Java EE の概要と特徴

OSSユーザーのための勉強会<OSS X Users Meeting>
#21 Java EE

Akihiro Nishikawa
Oracle Corporation Japan

December 12, 2017

JavaYourNext (Cloud)

Safe Harbor Statement

The following 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.

Program Agenda

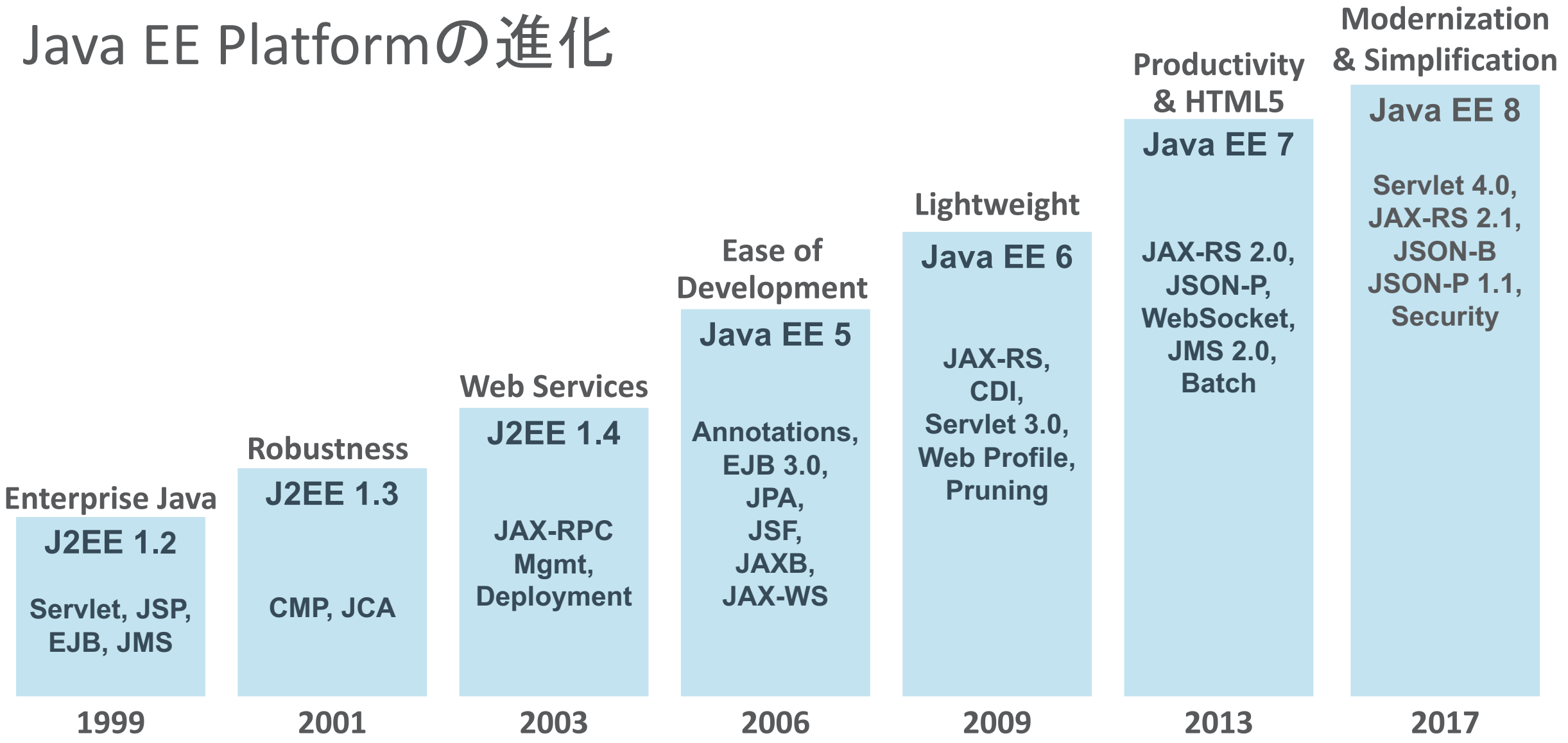
1. Java Enterprise Edition (Java EE)
2. The Road to Java EE 8
3. Java EE 8 Contents : JSRs and MRs
4. Summary

Java Enterprise Edition (Java EE)

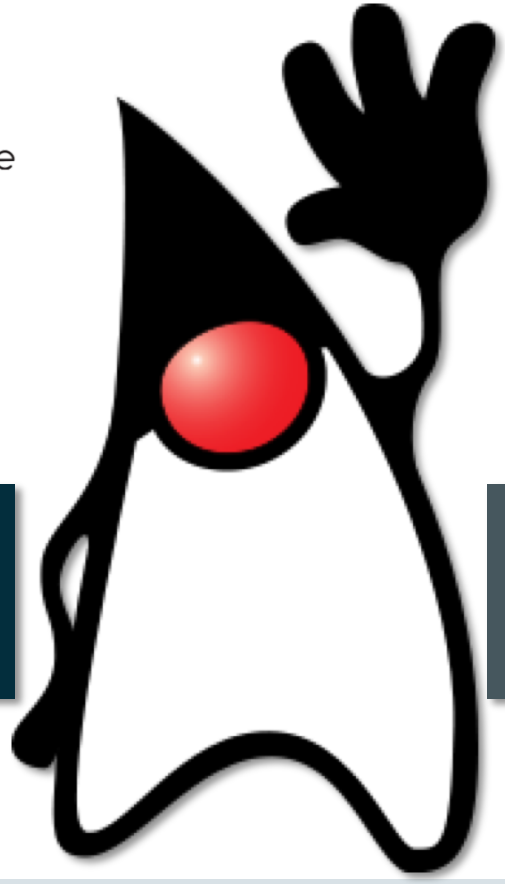
JavaYourNext (Cloud)

