The Java EE 7 Platform

Arun Gupta · Red Hat · @arungupta

Arun Gupta

- Director, Developer Advocacy, Red Hat Inc.
- O'Reilly and McGraw Hill author
- Fitness freak

Java EE 7 Themes

- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

Top 10 Features in Java EE 7

- WebSocket client/server endpoints

- WebSocket client/server endpoints
- **Batch Applications**

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- **JSON Processing**

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- **Concurrency Utilities**

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API
- `@Transactional` and `@TransactionScoped`

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API
- `@Transactional` and `@TransactionScoped`
- JAX-RS Client API

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API
- `@Transactional` and `@TransactionScoped`
- JAX-RS Client API
- **Default Resources**

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API
- `@Transactional` and `@TransactionScoped`
- JAX-RS Client API
- Default Resources
- More annotated POJOs

Top 10 Features in Java EE 7

- WebSocket client/server endpoints
- Batch Applications
- JSON Processing
- Concurrency Utilities
- Simplified JMS API
- `@Transactional` and `@TransactionScoped`
- JAX-RS Client API
- Default Resources
- More annotated POJOs
- **Faces Flow**

Java API for WebSocket 1.0 (JSR 356)

- **Server and Client WebSocket Endpoint**
  - Annotated: `@ServerEndpoint`, `@ClientEndpoint`
  - Programmatic: `Endpoint`
- **Lifecycle methods**
  - `@OnOpen`, `@OnClose`, `@OnError`, `@OnMessage`
- **Packaging and Deployment**

WebSocket Sample

ChatServer.java

```
@ServerEndpoint("/chat") (1)
public class ChatEndpoint {
    @OnMessage (2)
    public void message(String message,
                        Session client) (3)
                    throws IOException, EncodeException {
        for (Session peer : client.getOpenSessions()) {
            peer.getBasicRemote().sendText(message);
        }
    }
}
```

(1) Creates a WebSocket endpoint, defines the listening URL

(2) Marks the method that receives incoming WebSocket message

(3) Payload of the WebSocket message

Java API for JSON Processing 1.0 (JSR 353)

- API to parse and generate JSON
- Streaming API
  - Low-level, efficient way to parse/generate JSON
  - Similar to StAX API in XML world
- Object Model API
  - Simple, easy to use high-level API
  - Similar to DOM API in XML world

JSON-P Sample

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumber": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "fax", "number": "646 555-4567" }
  ]
}
```

```
JsonParser p = Json.createParser("...");
JsonParser.Event event = p.next();          // START_OBJECT
event = p.next();                           // KEY_NAME
event = p.next();                           // VALUE_STRING
String name = p.getString();                // "John”
```

Batch Applications for Java Platform 1.0 (JSR 352)

- Suited for non-interactive, bulk-oriented, and long-running tasks
- Batch execution: sequential, parallel, decision-based
- Processing Styles
  - Item-oriented: Chunked (primary)
  - Task-oriented: Batchlet

Batch Sample

myJob.xml

```
<step id=”sendStatements”>
  <chunk item-count=“3”>
    <reader ref=”accountReader”/>
    <processor ref=”accountProcessor”/>
    <writer ref=”emailWriter”/>
</step>
```

Batch Sample (contd)

AccountReader.java

```
... implements ItemReader {
  public Object readItem() {
    // read account using JPA
  }
```

AccountProcessor.java

```
...implements ItemProcessor {
  public Object processItems(Object account) {
    // read Account, return Statement
  }
```

EmailWriter.java

```
...implements ItemWriter {
  public void writeItems(List accounts) {
    // use JavaMail to send email
  }
```

Concurrency Utilities for Java EE 1.0 (JSR 236)

- Extension of Java SE Concurrency Utilities API
- Provide asynchronous capabilities to Java EE application components
- Provides 4 types of managed objects
  - `ManagedExecutorService`
  - `ManagedScheduledExecutorService`
  - `ManagedThreadFactory`
  - `ContextService`
- Context Propagation

Concurrency Utilities Sample

```
@Resource(name="java:comp/DefaultManagedExecutorService")
ManagedExecutorService executor; (1)

Future future = executor.submit(new MyTask()); (2)

class MyTask implements Runnable {
  public void run() {
    . . . (3)
  }
}
```

1. ManagedExecutorService using default resource
2. Standard Java SE Concurrency APIs
3. Task logic

Java Message Service 2.0

- New JMSContext interface
- AutoCloseable `JMSContext`, `Connection`, `Session`, ...
- Use of runtime exceptions
- Method chaining on `JMSProducer`
- Simplified message sending

JMS 2.0 Sample

```
@JMSDestinationDefinition(name="myQueue", interfaceName="javax.jms.Queue") 1

@Resource(mappedName="myQueue")
Queue syncQueue;

@Inject
// @JMSConnectionFactory("java:comp/DefaultJMSConnectionFactory") 2
private JMSContext context; 3

context.createProducer().send(syncQueue, "..."); 4
```

1. Create destination resource during deployment
2. Default JMS connection factory
3. Main interface of the simplified API
4. Fluent builder API, runtime exceptions

- Client API
- Message Filters and Entity Interceptors
- Asynchronous Processing – Server and Client
- Common Configuration

JAX-RS Client API Sample

`RunClient.java`

