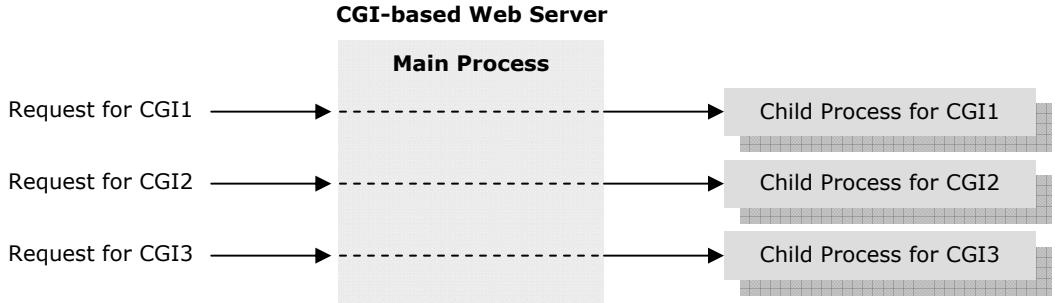
برنامه نویسی سرولت

مقدمه

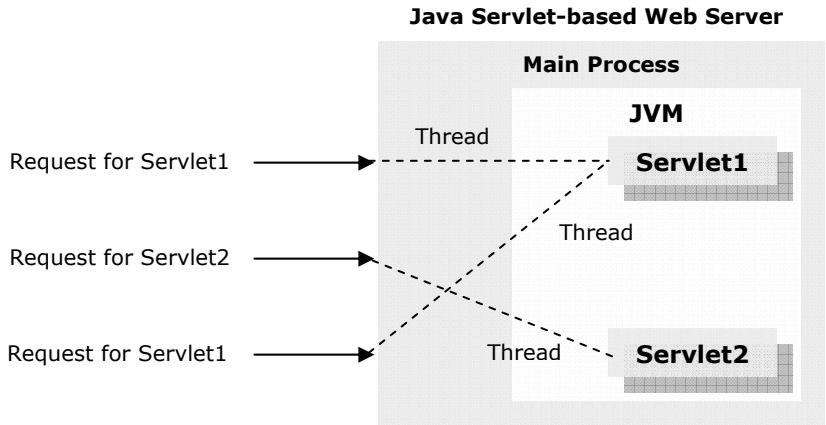
با گذر از دوره برنامه های سنتی دو لایه (معماری سرویس دهنده/سرویس گیرنده [1]) و پیدایش برنامه های چند لایه و مبتنی بر وب ، لایه ای بنام لایه وب بوجود آمد. این لایه وظیفه رسیدگی و کنترل نحوه نمایش برنامه های مبتنی بر وب را برعهده دارد.

در مدلهای اولیه برنامه های وب ، برای تولید پویای صفحات از تکنولوژی CGI[2] استفاده می گردید. این تکنیک از زبانهای برنامه نویسی همانند Perl یا C برای پردازش درخواست سرویس گیرنده ، انجام منطق برنامه و نیز تولید خروجی یا پاسخ بهره می گرفت. در این روش به ازای هر درخواست کاربر ، یک نمونه از برنامه CGI اجرا می شود. در نتیجه اگر چندین درخواست همزمان و اجرای همان تعداد برنامه در سرویس دهنده منجر به کاهش سرعت و کارآیی و حتی ازکار افتادن آن می شود. همچنین با توجه به طبیعت زبانهای برنامه نویسی استفاده شده ، مقیاس پذیری برنامه و توسعه آن غیر ممکن به نظر می رسد و از طرف دیگر فرآیند های تست ، ردیابی خطا و نگهداشت و نیز یادگیری آن برای تولیدکنندگان بسیار بسختی صورت می گیرد.



[1] Client/Server
[2] Common Gateway Interface

با پیدایش تکنیک های پیشرفته تر ، جاوا نیز مدل برنامه نویس سرولت[3] را عرضه نمود. این مدل یک راه حل مستقل از پلاتفرم ، قابل حمل ، مقیاس پذیر ، قدرتمند و آسان برای تولید برنامه های وب است. سرولت یک کلاس جاوا است که توسط سرویس دهنده وب یا حامل وب[4] فراخوانی و در داخل ماشین مجازی جاوا (JVM) سرویس دهنده اجرا می شود. برخلاف CGI که به ازای هر درخواست سرویس گیرنده یک نمونه از برنامه CGI اجرا می شوند ، در سرولت ها به ازای هر درخواست یک ریسمان از سرولت توسط حامل وب اجرا می شود. بنابراین عملکرد و کارایی سرولت ها بیشتر از برنامه های سنتی CGI و سایر تکنولوژیهای مشابه است .



یکی دیگر از ویژگیهای سرولت جاوا قابلیت حمل آن است ، سرولت ها نیز مشابه سایر برنامه های جاوا در تمام سیستم عامل های که JVM را پشتیبانی می کنند قابل اجرا است.

[3] Java Servlet
[4] Web Container

در این فصل مواردی چون مدل برنامه نویسی سرولت ، چرخه حیات سرولت ، اداره کردن درخواست و پاسخ و چند مثال کاربردی بیان خواهد شد.

سرویس گیرنده وب

در معماریJ2EE دو نوع سرویس گیرنده عمده وجود دارند : سرویس گیرنده وب [5] و سرویس گیرنده کاربردی [6].

سرویس گیرنده کاربردی در معماری سرویس گیرنده ، تعامل کاربر را از طریق یک رابط گرافیکی کاربر انجام می دهد که منطق برنامه را نیز شامل می شود. از اینرو به این نوع سرویس گیرنده کاربردی Fat Client نیز گفته می شود. این سرویس گیرنده منطق برنامه را انجام می دهد ، در حالیکه در یک معماری چند لایه سرویس گیرنده کاربردی می توان بخشی از منطق برنامه و کدهای دسترسی به منابع اطلاعاتی را به اجزای میانی [7] واگذار نمود .علیرغم واگذاری منطق برنامه به اجزای میانی ، این سرویس گیرنده همچنان Fat Client می ماند چرا که باید در دستگاه تمام کاربران نصب شود.

با پیدایش و رشد اینترنت ، سرویس گیرنده وب جایگزین سرویس گیرنده سنتی کاربردی شد . در معماری سرویس گیرنده مبتنی بر وب ، لایه تعامل با کاربر از لایه سرویس گیرنده جدا می شود. مرورگر وب یا برنامه های مشابه ، تعامل با کاربر را مدیریت می کند و رسیدگی به درخواستهای کاربر و نیز ارتباط با اجزای میانی و دسترسی به بانک های اطلاعاتی به لایه سرویس دهنده وب منتقل می شود.
در این معماری مرورگروب نقش یک سرویس گیرنده برای برنامه های مبتنی بر وب را ایفا می کند. با این تفاوت که نیازی به نصب برنامه در سمت سرویس گیرنده نیست. از اینرو به این نوع سرویس گیرنده Thin Client نیز گفته می شود.

ویژگیهای سرویس گیرنده وب

- یک مرورگر یا هر برنامه مشابه دیگر تعامل با کاربر را مدیریت می کند.

---

[5] Web Client
[6] Application Client
[7] Middle Component

- رابط کاربر با زبانهای HTML (بهمراه JavaScript یا DHTML) ، XHTML و یا XML(بهمراه XSL)تعریف می شود.
- پروتکل HTTP/HTTPS برای تبادل اطلاعات بین سرویس گیرنده وب و سرویس دهنده وب را بر عهده دارد و منطق برنامه در سمت سرویس دهنده اجرا می شود.

معماری J2EE یک مدل برنامه نویسی انعطاف پذیر و قدرتمند را برای ساختن برنامه های پویای مبتنی بر وب ارائه می کند. معماری J2EE حامل وب ، سرولت و Java Server Page را برای ساختن و مدیریت برنامه های وب عرضه می کند.
حامل وب یک محیط زمان اجرا برای اداره و کنترل نمودن اجرای سرولتهای جاوا و صفحات JSP است.
تولید برنامه های مبتنی بر وب را سرولتها و صفحات JSP بر عهده دارند.

پروتکل HTTP

در تولید برنامه های توزیع شده ، نوع پروتکل ارتباطی ساختار و نوع سرویس دهنده و سرویس گیرنده را مشخص می کند. این قانون برای برنامه های مبتنی بر وب نیز صدق می کند.

HTTP[8] یک پروتکل ارتباطی ناپایدار[9] است که بروی پروتکل TCP/IP[10] قرار می گیرد و بر پایه درخواست[11] و پاسخ[12] بنا شده است. در مدل ارتباطی این پروتکل یک سرویس گیرنده (مرورگر وب) درخواستهایی را برای سرویس دهنده (سرویس دهنده وب یک فروشگاه Online) برای گرفتن اطلاعات (درخواست یک کاتالوگ محصول) یا راه اندازی یک فرآیند در سرویس دهنده (ثبت یک سفارش خرید) ارسال می کند. متعاقبا سرویس دهنده پردازش های لازم را انجام و پاسخ مناسب را برای سرویس گیرنده ارسال می کند.

[8] Hypertext Transfer Protocol
[9] Stateless
[10] Transfer Control Protocol/Internet Protocol
[11] Request
[12] Response

متد های درخواست HTTP

در هر پروتکلی چندین متد برای ارسال درخواست و پاسخ وجود دارد. این امر در مورد پروتکل HTTP نیز صدق می کند. HTTP1.0 سه متد ارسال را تعریف می کند: GET ، POST و HEAD. در HTTP1.1پنج متد جدید OPTION ، PUT ، TRACE ، DELETE و CONNECT نیز اضافه شد. با این حال GET و POST رایج ترین و متد های مورد استفاده است.

متد درخواست GET

در بین متدهای ارسال درخواست ، متد GET آسانترین و پر استفاده ترین متد برای دسترسی به منابع ثابت بعنوان مثال صفحات HTML ، تصاویر و غیره است. درخواست های GET همچنین برای بازیابی صفحات پویا با اضافه کردن پارامتر به انتهای آدرس درخواست نیز استفاده کرد. برای مثال می توان پارامتر name=joe را به انتهای یک آدرس بصورت زیر اضافه کرد و در سمت سرویس دهنده با توجه به پارامتر ارسال شده ، پاسخ مناسب آن را تولید نمود.

```
http://www.domain.com?name=joe
```

متد درخواست POST

متد POST بیشتر برای دسترسی به منابع دینامیک استفاده می شود. درخواست های POST روشی برای انتقال اطلاعاتی است که به درخواست وابسته اند و نیز زمانی که حجم اطلاعات زیاد باشد ، بکار گرفته می شود. درخواست POST اجازه می دهد انواع اطلاعات را در قالب یک درخواست کپسوله نمود. برای مثال می توان از درخواست POST برای بارگذاری فایل های متنی یا باینری استفاده کرد. به همین شکل یک اپلت می تواند با استفاده از درخواست POST اشیاء سریال شده یا حتی اطلاعات خام را برای سرویس دهنده ارسال کند.

چندین تفاوت اساسی بین متدهای GET و POST وجود دارد. در درخواست GET ، پارامترهای درخواست از طریق یک رشته اضافه شده به آدرس درخواست ارسال می شود. در مورد درخواست های POST ، این پارامترها در بدنه درخواست قرار می گیرند. بعبارت دیگر چون درخواست GET تمام اطلاعات درخواست را به انتهای آدرس آن اضافه می کند ، مرورگر وب می تواند آدرس درخواست را در خود

ثبت کند و دوباره به آن مراجعه کند. علاوه براین بعضی از سرویس گیرنده ها محدودیت طول آدرس درخواست دارند ، بنابراین اگر طول پارامترهای درخواست بیشتر از حد مجاز باشد بخشی از اطلاعات برای سرویس دهنده ارسال نمی شود.

پاسخ HTTP

برای پاسخ به درخواست HTTP ، سرویس دهنده با استفاده از یکسری شبه داده و نیز کد وضعیت که در Header پاسخ قرار می گیرند، پاسخ را برای سرویس دهنده ارسال می کند. بعضی از فیلدهای Header پاسخ عبارتند از "Date" ، "Content-Type" و "Expires" . برای مثال اگر فیلد "Date" و "Expires" یکسان باشد ، مرورگر وب نمی تواند درخواست را در حافظه نهان خود نگاه دارد و درخواست را از سرویس دهنده فراخوانی نماید.

در پروتکل HTTP ، سرویس دهنده و سرویس گیرنده از MIME[13] برای تعیین نوع محتوای درخواست یا پاسخ استفاده می کنند. این فیلد می تواند text/html ، image/gif یا غیره باشد. قسمت اول فیلد MIME نوع اطلاعات (برای مثال متن یا تصویر) و قسمت دوم قالب درخواست (برای مثال html ، یا gif) را مشخص می کند. MIME این امکان ارسال انواع اطلاعات را از طریق اینترنت فراهم می سازد. سرویس دهنده HTTP در ابتدای هر انتقال نوع MIME را مشخص می کند و مرورگر با استفاده از آن تصمیم می گیرد که چگونه اطلاعات دریافت شده را تجزیه و تحلیل نماید.

ویژگیهای پروتکل HTTP عبارتست از :

- HTTP یک پروتکل ساده و سبک است.
- در این پروتکل ، همیشه سرویس گیرنده درخواست را پی ریزی می کند.
- در پروتکل HTTP ، سرویس گیرنده می بایست قبل از ارسال درخواست ، ارتباط را برقرار کند و نیز سرویس دهنده پس از فرستادن پاسخ ، ارتباط را قطع نماید. این امر تضمین می کند که سرویس گیرنده نتواند ارتباط را پس از دریافت درخواست نگه دارد. علاوه براین سرویس دهنده و سرویس گیرنده می تواند ارتباط برقرارشده را قبل از موعد قطع کند.

[13] Multi-Purpose Internet Mail Extensions

حامل های وب و برنامه های وب

برنامه وب ، در سمت سرویس دهنده قرار می گیرد. تولید هر برنامه سمت سرویس دهنده نیازمندیهای زیر را شامل می شود:

- یک مدل برنامه نویسی و یک API : برای تولید برنامه
- محیط زمان اجرای سمت سرویس دهنده : پشتیبانی سرویس های شبکه و یک محیط برای اجرای برنامه
- توسعه[14] : توسعه ، فرآیند نصب برنامه روی سرویس دهنده است. این توسعه می تواند سفارشی کردن[15] برنامه نیز باشد.

برای ساختن و اجرای برنامه های مبتنی بر وب ، J2EE تمام این نیازمندیها را پاسخ داده است :

- **سرولت جاوا یا صفحات JSP**

  سرولت جاوا یا صفحات JSP بلاک های تولید برنامه های وب هستند.

- **برنامه های وب**

  یک برنامه وب ، مجموعه ای از سرولت های جاوا ، صفحات JSP ، کلاس های کمکی ، منابع ایستا شامل HTML، XHTML یا سندهای XML، تصاویر و غیره است.

- **حامل وب برای میزبانی برنامه های وب**

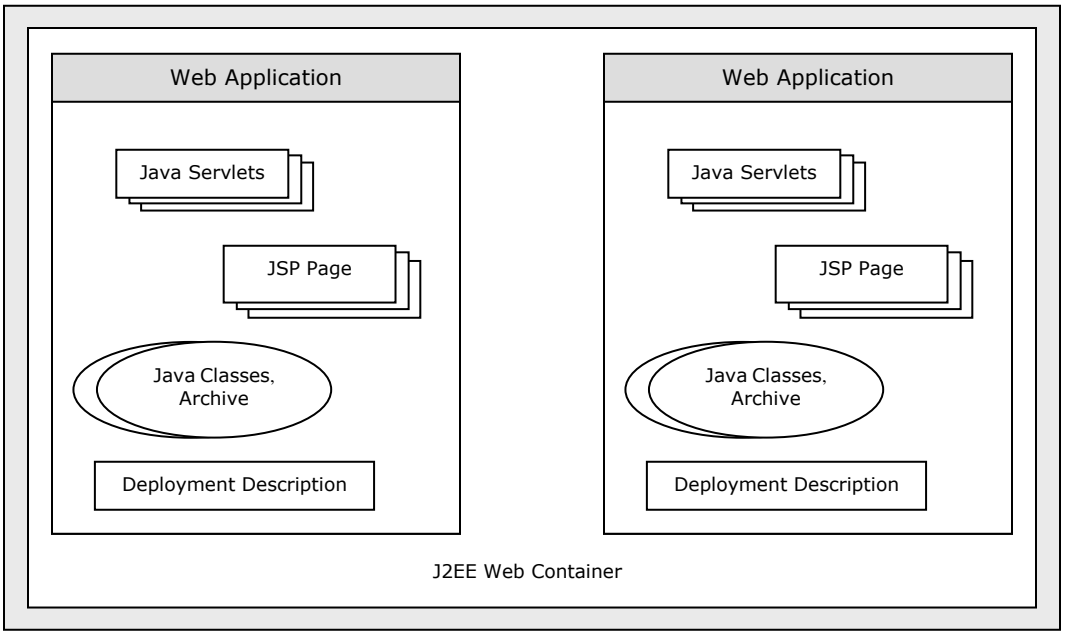
  حامل وب یک محیط زمان اجرای جاوا برای پیاده سازی API سرولت جاوا و سایر امکانات برای صفحات JSP است. حامل وب مسئول راه اندازی اولیه ، فراخوانی و مدیریت چرخه حیات سرولت ها و صفحات JSP است.