Java EE is a set of specifications

Java EE Platformの進化



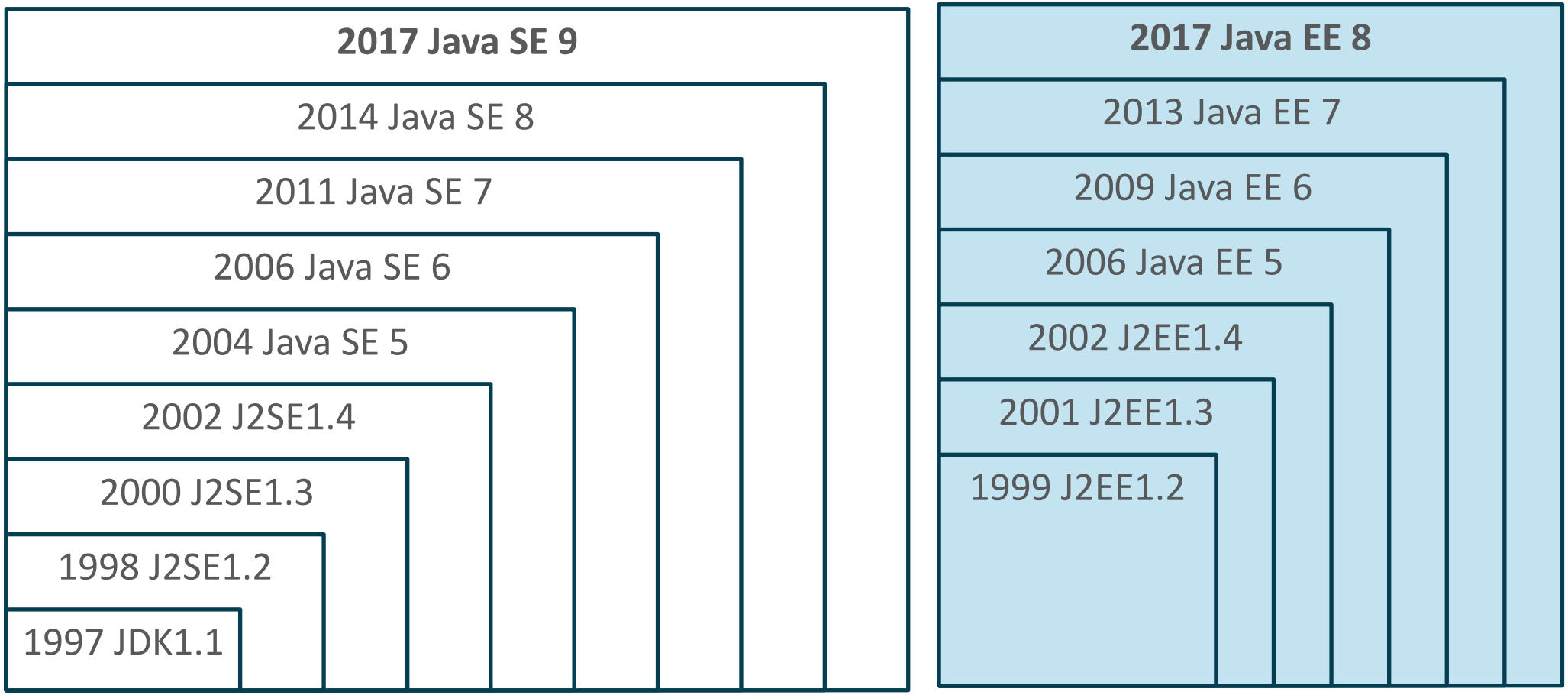
Java EE APIs - Backbone of Leading Open Source Projects



Web Frameworks	REST	Microservices	Web Containers	Java EE Containers	PaaS
PrimeFaces, Vaadin	Jersey, RESTEasy	MicroProfile, Spring Boot	Jetty, Resin, Tomcat	WildFly, GlassFish	Oracle Cloud, OpenShift by Red Hat, Cloud Foundry

後方互換性

上位バージョンは過去のバージョンを内包



仕様策定の透明性とロードマップ開示

OpenJDK

Java Community Process

• Java仕様の提案と策定

ORACLE

•標準Java VMの提供
•標準 Java EE Application Serverの提供
（Reference Implementation）

Java

コミュニティによる議論から仕様策定まで全て公開

世界中の企業、個人が
Javaベースのシステムや
製品を活用

世界中のユーザーが
システム構築に利用

世界中のITベンダーが
仕様に基づき製品開発

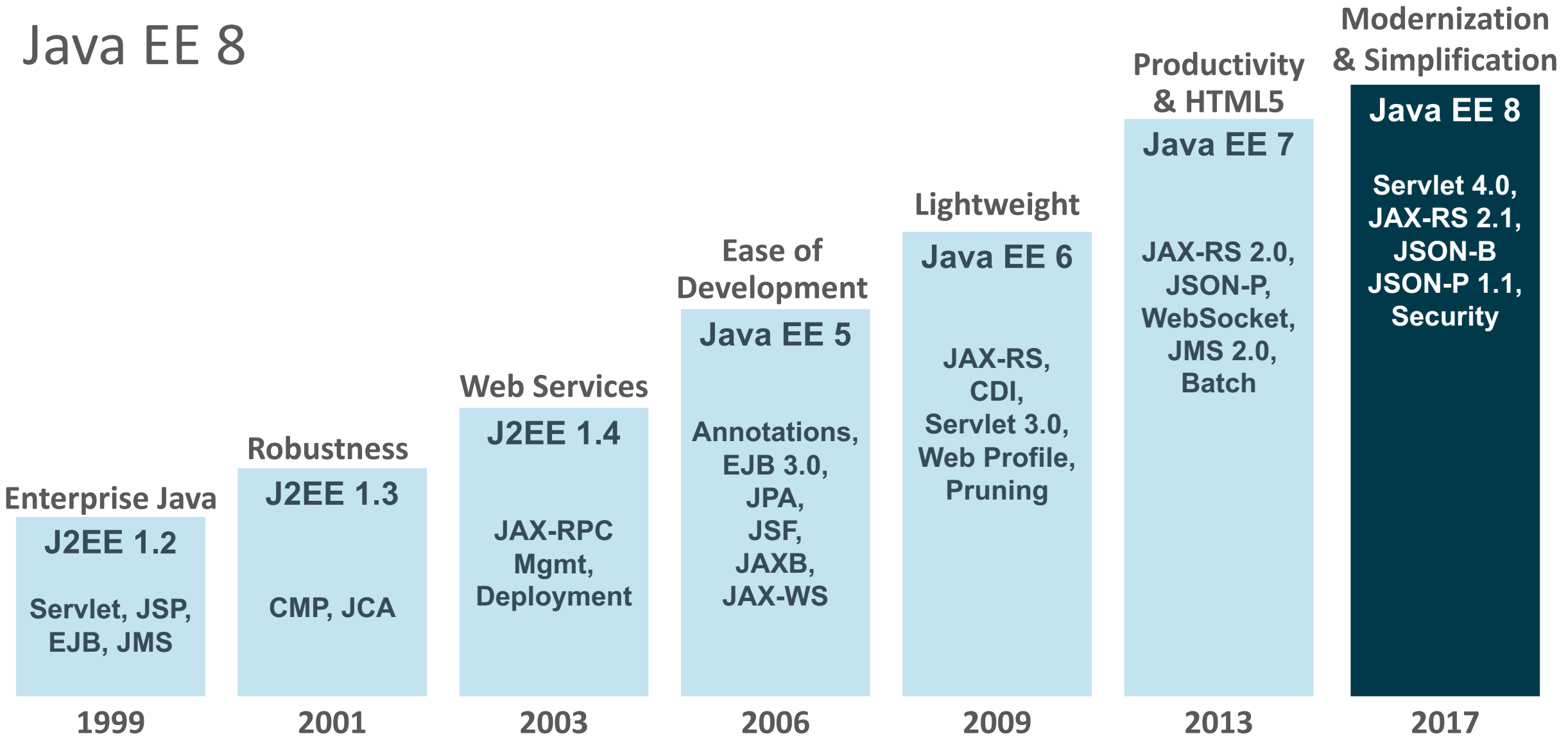
ORACLE 12c
FUSION MIDDLEWARE
WEBLOGIC SERVER