```
Client client = ClientBuilder.newClient(); (1)
WebTarget target = client.target(...); (2)
target.register(Person.class);
Person p = target
              .path("{id}") (3)
              .resolveTemplate("id", "1")
              .request(MediaType.APPLICATION_XML) (4)
              .get(Person.class); (5)
```

1. `ClientBuilder` is the entry point
2. Build a new web resource target, specifies the path
3. Sub resource URI
4. Define the accepted response media types
5. Call HTTP GET, specify the type of resource

Contexts and Dependency Injection 1.1 (JSR 346)

- Automatic enablement for beans with scope annotation and EJBs
- “beans.xml” is optional
- Bean discovery mode
  - `all`: All types
  - `annotated`: Types with bean defining annotation
  - `none`: Disable CDI
- `@Vetoed` for programmatic disablement of classes
- Global ordering/priority of interceptors and decorators

Bean Validation 1.1 (JSR 349)

- Alignment with Dependency Injection
- Method-level validation
  - Constraints on parameters and return values
  - Check pre-/post-conditions
- Integration with JAX-RS

Bean Validation Sample

```
public void placeOrder(
  @NotNull String productName, (1)
  @NotNull @Max("10") Integer quantity, (1)
  @Customer String customer) { (2)
  //. . .
}
```

(1) Built-in constraints on method parameters

(2) Custom constraints on method parameters

```
@Future (1)
public Date getAppointment() {
  //. . .
}
```

(1) Built-constraint on return value

Java Persistence API 2.1 (JSR 338)

- Schema Generation
  - `javax.persistence.schema-generation.*` properties
- Unsynchronized Persistence Contexts
- Bulk update/delete using Criteria
- User-defined functions using FUNCTION
- Stored Procedure Query

Servlet 3.1 (JSR 340)

- **Non-blocking I/O**
  - `ReadListener` and `WriteListener`
- **Protocol Upgrade**
- **Security Enhancements**
  - `<deny-uncovered-http-methods>` Deny request to HTTP methods not explicitly covered

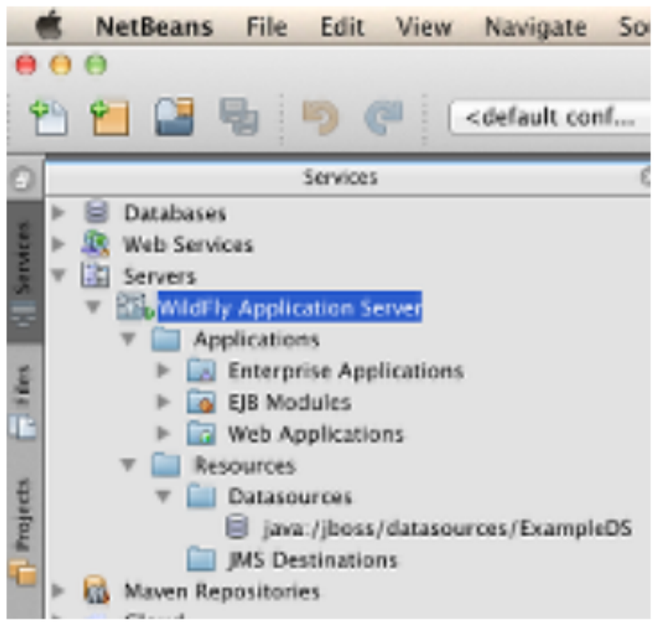
Java Server Faces 2.2 (JSR 344)

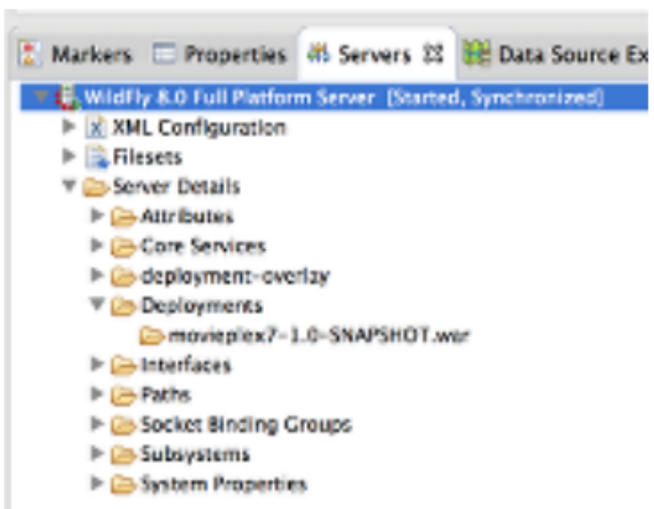
- Faces Flow
- Resource Library Contracts
- HTML5 Friendly Markup Support
  - Pass through attributes and elements
- Cross Site Request Forgery Protection
- Loading Facelets via ResourceHandler
- `h:inputFile` New File Upload Component

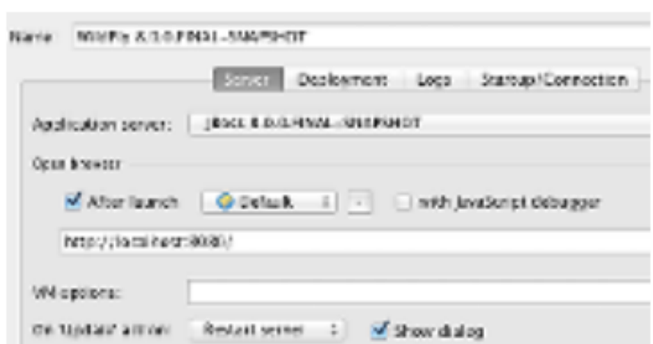
Java Transaction API 1.2 (JSR 907)

- `@Transactional` Define transaction boundaries on CDI managed beans
- `@TransactionScoped` CDI scope for bean instances scoped to the active JTA transaction

Java EE 7 IDEs







References

- Java EE 7 samples - https://github.com/javaee-samples/javaee7-samples
- Reference Implementaion: GlassFish - http://glassfish.org, @glassfish
- WildFly 8 - http://wildfly.org, http://github.com/wildfly, @WildFlyAS
- Slides generated with Asciidoctor and DZSlides backend
- Original slide template - Dan Allen & Sarah White

Arun Gupta

@arungupta