المواضيع التي سيتم ذكرها في المحاضرة:

- JSP
- Client side & Server side
- Front end & Back end

❖مراجعة لما سبق:

- مشكلتنا الأساسية كانت كتابة برامج شبكية متعددة المستخدمين، نتج منها عدد من المشاكل المقترحة التي بحاجة إلى معالجة فكانت أهم المشاكل التقنية هي مشكلة إدارة الموارد، إدارة التنافس، الأمن، الأداء ...
- عند تخاطب البرمجيات مع بعضها احتاجت إلى فتح socket التي هي مورد كبير وغالي، وكلما توقف تحتاج إلى إعادة تشغيل والكثير من المشاكل التي نتج عنها مشكلة إدارة الموارد.
- لو تحدثنا عن إدارة الموارد في الـ Middleware لوجدنا أنّ الـ Socket كانت تخصص موارد لكل زبون على حدى، وهي بداية كانت تخصص مورد للجميع، لكن عندما وضعنا Thread + Socket أصبح لدينا مورد لكل زبون وهي كلفة عالية، كما أنّ الـ Socket لوحدها كلفة عالية، وقد أضاف إليها البطء لأنه كما تحدثنا أنّ فتح الـ Threadيحتاج 200 ms وهو زمن طويل جداً بالنسبة للحاسب، كما أنها تستهلك Stack حجمه 1MB، لذلك اقتُرحت الـ Thread Pooling فأصبح لدينا Socket Thread Pooling، لكن في حال موت أحد الـ Threads نواجه مشكلة من سيقوم بإنعاشه، نكون قد دخلنا في مشكلة Fault Tolerance، فأصبح لابد من إضافة Resource Manger:
- Socket + Thread Pool + Resource Manager نتجت مشكلة عدم وجود بروتوكول تخاطب

- RPC = Socket + Thread Pool + Resource Manager + Protocol

- مشكلة الفريق (مثلاً: المحلل لا يستطيع انتظار المبرمج ولا يعرف ما هي Procedures ليقوم بتقسيمها) فنشأت هذه المشكلة التي منعت من استخدام الـ RPC في المشاريع الكبيرة، ودعت إلى ظهور Object Oriented Middleware.
- OOM: تمتلك نفس مشكلة RPC بحيث نضطر لإضافة الخاصة ACID واستعمال تقنيات مثل CORBA التي هي صعبة الاستعمال، ثم ظهرت EJB التي تحقق RMI (Over IIOP) أي عملت خليط بين Java RMI وبين CORRA ، EJB : هي أحد مكونات J2EE
- J2EE تُعتبر Package كاملة فيها EJB الذي حلّ مجموعة من المشاكل مثل التخاطب بين 2Application ، وإدارة الموارد JMS : طريقة لتغلف الـ Queue؛ أي إرسال رسالة للـ Queue واستقبال رسالة منه دون معرفة من الـ Provider الخاص به
- إدارة الـ Threads: مثل ضبط دورة حياة الـ Thread، ودعم قضية OOP، ودعم خاصية ACID .
- J2EE حققت مجموعة من الادوات لكن من بداية دورة حياتها (خلال أقل من عام) ساء استعمالها
- يتطلب ذلك وضع مجموعة من التوصيات أي Design Pattern التي اقترحت مجموعة من الادوات مثل JDBC, JSP (للتخاطب مع الـ DB) لكن الناس لم تستوعبها بسهولة
- تم الانتقال إلى المستوى الثالث الذي يُسمّى بيئات العمل (Framework) التي هي على عكس قائمة الأدوات حيث هنا يتم الحصول على التوصيات وتوفير بيئة العمل المناسبة
- من أمثلة بيئات العمل : Spring ، وقد عملت Microsoft لوحدها NET Framework. الهدف منها تسهيل البرمجة.
- بيئات العمل منها : MDA (Model Driven) ومنها ما يحقق ذلك كأن يطلب إدخال XML تعبر عن الـ Classes , XML تعبر عما نريد طباعته في الواجهات ، XML يعبر عن خصائص الـ Enterprise Component المرادة وتقوم الـ Framework بتوليدها جميعها، كما فعلت EJB وذلك ضمن لغة البرمجة، أو كما يعمل الـ Compiler.
- في كل مرة نحاول الصعود إلى level أعلى حيث تخفّف عدد المدخلات التي يقوم بها المبرمج والإكثار من المدخلات المولّدة أوتوماتيكياً والتي تخضع لجميع التوصيات الممكنة.

JSP (Java Server Pages): (slide 87)

- يوجد ما يُسمّى Server Side و Client Side وبينهما مسافة
- متى نقول أنّ الكود هو Server Side ومتى يكون Client Side وما Dynamic Html ؟
- سنتعرف على مصطلحات جديدة منها Front-end, Back-end وهذه القضايا مرتبطة بمفهومين: Client - طبقة Presentation.

