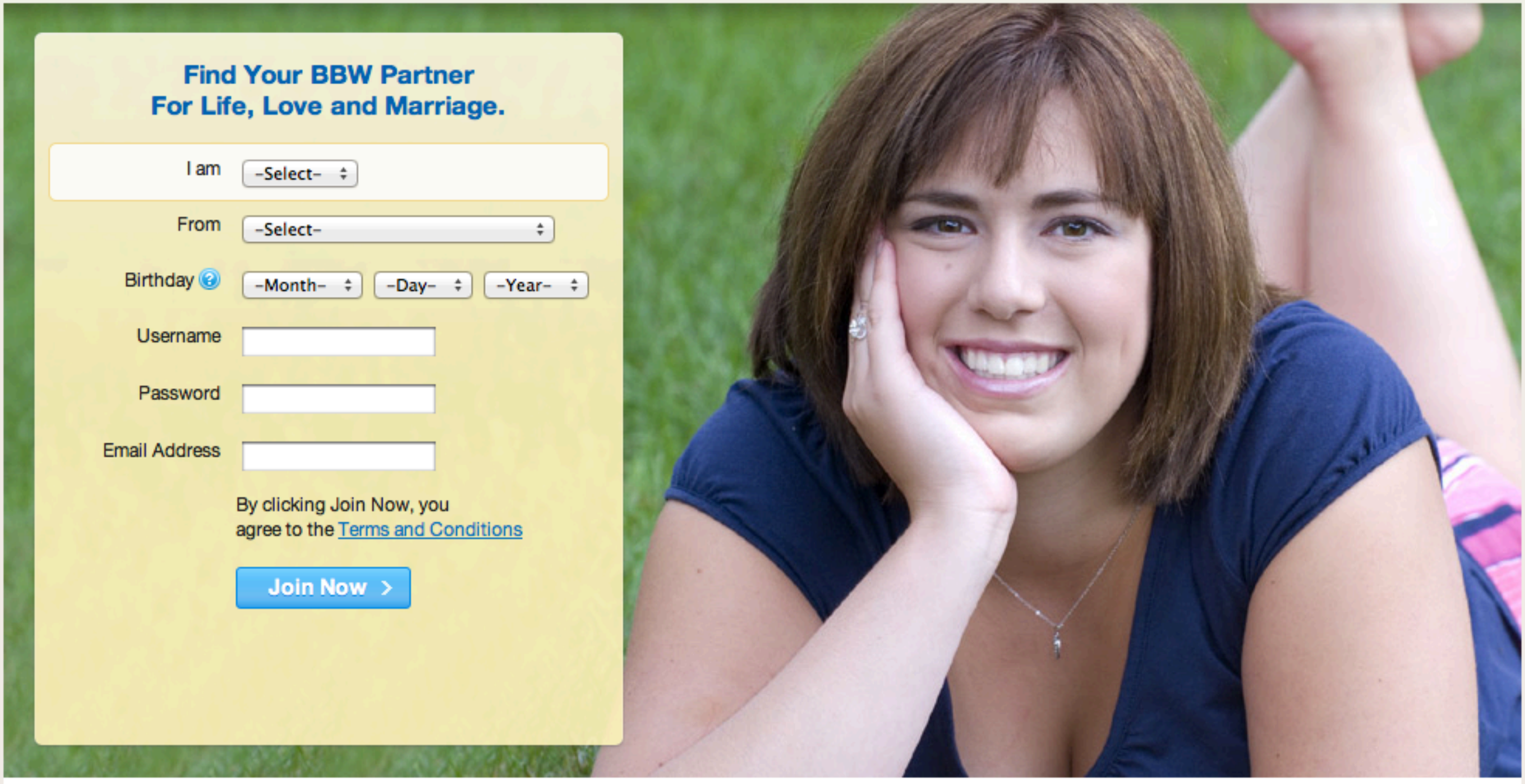
Username

Password

Email Address

By clicking Join Now, you agree to the Terms and Conditions

Join Now >



I met someone very special on this site. This is the best site I've ever been on and I've tried EVERY SINGLE ONE OF THEM. Thank you LovingBBW. We are talking about getting married and buying a home next year. Thanks again!

- Rick (HonestOne8)

HaitiDating.com
Connecting Haitians Worldwide

English français

Déjà membre? Se connecter

Trouvez votre partenaire Haitien(ne) pour la vie, l'amour et le mariage.

Je suis – Sélectionnez –

De – Sélectionnez –

Anniversaire – Jour – – Mois – – Année –

Pseudo

Mot de passe

Adresse email

En cliquant sur S'inscrire maintenant, vous acceptez les Conditions Générales

S'inscrire maintenant >

Au début, je n'étais pas sûr que les rencontres en ligne étaient faites pour moi. J'ai hésité à créer mon profil. Mais le lendemain, j'ai reçu un message de ma future âme sœur! Nous avons bavardé pendant quelques jours puis nous nous sommes rencontrés. C'était le jour le plus fou de ma vie. Il était doux et beau. Nous avions tous les deux fréquenté la même université, mais à des moments différents. Nous étions faits pour être ensemble. Milles mercis d'avoir rendu cette rencontre possible !!