Java ORACLE

The Road to Java EE 8

JavaYourNext (Cloud)

Java EE 8



Java EE 8

- 2014: 仕様策定開始

  …

- 2016: 開発に注力

  …

- 2017.9.21 : Java EE 8リリース

Java EE 8

2014年当初計画されていたJSR

- Java EE 8 Platform and Web Profile
- Contexts and Dependency Injection 2.0 (CDI)
- Java API for JSON Binding 1.0 (JSON-B)
- Java Message Service 2.1 (JMS)
- Java Servlet 4.0
- Java API for RESTful Web Services 2.1 (JAX-RS)
- Model-View-Controller 1.0 (MVC)
- JavaServer Faces 2.3 (JSF)
- Java EE Management API 2.0
- Java API for JSON Processing 1.1 (JSON-P)
- Java EE Security API 1.0
- Bean Validation 2.0

Java EE Community Survey

Java One 2016後にアンケート

- どのテクノロジーをJava EE 8に取り込むべきか、コミュニティに対しアンケートを実施
- 当初のJava EE 8の提案と比較
- その結果
  - 優先度高：Servlet、REST、JSON
  - 早期のリリースを求める
  - 優先度低：Management、JMS、MVC

当初の Java EE 8提案	調査対象	調査結果	改訂後の Java EE 8提案
JAX-RS 2.1	REST Services	◎	そのまま
Servlet 4.0	HTTP/2	◎	
JSON-B 1.0	JSON-B	◎	
JSON-P 1.1	JSON-P	○	
CDI 2.0	調査対象外	N/A	
Bean Validation 2.0	調査対象外	N/A	
JSF 2.3	調査対象外	N/A	
Security 1.0	調査対象外	N/A	
Management 2.0	Management	△	Management 2.0は取下げ
JMS 2.1	JMS	△	JMS 2.1は取下げ
MVC 1.0	MVC	△	取下げ

Java EE 8

2016年のアンケートに基づいて変更された提案

- Java EE 8 Platform and Web Profile
- Contexts and Dependency Injection 2.0 (CDI)
- Java API for JSON Binding 1.0 (JSON-B)
- Java Message Service 2.1 (JMS)
- Java Servlet 4.0
- Java API for RESTful Web Services 2.1 (JAX-RS)
- Model-View-Controller 1.0 (MVC)
- JavaServer Faces 2.3 (JSF)
- Java EE Management API 2.0
- Java API for JSON Processing 1.1 (JSON-P)
- Java EE Security API 1.0
- Bean Validation 2.0

Java EE 8 Contents: JSRs and MRs

Java EE 8での注力ポイント

Web Tier技術

機能強化
HTTP/2のサポート

開発生産性

かんたん開発
CDI利用領域の拡大

セキュリティ

標準化
モダナイズ

Java EE 8 API Updates

JSR 374	**JSON-P 1.1**	JSON標準への対応、新機能のサポート
JSR 367	**JSON-B 1.0**	JSON <-> Javaバインディング、デフォルトマッピングなど
JSR 370	**JAX-RS 2.1**	Reactive クライアントAPI, Server-sent events
JSR 365	**CDI 2.0**	非同期イベント、イベントの優先度、Java SEのサポート
JSR 369	**Servlet 4.0**	HTTP/2 performance – 多重化、server push
JSR 372	**JSF 2.3**	CDI、WebSocket、Bean Validationとの統合、Java SE 8対応
JSR 380	**Bean Validation 2.0**	Java SE 8 (date/time, collections) 対応、新たな制約
JSR 375	**Security 1.0**	認証、アイデンティティストア、セキュリティコンテキスト

Maintenance Releases

JSR 338	Java Persistence 2.2
JSR 919	JavaMail 1.6
JSR 250	Common Annotations 1.3
JSR 318	Interceptors 1.2 rev A
JSR 356	WebSocket 1.1

For Java SE 9
(GlassFish 5.0に包含)

JSR 224	JAX-WS 2.0
JSR 67	SAAJ 1.0
JSR 222	JAXB 2.0
JSR 925	JAF 1.2

JSON-P

JSON-P 1.1

JSON-P仕様を最新の標準に追随するためのアップデート

- 対応する標準
  - RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format
  - RFC 6901 – JSON Pointer
  - RFC 6902 – JSON Patch
  - RFC 7396 – JSON Merge Patch
- `JsonObject` と `JsonArray` に編集・変換操作を追加
- Java SE 8で導入されたStreams API活用のためのヘルパークラスを追加
  - `JSONCollectors`

JSON-Pointer

IETF RFC 6901

- JSON値を参照するための文字列構文

  例）"/0/user/address"

- JsonPointerのメソッド
  - getValue()
  - add()
  - remove()
  - replace()
  - containsValue()

値の取得

```
JsonArray contacts = ...
JsonPointer p =
        Json.createPointer("/0/phones/mobile");

JsonValue v = p.getValue(contacts);
```

```
[
   {
      "name":"Duke",
      "gender":"M",
      "phones":{
         "home":"650-123-4567",
         "mobile":"650-234-5678"}},
   {
      "name":"Jane",
      "gender":"F",
      "phones":{
         "mobile":"707-555-9999}}
]
```

値の置換

```
JsonArray contacts = ...
JsonPointer p =
        Json.createPointer("/0/phones/mobile");
JsonReader reader =
        Json.createReader(
            new StringReader("¥"650-555-1212¥""));
JsonValue jsonValue = reader.readValue();
contacts = p.replace(contacts, jsonValue);
```

```
[
   {
     "name":"Duke",
     "gender":"M",
     "phones":{
        "home":"650-123-4567",
        "mobile":"650-555-1212"}},
   {
     "name":"Jane",
     "gender":"F",
     "phones":{
        "mobile":"707-555-9999}}
]
```