➢ ماهي المشكلة التي أدّت إلى استخدام JSP أو ASP أو PHP رغم وجود Html منذ القدم ؟!

نحتاج إلى شيء يولّد الـ Html وذلك في الـ server لأنَّ الـ Browser لا يعرف لغة Html ليكتبها، يتم توليدها في الـ Server Side عن طريق Program، على سبيل المثال servlet في الـ input الخاص به يأخذ الـ request output stream من الـ Client.

Server Page: هي عبارة Html تحتوي Tags و Code بوضعه ضمن Tag لتمييزه، ولها عدة أنواع:

JSP: هي Server page على جافا

ASP: هي server page على VB

PHP: هي server page على C++

- Server Page: حقيقة عندما يقوم الـ Compiler بالتشغيل يأخذ الـ Code ويقوم بتحويله إلى Servlet وإرساله عبر الـ Request Output Stream

- سمحت Server Pages بتوليد صفحات Html وفقاً لطلبات (رغبات) الزبون دون خلط java مع Html ضمن برنامج (أي تم تضمين Java في Html بدلاً من السابق حيث كانت Html في كود Java)

- اتّسمت Server Pages بتكرار التخاطب بين الزبون والمخدّم، من أجل تعديل بسيط وهذا يستهلك Bandwidth لدينا

- فظهر عندها مفهوم Dynamic Html وهو أن نستطيع تغيير جزء من الـ Html

- سمحت Dynamic Html بتعديل الـ HTML من طرف الزبون بحيث يتمكن من تحديث جزء بسيط منها

الآليات التي تسمح بإنشاء dynamic html:

1. JavaScript: نستخدمها لتغيير شجرة الـ Html
2. Ajax: مهمته جلب التعديل المراد للزبون.

إذاً ما تمّ عمله في JSP هو جعل الصفحة متغيرة لكن ليس هو ما يُسمّى Dynamic Html.

❖ Client Side:

هو كل ما نستطيع عمله عند الـ Client.

- أسئلة هامة:

- كيف نميز الـ Code انه Client Side أو Server Side ؟

يمكن القول بأنّ كل الأكواد المكتوبة بـ JavaScript فهي في الـ Client Side ، و أنّ كل الأكواد المكتوبة بـ Java فهي Server Side

- متى نقرر استخدام Client Side أو Server Side ؟

1. الأمن: إذا كانت العملية هامة أمنياً تكتب في الـ Server Side.
2. العلاقة مع قاعدة البيانات: مثلاً في Google يوجد: حجم زمن التنفيذ، حجم الـ Code.

- Java script هي scripting language لـ Code بسيط، مثلاً لا يمكن وضع الـ Search Engine الخاص بـ Google في JavaScript وذلك لأن طبقية الـ Code كبير جداً ويعمل Access بشكل متكرر على قاعدة البيانات فهو بحاجة لأن يكون بجانبها
- هناك فرق بين تخزين الـ Data على Client side أو كتابة الـ Code في الـClient Side
- حدود استعمال Javascript: وُجِدت لإنشاء Dynamic Html، لكن حقيقةً لا يمكن استخدمها كَـ Business أي لا يجوز برمجة مناقلة بين حسابين أو عملية بيع أو شراء أسهم بـ Javascript، حيث أن JavaScript والـ Html عملها هو إدخال الـ Parameters اللازمة لعملية القيام بالأعمال، وكل ما نكتبه بـ Javascript هو فقط مساعدة بالإدخال.
- إذاً بمجرد كون الرماز Business لا يمكن كتابته بـ Client side نهائياً.

❖ Front End & Back End:

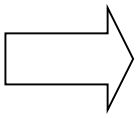
الاثنان بطرف Server side

عند وجود أكثر من Server، يقوم كل Server بوضع في الـ Front-end ما يلي:

- Webserver
- Enterprise App server بداخله container، ليعالج عدد من الطلبات بعدد محدود من الـ Threads، مُطبّق عليه Optimization لهذه الطلبات
- أما الـ Server الذي يستقبل الطلبات فيقوم بعملية Dispatching أي هو الذي يوزع الـ Requests إلى أماكنها المناسبة.
- أنه لا يتعامل مع Servers إنما يرى Webserver backend ، إذاً قد يوجد One Front-end و n Back-end، ويمكن وضع Frontend وخلفه n Front-end تعمل على n Back-end