---

[14] Deployment
[15] Customizing

- **ساختار بسته ها و توصیف گر توسعه**

استاندارد J2EE ساختار برنامه های مبتنی بر وب را نیز تعریف می کند. این استاندارد یک توصیف گر توسعه[16] برای برنامه های وب در نظر گرفته است. این توصیف گر یک فایل XML است. شکل زیر یک حامل وب J2EE با دو برنامه وب توسعه یافته را نشان می دهد.



توصیف گر توسعه

توصیف گر توسعه یکی از اجزای مهم برنامه های وب J2EE محسوب می شود. توصیف گر توسعه ، مدیریت پیکربندی برنامه های وب را برعهده دارد. این فایل web.xml در دایرکتوری/WEB-INF

[16] Deployment Descriptor

برنامه وب قرار می گیرد. ساختار این فایل طبق DTD زیر تعریف می شود :

http://java.sun.com/j2ee/dtds/web-app_2_2.dtd

تنظیمات زیر در توصیف گر توسعه انجام می شود :

- تعریف پارامترهای اولیه سرولت و برنامه وب
  این امکان ، حجم کدهای مربوط به مقداردهی اولیه در داخل برنامه ها را به حداقل می رساند. برای مثال اگر یک سرولت نیاز دارد به بانک اطلاعاتی دسترسی پیدا کند ، بجای تعریف مشخصات بانک اطلاعاتی شامل شناسه کاربر ، رمز عبور و مشخصات سرویس دهنده در کد سرولت ، آنرا در توصیف گر توسعه قرار داد. در نتیجه می توان این مشخصات را بدون نیاز به دستکاری کد و کامپایل مجدد برنامه ها تغییر داد.

- معرفی سرولت/ JSP
  هر سرولت استفاده شده در یک برنامه وب بایستی در توصیف گر توسعه تعریف شود. این تعریف شامل نام واقعی یا مجازی سرولت و نام کلاس سرولت است.

- نگاشت سرولت/JSP
  حامل های وب از این اطلاعات برای واگذاری و هدایت کردن درخواست های سرویس گیرنده به سرولت استفاده می کند.

- تعریف انواع MIME
  از آنجاییکه هر برنامه وب می توان چندین نوع MIME را پشتیبانی کند ، می توان نوع اطلاعات را در توصیف گر توسعه تعریف کرد.

- کنترل امنیت
  نحوه دسترسی کاربران به برنامه با استفاده از توصیف گر توسعه قابل کنترل است. برای مثال اگر صفحاتی از برنامه نیاز به login کاربر دارد ، می توان نام صفحه login و سطح دسترسی مورد نیاز کاربر را در توصیف گر توسعه تعریف نمود.

- معرفی صفحه اصلی/صفحه خطا

صفحه اصلی یا پیش فرض و صفحه کنترل خطاهای برنامه قابل تعریف است. صفحه اصلی یا پیش فرض ، صفحه ای است که حامل وب هنگام فراخوانی آدرس شاخه اصلی برنامه (برای مثال http://server-name/app-root/) توسط کاربر آنرا نمایش می دهد. علاوه براین می توان برای نمایش پیغام های مناسب تر در زمان بروز خطای زمان اجرا ، صفحاتی تعریف کرد.

انواع حامل های وب

قبل از پرداختن به جزئیات حامل های وب ، بد نیست درباره انواع حامل های وب مطالبی گفته شود. یک حامل وب می تواند سرویس های HTTP را پیاده سازی کند یا آنرا به سرویس دهنده های وب واگذار نماید. سه روش اصلی برای پیکربندی حامل های وب وجود دارد.

- **حامل وب در یک سرویس دهنده کاربردی J2EE**

بیشتر سرویس دهند های کاربردی مبتنی بر J2EE مانند Bea WebLogic ، Inprise Application Server ، iPlanet Application Server ، IBM WebSphere Application Server و غیره ، حامل وب را در داخل خود پیاده سازی کرده اند.

- **حامل وب در کنار سرویس دهنده وب**

سرویس دهنده های وب که بصورت کامل با جاوا پیاده سازی می شوند ، برای مثال Sun's Java WebServer حاوی یک حامل وب یکپارچه شده است. سرویس دهنده Jakarta Tomcat(http://jakarta.apache.org) که شامل یک سرویس دهنده وب در کنار یک حامل وب است ، در این طبقه بندی قرار می گیرد.

- **حامل وب مستقل**

در دنیای خارج از J2EE ، این مدل بیشتر مورد استفاده قرار می گیرد. سرویس دهنده هایی مانند Apache یا Microsoft IIS[17] برای اجرای سرولت های جاوا از یک plug-in برای ارتباط با محیط زمان اجرای جاوا و اجرای سرولت های جاوا استفاده می نمایند. این plug-in بین سرویس دهنده وب و حامل وب ارتباط برقرار می کند. موتورهای سرولت مانند Allaire JRun و ServletExec دارای plug-in زمان اجرای جاوا برای ارتباط با سرویس دهنده وب هستند.این دو موتور سرولت جزو سرویس دهنده های کاربردی J2EE محسوب می شوند. سرویس دهنده Tomcat در کنار سرویس دهنده وب Apache ، Microsoft's IIS و Enterprise Server Netscape نیز در این طبقه بندی قرار می گیرد.

انتخاب هر یک از این راه حلها ، بستگی به نیازمندیهای برنامه مورد نظر دارند. برای برنامه های وب مبتنی بر جاوا بیشتر پیکربندی اول مورد استفاده قرار می گیرد. در صورتی که یک مدل جدید برنامه مد نظر باشد ،استفاده از پیکربندی های دوم و سوم توصیه می گردد.

ساختار برنامه های وب

یک برنامه وب دارای چهار قسمت است :

- دایرکتوری عمومی
- فایل WEB-INF/web.xml
- دایرکتوری WEB-INF/classes
- دایرکتوری WEB-INF/lib

دایرکتوری عمومی ، مسیر اصلی برنامه و محل قرار گرفتن دایرکتوری WEB-INF است. در سرویس دهنده وب Apache این دایرکتوری معادل دایرکتوری htdocs است که تمام صفحات HTML در آن قرار می گیرد.

[17] Internet Information Service

دایرکتوری WEB-INF ناحیه اختصاصی و محرمانه یک حامل وب است. بدین معنی که کاربران یا سرویس گیرنده ها اجازه دسترسی مستقیم به این دایرکتوری را ندارند و فایل های موجود در این دایرکتوری فقط توسط حامل وب مورد استفاده قرار می گیرند. محتویات این دایرکتوری شامل توصیف گر توسعه (web.xml) ، دایرکتوریclasses و دایرکتوری lib است. دایرکتوری classes برای نگهداری کلاس های کامپایل شده سرولت های جاوا و سایر برنامه های کمکی درنظر گرفته شده است. اگر برنامه وب دارای فایل های آرشیو جاوا (JAR) باشد (API های درایور بانک اطلاعاتی و غیره) در این دایرکتوری قرار می گیرند.حامل وب از این دو دایرکتوری برای سازماندهی سرولت های جاوا و کلاس های وابسته استفاده می کند.

اولین برنامه وب

برای شروع برنامه ای با مشخصات زیر در نظر گرفته شده است :

- فرم ورود اطلاعات برای گرفتن نام و آدرس الکترونیکی کاربر
- چاپ پیغام خوشآمد گویی به کاربر و تایید دریافت درخواست کاربر

اگرچه می توان از مدل ترکیبی سرولت های جاوا و صفحات JSP برای تولید این برنامه استفاده کرد. اما می توان برنامه را بصورت زیر پیاده سازی نمود :

- یک صفحه HTML حاوی فرم ورود اطلاعات برای گرفتن نام و آدرس الکترونیکی کاربر
- یک سرولت برای پردازش درخواست کاربر و تولید خروجیHTMLونمایش پیغام خوشآمدگویی
- یک توصیف گر توسعه

آماده سازی حامل وب

پیاده سازی این برنامه نیاز مند یک حامل وب یا موتور سرولت مطابق با استاندارد Java Servlet Specification 2.2 و JSP1.1 است. چندین محصول مطابق با این استاندارد ها وجود دارد. برای شروع ،سرویس دهنده Apache Tomcat توصیه می شود.

http://jakarta.apache.org

با فرض اینکه سرویس دهنده Tomcat در سیستم عامل ویندوز نصب شده است ، دایرکتوری %TOMCAT_HOME% به دایرکتوریی که Tomcat در آن نصب شده است اشاره می کند. (برای مثال C:\jakarta-tomcat)سرویس دهنده Tomcat نیازمند JDK1.1 یا بالاتر است. پس از نصب این سرویس دهنده ، اجرای مثال های آن نشان می دهد که درست نصب شده است.

ایجاد فایل HTML

یک فایل HTML با محتوای زیر در آدرس TOMCAT_HOME%\webapps\greeting\index.html ایجاد نمایید. تگ <FORM> این HTML از متد POST برای گرفتن فیلدهای "name" و "email" استفاده می کند:

```
<HTML>
  <HEAD>
    <TITLE>ProJava Registration </TITLE>
  </HEAD>
  <BODY>

      <H1>Welcome</H1>

      <FORM ACTION= "/greeting/servlet/GreetingServlet" METHOD="POST">
          <P>Your Name <INPUT TYPE="text" SIZE="40" NAME="name"></P>
          <P>Your Email <INPUT TYPE="text" SIZE="40" NAME="email ">
          <INPUT TYPE="Submit" VALUE="Submit"></P>
       </FORM>
    </BODY>
</HTML>
```

تعریف سرولت

مرحله بعدی تعریف سرولت است. این سرولت درخواست HTTP ارسال شده توسط صفحه index.html را پردازش می کند.

کلاس GreetingServlet را در یک فایل جدید بنام GreetingServlet.java در دایرکتوریTOMCAT_HOME%\webapps\greeting\src% با کد زیر ایجاد نمایید:

```
// Import Servlet Packages
import javax.servlet.*;
import javax.servlet.http.*;
// Import other Java Packages
import java.io.*;
import java.util.*;

public class GreetingServlet extends HttpServlet {

   protected void dopost (HttpServletRequest request , HttpServletResponse response)
          throws ServletException , IOException {

      // Get parameters from the request .
      String name = request.getParameter("name");
      String email = request.getParameter("email");
```

```
      // Compute a greeting message
       String message = null;
       GregorianCalendar calendar = new GregorianCalendar();
       if(calendar.get(calendar.AM_PM) == Calendar.AM) {
          message = "Good Morning";
       } else {
          message = "Good Afternoon";
       }

       // Set MIME type for the reponse
       response.setContentType("text/html");
       // Obtain a print writer object
       PrintWriter out = response.getWriter();

       // Write the content
       out.println("HTML>");
       out.println("<BODY>");
       out.println("<P>" + message + "," + name + "</P>");
       out.println("<P> Thanks for registering your email (" + email +
                        ") with us.</P>");
       out.println("<P> - The Pro Java Team. </P>");
       out.println("</BODY>");
       out.println("</HTML>");
       out.close();
   }
}
```

کامپایل کردن سرولت

برای کامپایل سرولت ، فایل `%TOMCAT_HOME%\lib\servlet.jar` بایستی در `CLASSPATH` تعریف شود و همچنین دایرکتوری های `WEB-INF` و `classes` بصورت زیر ایجاد شوند :

```
%TOMCAT_HOME%\webapps\greeting
%TOMCAT_HOME%\webapps\greeting\WEB-INF
```

حال در دایرکتوری کد سرولت دستور زیر اجرا شود :

```
javac -d  ..\WEB-INF\classes\  *.java
```

این دستور کلیه کلاس های جاوا را کامپایل و در دایرکتوری زیر قرار می دهد :

```
%TOMCAT_HOME%\webapps\greeting\WEB-INF\classes
```

نوشتن توصیف گر توسعه

آخرین مرحله برنامه ، نوشتن توصیف گر توسعه است. فایل web.xml را در دایرکتوری %TOMCAT_HOME%\webapps\greeting\WEB-INF با کد زیر ایجاد نمایید :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD web Application 2.2// EN"
   "http://java.sun.com/j2ee/dtds/web-app-2.2.dtd">

<web-app>

   <servlet>
      <!--servlet alias -->
      <servlet-name>GreetingServlet</servlet-name>

      <!--Fully qualified Servlet class -->
      <servlet-class>GreetingServlet</servlet-class>
    </servlet>

   <servlet-mapping>
      <servlet-name>GreetingServlet</servlet-name>
      <url-pattern>/servlet/ GreetingServlet </url-pattern>
   </servlet-mapping>

</web-app>
```

تست برنامه

برنامه آماده تست است. اگر سرویس دهنده Tomcat بدرستی نصب شده باشد. با وارد کردن آدرس http://localhost:8080/greeting در مرورگر وب ، صفحه زیر نمایش داده می شود :

پس از وارد کردن نام و آدرس الکترونیکی روی دکمه Submit کلیک نمایید. مرورگر وب خروجی مشابه شکل زیر نمایش خواهد داد. البته خروجی واقعی به اطلاعات وارد شده و نیز زمان ارسال درخواست بستگی دارد.

ساختار برنامه

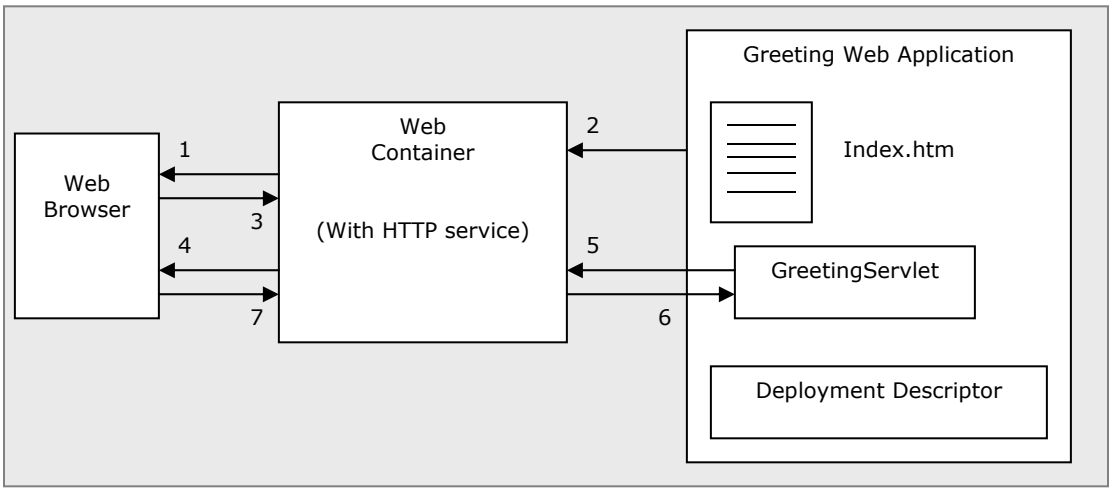
در این مثال فایل های زیر ایجاد شده اند :

- %TOMCAT_HOME%\webapps\greeting\index.html
- %TOMCAT_HOME%\webapps\greeting\src\GreetingServlet.java
- %TOMCAT_HOME%\webapps\greeting\WEB-INF\web.xml
- %TOMCAT_HOME%\webapps\greeting\WEB-INF\classes\GreetingServlet.class

این چهارفایل اولین برنامه وب جاوا را تشکیل می دهند.

برنامه چگونه کار می کند

شکل زیر فرآیند و ترتیب اجرای برنامه را نمایش می دهد. در این مثال مرورگر وب دو درخواست HTTP برای حامل وب ارسال می کند. برای پاسخ به اولین درخواست (http://localhost:8080/greeting) حامل وب فایل index.html را از دایرکتوری %TOMCAT_HOME%\webapps\greeting\index.html فراخوانی می کند. در فایل پیکربندی برنامه (web.xml) ، مسیر مجازی greeting برای یک برنامه وب موجود در دایرکتوری %TOMCAT_HOME%\webapps\greeting\ تعریف شده است. فایل index.html در بیشتر حامل/سرویس دهنده های وب بعنوان فایل پیش فرض برای هر دایرکتوری در نظر گرفته می شود. فلشهای (1)، (2) و (3) مراحل پاسخگویی سرویس دهنده وب به درخواست سرویس دهنده را نشان می دهد.



درخواست دوم

(http://localhost:8080/greeting/servlet/GreetingServlet) درخواست HTTP POST با دو پارامتر name و email است. با توجه به آدرس فراخوانی شده در فرم فایل index.html و مسیر های تعریف شده در فایل توصیف گر توسعه (web.xml) ، حامل وب این درخواست را به سرولت GreetingServlet واگذار می کند. این سرولت وظایف زیر را انجام می دهد:

- استخراج پارامترهای درخواست : گرفتن مقدار پارامترهای name و email از درخواست HTTP
- اجرای منطق برنامه : تولید خروجی شامل پیغام خوشامدگویی و اطلاعات وارد شده توسط کاربر
- تولید خروجی : پس از پردازش درخواست ، نمایش یک خروجی HTML حاوی پیغام خوشامدگویی

بدین ترتیب مرورگر وب، پاسخ تولید شده به این روش را دریافت می کند. فلش های (4)، (5) ، (6) و (7) این فرآیند را نشان می دهد.

ساختار سرولت GreetingServlet

- وارد کردن بسته های سرولت

  Java Servlet API شامل دو بسته است: javax.servlet و javax.servlet.http. ذکر این نکته ضروری است که بسته javax بخشی از Java 2 SDK, Enterprise Edition می باشد.

  بسته javax.servlet شامل کلاس ها و رابط های مستقل از پروتکل HTTP است. کلاس ها و رابط های مربوط به HTTP در بسته javax.servlet.http قرار گرفته اند. علاوه بر این دو بسته ، سرولت GreetingServlet به بسته های java.io و java.util نیزنیاز دارد :