JSON-Patch

IETF RFC 6902

- JSONドキュメントに適用する一連の操作を記述
  - "add", "remove", "replace", "move", "copy", "test"
- JsonPatchBuilderメソッド
  - add, copy, move, remove, replace, test
- JsonPatchメソッド
  - apply()
  - toJsonArray()

JSONドキュメントの操作

```
JsonPatchBuilder builder =
        Json.createPatchBuilder();

JsonPatch patch =
        builder.replace("0/phones/mobile",
                "650-111-2222")
                .remove("/1").build();

JsonArray result = patch.apply(contacts);
```

```
[
   {
      "name":"Duke",
      "gender":"M",
      "phones":{
         "home":"650-123-4567",
         "mobile":"650-111-2222"}},
   {
      "name":"Jane",
      "gender":"F",
      "phones":{
         "mobile":"707-555-9999}}
]
```

JSON-Merge Patch

IETF RFC 7396

Operation	Original	Patch	Result
置換（Replace）	{"a" : "b"}	{"a" : "c"}	{"a" : "c"}
	{"a" : "b"}	{"a" : null}	{}
追加（Add）	{"a" : "b"}	{"b" : "c"}	{"a" : "b", "b" : "c" }
削除（Remove）	{"a" : "b", "b" : "c" }	{"a" : null}	{"b" : "c"}

MergePatchでの操作

Patchの適用

```
JsonValue source = Json.createValue("{¥"color¥":¥"blue¥"}");

JsonValue patch = Json.createValue("{¥"color¥":¥"red¥"}");

JsonMergePatch mergePatch = Json.createMergePatch(patch);

JsonValue result = mergePatch.apply(source);
// {"color":"red"}
```

MergePatchでの操作

差分の取得

```
JsonValue source = Json.createValue("{¥"color¥":¥"red¥"}");

JsonValue target = Json.createValue("{¥"color¥":¥"blue¥"}");

JsonMergePatch mergePatch = Json.createMergeDiff(source, target);
// {"color":"blue"}
```

JsonCollectors

従来、Lambda式を使ってJSONのQueryをする場合、JSONから一旦離れる必要があった

```
JsonArray contacts = ...

List<String> femaleNames =
        contacts.getValuesAs(JsonObject.class).stream()
                .filter(x -> "F".equals(x.getString("gender")))
                .map(x -> (x.getString("name")))
                .collect(Collectors.toList());
```

JsonCollectors

JsonCollectorsを使うと、JSONの世界から出る必要がなくなる

```
JsonArray contacts = ...;

JsonArray femaleNames =
        contacts.getValuesAs(JsonObject.class).stream()
                .filter(x -> "F".equals(x.getString("gender")))
                .map(x -> (x.get("name")))
                .collect(JsonCollectors.toJsonArray());
```

JSON-B

JSON-B 1.0

Java API for JSON Binding

- JavaオブジェクトとJSONドキュメントをマーシャル/アンマーシャルするAPI
  - マーシャル : Javaオブジェクト → JSONドキュメント
  - アンマーシャル : JSONドキュメント → Javaオブジェクト

- APIのカスタマイズ
  - Annotationの利用 (`@JsonbProperty` と `@JsonbNillable`)
  - Runtime configuration builderの利用

- JSON Bindingプロバイダを変更可能

JSON-B 1.0

Java API for JSON Binding

- `JsonBuilder`
  - JSONバインディングAPIのためのクライアントのエンドポイント
  - プロバイダ実装を選択可能（構成プロパティを設定）

- `Jsonb`
  - JSONバインディングフレームワーク操作を抽象化
  - `fromJson`: JSONを読み取り、Javaオブジェクト・コンテンツ・ツリーにデシリアライズ
  - `toJson`: Javaオブジェクト・コンテンツ・ツリーをJSONテキストにシリアライズ

JSON-B 1.0

```
Car car1 = new Car();
car1.setBrand("Toyota");
car1.setModel("Prius");
car1.setStock(20);
Car car2 = new Car();
car2.setBrand("Tesla");

...
List<Car> inventory = new ArrayList<>();
inventory.add(car1);
inventory.add(car2);

Jsonb jsonb = JsonbBuilder.create();
String json = jsonb.toJson(inventory);
```

```
[
  {
    "brand":"Toyota",
    "model":"Prius",
    "stock":20
  },
  {
    "brand":"Tesla",
    "model":"Model S",
    "stock":0
  }
]
```

アノテーションを使ったカスタマイズ

```
public class Customer {
    private int id;

    @JsonbProperty(“name”)
    private String firstName;

    private String getFirstName() {
        return firstName;
    }
}
```

```
@JsonbNillable
@JsonbPropertyOrder(PropertyOrderStrat
gy.REVERSE)
public class Customer {
    @JsonbNumberFormat("#0")
    private int id;

    @JsonbProperty(“name”)
    private String firstName;

    private String getFirstName() {
        return firstName;
    }
}
```

カスタマイズ

- プロパティの命名
- プロパティの順序
- 無視すべきプロパティ
- Nullの取り扱い
- インスタンス生成のカスタマイズ
- フィールドやプロパティの可視性
- 日付や数値のフォーマット
- Encoding形式
- Adapters

…

カスタマイズ

Runtime configuration builder

```
Jsonb jsonb = JsonbBuilder.create();

//Ordering, naming strategy, encoding, Locale, ...
JsonbConfig config = new JsonbConfig().withFormatting(true)
                                      .withAdapters(new CarAdapter());

Jsonb jsonb = JsonbBuilder.newBuilder("myProvider");

Jsonb jsonb = JsonbBuilder.create(config);
```

JAX-RS

JAX-RS 2.1

- Reactive Client API
- Server-sent events
- Hypermedia API enhancements
- 他のJSRやフレームワークとの統合
  - JSON-Bエンティティプロバイダのサポート
  - HTTP PATCHリクエストのサポート

JAX-RS 2.0のClient API（同期）

```
// http://example.com/api/read/doe?dpt=1
        WebTarget myResource = client.target("http://example.com/api/read")
                .register(SomeFilter.class)
                .path("{user}")
                .resolveTemplate("user", "joe")
                .queryParam("dpt", "1")
                .header("some-header", "true");
        Response response = myResource.request(...).get();

// ...
        client.close();
```

