The Java EE 7 Platform: Higher Productivity and Embracing HTML5

Arun Gupta
Java EE & GlassFish Guy, @arungupta
The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Java EE 6 Platform
December 10, 2009
Java EE 6 – Key Statistics

- 40+ Million Java EE 6 Component Downloads
- #1 Choice for Enterprise Developers
- #1 Application Development Platform
- Fastest implementation of a Java EE release
Java EE 7 Revised Scope
Productivity and HTML5

• Higher Productivity
  – Less Boilerplate
  – Richer Functionality
  – More Defaults

• HTML5 Support
  – WebSocket
  – JSON
  – HTML5 Forms
Java EE 7 – Candidate JSRs

- Portable Extensions
  - JSP 2.2
  - JSF 2.2
  - JAX-RS 2.0
  - EL 3.0

- Servlet 3.1
  - Servlet 3.1

- Common Annotations 1.1
  - Interceptors 1.1

- Managed Beans 1.0
  - EJB 3.2

- Connector 1.6
  - JPA 2.1
  - JTA 1.2
  - JMS 2.0

- Bean Validation 1.1
  - Java Caching API (JSR 107)
  - Concurrency Utilities (JSR 236)
  - Batch Applications (JSR 352)
  - Java API for JSON (JSR 353)
  - Java API for WebSocket (JSR 356)

- New
- Major Release
- Updated
Java API for RESTful Web Services 2.0

- Client API
- Message Filters & Entity Interceptors
- Asynchronous Processing – Server & Client
- Hypermedia Support
- Common Configuration
// Get instance of Client
Client client = ClientFactory.newClient();

// Get customer name for the shipped products
String name = client.target("../orders/{orderId}/customer")
    .resolveTemplate("orderId", "10")
    .QueryParam("shipped", "true")
    .request()
    .get(String.class);
Java Message Service 2.0
Simplify the existing API

• Less verbose
• Reduce boilerplate code
• Resource injection
• Connection, Session, and other objects are AutoCloseable
• Requires Resource Adapter for Java EE containers
• Simplified API in both Java SE and EE
Java Message Service 2.0
Sending a Message using JMS 1.1

@Resource(lookup = "myConnectionFactory")
ConnectionFactory connectionFactory;

@Resource(lookup = "myQueue")
Queue myQueue;

public void sendMessage (String payload) {
    Connection connection = null;
    try {
        connection = connectionFactory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer messageProducer = session.createProducer(myQueue);
        TextMessage textMessage = session.createTextMessage(payload);
        messageProducer.send(textMessage);
    } catch (JMSException ex) {
        // . . .
    } finally {
        if (connection != null) {
            try {
                connection.close();
            } catch (JMSException ex) {
                // . . .
            }
        }
    }
}
Java Message Service 2.0
Sending message using JMS 2.0

@Inject
JMSContext context;

@Resource(lookup = "java:global/jms/demoQueue")
Queue demoQueue;

public void sendMessage(String payload) {
    context.createProducer().send(demoQueue, payload);
}
Java API for JSON Processing 1.0

- API to parse and generate JSON
- Streaming API
  - Low-level, efficient way to parse/generate JSON
  - Provides pluggability for parsers/generators
- Object Model
  - Simple, easy-to-use high-level API
  - Implemented on top of Streaming API
- Binding JSON to Java objects forthcoming
Java API for JSON Processing 1.0
Streaming API – JsonParser

```java
{
   "firstName": "John", "lastName": "Smith", "age": 25,
   "phoneNumber": [
   { "type": "home", "number": "212 555-1234" },
   { "type": "fax", "number": "646 555-4567" }
   ]
}
Iterator<Event> it = parser.iterator();
Event event = it.next(); // START_OBJECT
event = it.next(); // KEY_NAME
event = it.next(); // VALUE_STRING
String name = parser.getString(); // "John"
```
Java API for WebSocket 1.0

- **API for WebSocket Client/Endpoints**
  - Annotation-driven (@WebSocketEndpoint)
  - Interface-driven (Endpoint)
  - Client (@WebSocketClient)

- **SPI for data frames**
  - WebSocket opening handshake negotiation

- **Integration with Java EE Web container**
Java API for WebSocket 1.0
Hello World – POJO/Annotation-driven

```java
import javax.websocket.*;

@WebSocketEndpoint("/hello")
public class HelloBean {

    @WebSocketMessage
    public String sayHello(String name) {
        return "Hello " + name;
    }
}
```
@WebSocketEndpoint("/chat")
public class ChatBean {
    Set<Session> peers = Collections.synchronizedSet(…);

    @WebSocketOpen
    public void onOpen(Session peer) {
        peers.add(peer);
    }

    @WebSocketClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }

    ...
}
@WebSocketMessage

public void message(String message, Session client) {
    for (Session peer : peers) {
        peer.getRemote().sendObject(message);
    }
}
Bean Validation 1.1

- Open: Spec, Reference Implementation, TCK
- Alignment with Dependency Injection
- Method-level validation
  - Constraints on parameters and return values
  - Check pre-/post-conditions
public void placeOrder(
    @NotNull String productName,
    @NotNull @Max("10") Integer quantity,
    @Customer String customer) {
    //...
Batch Applications for the Java Platform 1.0

- Suited for non-interactive, bulk-oriented and long-running tasks
- Computationally intensive
- Can execute sequentially/parallel
- May be initiated
  - Adhoc
  - Scheduled
    - No scheduling APIs included
Batch Applications for the Java Platform 1.0

Concepts

- **Job**: Entire batch process
  - Put together through a Job Specification Language (XML)

- **Step**: Independent, sequential phase of a job
  - **ItemReader**: Retrieval of input for a step, one at a time
  - **ItemProcessor**: Business processing of an item
  - **ItemWriter**: Output of an item, chunks of items at a time