```
// Import Servlet Packages
import javax.servlet.*;
import javax.servlet.http.*;
// Import other Java Packages
import java.io.*;
import java.util.*;
```

- تعریف کلاس

هر سرولت باید رابط javax.servlet.Servlet را پیاده سازی نماید. برای سرولت دربرنامه های کاربردی وب ، کلاس javax.servlet.http.HttpServlet این پیاده سازی را برعهده دارد.

بنابراین کلاس GreetingServlet از کلاس HttpServlet به ارث گرفته می شود :

```
public class GreetingServlet extends HttpServlet {
 ...
}
```

- پردازش درخواست HTTP POST

در سرولت GreetingServlet ، متد doPost () درخواست HTTP POST را اداره می کند. این متد یک شیء از نوع HttpServletRequest حاوی اطلاعات درخواست HTTP و یک شیء از نوعHttpServletResponse که پاسخ HTTP را کپسوله می کند را بعنوان آرگومان دریافت می کند.

متد doPost () دو وظیفه را انجام می دهد : استخراج پارامترهای درخواست و تولید خروجی.

- استخراج پارامترها از HttpServletRequest

متد getParameter () از رابط HttpServletRequest ، پارامترهای درخواست HTTP را استخراج می کند :

```
// Get parameters from the request .
 String name = request.getParameter("name");
 String email = request.getParameter("email");
```

اولین خط پارامتر name تعریف شده در تگ <FORM> صفحه index.html را دریافت می کند:

```
<P>Your Name <INPUT TYPE="text" SIZE="40" NAME="name"></P>
```

متد getParameter() پارامترهای قابل دسترس درخواست HTTP را جستجو و مقدار پارامتر name را استخراج می کند. به همین ترتیب مقدار پارامتر email نیز مشخص می شود. ترتیب گرفتن پارامتر های درخواست مهم نیست ، بدین معنی که می توان ابتدا پارامتر email و سپس پارامتر name را استخراج کرد.مرحله پردازش درخواست کاربر در همین جا به اتمام می رسد. مرحله بعدی آماده سازی پاسخ است.

- تولید پاسخ

این مرحله شامل آماده سازی سرولت برای تولید خروجی است. در این مثال خروجی مشابه زیر است :

```
Good Afternoon, Simon Sanders
Thanks for registering your email (ssanders@bar.com) with us.
-The Pro java Team.
```

در سرولت GreetingServlet از استریم اختصاص یافته برای چاپ محتوای پاسخ استفاده می کند. قطعه کد زیر، یک پیغام خوشآمدگویی با توجه به زمان دریافت درخواست چاپ می کند. این تنها منطق این سرولت است:

```
// Compute a greeting message
 String message = null;
 GregorianCalendar calendar = new GregorianCalendar();
 if(calendar.get(calendar.AM_PM) == Calendar.AM) {
    message = "Good Morning";
 } else {
    message = "Good Afternoon";
 }
```

کد فوق از کلاس java.util.GregorianCalendar برای گرفتن تاریخ و زمان جاری سرویس دهنده استفاده می کند. پس از مشخص شدن پیغام ، سرولت آماده تولید خروجی است. این آماده سازی شامل مراحل زیر است :

- تعیین نوع خروجی (MIME) پاسخ. در این مثال خروجی از نوع "text/html" است:

```
// Set MIME type for the reponse
response.setContentType("text/html");
```

- تعریف شیء java.io.PrintWriter با استفاده از شیء تعریف شده از نوع HttpServletResponse

```
// Obtain a print writer object
PrintWriter out = response.getWriter();
```

- چاپ خروجی با استفاده از شیء PrintWriter

```
// Write the content
out.println("HTML>");
out.println("<BODY>");
out.println("<P>" + message + "," + name + "</P>");
out.println("<P> Thanks for registering your email (" + email +
                ") with us.</P>");
out.println("<P> - The Pro Java Team. </P>");
out.println("</BODY>");
out.println("</HTML>");
```

- بستن شیء PrintWriter

```
out.close();
```

تعریف توصیف گر توسعه

در مثال ارائه شده ، فایل توصیف گر توسعه (web.xml) پیکربندی برنامه را مشخص می کند. اولین خط این فایل ، شامل تعریف سند XML و نوع Encoding آن است :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

خط دوم ، آدرس DTD تعریف شده برای این سند XML را مشخص می کند :

```
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD web Application 2.2// EN"
   "http://java.sun.com/j2ee/dtds/web-app-2.2.dtd">
```

طبق استاندارد J2EE فایل web-app-2.2.dtd برای برنامه های وب تعریف شده است. این دو خط برای تمام برنامه های وب مبتنی بر جاوا یکسان است.

معرفی سرولت شامل نام سرولت و نام کلاس آن توسط تگ <servlet> در داخل تگ <web-app> صورت می گیرد :

```
<web-app>
   <servlet>
      <!--servlet alias -->
      <servlet-name> GreetingServlet </servlet-name>

      <!--Fully qualified Servlet class -->
      <servlet-class>GreetingServlet</servlet-class>
    </servlet>
    <servlet-mapping>
      <servlet-name>GreetingServlet</servlet-name>
      <url-pattern>/servlet/ GreetingServlet </url-pattern>
   </servlet-mapping>
</web-app>
```

همانطور که می دانید تگ <FORM> فایل index.html آدرس نسبی /greeting/servlet/GreetingServlet را فراخوانی می کند. برای درک بیشتر این مطلب که حامل وب چگونه از تعاریف فایل web.xml استفاده می کند ، در تگ <FORM> فایل index.html تغییر زیر را اعمال نمایید :

```
<FORM ACTION= "greeting/SERVLET/GreetingServle" METHOD="POST">

        <P>Your Name <INPUT TYPE="text" SIZE="40" NAME="name"></P>
        <P>Your Email <INPUT TYPE="text" SIZE="40" NAME="email ">
        <INPUT TYPE="Submit" VALUE="Submit"></P>

</FORM>
```

فایل web.xml را نیز بصورت زیر تغییر دهید :

```
<servlet>
    <!--servlet alias -->
    <servlet-name> GreetingServlet </servlet-name>

    <!--Fully qualified Servlet class -->
    <servlet-class>GreetingServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name> GreetingServlet </servlet-name>
    <url-pattern>/servlet/ GreetingServlet </url-pattern>
</servlet-mapping>
```

سرویس دهنده Tomcat را دوباره اجرا و آدرس http://localhost:8080/greeting/ را در مرورگر وب فراخوانی نمایید. نتیجه حاصل با حالت قبل یکسان است. این امر بخاطر تعاریف انجام شده در تگ <servlet> در فایل web.xml می باشد. تگ <servlet-name> یک آدرس مجازی یا مستعار برای سرولت واقعی تعریف می کند. این ویژگی امکان تغییر کلاس سرولت را بدون نیاز به دستکاری کد HTML یا سایرسرولت هایی که این سرولت را فراخوانی می نمایند ، فراهم می سازد. بعبارت دیگر می توان از یک کلاس سرولت با چندین نام مستعار استفاده کرد.

محل قرار گرفتن برنامه

ساختار مثال بصورت زیر است:

```
%TOMCAT_HOME%\webapps
                    \greeting
                              \src
                                  \GreetingServlet.java
                              \index.html
                              \WEB-INF
                                  \web.xml
                                  \classes
                                      \ GreetingServlet.class
```

ذکر چند نکته ضروری به نظر می رسد . به دایرکتوری src توجه نمایید ، لزوما نیازی نیست که دایرکتوری src تحت دایرکتوری webapps قرار بگیرد . دایرکتوری greeting نیز تحت دایرکتوری webapps قرار گرفته است و آدرس فراخوانی برنامه http://localhost:8080/greeting نیز به این دایرکتوری اشاره می کند. در حقیقت آدرس برنامه با نام دایرکتوری greeting یکسان در نظر گرفته شده است. یک راه حل بهتر اینست که کلیه این فایل ها را از زیر مجموعه tomcat خارج کرد. برای مثال کل برنامه را می توان به دایرکتوری c:\JavaPro\Chapter14 منتقل نمود. حال ساختار برنامه بصورت زیر است :

```
C:\JavaPro\Chapter14
                \greeting
                        \index.html
                          \WEB-INF
                                \web.xml
                                 \classes
                                     \ GreetingServlet.class
```

برای آگاه ساختن سرویس دهنده Tomcat از این تغییرات فایل %TOMCAT%\conf\server.xml را باز کرده و تگ <ContextManager> و </ContextManager> را بیابید. قبل از تگ </ContextManager> تگ تعریف محل جدید برنامه را وارد نمایید :