JAX-RS 2.0のClient API（非同期）

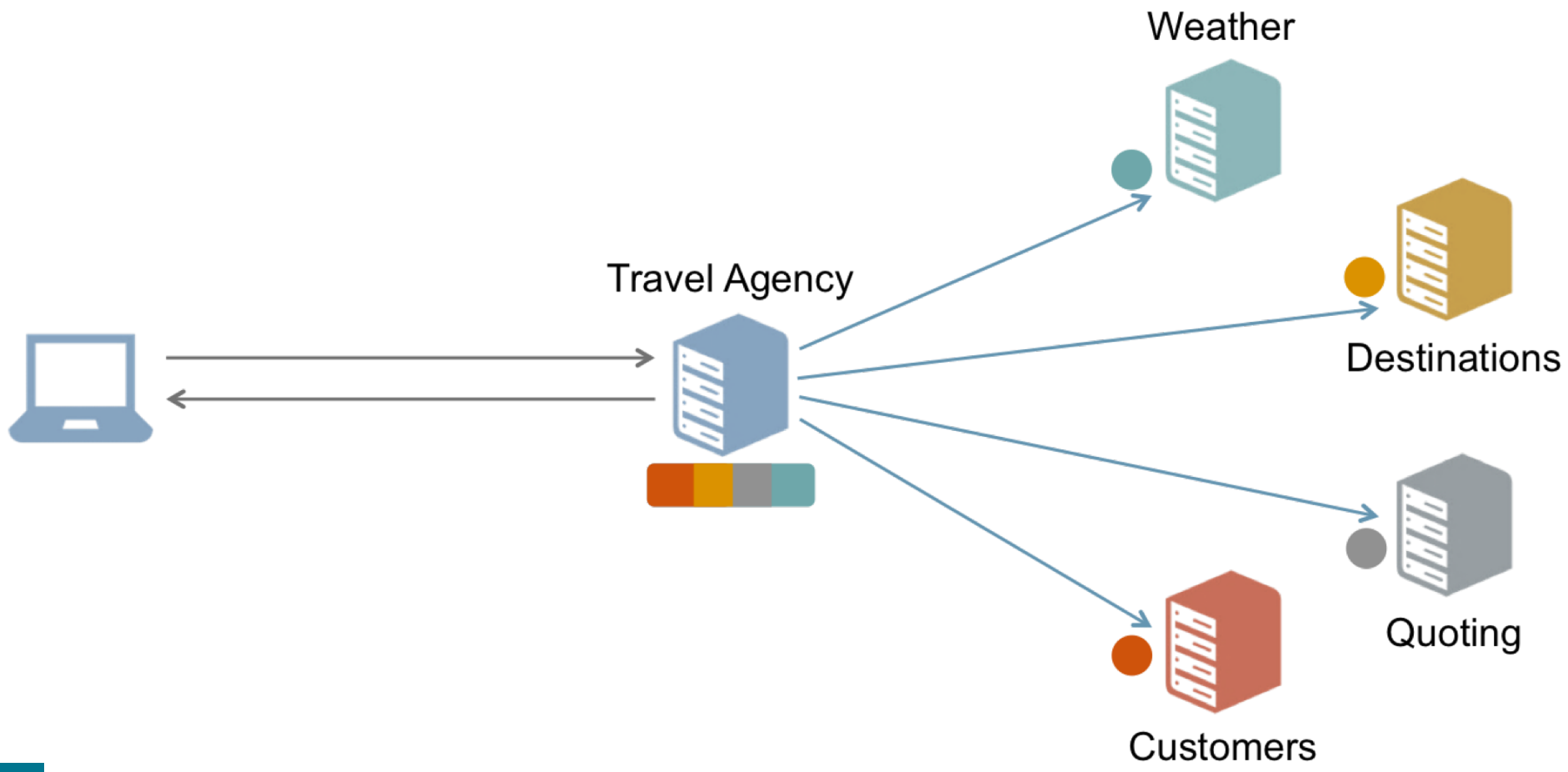
```
Client client = ClientBuilder.newClient();

WebTarget myResource = client.target("http://example.com/api/read");

Future<String> response = myResource.request(MediaType.TEXT_PLAIN)
      .async()
      .get(String.class);
// ...
```

JAX-RS 2.0のClient API（非同期）

```
Client client = ClientBuilder.newClient();
WebTarget myResource = client.target("http://example.com/api/read");
Future<Customer> fCustomer = myResource.request(MediaType.TEXT_PLAIN)
        .async()
        .get(new InvocationCallback<Customer>(){
            @Override
            public void completed(Customer customer) {
                // work on the customer
            }
            @Override
            public void failed(Throwable throwable) {
                // Oops!
            }
        });
```

JAX-RS 2.0

下図のようなシナリオの場合、Orchestrationの実装が必要だが・・・



JAX-RS 2.0で実装した場合 (1/4)

```
destination.path("recommended")
    .request()
    .header("Rx-User", "Async")
    .async()
    .get(new InvocationCallback<List<Destination>>() {

        @Override
        public void completed(final List<Destination> recommended) {
            final CountDownLatch innerLatch =
                    new CountDownLatch(recommended.size());

            final Map<String, Forecast> forecasts =
                    Collections.synchronizedMap(new HashMap<>());
           ...
```

JAX-RS 2.0で実装した場合 (2/4)

```
            for (final Destination dest : recommended) {
                forecasts.resolveTemplate("dest", dest.getDestination())
                        .request()
                        .async()
                        .get(new InvocationCallback<Forecast>() {
                            @Override
                            public void completed(final Forecast forecast) {
                                forecasts.put(dest.getDestination(), forecast);
                                innerLatch.countDown();
                            }

...
```

JAX-RS 2.0で実装した場合 (3/4)

```
                    @Override
                    public void failed(final Throwable throwable) {
                        innerLatch.countDown();
                    }
                });
        }

        try {
            if (!innerLatch.await(10, TimeUnit.SECONDS)) {
                // timeout
            }
        } catch (final InterruptedException e) {
            // Ooops, interrupted!
        }
...
```

JAX-RS 2.0で実装した場合 (4/4)

```
            // Continue with processing…
            }

            @Override
            public void failed(final Throwable throwable) {
                // Recommendation error
            }
        });

// Continue...
```

JAX-RS 2.1

Reactive Client API

```
CompletionStage<String> cs1 = ClientBuilder.newClient()
        .target("http://example.com/api")
        .request()
        .rx()
        .get(String.class);
cs1.thenAccept(System.out::println);
```