- **JobOperator**: Manage batch processing

- **JobRepository**: Metadata for jobs
<step id="sendStatements">
  <chunk reader="AccountReader" processor="AccountProcessor" writer="EmailWriter" chunk-size="10" />
  <ReadItem>
    public Account readAccount() {
      // read account using JPA
    }
  </ReadItem>
  <ProcessItem>
    public Account processAccount(Account account) {
      // calculate balance
    }
  </ProcessItem>
  <WriteItems>
    public void sendEmail(List<Account> accounts) {
      // use JavaMail to send email
    }
  </WriteItems>
</step>
Java Temporary Caching API 1.0

- **API** and semantics for temporary, in-memory caching of Java objects
  - Object creation
  - Shared access
  - Spooling
  - Invalidation
  - Consistency across JVMs
- **SPI** for implementers
Java Temporary Caching API 1.0

Code Sample – Blog

```java
public class BlogManager {
    @CachePut(cacheName="blogManager")
    public void createEntry(
        @CacheKeyParam String title,
        @CacheValue Blog blog) {
        ...
    }
    ...
}
```
Java Temporary Caching API 1.0

Code Sample – Blog

```java
public Blog getBlogEntry(String title) {...}

@CacheResult(cacheName="blogManager")
public Blog getEntryCached(
    String randomArg,
    @CacheKeyParam String title) {...}
```

```java
@CacheResult(cacheName="blogManager")
public Blog getEntryCached(
    String randomArg,
    @CacheKeyParam String title) {...}
```

```java
... ...
```
Java Temporary Caching API 1.0

Annotations – Blog Sample

```java
@CacheRemoveEntry(cacheName="blogManager")
public void removeBlogEntry(String title) {
...

@CacheRemoveAll(cacheName="blogManager")
public void removeAllBlogs() {
...
```
Java Persistence API 2.1

- Schema Generation
- Unsynchronized Persistence Contexts
- Converters
- Bulk update/delete using Criteria
- User-defined functions using FUNCTION
- Stored Procedure Query
Servlet 3.1

- Non-blocking I/O
- Protocol Upgrade
- Security Enhancements
public class TestServlet extends HttpServlet
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
    ServletInputStream input = request.getInputStream();
    byte[] b = new byte[1024];
    int len = -1;
    while ((len = input.read(b)) != -1) {
        . . .
    }
}
AsyncContext context = request.startAsync();
ServletInputStream input = request.getInputStream();
input.setReadListener(
    new MyReadListener(input, context));
@Override
public void onDataAvailable() {
    try {
        StringBuilder sb = new StringBuilder();
        int len = -1;
        byte b[] = new byte[1024];
        while (input.isReady() && (len = input.read(b)) != -1) {
            String data = new String(b, 0, len);
            System.out.println("--> " + data);
        }
    } catch (IOException ex) {
        ...
    }
    ...
}

Non-blocking I/O: MyReadListener Code Sample
Concurrency Utilities for Java EE 1.0

Goals

- Provide concurrency capabilities to Java EE application components
  - Without compromising container integrity
- Support simple (common) and advanced concurrency patterns
Recommended to bind in java:comp/env/concurrent subcontext

```xml
<resource-env-ref>
  <resource-env-ref-name>concurrent/BatchExecutor</resource-env-ref-name>
  <resource-env-ref-type>javax.enterprise.concurrent.ManagedExecutorService</resource-env-ref-type>
</resource-env-ref>
```
Concurrency Utilities for Java EE 1.0

Submit Tasks to ManagedExecutorService using JNDI

public class TestServlet extends HTTPServlet {
    @Resource(name="concurrent/BatchExecutor")
    ManagedExecutorServiceService executor;

    Future future = executor.submit(new MyTask());

    class MyTask implements Runnable {
        public void run() {
            .. // task logic
        }
    }
}
JavaServer Faces 2.2

- @FlowScoped
- HTML5 Friendly Markup Support
  - Pass through attributes and elements
- Cross Site Request Forgery Protection
- Loading Facelets via ResourceHandler
- File Upload Component
- Multi-templating
Java EE 7 – Implementation Status

download.java.net/glassfish/4.0/promoted/
Java EE 8 and Beyond
Standards-based cloud programming model

• Deliver cloud architecture
• Multi tenancy for SaaS applications
• Incremental delivery of JSRs
• Modularity based on Jigsaw
Adopt-a-JSR

How do I get started? – glassfish.org/adoptajsr

- Java API for Temporary Caching 1.0 (JSR 107)
- Concurrency Utilities for Java EE 1.0 (JSR 236)
- Java Persistence API 2.1 (JSR 338)
- Java API for RESTful Web Services 2.0 (JSR 339)
- Servlet 3.1 (JSR 340)
- Expression Language 3.0 (JSR 341)
- Java Message Service 2.0 (JSR 343)
- JavaServer Faces 2.2 (JSR 344)
- Enterprise JavaBeans 3.2 (JSR 345)
- Contexts and Dependency Injection 1.1 (JSR 346)
- Bean Validation 1.1 (JSR 349)
- Batch Applications for the Java Platform 1.0 (JSR 352)
- Java API for JSON Processing 1.0 (JSR 353)
- Java API for WebSocket 1.0 (JSR 356)
- Java Transaction API 1.2 (JSR 907)
Adopt-a-JSR

Participating JUGs
Call to Action

• Java EE 7 Transparent Expert Groups
  – javaee-spec.java.net

• Java EE 7 Reference Implementation
  – glassfish.org

• The Aquarium
  – blogs.oracle.com/theaquarium

• Adopt a JSR
  – glassfish.org/adoptajsr