```
<Context path= "/greeting"
     codeBase= "c:\JavaPro\Chapter14\greeting">
</Context>
```

پس از اجرای مجدد Tomcat آدرس http://localhost:8080/greeting را در مرورگر وب فراخوانی نمایید. تگ <Context> دو مشخصه دارد ، مشخصه path مسیر نسبی برنامه و مشخصه codeBase محل فیزیکی قرار گرفتن برنامه را تعریف می کند. با توجه به این تعریف ، سرویس دهنده Tomcat تمام آدرس های را که دارای /greeting است را از دایرکتوری C:\JavaPro\Chapter14\greeting جستجو و فراخوانی می کند.

مروری بر Java Servlet API

تا اینجا مطالبی درباره برنامه های وب ، حامل های وب ، نیازهای برنامه نویسی و ساختار اولیه برنامه های وب گفته شد. در ادامه شما با Java Servlet API آشنا خواهید شد . این مجموعه شامل دو بسته javax.servlet و javax.servlet.http است و نیازهای زیر را برآورده می سازد :

- کلاس و رابط برای پیاده سازی سرولت
- کنترل خطا در سرولت
- پیکربندی سرولت
- چرخه حیات سرولت
- درخواست و پاسخ

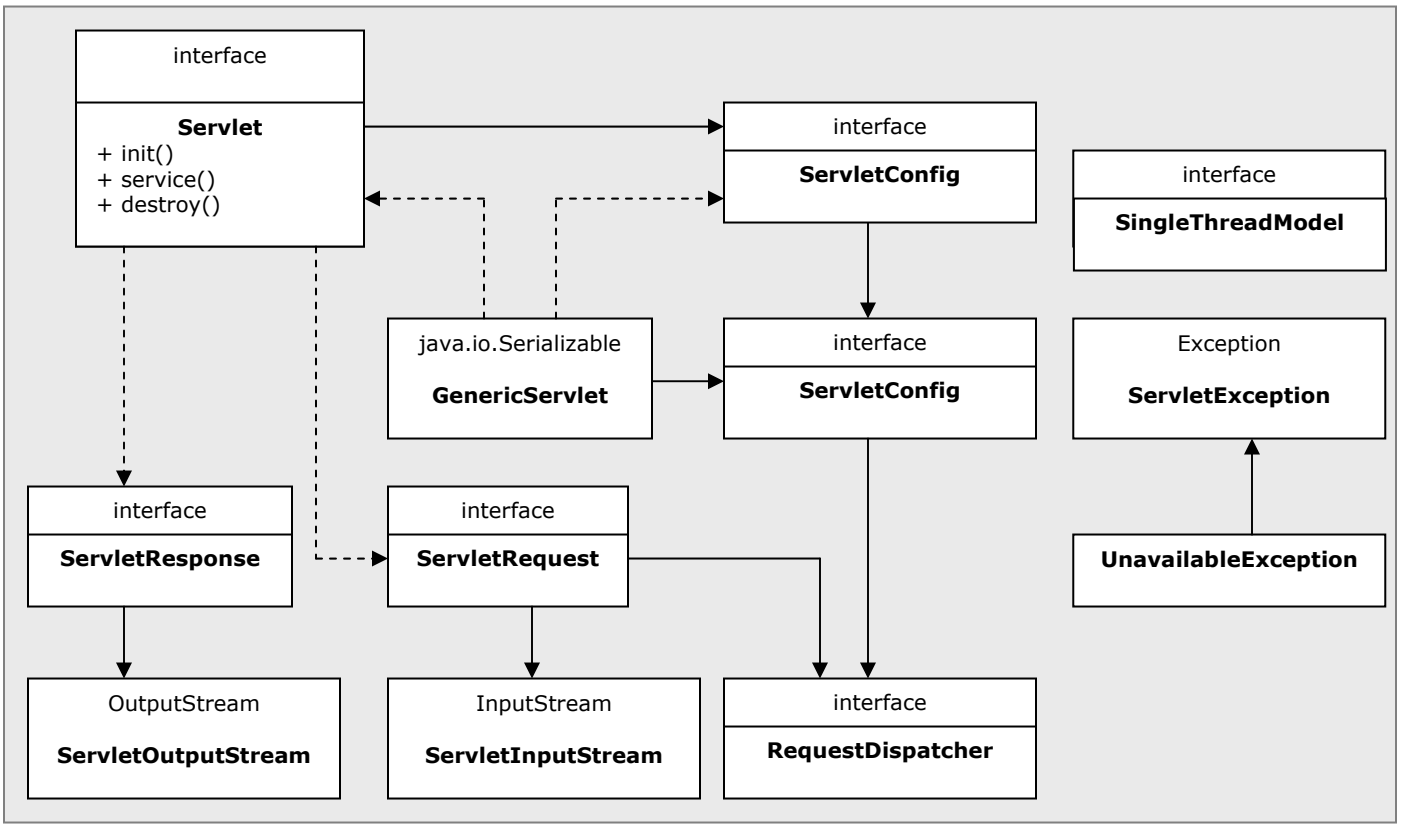
کلاس و رابط های موجود در بسته javax.servlet عمومی و مستقل از پروتکل است ، درحالیکه بسته javax.servlet.http شامل کلاس و رابط های مختص HTTP است و نیز تعدادی از کلاس و رابط های موجود در javax.servlet.http از کلاس و رابط های javax.servlet ارث می گیرند.

شکل زیر نمودار کلاس های بسته javax.servlet و روابط بین آنها را نشان می دهد. خطوط ساده بیانگر رابطه تجمعی و خطوط نقطه چین بیانگر وابستگی کلاس ها است.

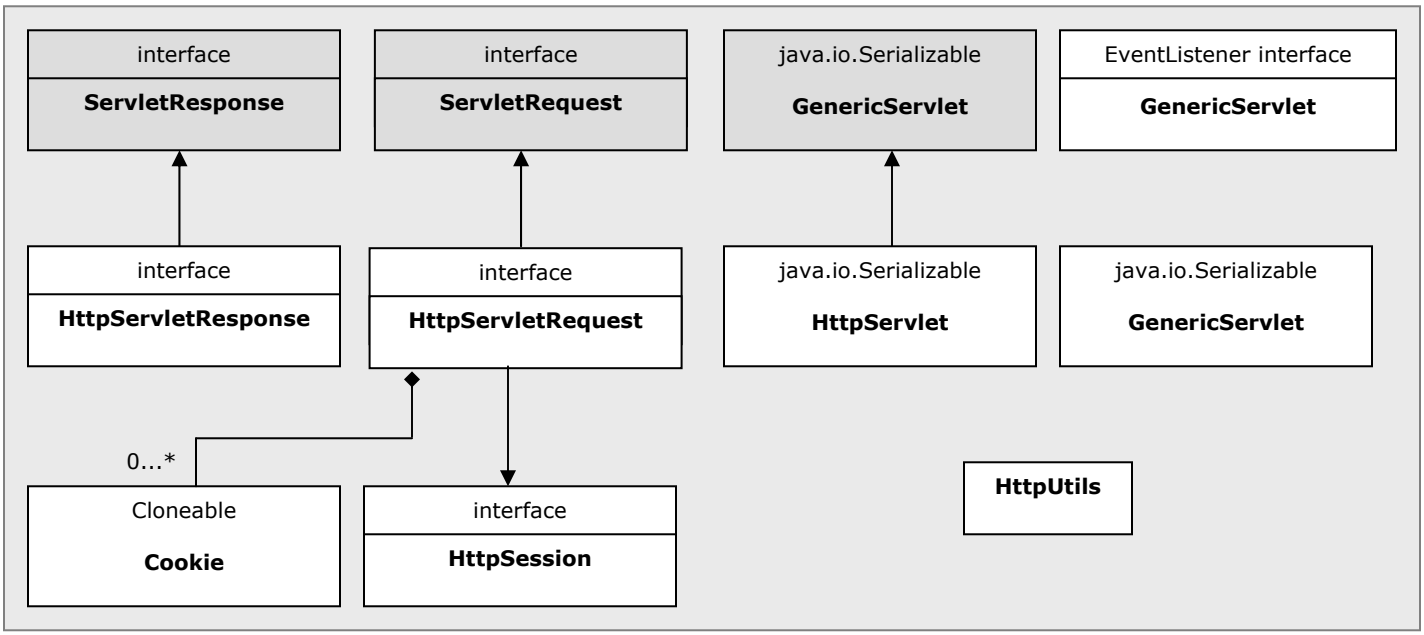
سوالی که مطرح می شود اینست که هدف از طراحی این کلاس ها و رابط ها چیست ؟سوالات زیر را نیز در نظر بگیرید :

- مدیریت سرولت ها بر عهده حامل است ، چگونه حامل وب چرخه حیات سرولت را مدیریت می کند ؟
- منطق برنامه چگونه اجرا می شود ؟
- چگونه سرولت پارامترهای درخواست را خوانده و پاسخ را تولید می کند ؟
- خطای زمان اجرای سرولت چگونه کنترل می شود ؟
- چگونه سرولت ها با حامل وب ارتباط برقرار می کند؟

پس از خواندن ادامه این فصل شما می توانید به این سوالات پاسخ دهید.



شکل زیر دیاگرام کلاس و رابط های بسته `javax.servlet.http` را نمایش می دهد.مستطیل های خاکستری کلاس های بسته `javax.servlet` را مشخص می کند :



جدول زیر مجموعه Java Servlet API را نشان می دهد. رابط ها بصورت پررنگ مشخص شده اند.

کلاس/ربط	نیاز
javax.servlet.Servlet **javax.servlet.SingleThreadModel** javax.servlet.GenericServlet javax.servlet.HttpServlet	پیاده سازی سرولت
javax.servlet.ServletConfig	پیکربندی سرولت
javax.servlet.ServletException javax.servlet.UnavailableException	کنترل خطا
javax.servlet.ServletRequest **javax.servlet.ServletResponse** javax.servlet.ServletInputStream javax.servlet.ServletOutputStream **javax.servlet.HttpServletRequest** **javax.servlet.HttpServletResponse**	درخواست و پاسخ
javax.servlet.http.HttpSession **javax.servlet.http.HttpSessionBindingListener** javax.servlet.http.HttpSessionBindingEvent	ردیابی تماس کاربر[18]
javax.servlet.ServletContext	محتوای سرولت[19]
javax.servlet.RequestDispatcher	تعامل بین سرولت ها[20]
javax.servlet.http.Cookie javax.servlet.http.HttpUtils	کلاس های کمکی

[18] Session Tracking
[19] Servlet Context
[20] Servlet Collaboration

جدول فوق کلاس ها و رابط های بسته های javax.servlet و javax.servlet.http را براساس نقش و وظیفه هر یک در تولید برنامه های مبتنی بر وب گروه بندی می کند.

- **پیاده سازی سرولت**

  کلاس و رابط های این گروه برای پیاده سازی سرولت در نظر گرفته شده اند. دو رابط javax.servlet.Servlet و javax.servlet.SingleThreadModel و دو کلاس انتزاعی javax.servlet.GenericServlet و javax.servlet.http.HttpServlet این نقش را برعهده دارند.

  برای نوشتن سرولت یک یا چند متد موجود در این رابط و کلاس ها باید پیاده سازی شوند و حامل وب با فراخوانی این متدها چرخه حیات سرولت ها را مدیریت و منطق برنامه را اجرا می کند.

- **پیکربندی سرولت**

  رابط javax.servlet.ServletConfig در این گروه قرار می گیرد. API سرولت روش های مختلفی برای دسترسی به شیء ServletConfig تخصیص یافته یک سرولت ارائه می کند. شیء تعریف شده از نوع ServletConfig امکان دسترسی به اطلاعات پیکربندی سرولت ، از جمله پارامترهای اولیه تعریف شده برای هر سرولت در فایل توصیف گر توسعه (web.xml) را فراهم می سازد.

- **کنترل خطا در سرولت**

  هر سرولت دو استثناء javax.servlet.ServletException و javax.servlet.UnavailableException را throws می کند و حامل وب آن را کنترل می نماید.

- **درخواست و پاسخ**

چهار رابط javax.servlet.ServletRequest ،
javax.servlet.ServletResponse ،
javax.servlet.http.HttpServletRequest و
javax.servlet.http.HttpServletResponse و دو کلاس انتزاعی
javax.servlet.ServletInputStream و javax.servlet.
ServletOutputStream در این گروه قرار گرفته اند. این اشیاء متدهای دسترسی به جریانات ورودی و خروجی برقرارشده با سرویس گیرنده ، خواندن اطلاعات درخواست و تولید پاسخ برای سرویس گیرنده را فراهم می سازند.

- **اشتراک اطلاعات**

رابط javax.servlet.ServletContext این امکان را فراهم می سازد که سرولت ها بتوانند در یک برنامه کاربردی اطلاعات مورد نیاز را به اشتراک بگذارند. علاوه براین سرولت ها با استفاده از متدهای این رابط می توانند به اطلاعات حامل وب دسترسی پیدا نمایند.

- **تعامل بین سرولت ها**

با استفاده از رابط javax.servlet.http.RequestDispatcher یک سرولت می تواند یک سرولت دیگر یا یک صفحه JSP و حتی صفحات HTML را فراخوانی نماید. این مکانیزم کمک می کند تا بتوان چرخه اجرای منطق برنامه را در چندین سرولت یا صفحات JSP کنترل کرد.

- **کلاس های کمکی**

در این گروه ، دو کلاس وجود دارد: javax.servlet.http.HttpUtils حاوی متدهای کمکی مختلف و کلاس javax.servlet.http.Cookie برای تعریف کوکی ها.

پیاده سازی سرولت

رابط Servlet

```
public interface Servlet
```

این رابط نحوه تعامل بین حامل وب و سرولت را تعریف می کند. برای نوشتن یک سرولت این رابط بصورت مستقیم یا غیر مستقیم پیاده سازی می شود. در بیشتر موارد این رابط با ارث بری کلاس سرولت از کلاس javax.servlet.GenericServlet یا javax.servlet.http.HttpServlet بطور غیر مستقیم پیاده سازی می شود.

با پیاده سازی رابط javax.servlet.Servlet پنج متد زیر نیز باید پیاده سازی شوند :

```
public void init(ServletConfig config)
public void service(ServletRequest request, ServletResponse response)
public void destroy()
public ServletConfig getServletConfig()
public String getServletInfo()
```

متد init()

```
public void init(ServletConfig config) throws ServletException
```

اولین باری که کلاس سرولت توسط حامل وب بارگذاری یا نمونه سازی[21] می شود، متد init() توسط حامل وب فراخوانی می شود.هدف این متد ، انجام اقدامات لازم برای مقداردهی های اولیه در سرولت قبل از پرداختن به درخواست HTTP است. حامل وب یک شیء از نوع ServletConfig را به متد init() ارسال می کند. همانطور که گفته شد یک سرولت می تواندبه اطلاعات پیکربندی برنامه از طریق شیء ServletConfig دسترسی پیدا کند. این اطلاعات در فایل web.xml تعریف شده است. متد init() استثناء ServletException را برای مواردی که در این متد خطای زمان اجرا رخ دهد ، throws می کند.

[21] Instantiated

طبق استاندارد های سرولت ، متد `init()` یکبار و تنها یکبار در زمان نمونه سازی کلاس سرولت توسط حامل وب فراخوانی می شود و قبلا از پرداختن به درخواست ارسال شده برای سرولت ، این متد بطور کامل اجرا شده است.

وظایف و اقداماتی که می توانند در متد `init()` انجام شوند عبارتند از :

- خواندن اطلاعات پیکربندی از منابع ماندگار مانند فایل های پیکربندی
- خواندن پارامترهای اولیه سرولت توسط شیء `javax.servlet.ServletConfig`
- انجام وظایفی که اجرای آن برای یکبار کافی است ، برای مثال بارگذاری درایور بانک اطلاعاتی ، فعال کردن یک منبع اتصال بانک اطلاعاتی[22]

متد `service()`

```
public void service(ServletRequest request, ServletResponse response
) throws ServletException, IOException
```

این متد نقطه ورودی یک سرولت برای اجرای منطق برنامه است. حامل وب متد `service()` را برای پاسخگویی به درخواست های سرویس گیرنده فراخوانی می کند. تنها در صورتیکه متد `init()` با موفقیت اجرا شده باشد ، متد `service()` قابل فراخوانی است. این متد دو آرگومان می پذیرد : شیء پیاده سازی شده از رابط `javax.servlet.ServletRequest` و شیء پیاده سازی شده از رابط `javax.servlet.ServletResponse`. شیء `request` متدهای دسترسی به اطلاعات درخواست سرویس گیرنده و شیء`response` متدهای آماده سازی و ساختن پاسخ را فراهم می سازد.

متد `destroy()`

```
public void destroy()
```

حامل وب این متد را قبل از حذف نمونه ایجاد شده یک سرولت فراخوانی می نماید. متد `destroy()` برای آزاد سازی منابع و حافظه اختصاص یافته برای یک سرولت در نظر گرفته شده است. قبل از اینکه

[22] Connection Pool

حامل وب این متد را فراخوانی نماید ، به ریسمان های در حال اجرای متد service() اجازه می دهد تا اجرایشان به اتمام برسد. بنابراین متد destroy() تا مادامی که متد service() در حال اجرا است ، فراخوانی نمی شود. پس از فراخوانی متد destroy() حامل وب درخواست های ارسال شده را برای سرولت راهنمایی نمی کند.

وظایف و اقداماتی که می توانند در متد destroy() انجام شوند عبارتند از :

- انجام اقدامات آزادسازی ، برای مثال آزاد سازی درایور بانک اطلاعاتی ، بستن یک منبع اتصال بانک اطلاعاتی و نیز آزاد سازی سیستم و برنامه های که سرولت از آن استفاده کرده است.

متد getServletConfig()

```
public ServletConfig getServletConfig()
```

این متد شیء ServletConfig که از طریق متد init() در اختیار سرولت قرار گرفته است را برمی گرداند.

متد getServletInfo()

```
public String getServletInfo()
```

این متد یک String حاوی اطلاعاتی درباره سرولت (برای مثال نویسنده سرولت ، تاریخ ایجاد وغیره) را برمی گرداند. می توان این متد را بمنظور سازماندهی اطلاعات یک سرولت و یا اضافه کردن موارد دیگر پیاده سازی نمود.

کلاس GenericServlet

```
public abstract class GenericServlet implements Servlet,
                                                ServletConfig,
                                                Serializable
```

کلاس GenericServlet یک پیاده سازی اولیه از رابط Servlet را فراهم می سازد. این کلاس انتزاعی است و در نتیجه تمام زیر کلاس های آن بایستی متد service() را پیاده سازی نمایند. این کلاس انتزاعی علاوه بر متدهای تعریف شده در javax.servlet.Servlet و javax.servlet.ServletConfigسه متد دیگر را نیز دارا است:

```
public init()
public void log(String message)
public void log(String message, Throwable t)
```

متد init(ServletConfig config) شیء ServletConfig را در یک متغیر ماندگار (بنام config) ذخیره می نماید. برای دسترسی به این شیء از متد getServletConfig() می توان استفاده کرد. اگر نیاز به بازنویسی این متد باشد ، عبارت super.init(config) در متد جدید باید ذکر گردد.

کلاس GenericServlet همچنین رابط ServletConfig را نیز پیاده سازی می کند. این امکان اجازه می دهد متدهای ServletConfig را بدون نیاز به داشتن یک شی ء از نوع ServletConfig فراخوانی نمود. این متدها عبارتند از : getInitParameter() ، getInitParameterNames() ، getServletContext() و getServletName().
هر یک از این متدها ، متد متناظر خود در شیء ServletConfig را فراخوانی می نمایند.

کلاس GenericServlet دو متد دیگر را نیز برای نوشتن log سرولت ارائه می کند که متدهای متناظرشان در ServletContext را فراخوانی می نماید. اولین متد log(String message) نام سرولت و پیغام مشخص شده توسط آرگومان message را در log حامل وب می نویسد. متد دوم log(String message, Throwable cause) علاوه بر نام سرولت و پیغام ، اطلاعات استثناء مشخص شده توسط آرگومان cause را در log حامل وب چاپ می کند. پیاده سازی واقعی مکانیزم log به نوع حامل وب برمی گردد ، اما بیشتر حامل ها از فایل های متن برای مکانیزم log استفاده می کنند. برای مثال Tomcat از فایل %TOMCAT_HOME%\log\servlet.log برای ثبت log سرولت های برنامه کمک می گیرد.

کلاس HttpServlet

```
public abstract class HttpServlet extends GenericServlet
                                  implements Serializable
```

کلاس HttpServlet از کلاس GenericServlet ارث می برد و یک پیاده سازی مختص HTTP از رابط Servlet را ارائه می نماید. این بدین معنی است که تمام سرولت ها از این کلاس ارث می برند. این کلاس دارای متدهای زیر است :

```
public void service(ServletRequest request, ServletResponse response)

protected void service(HttpServletRequest request,
                       HttpServletResponse response)

protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)

protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)

protected void doDelete(HttpServletRequest request,
                        HttpServletResponse response)

protected void doOptions(HttpServletRequest request,
                         HttpServletResponse response)

protected void doPut(HttpServletRequest request,
                     HttpServletResponse response)

protected void doTrace(HttpServletRequest request,
                       HttpServletResponse response)
```

متدهای service()

کلاس HttpServlet دو متد service() دارد :

```
public void service(ServletRequest request, ServletResponse response)
         throws ServletException, IOException
```

این متد پیاده سازی شده متد service() از کلاس GenericServlet است. این متد دو شیء request و response را به HttpServletRequest و HttpServletResponse تبدیل و متد service() زیر را فراخوانی می کند :