CompletionStage APIを使う

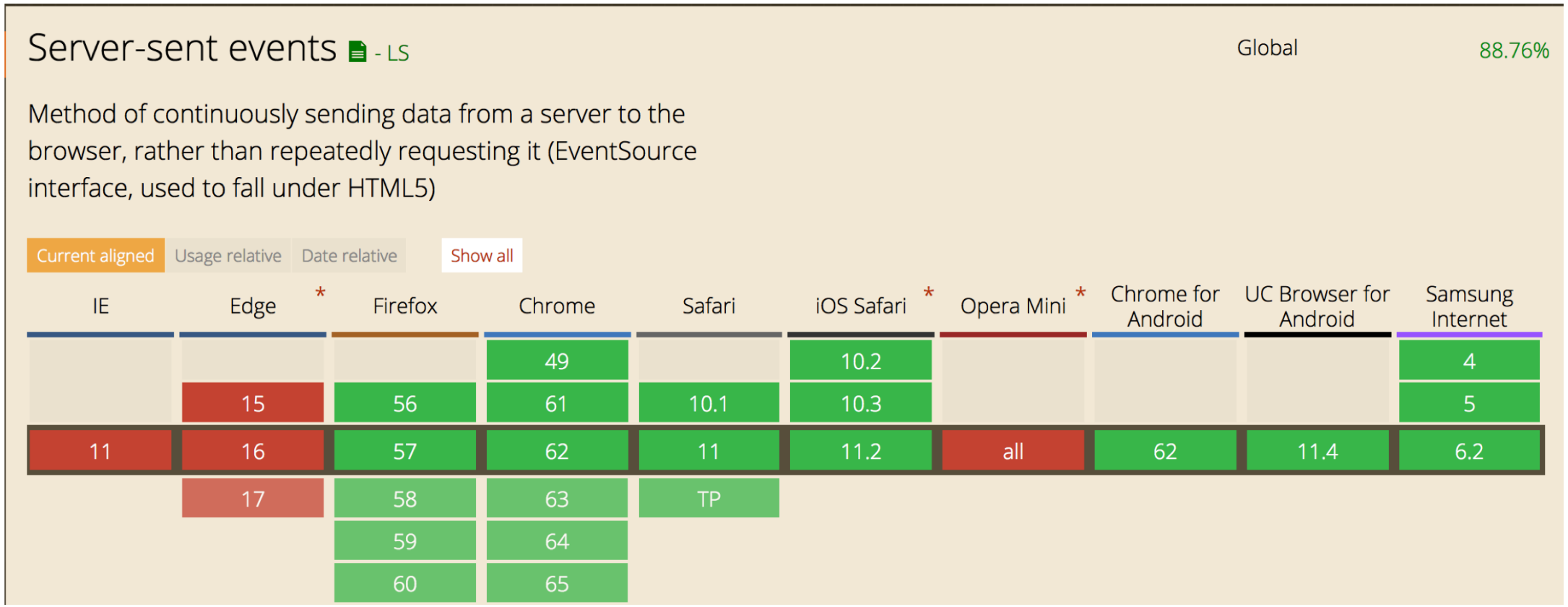
```
CompletionStage<String> cs1 = ClientBuilder.newClient()
        .target("http://example.com/service1/hello")
        .request()
        .rx()
        .get(String.class);
CompletionStage<String> cs2 = ClientBuilder.newClient()
        .target("http://example.com/service2/bonjour")
        .request()
        .rx()
        .get(String.class);
CompletionStage<String> concat = cs1.thenCombine(cs2, String::concat);
concat.thenAccept(System.out::println);
```

Server-Sent Events (SSE)

- サーバからクライアントへの一方向の永続チャネル
  - サーバから複数のイベント(メッセージ)を送信可
- HTTPプロトコルベース
  - メディアタイプ: "text/event-stream"
- イベントの構造
  - イベント名、データ、ID、リトライ、コメントのフィールド
  - イベントペイロードはデータフィールドに含まれる

Server-Sent Events (SSE) のサポート状況

Source : https://caniuse.com/



SSE Client API

- `SseEventSource`
  - Webターゲットへの接続を開く
  - イベントConsumerをサブスクライブ
  - 接続を閉じる

- `InboundSseEvent`
  - イベントデータの読み取り
  - その他のイベントフィールドの読み取り

SSE Client API

```
Client client = ClientBuilder.newClient();
WebTarget target = client.target( "http://example.com/service/subscribe" );
try( SseEventSource eventSource
             = SseEventSource.target( target )
                             .reconnectingEvery ( 5, TimeUnit.SECONDS )
                             .build () ) {
    eventSource.register( System.out::println );

    // ...
    eventSource.open();

    // ...
    eventSource.close();
}
```

SSE Server API

- SseEventSink
  - リソースメソッド・パラメータとして注入
  - 一つのクライアントHTTP接続に対応するインスタンス
  - クライアントにイベント（OutboundSseEvent）送信時に利用
- Sse
  - OutboundSseEventとSseBroadcasterを生成するために利用
- OutboundSseEvent
  - イベントを1個のクライアントもしくはブロードキャストする際に利用
- SseBroadcaster
  - イベントをブロードキャストする場合に利用

SSE Server API

```
@GET
@Path("subscribe")
@Produces(MediaType.SERVER_SENT_EVENTS)
public void subscribe(@Context SseEventSink eventSink, @Context Sse sse) {
    eventSink.send (
        sse.newEventBuilder ()
           .name ( "event-name" )
           .data ( String.class, "Welcome!" )
           .build () );

    eventSink.send( sse.newEvent ( "an event" ) );
    eventSink.send( sse.newEvent ( "another event" ) );
    eventSink.close ();
}
```

CDI

CDI 2.0

- 仕様を3分割
  - CDI Core
  - CDI for Java SE
    - Java SEで利用するためのCDIコンテナ起動API
  - CDI for Java EE
- Observerの順序制御
- 非同期イベント
- 組込みアノテーション・リテラルの追加

- Java SE 8の新機能（Stream API、Lambda式など）に対応
- 主要なSPI要素に対するConfigurator
- Observerメソッドの構成および廃棄
- ProducerへInterceptorを適用可能

CDI 1.1の場合

順序不定、同期イベントのみ、イベントがImmutableではなかった

```
@Inject
private Event<PaymentEvent> paymentEvent;

// event producer
paymentEvent.fire(new PaymentEvent(amt));

// event consumer A
public void aObserver(@Observes PaymentEvent p) {
    // ...
}
// event consumer B
public void bObserver(@Observes PaymentEvent p) {
    // ...
}
```

CDI 2.0

優先度を指定できる

```
@Inject
private Event<PaymentEvent> paymentEvent;

// event producer
paymentEvent.fire(new PaymentEvent(amt));

// event consumer A
public void aObserver(@Observes @Priority(10) PaymentEvent p) {
    // ...
}
// event consumer B
public void bObserver(@Observes @Priority(20) PaymentEvent p) {
    // ...
}
```