- إذاً لا مانع من تكرار Front end ، ففي حال وجود Data center كبير فعلينا التفكير بِـ Front-end أقل يحوّل إلى n Front-end خلفه وهي تحوّل إلى n Back-end
- Back end: هو المخدّم النهائي.
- نلاحظ الكود اليساري هو Java code، إلّا أن المحتوى المولّد هو Html.

```
<html>
  <body>
    <% for (int i = 0; i < 2;
i++) { %>
      <p>Hello World!</p>
    <% } %>
  </body>
</html>
```



```
<html>
  <body>

      <p>Hello
World!</p>
      <p>Hello
World!</p>
  </body>
</html>
```

(Slide 94) :Expression Tag

```
<html>
<body>
<p><%= Integer.toString( 5 * 5 )
%></p>
</body>
</html>
```

- يبدأ هذا الـ Expression بـ (=) وتعني أن هذا Expression نريد تخريجهُ إلى الـ Output Stream (أي نضع أي عبارة Java)
- أيضاً داخل JSP نستطيع كتابة أكواد جافا بحيث لا تكون Business logic بحيث علينا استخدام Java Bean التي تربط مع الـ Html (بحيث 90% Html و 10% Java) حيث إذا تجاوزت نسبة Java الـ 10% لا يعود بالإمكان قراءة الـ Html.
  - ما الفرق بين Database والـ File system؟
- هو Concurrent Access بحيث أن الـ DB يمكن أن يدخل n thread إلى DB وتدير قضية الوصول التعدد عليها.
- بينما الـ File على كل thread الانتظار لينتهي السابق له، تدير قضايا مثل security
- مثلاً البرامج التي تحتاج إلى مستخدم واحد مثل المحاسبة تُدار بالـ File وليس في الـ DB
- لا يجوز كتابة تعليمات SQL داخل Program
- ORM: عدد الـ SQL التي تتجه إلى DB محدود.
- مثلاً: لو عندنا Object x فيه حقول A,B,C,D,E,F، وأردنا عمل SQL لتنفيذ synchronized و SQL لتنفيذ select وما تبقى نقوم به بأنفسنا.

- JSP هي طبقة View.
- سلايد 95 : Declaration لتعريف Statement
- سلايد 96 : Directive تستخدم لجلب مكتبات
- سلايد 97 : use bean لاستعمال Class Java علينا الوصول لها والكتابة فيها

- يعرف الـ use bean عن الـ Class العاديّة: أنه لو أتى في الـHttp Request: حقول x,y,z وكان الـ Bean class فيه x,y,z يتم تعبئتهم أوتوماتيكياً (دون الاضطرار لعمل get و set)، كما يتم تنفيذ الـmethods أوتوماتيكياً.
- سلايد 98: comment معروف بديهياً.

- سلايد 99 : لدينا متحولات ثابتة والتي هي : 6 Maps في JSP تُبنى عند وصول الـ Request للصفحة و هي:
  - ✓ Request
  - ✓ Session
  - ✓ Page
- لكل منها Parameters، مثلاً البارامترات التي توضع في الـ Page تبقى طالما الصفحة على قيد الحياة، والتي توضع في الـ Request تبقى خلال دورة حياة Request، حيث أن دورة حياة الـ Request تعني أنه لو أنها جاءت إلى صفحة الـ JSP وعملت Forward لـ Request أخرى وأخرى ثالثة جميعها تُسمّى Request واحدة
- لو قمنا بعمل Request في وقت لاحق: تعدّ Request جديدة.
- إذاً جميع الـ Parameters السابقة التي قمنا بتعبئتها تضيع لذلك نخزنا في الـ Session
- تموت الـ Session عند إغلاق الـ Browser فقط
- Application : وإن وضُعت فيها Data نستطيع أن نجعلها مرئية لجميع المكونات.
- Out: هنا المتحول نكتب به على الـ Output Stream.

○ هل يمكن معرفة الـ Session هل هي Valid أم لا؟

نقوم بعمل Session timeout لتحديد مدة الصلاحية الأعظميّة للـ Session

ونعمل setAttribute: لتعبئة متحول داخل الـ Session.

- في حال قام أحد الـ Users بعمل Register فيتم وضع Attribute لتمييز الـ User:

- Authenticated user أو Registered user: بحيث أن Authentication له حقوق خاصة له تبقى على مستوى الـ Session الخاصة به طيلة فترة حياته.

- ملاحظة:

الاطلاع على كامـــــــل السلايدات من 87 ← 102

** و كل عام و أنتم بخير **

تمّ بعون الله إنهاء محاضرات مقرّر برمجة التطبيقات الشبكية

ما كان مِن صوابٍ فمِنَ الله وما كان من خطأ فمِن أنفسنا

ولا تنسونا من صالح دعائكم

فريق برمجة التطبيقات الشبكية

Ghofran Bakuor - Aroub Arar - Sawiem Msouti - Muhammad Arafat

Asmaa Najm - Alaa Deban