```
protected void service(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException
```

این متد تحریف شده ، دو شیء request و response مختص HTTP را بعنوان آرگومان دریافت کرده و متد service() نخست را فراخوانی می کند. کلاس HttpServlet این متد را برای اداره کردن درخواست های HTTP پیاده سازی کرده است. متد getMethod() در رابط javax.servlet.ServletRequest نوع متد درخواست HTTP را مشخص می کند. بعنوان مثال برای درخواست های GET ، این متد مقدار "GET" را برمی گرداند. متد service() با استفاده از این متد درخواست را به یکی از متدهای doGet() ، doPost() ، doOptions() ، doDelete() ، doPut() و doTrace() واگذار می کند.

ترتیب فراخوانی متد های service() هنگامیکه حامل وب درخواستی را برای یک سرولت دریافت می کند ، بصورت زیر است :

- حامل وب متد عمومی service() (متد اول) را فراخوانی می نماید.
- متد عمومی service() ، متد محافظت شده service() (متد دوم) را پس از تبدیل آرگومان های آن به HttpServletRequest و HttpServletResponse فراخوانی می کند.
- متد محافظت شده service() یکی از متدهای doXXX() را با توجه به نوع متد درخواست HTTP فراخوانی می کند.

متدهای doXXX()

کلاس HttpServlet متدهای محافظت شده زیر را به ازای هر متد درخواست HTTP پیاده سازی می کند :

```
protected void doGet(HttpServletRequest request, HttpServletResponse
                     response) throws ServletException, IOException

protected void doPost(HttpServletRequest request, HttpServletResponse
                      response) throws ServletException, IOException

protected void doDelete(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException

protected void doOptions(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException


protected void doPut(HttpServletRequest request, HttpServletResponse
                     response) throws ServletException, IOException

protected void doTrace(HttpServletRequest request, HttpServletResponse
                       response) throws ServletException, IOException
```

امضای هر یک از این متدهای `doXXX()` مشابه متد محافظت شده `service()` است ، هر متد دارای دو آرگومان `HttpServletRequest` و `HttpServletResponse` است و دو استثنای `ServletException` و `IOException` را `throws` می کنند.

کلاس `HttpServlet` متدهای متناسب با درخواست های `TRACE` و `OPTIONS` را پیاده سازی کرده است و نیازی به بازنویسی متدهای `doTrace()` و `doOptions()` نیست.

متد `getLastModified()`

```
protected long getLastModified(HttpServletRequest request)
```

این متد زمان آخرین تغییر سرولت را از مبدا `January 1, 1970 00:00:00 GMT` برمی گرداند. پیاده سازی اولیه این متد ، درصورتی که زمان تغییر سرولت مشخص نباشد ، مقدار `-1` را برمی گرداند.

پیکربندی سرولت

یک شیء از نوع javax.servlet.ServletConfig اطلاعات پیکربندی سرولت ها را در خود نگه می دارد. اطلاعات پیکربندی عبارتند از : پارامترهای اولیه (شامل نام و مقدار) ، نام سرولت و یک شیء ازنوع javax.servlet.ServletContext حاوی اطلاعاتی درباره حامل وب.

پارامترهای اولیه و نام سرولت در توصیف گرتوسعه (web.xml) تعریف می شوند :

```
<web-app>
  <servlet>
    <servlet-name>Admin</servlet-name>
    <servlet-class>com.wrox.admin.AdminServlet</servlet-class>
    <init-param>
      <param-name>email</param-name>
      <param-value>admin@admin.wrox.com</param-value>
    </init-param>
    <init-param>
      <param-name>helpURL</param-name>
      <param-value>/admin/help/index.html</param-value>
    </init-param>
  </servlet>
</web-app>
```

این مثال یک سرولت بنام Admin بهمراه دو پارامتر اولیه بنام email و helpURL تعریف می کند. حامل وب این اطلاعات را از طریق javax.servlet.ServletConfig بازیابی و در اختیار سرولت com.wrox.admin.AdminServlet قرار می دهد. بدیهی است تغییر این پارامترها بدون نیاز به کامپایل مجدد سرولت امکان پذیر است.

رابط ServletConfig

```
public interface ServletConfig
```

این رابط متدهای زیر را تعریف می کند :

```
public String getInitParameter(String name)
public Enumeration getInitParameterNames()
public ServletContext getServletContext()
```

```
public String getServletName()
```

متد getInitParameter()

```
public String getInitParameter(String name)
```

این متد مقدار پارامتر اولیه با نام name را برمی گرداند. اگر پارامتر مورد نظر در فایل web.xml تعریف شده باشد ، این متد مقدار آن و در غیر این صورت مقدار null را برمی گرداند. در مثال بالا ، فراخوانی متد getInitParameter() با آرگومان "email" مقدار "admin@admin.wrox.com" را برمی گرداند.

متد getInitParameterNames()

```
public Enumeration getInitParameterNames()
```

این متد یک مجموعه شمارشی Enumeration حاوی تمام پارامترهای اولیه یک سرولت را بر می گرداند. با استفاده از این مجموعه می توان نام همه پارامترهای اولیه سرولت را یکی پس از دیگری مشخص کرد. در مثال بالا فراخوانی متد getInitParameterNames() یک مجموعه شمارشی شامل دو شیء از نوع String را برمی گرداند : "email" و "helpURL".

متد getServletContext()

```
public ServletContext getServletContext()
```

این متد شیء ServletContext تخصیص یافته برای حامل وب را بر می گرداند. این شیء اطلاعات مشترک بین تمام سرولت ها و نیز مکانیزم هایی مانند logging را عرضه می کند.

متد getServletName()

```
public String getServletName()
```

این متد نام تعریف شده در فایل web.xml برای یک سرولت را برمی گرداند. اگر نامی برای سرولت تعریف نشده باشد ، این متد نام کلاس سرولت را برمی گرداند.

راههای دسترسی به ServletConfig

یک سرولت می تواند به دو روش زیر به شیء javax.servlet.ServletConfig دسترسی داشته باشد :

- درزمان مقداردهی اولیه سرولت

همانطور که قبلا گفته شد ، متد init() رابط Servlet و کلاس GenericServlet یک آرگومان از نوع javax.servlet.ServletConfig دارد. در زمان مقدار دهی اولیه سرولت ، حامل وب این آرگومان را تولید و به متد init() ارسال می نماید. با بازنویسی این متد در کلاس سرولت می توان به شیء ServletConfig دسترسی پیدا کرد.
در صورت بازنویسی این متد در یک سرولت ، ذکر جمله super.init() الزامی است :

```
public void init(ServletConfig config) {
      super.init(config);

     //Initialization here
}
```

یک سرولت می تواند با فراخوانی متد getServletConfig() نیز به شیء ServletConfig دسترسی داشته باشد. این متد در رابط javax.servlet.Servlet تعریف شده است.

سرولت هایی که از کلاس GenericServlet و یا زیر کلاس آن HttpServlet ارث بری دارند ، می توانند به متدهای رابط ServletConfig دسترسی داشته باشند. چراکه کلاس GenericServlet رابط ServletConfig را نیز پیاده سازی می کند.

کنترل خطا در سرولت

بسته javax.servlet دو کلاس برای کنترل استثناء تعریف کرده است:

```
javax.servlet.ServletException
javax.servlet.UnavailableException
```

کلاس ServletException

```
public class ServletException extends java.lang.Exception
```

این کلاس یک استثنای عمومی است که توسط متدهای init() ، service() ، doXXX() و destroy() اداره می شود. این کلاس دو سازنده دارد :

```
public ServletException()
public ServletException(String message)
```

کلاس UnavailableException

```
public class UnavailableException extends ServletException
```

این کلاس javax.servlet.UnavailableException نوع خاصی از استثنات سرولت است. بدلیل اینکه این کلاس از javax.servlet.ServletException ارث می برد ، تمام متدهای سرولت ها می توانند استثنای javax.servlet.Exception راthrows نمایند.
این استثناء به حامل وب نشان می دهد (نمایان می کند) که سرولت بصورت موقت یا دائم قابل دسترس نیست.این کلاس دو سازنده دارد:

```
public UnavailableException(String message)
```

این سازنده یک استثنای قابل دسترس نبودن سرولت را با پیغام داده شده ایجاد می کند.

```
public UnavailableException(String message, int seconds)
```

آرگومان seconds مدت زمانی که سرولت در دسترس نیست را برحسب ثانیه مشخص می کند.

چرخه حیات سرولت

همانطور که قبلا گفته شد ، حامل وب یک محیط زمان اجرا است که اجرای سرولت و صفحات JSP را مدیریت می کند. یکی از مسئولیت های یک حامل ، مدیریت چرخه حیات سرولت است. رویدادهای چرخه حیات در رابط javax.servlet.Servlet تعریف شده اند. اگرچه مدیریت و کنترل چرخه حیات سرولت برعهده حامل سرولت است ، اما برنامه نویس نیز می تواند این رویدادها را دنبال نماید. متدهای مربوط به چرخه حیات سرولت عبارتند از : init() ، service() و destroy() . چرخه حیات سرولت با فراخوانی متد init() توسط حامل وب شروع و با فراخوانی متد destroy() توسط حامل وب به پایان می رسد.

معمولا چرخه حیات سرولت شامل مراحل زیر است :

- نمونه سازی[23] : حامل وب یک نمونه از کلاس سرولت ایجاد می کند.
- مقداردهی اولیه[24] : حامل وب متد init() سرولت را فراخوانی می کند.
- سرویس [25] : اگر درخواستی برای یک سرولت وجود داشته باشد ، حامل وب متد service() سرولت را فراخوانی می کند.
- از بین رفتن[26] : قبل از بین بردن نمونه سرولت ، حامل وب متد destroy() سرولت را فراخوانی می کند.
- عدم دسترسی[27] : نمونه سرولت غیرقابل دسترس می شود.

حامل وب یک نمونه از کلاس سرولت را برای پاسخگویی به درخواست HTTP سرویس گیرنده یا اینکه در startup حامل وب ایجاد می کند. پس از ساختن (ایجاد) نمونه ، حامل وب متد init() سرولت را فراخوانی می کند. پس از این مقدار دهی اولیه ، نمونه سرولت آماده پاسخگویی به درخواستهای

---

[23] Instantiation
[24] Initialization
[25] Service
[26] Destroy
[27] Unavailable

ارسال شده توسط سرویس گیرنده ها است . هدف از این مقدار دهی اولیه ، بارگذاری پارامترهای اولیه مورد نیاز سرولت است.
در طی فرآیند مقداردهی اولیه ، یک نمونه سرولت می تواند استثنای ServletException یا UnavailableException را throws کند. در صورتی که مقدار دهی اولیه با خطای زمان اجرا مواجه شود، فراخوانی متد service() سرولت غیر ممکن خواهد بود.

حامل تضمین می کند که متد init() هر سرولت قبل از فراخوانی متد service() و نیز متد destroy() قبل از بین بردن نمونه سرولت اجرا می شود.

در عمل ، حامل وب سرولت ها را در زمان startup یا با اولین فراخوانی سرولت ، مراحل بارگذاری و مقداردهی اولیه را انجام می دهد ، و نمونه سرولت را برای پاسخگوئی به درخواست های سرویس گیرنده در حافظه نگهداری می کند. حامل وب ممکن است هر زمان که تصمیم بگیرد نمونه سرولت را از حافظه پاک کند ، چرخه حیات سرولت پایان می پذیرد. برای مثال اگر سرولت برای مدت زیادی فراخوانی نشود یا حامل وب shut down شود ، حامل وب متد destroy() سرولت را فراخوانی می کند.

در یک مدل معمولی چرخه حیات سرولت ، حامل وب یک نمونه واحد از هر سرولت ایجاد می کند. در این صورت اگر حامل وب در حال اجرای متد service() یک سرولت باشد و درخواست دیگری برای سرولت دریافت شود ، چه اتفاقی می افتد ؟ برای سرولت هایی که رابط javax.servlet.SingleThreadModel را پیاده سازی نکنند ، حامل وب از تنها نمونه سرولت استفاده می کند و به هر درخواست سرویس گیرنده یک ریسمان از سرولت را اختصاص می دهد. برای سرولت های که رابط SingleThreadModel را پیاده سازی می کنند ، حامل وب یا درخواست های ارسال شده برای سرولت را سریال بندی می کند یا منبعی از نمونه های یک سرولت ایجاد می کند و به هر درخواست یک نمونه سرولت جداگانه اختصاص می دهد.

یک حالت دیگر نیز وجود دارد که یک حامل وب می تواند چند نمونه از یک سرولت را بطور همزمان نگهداری کند ، اگر در فایل توصیف گر توسعه (web.xml) یک سرولت بیش از یکبار و با چند نام مختلف معرفی شده باشد ، در اینصورت درآن واحد چندین نمونه از یک سرولت توسط حامل وب ایجاد و مدیریت می شود.

مثال چرخه حیات سرولت

برای فهم بهتر چرخه حیات سرولت و مراحل آن به یک مثال می پردازیم. در این مثال یک سرولت بنام `FreakServlet` مراحل مختلف چرخه حیات یک سرولت را دنبال می کند. کد زیر را در فایل `C:\JavaPro\Chapter14\freakServlet\src\FreakServlet.java` ذخیره نمایید:

```
//Freakservlet.java

//Import servlet packages
import javax.servlet.*;
import javax.servlet.http.*;

//import other Java packages
import java.io.*;
import java.util.*;

public class Freakservlet extends HttpServlet {
    java.util.Vector states;
    java.util.Random random;
    int waitInterval;
    public static final int DEFAULT_WAIT_INTERVAL = 10;

    public Freakservlet() {
        states = new java.util.Vector();
        random = new java.util.Random();
        waitInterval = DEFAULT_WAIT_INTERVAL;
        states.add(createState("Instantiation"));
    }

    public void init() throws ServletException {
        states.add(createState("Initialization"));
        String waitIntervalstring =
            getServletConfig().getInitParameter("waitInterval");
        if (waitIntervalstring != null) {
            waitInterval = new Integer(waitIntervalstring).intValue();
        }
    }
```

```
    protected void doGet(HttpServletRequest request,
                HttpServletResponse response) throws ServletException,
            IOException {
        if (random.nextBoolean()) {
            // Not available for waitInterval seconds.
            states.add(createState("Unavailable from doGet"));
            throw new UnavailableException("Unavailable from doGet",  waitInterval);
        }

        states.add(createState("Service"));
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Send acknowledgment to the browser
        out.println("<HTML>");
        out.println("<META HTTP-EQUIV=\"pragma\"CONTENT=\"no-cache\">");
        out.println("<HEAD><TITLE>");

        out.println("FreakServlet: State History");
        out.println("</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<H1>FreakServlet: State History</H1>");

        out.println("<a href=\"/lifeCycle/servlet/freak\">Reload</a></p>");

        for (int i = 0; i < states.size(); i++) {
            out.println("<P> " + states.elementAt(i) + "</P>");
        }
        out.println("</BODY></HTML>");
        out.close();
    }

    public void destroy() {
        states.add(createState("Destroy"));
        log("Flushing state history of LifeCycleTest servelt.");
        for (int i = 0; i < states.size(); i++) {
            log(states.elementAt(i).toString());
        }
    }

    private String createState(String message) {
        return "[" + (new java.util.Date()).toString() + "] " + message;
    }
}
```

در فایل توصیف گر توسعه web.xml موجود در دایرکتوری زیر ، سرولت و صفحه کنترل خطا را بصورت زیر تعریف کنید :

C:\JavaPro\Chapter14\freakServlet\WEB-INF

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app-2.2.dtd">

<web-app>
   <servlet>
      <servlet-name>freak</servlet-name>
      <servlet-class>FreakServlet</servlet-class>
      <init-param>
         <param-name>waitInterval</param-name>
         <param-value>5</param-value>
      </init-param>
   </servlet>

   <error-page>
      <exception-type>javax.servlet.UnavailableException</exception-type>
      <location>/unavailable.html</location>
   </error-page>
</web-app>
```

در فایل %TOMCAT_HOME%\conf\server.xml نیز تگ زیر را اضافه کنید :

```
<Context path="/lifeCycle"
   docBase="c:\JavaPro\chapter14\freakServlet">
</Context>
```

حال سرولت را کامپایل و کلاس آنرا در دایرکتوری

C:\JavaPro\Chapter14\freakServlet\WEB-INF\classes قرار دهید.