CDI 2.0

非同期イベント

```
@Inject
private Event<PaymentEvent> paymentEvent;

// event producer
CompletionStage<PaymentEvent> stage
        = paymentEvent.fireAsync(new PaymentEvent(amt));

// event consumer A
public void aObserver(@ObservesAsync PaymentEvent p) {
    // ...
}
// event consumer B
public void bObserver(@ObservesAsync PaymentEvent p) {
    // ...
}
```

EventとObserver

- 同期・非同期で同じ種類のEventを送出できる
  - `paymentEvent.fire(new PaymentEvent(100));`
  - `paymentEvent.fireAsync(new PaymentEvent(200));`
- 同期Eventは同期Observerでのみ、非同期Eventは非同期Observerでのみ取り扱い可能
- 非同期Observerの場合、新たなライフサイクル・コンテキストとトランザクション・コンテキストで呼ばれる

Web Tier

Servlet, HTTP/2, and JSF

HTTP/2

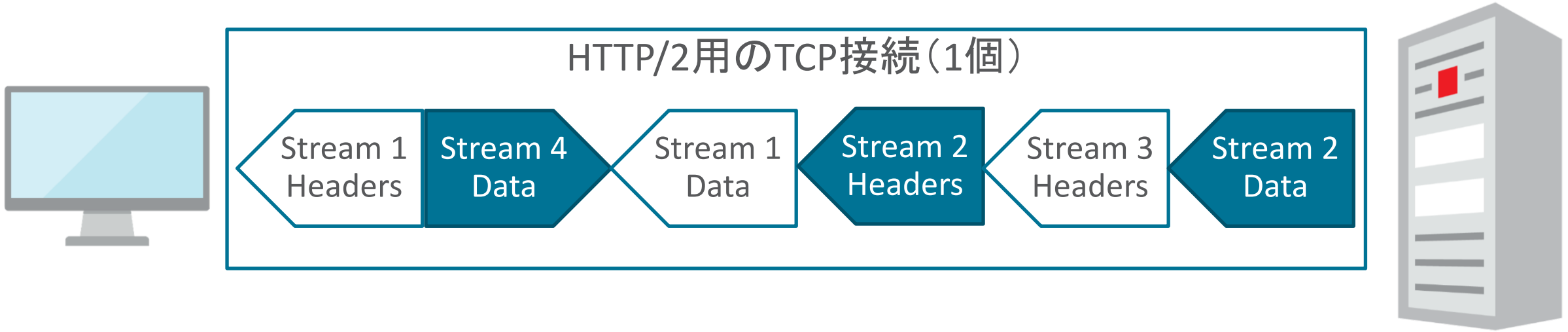
HTTP 1.xの制限を解決するために

- レイテンシの削減
- （複数接続をしなくても）並行処理をサポート
- 「ヘッドオブラインブロッキング」（Head-of-Line Blocking）の回避
- HTTP 1.1のセマンティクスはそのまま利用可

HTTP/2

機能・特長

- 1接続でリクエスト・レスポンスを多重化できる
  - 完全双方向
  - 複数のストリーム
- バイナリフレーミング
- Streamの優先度付け
- サーバー・プッシュ
- ヘッダ圧縮
- HTTP 1.1からのアップグレード



Servlet 4.0

- HTTP/2のサポート
  - リクエスト・レスポンスの多重化
  - サーバー・プッシュ
  - HTTP 1.1からのアップグレード
- HTTP 1.1 RFCに対する互換性を確保
- コミュニティからの要望に基づいた改良
  - コンテナ固有の設定に頼らずにデフォルトのコンテキスト・パス設定を許可
  - プログラムによるjspファイルの設定の許可
  - デプロイメント・ディスクリプタで設定したエンコーディングの許可
  - Servlet Mapping API

Server Push

```
...
PushBuilder pushBuilder = request.newPushBuilder();
pushBuilder.path( "images/myPhoto.png" )
           .addHeader( "content-type", "image/png" )
           .push ();
```

URLマッピングのランタイムディスカバリ

HttpServletMapping

- getMappingMatch()
  - MappingMatchのタイプ
- getPattern()
  - Servletリクエストをアクティブにするurlパターン
- getMatchValue()
  - このリクエストの元になったURIパスの一部分を返す
- getServletName()
  - Servlet名（完全修飾名）

```
@WebServlet(urlPatterns = {"", "/", "/main", "*.html", "/main/*"},
            name = "MainServlet")

public class MainServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;
  static final Logger logger = Logger.getLogger(MainServlet.class.getName());

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) {

    HttpServletMapping httpServletMapping = req.getHttpServletMapping();
    MappingMatch mappingMatch = httpServletMapping.getMappingMatch();

    ...
```

Matchingの結果

getServletNameは常にMainServlet

	getMappingMatch	getMatchValue	getPattern
""	CONTEXT_ROOT	""	""
"/"	DEFAULT		/
"/main"	EXACT	main	/main
"/index.html"	EXTENSION	index	*.html
"/main/hello"	PATH	hello	/main/*

JSF 2.3

- CDIとの統合が進化
  - より多くのものが注入可能
  - 以前のmanaged beansは非推奨
- Date and Time APIのサポート
- WebSocketとの統合
- Ajaxメソッドの呼び出し
- クラスレベルのBean Validation
- UIDataとUIRepeatの改善

Bean Validation

Bean Validation 2.0

- Java SE 8の新機能をサポート
  - Date and Time APIのサポート
  - Collection要素に適用される制約
  - Optionalラッパー
  - アノテーションの重複指定
- 新たな制約の追加
  - `@NotEmpty, @NotBlank, @Email, @Positive, @Negative, @PositiveOrZero, @NegativeOrZero, @PastOrPresent, @FutureOrPresent`
- その他コミュニティからの追加機能リクエストに対応

Constraints (1/2)

```
@Past
Year startYear = Year.of(2016);

// repeating annotations
@Size(min = 8, group = Default.class)
@Size(min = 12, group = Admin.class)
private String password;

List<@NotNull @Email String> emails;

String @NotNull @Email[] email;

// constraints with container elements
Map<@Valid Customer, @Valid Account> customerAccountInfo;
```

Constraints (2/2)

```
// Optional
Optional<@Past LocalDate> getEnrollmentDate();

// new Annotations
@NotEmpty
List<@NotBlank @Email String> customerEmails;
```