- Martine (LaBelle75)

clojure & us

- needed to process large amounts of data
  - (irony: one of clojure's touted strengths)
- 2009 introduced scala for "heavy lifting"
  - not a good cultural fit for our team
- 2010 started evaluating clojure
- 2011 first production usage

introducing clojure

- pick small, low-level components
- reusable from java / other languages
  - :gen-class / clojure.lang.RT
- logging (tools.logging, log4j)
- environment control (dev/ci/qa/prod)

clojure & us

- email (html generation & sending)
- environment control
- geo location
- i18n
- logging
- persistence
- search engine interaction (json/xml)

clojure & us

- time is short - focus on
- persistence
- search engine interaction

persistence

- crud wrapper around clojure.java.jdbc
- (get-by-id :table id)
- (find-by-keys f :table {:key val})
- (save-row :table {:id pk ...})
- (execute f "sql statement" [params])

persistence

```
(defn get-by-id
  ([table id] (get-by-id table id :id))
  ([table id pk]
     (sql/with-naming-strategy qi
       (sql/with-connection
         (ws/worldsingles-db-readonly)
         (sql/with-query-results rows
           [(str "SELECT * FROM " (q table)
                 " WHERE " (q pk) " = ?") id]
           (first rows))))))
```

persistence

```
(def q
  "Convert keyword to MySQL quoted entity name."
  (partial sql/as-quoted-identifier \`))
```

```
(def qi
  "Naming strategy for MySQL quoting and no lowercasing in results."
  { :entity (partial sql/as-quoted-str \`) :keyword identity })
```

persistence

- same crud wrapper around congomongo!
  - (except for "execute")
- abstraction allows us to mix'n'match mysql data and mongodb documents

persistence

```
(defn get-by-id
  ([table id] (get-by-id table id :id))
  ([table id pk]
     (if-let [mongo-metadata
          ((sql/as-keyword table) ws/mongo-tables)]
(mongo/with-mongo (ws/mongo-db)
  (mongo/fetch-by-id (:collection mongo-metadata) id))
(... previous sql version of get-by-id...))))
```

persistence

- generic application code uses mysql or mongodb transparently based on :table
  - (get-by-id :table id)
  - (find-by-keys f :table {:key val})
  - (save-row :table {:id pk ...})

search engine

- discovery by transparensee
  - powerful, fast, fuzzy searching
- http post xml searchable data
- http post json queries, returns json

search engine

- scan database for changed profiles
- create xml via hiccup
- posted via clj-http
- (heavy lifting we introduced scala for)

search engine

```
(defn- post [options xml]
  (client/post (str (:transparensee options) "/ws/changeset")
               {:headers {"Content-Type" "text/xml"}
                :body xml}))

(defn- process-users [options users]
  (let [xml (render/html [:changeset (map render-user users)])]
    (post options xml)))
```

search engine

```
(defn- render-user [user]
  (if (excluded-from-search user)
    [:remove-item {:id (:id user)}]
    [:set-item {:id (:id user)}
     [:properties
      [:struct
       (let [...]
         (filter identity (map render-item dims)))]]]))
```

search engine

- json created via clojure.data.json
- posted via clj-http
- result parsed via clojure.data.json

search engine

```
(defn- transparensee-search [data & {:keys [skip-exact]}]
  (try
    (let [json-query (json/json-str data)
          result (post json-query)]
      (if (= 200 (:status result))
        (let [result (json/read-json (:body result))
              user-query (if (empty? (:itemIds result)) []
                             (get-users-by-item-ids result))]
          {:success true :matches result
           :query (make-ordered-query user-query result)})
        {:success false :httpStatusCode (:status result)
         :httpFileContent (:body result)}))
    (catch Exception e
      (log/error my-ns {:message "transparensee-search failed
                        :exception (.toString e)
                        :query (json/json-str data)})
      {:success false :exception e})))
```

search engine

```
(let [json-query (json/json-str data)
      result (post json-query)]
  (if (= 200 (:status result))
    (let [result (json/read-json (:body result))
```

search engine

```
(defn- transparensee-search [data & {:keys [skip-exact]}]
  (try
    (let [json-query (json/json-str data)
          result (post json-query)]
      (if (= 200 (:status result))
        (let [result (json/read-json (:body result))
              user-query (if (empty? (:itemIds result)) []
                             (get-users-by-item-ids result))]
          {:success true :matches result
           :query (make-ordered-query user-query result)})
        {:success false :httpStatusCode (:status result)
         :httpFileContent (:body result)}))
    (catch Exception e
      (log/error my-ns {:message "transparensee-search failed
                        :exception (.toString e)
                        :query (json/json-str data)})
      {:success false :exception e})))
```

search engine

```
(defn- post [json-query]
  (let [s @env/my-settings]
    (client/post (str (:TransparenseeQuery s) "/json/query")
                 {:headers {"Content-Type" "application/json"}
                  :body json-query})))
```

```
(defn- get-users-by-item-ids [result]
  (crud/execute
   doall
   (str "SELECT u.*, p.photoName, p.photoExt FROM user u
         LEFT OUTER JOIN userPhoto p ON u.id = p.userId AND p.c
         WHERE u.id in ("
        (str/join "," (:itemIds result))
        ")")))
```

clojure & us

- email (html generation & sending)
- environment control
- geo location
- i18n
- logging
- persistence
- search engine interaction (json/xml)

libraries we use

- clojure contrib
  - data.json, java.jdbc, tools.cli, tools.logging
- other clojure
  - clj-http, clj-time, clojure-csv, congomongo, date-clj, enlive, hiccup
- java
  - c3p0, log4j, javax.mail, mysql

code stats

- 35 src .clj files
- 3,435 lines
- 19 test .clj files
- 746 lines
- lein multi test against 1.3.0 & 1.4.0

clojure benefits

- performs well
- scales well for data complexity
- immutable data provides thread safety
- maintenance / flexibility is good

clojure & the future

- training more of our team
- clojure code base is growing
- most new backend code will be clojure
- looking at cascalog

we're hiring!

linkd.in/worldsingles

questions? / contact

- @seancorfield
- sean@worldsingles.com
- http://worldsingles.com
- http://corfield.org