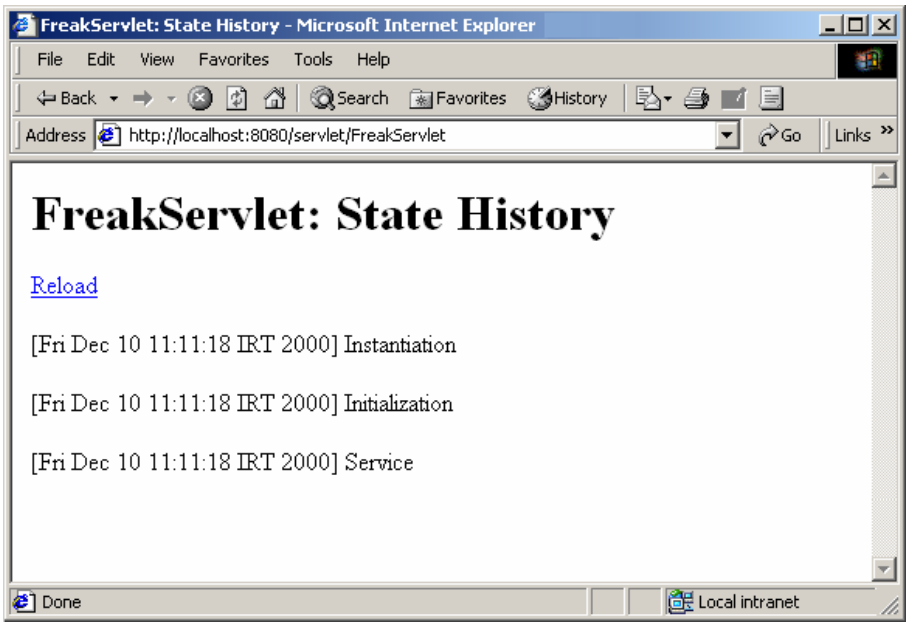
و در انتها کد زیر را در فایل

C:\JavaPro\Chapter14\freakServlet\unavailable.html ذخیره نمایید :

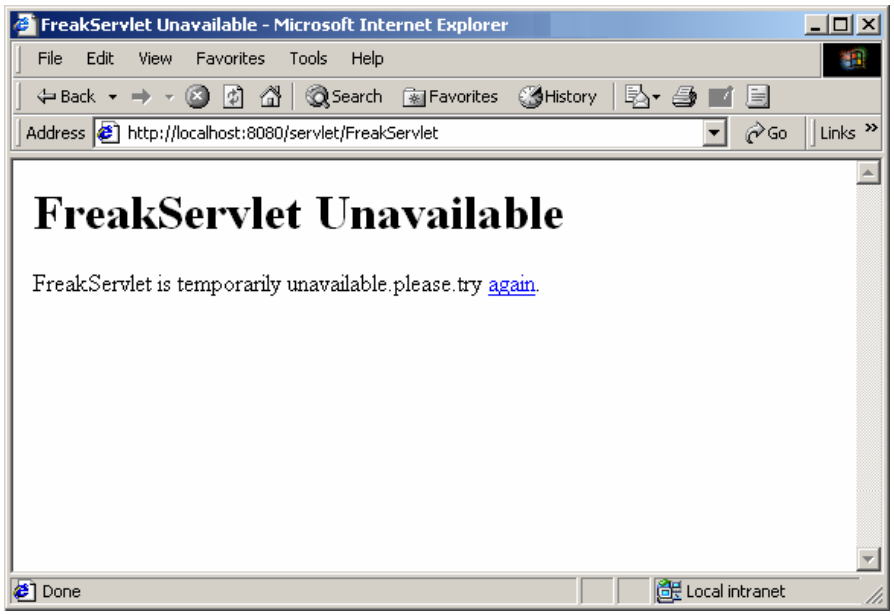
```
<HTML>
   <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
   <HEAD>
   <TITLE>FreakServlet Unavailable</TITLE>
   </HEAD>
   <BODY>
      <H1>FreakServlet Unavailable</H1>

      <P>FreakServlet is temporarily unavailable.please.try
      <A HREF="/lifeCycle/servlet/freak">again</A>.</P>
   </BODY>
</HTML>
```

برای تست برنامه ، آدرس `http://localhost:8080/lifecycle/servlet/freak` را در مرورگر وب وارد نمایید. با استفاده از دکمه `reload/refresh` مرورگر چند بار سرولت را فراخوانی کنید. دو نوع پاسخ در مرورگر مشاهده می شود. مراحل مختلف چرخه حیات سرولت در سرولت `FreakServlet` نشان داده می شود :



زمانیکه سرولت غیرقابل دسترس می شود ، خروجی سرولت بصورت زیر است :



برای اتمام تست سرولت `FreakServlet` سرویس دهنده `Tomcat` را `Shutdown` نمایید. برای این کار از `Ctrl-C` استفاده نکنید ، بلکه فایل اجرای مربوطه را در مسیر `%TOMCAT_HOME%\bin` را اجرا نمایید. این کار به سرویس دهنده `Tomcat` این شانس/اجازه را می دهد تا مکانیزم `logging` را کامل کند. فایل `%TOMCAT_HOME%\bin\logs\servlet.log` را باز کنید ، پیغام هایی مشابه زیر در آن مشاهده خواهید کرد:

```
freak:  [Fri Dec 10 11:11:18 GMT+01:00 2000] Instantiation
freak:  [Fri Dec 10 11:11:18 GMT+01:00 2000] Initialization
freak:  [Fri Dec 10 11:11:18 GMT+01:00 2000] Unavailable from doGet
freak:  [Fri Dec 10 11:11:27 GMT+01:00 2000] Unavailable from doGet
freak:  [Fri Dec 10 11:11:28 GMT+01:00 2000] Service
freak:  [Fri Dec 10 11:11:35 GMT+01:00 2000] Service
freak:  [Fri Dec 10 11:11:35 GMT+01:00 2000] Service
freak:  [Fri Dec 10 11:11:36 GMT+01:00 2000] Unavailable from doGet
freak:  [Fri Dec 10 11:12:96 GMT+01:00 2000] Destroy
```

لیست بالا همه وضعیت های چرخه حیات این سرولت را نمایش می دهد. قابل توجه است که ترتیب پیغامی که شما خواهید دید با لیست بالا متفاوت خواهد بود.

حال چگونگی تولید این پیغام ها مورد بررسی قرار می گیرد .

نمونه سازی

در این مرحله نمونه سرولت توسط حامل وب ایجاد می شود:

```
public Freakservlet() {
      states = new java.util.Vector();
      random = new java.util.Random();
      waitInterval = DEFAULT_WAIT_INTERVAL;
      states.add(createState("Instantiation"));
 }
```

این سازنده یک `Vector` و یک تولید کننده عدد تصادفی را مقداردهی می کند. در این مثال از یک `Vector` بنام `state` برای ذخیره کردن مراحل چرخه حیات سرولت استفاده می شود. هر مرحله با یک رشته متناسب بهمراه تاریخ و زمان وقوع آن نمایش داده می شود. از تولید کننده عدد تصادفی نیز برای `throws` تصادفی استثنای `UnavailableException` در متد `doGet()` استفاده می شود.

مقداردهی اولیه

دومین مرحله در هر سرولت ، مرحله مقداردهی اولیه سرولت است :

```
public void init() throws ServletException {
      states.add(createState("Initialization"));
      String waitIntervalstring =
         getServletConfig().getInitParameter("waitInterval");
      if (waitIntervalstring != null) {
         waitInterval = new Integer(waitIntervalstring).intValue();
      }
   }
```

این متد وضعیت "Initialization" را به Vector نگهدارنده state اضافه می کند. همچنین پارامتر اولیه "waitInterval" را از اطلاعات پیکربندی سرولت استخراج می کند. سرولت FreakServlet از این متغیر در زمان throws کردن استثنای UnavailableException استفاده می کند. پارامترهای اولیه سرولت در فایل web.xml و توسط تگ <init-param> صورت می گیرد.

```
<init-param>
        <param-name>waitInterval</param-name>
        <param-value>5</param-value>
</init-param>
```

در تگ بالا پارامتر اولیه بنام "waitInterval" با مقدار 5 ثانیه تعریف می شود.

زمانیکه حامل وب یک برنامه وب را بارگذاری می کند ، پارامترهای اولیه تعریف شده برای برنامه را نیز بارگذاری می کند. در حین بارگذاری برنامه ،حامل وب یک شیء از نوع ServletConfig را به ازای هر سرولت ایجاد می کند. این شیء توسط متد getServletConfig() در دسترس است. متد getInitParameter() در حالتی که نام پارامتر خواسته شده در فایل web.xml تعریف نشده باشد ، مقدار null را برمی گرداند.

سرویس

سومین مرحله از چرخه حیات سرولت ، مرحله سرویس است. در متد doGet() سرولت FreakServlet این مرحله انجام می شود. این متد یک متغیر تصادفی از نوع boolean تولید می شود ، اگر این متغیر برابر true باشد استثنای UnavailableException کنترل می شود این بدین معنی است که سرولت تا طی شدن زمان waitInterval در دسترس نخواهد بود :

```
if (random.nextBoolean()) {
    // Not available for waitInterval seconds.
    states.add(createState("Unavailable from doGet"));
    throw new UnavailableException("Unavailable from doGet",  waitInterval);
}
```

قبل از throws کردن استثناء ، سرولت وضعیت "Unavailable from doGet" را به Vector نگهدارنده state اضافه می کند. در این حالت مرورگر وب یک صفحه مبنی بر در دسترس نبودن سرولت را نمایش می دهد. این صفحه چگونه فراخوانی می شود ؟

زمانی که حامل وب استثنای UnavailableException (یا هر استثنای رخ داده در متد doGet ()) را دریافت کند ، در صورتی که برای این استثناء در فایل web.xml صفحه ای تعریف شده باشد ، این صفحه را نمایش می دهد. در مثال FreakServlet ، توصیف گر توسعه صفحه unavailable.html را تعریف می کند :

```
<error-page>
    <exception-type>javax.servlet.UnavailableException</exception-type>
    <location>/unavailable.html</location>
</error-page>
```

خواهد داد. از این مکانیزم می توان برای نمایش پیغام های مناسب تر برای خطاها استفاده کرد. اگر از این مکانیزم استفاده نشود ، در صورت بروز استثناء در یک سرولت ، حامل وب پیغام اصلی خطا را نمایش می دهد.

اگر متغیر تصادفی تولید شده برابر false باشد ، سرولت FreakServlet یک صفحه برای نمایش وضعیت و مراحل چرخه حیات سرولت تولید می کند. از متدهای رابط HttpServletResponse برای نوع محتوای خروجی و تولید آن استفاده شده است :

```
    states.add(createState("Service"));
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Send acknowledgment to the browser
    out.println("<HTML>");
    out.println("<META HTTP-EQUIV=\"pragma\"CONTENT=\"no-cache\">");
    out.println("<HEAD><TITLE>");

    out.println("FreakServlet: State History");
    out.println("</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<H1>FreakServlet: State History</H1>");

    out.println("<a href=\"/lifeCycle/servlet/freak\">Reload</a></p>");

    for (int i = 0; i < states.size(); i++) {
        out.println("<P> " + states.elementAt(i) + "</P>");
    }
    out.println("</BODY></HTML>");
    out.close();
}
```

تولید پاسخ شامل چهار مرحله زیر است :

- اولین مرحله ، تعریف نوع محتوای پاسخ است. برنامه دریافت کننده پاسخ (مرورگر) از این اطلاعات برای چگونگی رفتار با پاسخ استفاده می کند. در این مثال خروجی از نوع HTML است ، بنابراین مقدار "text/html" انتخاب شده است.

- دومین مرحله ، گرفتن یک شیء از نوع PrintWriter از شیء پاسخ است. کلاس java.io.PrintWriter از کلاس انتزاعی java.io.Writer ارث می گیرد. در سرولت ها ، حامل وب یک شی ء PrintWriter را با استفاده از شیء java.io.OutputStream اختصاص یافته به اتصال بین سرویس گیرنده و حامل وب ، می سازد. در نتیجه برنامه نویس قادر به تولید خروجی خواهد بود.

- به تگ META صفحه HTML تولید شده توجه نمایید ، این تگ به مرورگر اعلام می کند که هیچ گاه این صفحه را در خود ذخیره نکند ، تعریف این تگ در صفحه HTML باعث می شود که با هر فراخوانی صفحه ، مرورگر صفحه را از سمت سرویس دهنده دوباره دریافت کند.

- کلاس PrintWriter دارای چندین متد برای چاپ انواع داده در خروجی است. در این مثال از متد println() با آرگومانی از نوع java.lang.String استفاده شده است.

در پایان کار ، بهتر است شیء PrintWriterبسته شود .

ازبین رفتن

این مرحله آخرین مرحله از چرخه حیات یک سرولت است. در مثال FreakServlet متد destroy() قبل از shut down شدن سرویس دهنده Tomcat فراخوانی شده است:

```
public void destroy() {
      states.add(createState("Destroy"));
      log("Flushing state history of LifeCycleTest servelt.");
      for (int i = 0; i < states.size(); i++) {
         log(states.elementAt(i).toString());
      }
   }
```

این متد رشته "Destroy" را به Vector نگهدارنده state اضافه می کند و بدین ترتیب مکانیزم

دنبال کردن مراحل مختلف چرخه حیات سرولت را با ثبت این مراحل با استفاده از متد log() به پایا می رسد. بیشتر حامل ها از فایل های متنی برای مکانیزم logging استفاده می کند. در سرویس دهنده Tomcat پیغام های logدر مسیر %TOMCAT_HOME%\logs\ ذخیره می شوند.

مبحث چرخه حیات سرولت در همین جا به پایان می رسد ، نکات زیر در مورد چرخه حیات سرولت ها قابل توجه است :

- اشیای تعریف شده از نوع ServletConfig ، ServletContext ، HttpServletRequest و HttpServletResponse فقط در محدوده بین فراخوانی متدهای init() و destroy() معتبر هستند. این بدین معنی است که نباید این اشیاء قبل از فراخوانی متد init() و نیز بعد از فراخوانی متد destroy() مورد استفاده قرار گیرند. برای مثال سازنده سرولت نمی تواند از متد log() استفاده نماید ، چرا که این متد توسط شیء ServletContext سرولت پیاده سازی می شود. بنابراین نمونه سرولت قبل از فراخوانی متد init() معتبر نخواهد بود.

- Vector در مثال سرولت FreakServlet بعنوان یک متغیر عضو استفاده شده است. از آنجاییکه این سرولت از نوع SingleThreadModel نیست ، حامل وب تنها یک نمونه از این سرولت را ایجاد می کند وتمام درخواست ها بصورت همزمان در متد doGet() اجرا می شوند. علاوه براین چون متغیر state متغیر عضو است ، عمل خواندن و نوشتن بر روی این متغیر بایستی بصورت Thread-Safe صورت گیرد. در کلاس java.util.Vector متدهای add() و size() بصورت synchronized هستند. بنابراین چندین کاربر می توانند بصورت همزمان به سرولت FreakServlet دسترسی پیدا کنند.

درخواست و پاسخ

رابط های javax.servlet.HttpServletRequest و
javax.servlet.HttpServletResponse کلاس هایی هستند که سرولت ها برای دسترسی به درخواست و پاسخ HTTP از آنها استفاده می کنند. کلاس ها و رابط های مربوط به درخواست و پاسخ عبارتند از :

- javax.servlet.ServletRequest
- javax.servlet.ServletResponse
- javax.servlet.ServletInputStream
- javax.servlet.ServletOutputStream
- javax.servlet.http.HttpServletRequest
- javax.servlet.http.HttpServletRequest

رابط HttpServletRequest و HttpServletResponse مختص HTTP و از
ServletRequest وServletResponse ارث می گیرند. دو کلاس
ServletInputStream و ServletOutputStream برای خواندن و نوشتن در استریم ورودی و خروجی بین سرولت و سرویس دهنده استفاده می شود.