Security

Security API for Java EE

Java EEセキュリティAPIの近代化および簡素化

- 新規APIの追加
  - SecurityContext
  - HttpAuthenticationMechanism
  - IdentityStore
- 共通のアプローチやメカニズムを使い、アプリケーションがアクセス可能な認証メカニズムを簡素化
- CDIの活用

SecurityContext

プログラムでセキュリティ機構にアクセスする際のエントリポイント

- 認証のためのメソッド
  - authenticate()
  - getCallerPrincipal(), getPrincipalsByType()
  - isCallerinRole()
  - hasAccessToWebResource()
- 以下のメソッドの置き換え
  - HttpServletRequest.getUserPrincipal()
    HttpServletRequest.isUserInRole()
  - EJBContext.getCallerPrincipal()
    EJBContext.isCallerInRole()

HttpAuthenticationMechanism

Servletコンテナの認証をサポート

- WebアプリケーションのCallerの認証に使用
  - Servletコンテナでの利用のみ想定
- JASPIC ServerAuthModule SPIをモデルにしたInterface
  - validateRequest() : doFilter()もしくはservice()メソッド呼び出し**前**
  - secureResponse() : doFilter()もしくはservice()メソッド呼び出し**後**
  - cleanSubject() : logout() メソッドの呼び出しに対応して呼び出す
- アプリケーションにパッケージングもしくはコンテナが提供し、コンテナもしくはアプリケーションから呼び出す
- IdentityStoreに委譲可能

IdentityStore

JAASログインモジュールの標準化

- ユーザーストアと対話してユーザーを認証、ユーザーグループ情報を取得するために利用
- タイプ
  - LDAP、Database
  - Authentication、Authorization、両方
- メソッド
  - 認証：`validate(Credential)` - `CredentialValidationResult`を返す
  - 認可：`getCallerGroups(CredentialValidationResult)`

メンテナンスリリース

Java Persistence 2.2

- 重複アノテーション
- Date and Time APIのサポート
  - java.timeタイプのLocalDate, LocalTime, LocalDateTime, OffsetTime, OffsetDateTimeへのマッピング
- クエリ実行結果をStream化
  - Query : Stream getResultStream()
  - TypedQuery : Stream<x> getResultStream()
- AttributeConverterでCDIを利用可

Summary

JavaYourNext (Cloud)

This organization | Search | Pull requests | Issues | Marketplace | Explore

Java EE

Java Enterprise Edition

http://oracle.com/javaee

Java EE Development has been migrated from Java.net to GitHub

Repositories 120 | People 55

Pinned repositories

https://javaee.github.io/

https://github.com/javaee

glassfish

The Open Source Java EE Reference Implementation

Servlet Specification

grizzly

Writing scalable
programming lan
Before the adven
thread managem
...

Java EE 8 Full Profile

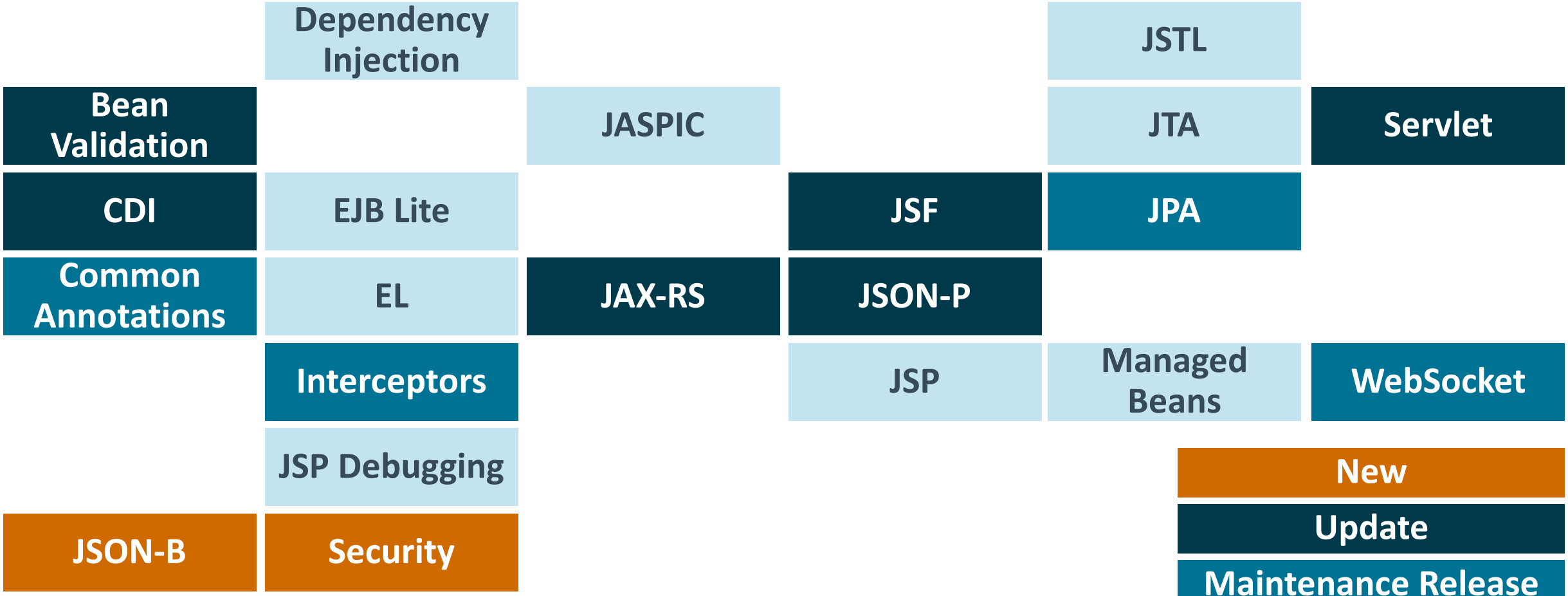
Batch	Dependency Injection	JACC	JAXR	JSTL	Management
Bean Validation	Deployment	JASPIC	JMS	JTA	Servlet
CDI	EJB	JAX-RPC	JSF	JPA	Web Services
Common Annotations	EL	JAX-RS	JSON-P	JavaMail	Web Services Metadata
Concurrency EE	Interceptors	JAX-WS	JSP	Managed Beans	WebSocket
Connector	JSP Debugging	JAXB			
JSON-B	Security				

New

Update

Maintenance Release

Java EE 8 Web Profile



Summary

- Final Java EE 8 was shipped!
- EE4J (Enterprise Eclipse for Java) project is now on going.
- Fore more details on EE4J, stay tuned for Ito-san's presentation!

Safe Harbor Statement

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.


ORACLE®


Java™
ORACLE®