رابط ServletRequest

```
public interface ServletRequest
```

این رابط یک درخواست انتزاعی برای یک سرولت تعریف می کند. حامل وب یک نمونه از این رابط را هنگام فراخوانی متد service() کلاس GenericServlet یا HttpServlet ایجاد می کند.
لیست کامل متدهای این رابط عبارتند از :

```
public Object getAttribute(String name)
public void setAttribute(String name, Object attribute)
public Enumeration getAttributeNames()
public void removeAttribute(String name)
public Locale getLocale()
```

```
public Enumeration getLocales()
public String getCharacterEncoding()
public int getContentLength()
public String getContentType()
public ServletInputStream getInputStream()
public String getParameter(String name)
public Enumeration getParameterNames()
public Enumeration getParameterValues()
public String getProtocol()
public String getScheme()
public String getServerName()
public int getServerPort()
public BufferReader getReader()
public String getRemoteAddr()
public String getRemoteHost()
public Boolean isSecure()
public RequestDispatcher getRequestDispatcher(String path)
```

در ادامه به تشریح متدهایی که بیشتر مورد استفاده قرار می گیرند ، پرداخته می شود.

متدهای پارامترهای درخواست

متدهای زیر برای دسترسی به پارامترهای درخواست سرویس گیرنده مورد استفاده قرار می گیرند. در مورد درخواست هایی از نوع HTTP این متد برای درخواست های POST و GET بکار گرفته می شود.

- متد getParameter()

```
public String getParameter(String key)
```

این متد پارامتر مشخص شده توسط آرگومان key (حساس نسبت به حروف کوچک و بزرگ)را در درخواست جستجو می کند و مقدار آنرا برمی گرداند، اگر چندین مقدار برای یک پارامتر مشخص شده باشد این متد اولین مقدار لیست را برمی گرداند. اگر پارامتر مورد نظر وجود نداشته باشد مقدار null حاصل فراخوانی این متد است.

- متد getParameterValues()

```
public String[] getParameterValues(String key)
```

اگر برای یک پارامتر چندین مقدار وجود داشته باشد ، برای مثال مجموعه ای از چند Check Box یا یک لیست چند انتخابی و یا چند کنترل با یک نام ، این متد یک آرایه حاوی مقادیر پارامتر را بر می گرداند.

- متد getParameterNames()

```
public Enumeration getParameterNames()
```

این متد یک مجموعه حاوی نام پارامترهای درخواست را برمی گرداند. اگر درخواست هیچ پارامتری نداشته باشد ، این متد یک مجموعه خالی برمی گرداند.

متدهای مشخصه درخواست

جدا از پارامترهای درخواست ، حامل وب ، سرولت و صفحات JSP می توانند مشخصه هایی را به درخواست ها ضمیمه نمایند. سه روش متفاوت برای ذخیره و بازیابی مشخصه های درخواست وجود دارد. در روش اول ضمیمه کردن مشخصه ها به شیء درخواست است. دو روش دیگر از HttpSession و ServletContext برای ذخیره و بازیابی مشخصه ها استفاده می نمایند.

هدف از تعریف مشخصه ها در محدوده درخواست اجازه دادن به حامل وب یا سرولت ها برای ارسال اطلاعات اضافه به یک سرولت یا صفحه JSP است. متدهای زیر برای مدیریت مشخصه های درخواست وجود دارد :

- متد getAttribute()

```
public Object getAttribute(String key)
```

این متد مقدار مشخصه داده شده را برمی گرداند (مقدار null در صورت وجود نداشتن مشخصه).

- متد getAttributeNames()

```
public Enumeration getAttributeNames()
```

این متد یک مجموعه از تمام مشخصه های درخواست را برمی گرداند.اگر درخواست هیچ مشخصه ای نداشته باشد ، یک مجموعه خالی نتیجه فراخوانی این متد است.

- متد setAttribute()

```
public void setAttribute(String name, Object o)
```

این متد یک مشخصه بنام name و مقدار o در شیء درخواست تعریف می کند.

- متد removeAttribute()

```
public void removeAttribute(String name)
```

این متد مشخصه name را از شیء درخواست حذف می کند

متدهای ورودی

همانطور قبلا گفته شد ، شیء ServletRequest یک مرجع از اتصال سرویس گیرنده را نگهداری می کند. با استفاده از متدهای زیر می توان به استریم اختصاص یافته به درخواست دسترسی پیدا کرد:

- متد getInputStream()

```
public ServletInputStream getInputStream()throws IOException
```

متد می تواند به بدنه درخواست با استفاده از ServletInputStream دسترسی پیدا کند.

- متد getReader()

```
public java.io.BufferedReader getReader()throws IOException
```

متد می تواند به بدنه درخواست با استفاده از BufferedReader دسترسی پیدا کند.

رابط HttpServletRequest

```
public interface HttpServletRequest extends ServletRequest
```

اکثر متدهای این رابط برای دسترسی به پارامترهای درخواست HTTP تعریف شده است. برای درک اینکه چگونه از این متدها باید استفاده کرد ، در نظر بگیرید که HTTP چگونه اطلاعات را برای سرویس دهنده وب ارسال می کند. همانطور که در ابتدای فصل گفته شد ، HTTP اجازه می دهد پارامترها از طریق یک درخواست برای سرویس دهنده فرستاده شوند. درخواست از نوع GET ، پارامترها را به انتهای آدرس درخواست اضافه می کند ، در حالیکه درخواست از نوع POST پارامترها را در بدنه درخواست و با قالب www-form-urlencoded ارسال می کند. در هر دو روش ، پارامترها بصورت زوج کلید/ مقدارتعریف می شوند و می توانند تکراری باشند . در اینصورت یک کلید می تواند چندین مقدار داشته باشد(برای مثال لیست های چند انتخابی).

هنگام ساختن یک فرم HTML برای درخواست POST یا GET ، بیشتر فیلدهای فرم با تگ <INPUT> تعریف می شوند. هر فیلد دارای یک مشخصه TYPE برای مثال CHECKBOX ، TEXT یا SUBMIT است. مشخصه NAME نام فیلد یا پارامتر HTTP را مشخص می کند و توسط سرویس دهنده قابل شناسایی است. مشخصه VALUE مقدار فیلد یا پارامتر را تعیین می کند و بازیابی آن نسبت به نوع فیلد متفاوت است. بدیهی است اگر چند تگ <INPUT> با یک نام وجود داشته باشد ، چندین زوج کلید/مقدار به ازای یک فیلد قابل بازیاب است. جدول زیر مقدار ارسال شده فیلدهای مختلف را برای سرویس دهنده نمایش می دهد :

نوع فیلد	شرح	مقدار ارسالی
TEXT	فیلد ورود متن یک خطی، با مشخصه پیش فرض VALUE	متن وارد شده توسط کاربر یا مقدار پیش فرض
TEXTAREA	فیلد ورود متن چند خطی، با مشخصه پیش فرض VALUE	متن وارد شده توسط کاربر یا مقدار پیش فرض
PASSWORD	فیلد ورود رمز عبور یک خطی(بجای کاراکتر وارد شده ، کاراکتر * نمایش می دهد).	متن وارد شده توسط کاربر

CHECKBOX	Checkbox استاندارد	اگر انتخاب شده باشد ، مقدار VALUE (یا مقدار on در صورتی که مشخصه VALUE تعریف نشده باشد). اگر انتخاب نشده باشد هیچ زوج کلید/مقداری برگردانده نمی شود.
RADIO	Radio button استاندارد، همه با یک نام و تنها یکی از آنها قابل انتخاب است.	مقدار مشخصه VALUE آیتم انتخاب شده
SELECT	ایجاد لیستی از آیتم ها که توسط کاربر قابل انتخاب است. این فیلد می تواند تک انتخابی یا چند انتخابی باشد.	مقدار مشخصه VALUE آیتم انتخاب شده در صورت تعریف ، در غیر اینصورت متن آیتم انتخاب شده
SUBMIT	دکمه Submit ، مشخصه VALUE عنوان دکمه را تعریف می کند.	اگر مشخصه NAME آن تعریف نشده باشد ، هیچ مقداری را برنمی گرداند.
HIDDEN	فیلد مخفی که در مرورگر وب نمایش داده نمی شود و قابل تغییر توسط کاربر نیست.	مشخصه VALUE

دلیل استفاده از چندین فیلد با یک نام ، ایجاد مجموعه ای از Radio button ، Checkbox و یا لیست چند انتخابی است.

هر شیء پیاده سازی شده از رابط HttpServletRequest (برای مثال شیء درخواست HTTP ارسال شده توسط حامل وب) امکان دسترسی به پارامترهای درخواست HTTP را برای سرولت ها فراهم

می سازد. متدهای دسترسی به پارامترهای درخواست عبارتند از :

```
public String getAuthType()
public Cookie[] getCookies()
public long getDateHeader()
public String getHeader(String name)
public Enumeration getHeaders(String name)
public Enumeration getHeaderNames()
public int getIntheader(string name)
public String getMethod()
public String getContextPath()
public String getPathInfo()
public String getPathTranslated()
public String getQueryString()
public String getRemoteUser()
public Boolean isUserInRole(String role)
public java.security.Principal getUserPrincipal()
public String getRequestedSessionId()
public boolean isRequestedSessionIdvalid()
public boolean isRequestedSessionIdFromCookie()
public boolean isRequestedSessionIdFromURL()
public String getRequestURI()
public String getServletPath()
public HttpSession getSession()
public HttpSession getSession(boolean create)
```

در ادامه به تشریح متدهایی که بیشتر مورد استفاده قرار می گیرند ، پرداخته می شود.

متدهای آدرس و مسیر درخواست

این گروه از متدها به سرولت اجازه می دهند به اطلاعات آدرس و مسیر درخواست HTTP دسترسی پیدا کند.

- متد getPathInfo()

```
public String getPathInfo()
```

این متد اطلاعات آدرس یک درخواست HTTP را برمی گرداند. بطور کلی هر سرولت از طریق نام کلاس آن یا نام تعریف شده در فایل web.xml ، فراخوانی می شود. متد getPathInfo() این آدرس نسبی را برمی گرداند.

- متد getPathTranslated()

```
public String getPathTranslated()
```

این متد مسیر فیزیکی محل ذخیره شدن کلاس سرولت را برمی گرداند. برای مثال اگر سرولت MyServlet در دایرکتوری C:\work\myApp\WEB-INF\classes قرار گرفته باشد ، این متد مقدار C:\work\myApp را برمی گرداند.

- متد getQueryString()

```
public String getQueryString()
```

این متد رشته حاوی پارامترهای درخواست را برمی گرداند.

- متد getServletPath()

```
public String getServletPath()
```

این متد آدرس کامل یک سرولت را برمی گرداند.

متدهای دسترسی به HTTP Headers

این گروه از متدها امکان خواندن اطلاعات HTTP Header فرستاده شده با درخواست را برای سرولت ها فراهم می سازد.

- متد getHeader()

```
public String getHeader(String name)
```

این متد مقدار HTTP Header مشخص شده توسط آرگومان name را برمی گرداند. اگر Header HTTP مورد نظر وجود نداشته باشد ، متد getHeader() مقدار null برمی گرداند.

- متد getHeaders()

```
public Enumeration getHeaders()
```

این متد مجموعه ای از مقادیر HTTP Header موجود در درخواست را برمی گرداند.

- متد getHeaderNames()

```
public Enumeration getHeaderNames()
```

این متد مجموعه ای از اسامی HTTP Header موجود در درخواست را برمی گرداند.

- متد getMethod()

```
public String getMethod()
```

این متد نوع درخواست HTTP شامل POST ، GET و غیره را برمی گرداند.

رابط ServletResponse

```
public interface ServletResponse
```

این رابط نقطه مقابل رابط ServletResponse محسوب می شود و متدهای آماده سازی و ساختن پاسخ را در برمی گیرد :

```
public String getCharacterEncoding()
public ServletOutputStream getOutputStream()
public PrintWriter getWriter()
public void getContentLength(int length)
public void setContentType(String type)
public void setBufferSize(int size)
public int getBufferSize()
public void reset()
public boolean isCommitted()
public void flushBuffer()
public void setLocale(locale)
public Locale getLocale()
```

متدهای دسترسی به طول و نوع محتوای پاسخ

این گروه از متدها امکان تعیین نوع MIME محتوای پاسخ و طول آنرا برای سرولت ها فراهم می سازد.

- متد setContentType()

```
public void setContentType(String type)
```

این متد نوع محتوای پاسخ را تعیین می کند. هنگام استفاده از کلاس PrintWriter برای تولید پاسخ باید نوع آن با استفاده از متد setContentType() مشخص شود. برای مثال مقدار "text/html" پاسخ از نوع HTML است .

- متد setContentLength()

```
public void setContentLength(int size)
```

این متد طول یا اندازه محتوای پاسخ را تعیین می کند.

متدهای خروجی

این گروه از متدها برای تولید خروجی متن یا باینری بکارمی روند.

- متد getOutputStream()

```
public ServletOutputStream getOutputStream()
        throws java.io.IOException
```

این متد یک شیء از نوع ServletOutputStream برمی گرداند. این شیء برای نوشتن اطلاعات باینری در پاسخ استفاده می شود. این کلاس از کلاس java.io.OutputStream ارث می برد . این متد فقط یکبار قابل فراخوانی است. اگر برای بار دوم فراخوانی شود ، استثنای IllegalStateException رخ می دهد.

- متد getWriter()

```
public java.io.PrintWriter getWriter()
        throws java.io.IOException
```

این متد یک شیء از نوع PrintWriter برمی گرداند. این شیء برای نوشتن رشته های کاراکتری در پاسخ استفاده می شود. کلاس PrintWriter بطور اتوماتیک کاراکترهای یونی کد جاوا را به Encoding خروجی مناسب و قابل نمایش برای سرویس گیرنده تبدیل می کند. برای چاپ رشته

خروجی از متد println(String string) استفاده می شود. مشابه متد getOutputStream() متد getWriter() نیز یکبار قابل فراخوانی است.

متدهای بافر کردن خروجی

در سرولت ها می توان درصورتی که حجم اطلاعات خروجی بالا باشد ، برای تسریع در تولید پاسخ و دریافت آن توسط کاربر از امکان بافر کردن خروجی استفاده کرد.

- متد setBufferSize()

```
public void setBufferSize(int size)
```

این متد اندازه بافر خروجی را برحسب کیلوبایت تعریف می کند.

- متد getBufferSize()

```
public int getBufferSize()
```

این متد اندازه بافر خروجی را برمی گرداند.

- متد flushBuffer()

```
public void flushBuffer() throws java.io.IOException
```

این متد اطلاعات بافر شده را در خروجی چاپ و بافر را خالی می کند.

- متد isCommitted()

```
public boolean isCommitted()
```

این متد تعین می کند که آیا اطلاعات بافر شده ، چاپ شده اند یا خیر.

- متد reset()

```
public void reset()
```

این متد برای خالی کردن بافر خروجی می تواند مفید باشد.

تعامل بین سرولت ها

در مدل معمولی برنامه های وب یک سرولت درخواست HTTP را دریافت ، منطق برنامه را اجرا و در نهایت خروجی را تولید می کند. اما دو سناریوی دیگر نیز وجود دارد :

- یک سرولت درخواست HTTP را از سرویس گیرنده دریافت می کند و تولید پاسخ را به یک صفحه JSP واگذار می کند. در این صورت سرولت دیگر مسئول ساختن پاسخ نیست .
- یک سرولت درخواست HTTP را از سرویس گیرنده دریافت می کند و پس از اجرای بخشی از منطق برنامه ، درخواست را به یک سرولت دیگر واگذار می کند. سرولت دوم منطق برنامه را کامل می کند و تولید پاسخ را برعهده سرولت سوم می گذارد.

در هر دو سناریو ، یک سرولت به تنهایی مسئول پردازش درخواست و تولید خروجی نیست و این فرآیند را به چند سرولت و صفحه JSP دیگر واگذار می کند.

در جاوا دو راه حل برای سناریوهای بالا وجود دارد :

- زنجیره سرولت[28] : این راه حل توسط تعدادی از عرضه کنندگان حامل وب/ موتور سرولت پشتیبانی می شود.
- واگذاری درخواست [29]: این راه حل اجازه می دهد یک سرولت درخواست را به سرولت یا صفحات JSP واگذار کند. قبل از Servelt API 2.2 از روش "ارتباط بین سرولت ها"[30] استفاده می شد. اما این روش منسوخ گشته و راه حل واگذاری درخواست جایگزین آن شده است.

[28] Servlet Chaining
[29] Request Dispatching
[30] Inter-Servlet Communication

زنجیره سرولت

ایده زنجیره سرولت بسیار ساده است : مجموعه ای از چندین سرولت طراحی می شوند که هر یک وظیفه خاصی را برعهده دارد. پس از پیاده سازی این سرولت ها ، در پیکربندی یرنامه سرولت ها بصورت یک زنجیره تعریف می شوند.پس از اینکه حامل وب درخواست سرویس گیرنده را دریافت کند ، سرولت ها را به ترتیب فراخوانی می کند. این روش شبیه مکانیزم صف چاپ در سیستم عامل یونیکس است.

واگذاری درخواست

واگذاری درخواست به یک سرولت یا JSP اجازه می دهد درخواست سرویس گیرنده را بمنظور کامل کردن اجرای منطق برنامه یا تولید پاسخ را به یک سرولت ، صفحه JSP و حتی یک فایل HTML واگذار کند.

زمانیکه یک حامل وب درخواست سرویس گیرنده را دریافت می کند ، یک شی request و یک شی response را می سازد و بعنوان آرگومان برای یکی از متدهای سرویس سرولت ارسال می کند . این مکانیزم واگذاری درخواست سرویس گیرنده به یک سرولت توسط حامل وب است. حال اگر یک سرولت بخواهد درخواست را به سرولت دیگری واگذار کند چگونه عمل می کند ؟ برای این کار سرولت اول بایستی مرجعی از سرولت دوم داشته باشد. با استفاده از این مرجع ، سرولت اول می تواند تمام اطلاعات درخواست را به سرولت دوم منتقل کند. رابط javax.servlet.RequestDispatcher برای راه حل واگذاری درخواست در نظر گرفته شده است.

رابط RequestDispatcher

این رابط یک مرجع برای رجوع به منابع دیگر موجود در برنامه شامل سرولت ها ، صفحات JSP و فایل های HTML تولید می کند. یک شی از نوع RequestDispatcher را می توان برای واگذاری درخواست به منابع دیگر بکار گرفت.

این رابط دارای دو متد زیر است :

- **متد forward()**

```
public void forward(ServletRequest request, ServletResponse
                    response) throws ServletException, IOException
```

این متد اجازه می دهد درخواست سرویس گیرنده به یک سرولت ، صفحه JSP و یا یک HTML **forward** شود.

- **متد include()**

```
public void include(ServletRequest request, ServletResponse
                    response) throws ServletException, IOException
```

این متد اجازه می دهد محتوای تولید شده توسط یک منبع را در یین خروجی یک سرولت قرار داد.

راه های دسترسی به شی RequestDispatcher

دو روش یرای دسترسی به یک شی RequestDispatcher وجود دارد :

- متدهای getRequestDispatcher() و getNamedDispatcher() از رابط javax.servlet.ServletContext

```
public RequestDispatcher getRequestDispatcher(String path)
public RequestDispatcher getNamedDispatcher(String name)
```

- متد getRequestDispatcher() از رابط javax.servlet.ServletRequest

```
public RequestDispatcher getRequestDispatcher(String path)
```

اگرچه این سه متد همگی برای یک هدف مورد استفاده قرار می گیرند ، ولی انتخاب هر متد بستگی به اطلاعات قابل دسترس برای برنامه نویس دارد.

دو متد getRequestDispatcher() آدرس منبع را بعنوان آرگومان می پذیرند. اما آدرس متد getRequestDispatcher() در رابط javax.servlet.ServletContext

بصورت مسیر مطلق[31] است. برای مثال اگر در سرولت /myWebApp/servlet/servlet1 بخواهید شی RequestDispatcher برای سرولت /myWebApp/servlet/servlet2 ایجاد کنید ، باید آدرس کامل را نسبت به مسیر ریشه برنامه تعیین نمایید.

متد مشابه در رابط javax.servlet.ServletRequest هر دو آدرس مطلق و نسبی را بعنوان آرگومان می پذیرد.

متد getNamedDispatcher() مشابه متد getRequestDispatcher() است با این تفاوت که بجای آدرس سرولت ، نام تعریف شده برای سرولت در فایل پیکربندی web.xml بوسیله تگ <servlet-name> را بعنوان آرگومان می پذیرد.

مثال کاربردی – درخواست پشتیبانی فنی

با مرور Java Servlet API این فصل را با یک مثال کاربردی به پایان می رسانیم. این مثال وظایف زیر را برعهده دارد:

- گرفتن درخواست پشتیبانی فنی

این بخش از برنامه مسئول جمع آوری پارامترها از درخواست HTTP ارسال شده توسط کاربر است.

- ثبت درخواست پشتیبانی فنی

این قسمت از برنامه شامل تولید یک شماره منحصربفرد و ثبت درخواست پشتیبانی فنی در بانک اطلاعاتی

- تولید یک صفحه اطلاع رسانی

[31] Absolute Path

این مثال برای ذخیره سازی اطلاعات نیاز به بانک اطلاعاتی دارد. بدین منظور بانک اطلاعاتی Cloudscape در نظر گرفته شده است. نسخه رایگان Cloudscape3.5 در سایت http://www.cloudscape.com/ موجود است. البته برای این مثال می توان از بانک های اطلاعاتی Microsoft Access و MySQL نیز استفاده کرد. تولید برنامه شامل مراحل زیر است :

- آماده سازی یک صفحه HTML بهمراه یک فرم برای جمع آوری اطلاعات
- آماده سازی جداول بانک اطلاعاتی با تعریف ساختار آن
- ایجاد یک کلاس Sequencer بمنظور تولید عدد منحصربفرد برای درخواست های پشتیبانی فنی
- نوشتن سرولت TechSupportServlet برای پردازش درخواست ، درج اطلاعات در بانک اطلاعاتی و تولید خروجی مناسب

آماده سازی صفحه HTML

در این صفحه HTML یک فرم ورود اطلاعات بمنظور گرفتن درخواست های پشتیبانی فنی طراحی می شود. کد زیر را در فایل C:\JavaPro\Chapter14\techSupport\techsupp.html ذخیره نمایید :

```
<HTML>
<HEAD>
 <TITLE>XYZ corporation, IT Department</title>
</HEAD>
<BODY>
   <H1>Technical Support Request</H1>
   <HR><BR>
   <CENTER>
     <FORM ACTION="/techSupport/servlet/techSupport" METHOD="POST">
     <TABLE ALIGN="center" WIDTH="100%" CELLSPACING="2" CELLPADDING="2">
       <TR>
         <TD ALIGN="right">First Name:</TD>
         <TD><INPUT TYPE="Text" NAME="firstName" ALIGN="LEFT" SIZE="15"></TD>
         <TD ALIGN="right">Last Name:</TD>
         <TD><INPUT TYPE="Text" NAME="lastName" ALIGN="LEFT" SIZE="15"></TD>
       </TR>
       <TR>
         <TD ALIGN="right">Email:</TD>
         <TD><INPUT TYPE="Text" NAME="email" ALIGN="LEFT" SIZE="25"></TD>
         <TD ALIGN="right">Phone:</TD>
         <TD><INPUT TYPE="Text" Name="phone" ALIGN="LEFT" SIZE="15"></TD>
       </TR>
       <TR>
         <TD ALIGN="right">Software:</TD>
         <TD>
           <SELECT NAME="software" SIZE="1">
             <OPTION VALUE="Word">Microsoft Word</OPTION>
             <OPTION VALUE="Excel">Microsoft Excel</OPTION>
             <OPTION VALUE="Access">Microsoft Access</OPTION>
           </SELECT>
        </TD>
        <TD ALIGN="right">Operating System:</TD>
```

```
      <TD>
        <SELECT NAME="os" size="1">
          <OPTION VALUE="95">Windows 95</OPTION>
          <OPTION VALUE="98">Windows 98</OPTION>
          <OPTION VALUE="NT">Windows NT</OPTION>
        </SELECT>
      </TD>
    </TR>
  </TABLE>

<BR>Problem Description
    <BR>
    <TEXTAREA NAME="problem" COLS="50" ROWS="4"></TEXTAREA>
    <HR><BR>
    <INPUT TYPE="Submit" NAME="submit" VALUE="Submit Request">
  </FORM>
 </CENTER>
</BODY>
</HTML>
```

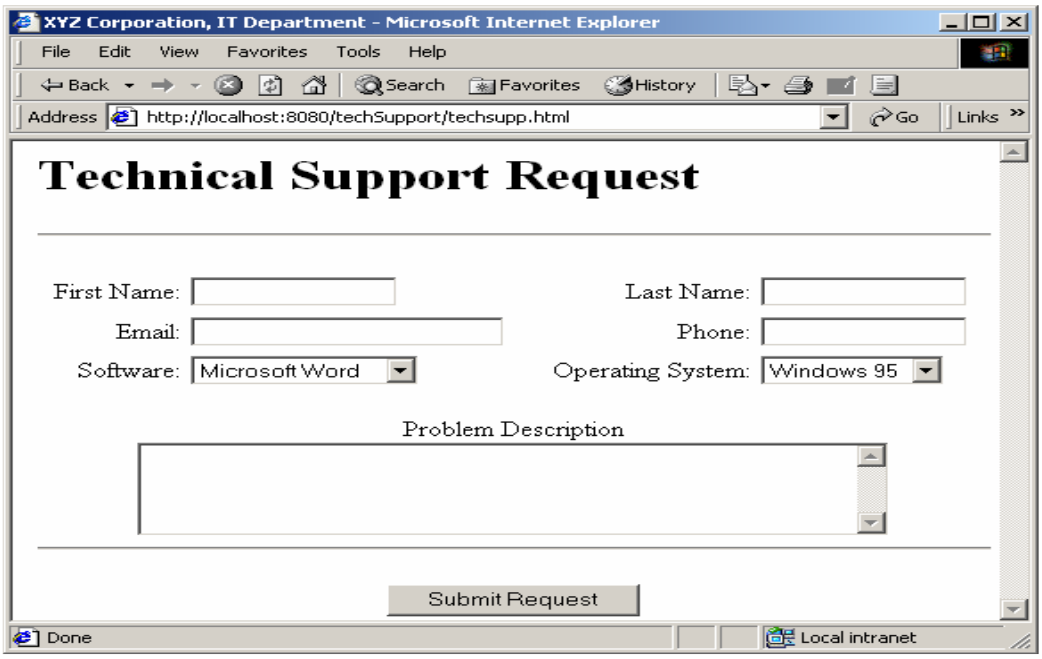
این فرم فیلدهای زیر را در برمی گیرد :

- یک فیلد از نوع TEXT بنام firstName برای گرفتن نام
- یک فیلد از نوع TEXT بنام lastName برای گرفتن نام خانوادگی
- یک فیلد از نوع TEXT بنام email برای گرفتن آدرس الکترونیکی
- یک فیلد از نوع TEXT بنام phone برای گرفتن شماره تلفن
- یک فیلد از نوع SELECT بنام software برای نمایش لیست نرم افزارها
- یک فیلد از نوع SELECT بنام os برای نمایش لیست سیستم عامل ها

اگرچه می توان این فایل را بصورت محلی و مستقیم با فراخوانی C:\JavaPro\Chapter14\techSupport\techsupp.html در مرورگر وب مشاهده نمود. همانطور که در اولین مثال این فصل گفته شد. می توان برنامه را در محلی خارج از دایرکتوری %TOMCAT_HOME% تولید و در فایل %TOMCAT_HOME%\conf\server.xml این مسیر را طبق الگوی زیر معرفی کرد :

```
<Context path="/techSupport"
   docBase="C:\JavaPro\chapter14\techSupport">
</Context>
```

آدرس http://localhost:8080/techSupport/techsupp.html را در مرورگر وب وارد نمایید. نتیجه بصورت زیر خواهد بود :



آماده سازی بانک اطلاعاتی

قبل از نوشتن سرولت برنامه ، باید جداول بانک اطلاعاتی برای ذخیره سازی درخواست های پشتیبانی فنی آماده شود. با استفاده از ابزارهای کمکی بانک اطلاعاتی (CloudView در Cloudscape یا SQL Plus در Oracle) می توان جداول را ایجاد نمود. اگر از بانک اطلاعاتی Cloudscape را برای این مثال در نظر گرفته اید ، بهتر است بانک اطلاعاتی در دایرکتوری C:\JavaPro\Chapter14\techSupport\WEB_INF\db تعریف شود.

در ابزار اجرای دستورات SQL بانک اطلاعاتی ، دستورات زیر را برای ساختن جداول بانک اطلاعاتی وارد نمایید:

```
--
-- Table for storing the technical support requests
--

CREATE TABLE SUPP_REQUESTS(REQUEST-ID  INTEGER PRIMARY KEY,
                    FIRST-NAME  VARCHAR(40),
                    LAST-NAME   VARCHAR(40),
                    EMAIL       VARCHAR(40),
                    PHONE       VARCHAR(15),
                    SOFTWARE    VARCHAR(40),
                    OS          VARCHAR(40),
                    PROBLEM     VARCHAR(256));

--
-- Sequencer table for generating sequence numbers for REQUEST-ID
--

CREATE TABLE SEQ-NO(NEXT-NO    INTEGER);

INSERT INTO SEQ-NO VALUES(0);
```

کلاس Sequencer

بمنظور اختصاص دادن یک شماره منحصر بفرد به هر درخواست پشتیبانی فنی ، نیاز به تولید کننده اعداد ترتیبی و منحصر بفرد است. اگرچه از امکانات خود بانک اطلاعاتی می توان برای رفع این مشکل استفاده نمود. اما در این مثال یک کلاس جاوا در کنار جدول SEQ_NO برای تولید اعداد ترتیبی در نظر گرفته شده است. کلاس Sequencer.java بصورت زیر تعریف می شود:

```
// Imports Java Packages
import java.sql.*;

public class Sequencer {
     static String updateStatementStr =
        "UPDATE SEQ-NO SET NEXT-NO = NEXT-NO + 1";
    static String selectstatementStr = "SELECT NEXT-NO FROM SEQ-NO";
     // Get tje next number
    public static int getNextNumber(String protocol) throws SQLException {
        Connection connection = DriverManager.getConnection(protocol);
         PreparedStatement updateStatement =
        connection.prepareStatement(updateStatementStr);
        PreparedStatement selectStatement =
        connection.prepareStatement(selectstatementStr);
        connection.setAutoCommit(false);
         // Increment the sequencer
        updateStatement.executeUpdate();
         // Retrieve the sequencer number
        ResultSet rs = selectStatement.executeQuery();
        rs.next();
        int next = rs.getInt(1);

        rs.close();
        updateStatement.close();
        selectStatement.close();
        connection.commit();
        connection.close();
        return next;
    }
}
```

دراین کلاس یک متد static بنام getNextNumber() تعریف شده است که پروتکل اتصال به بانک اطلاعاتی را بعنوان آرگومان دریافت می کند و یک شماره سریال جدید را برمی گرداند. این متد یک واحد به فیلد NEXT_NO در جدول SEQ_NOاضافه می کندو عدد افزایش یافته را بعنوان عدد جدید بر می گرداند. جملات SELECT و UPDATE در یک تراکنش تعریف شده اند(با تنظیم auto-commit-mode با مقدار false).

اتخاذ این تدبیر به این دلیل است که دسترسی همزمان چندین ریسمان به سرولت از طریق متد getNextNumber() به جدول SEQ_NO مشکلی در پی نداشته باشد.

نوشتن سرولت

از آنجاییکه تگ <FORM>صفحه techsupp.html از متد POST استفاده می کند. سرولت پاسخ دهنده باید

متد doPost() را پیاده سازی نماید. برای پردازش درخواست کاربر مراحل زیر طی می شود :

- مقدار دهی اولیه سرولت و بارگذاری درایور بانک اطلاعاتی
- استخراج پارامترهای HttpServletRequest با استفاده از متد getParameter()
- درج اطلاعات در بانک اطلاعاتی
- تولید پاسخ بهمراه شماره مرجع درخواست

تعریف کلاس TechSupportServlet با دستورات زیر شروع می شود :

```
//Import Servlet Packages
import javax.servlet.*;
import javax.servlet.http.*;

//Import other Java Packages
import java.io.*;
import java.sql.*;

public class TechSupportServlet extends HttpServlet {

   //Methods will go here …

}
```

مقدار دهی اولیه سرولت

در این مثال از راه حل استاندارد JDBC (java.sql) برای ثبت و بارگذاری درایور بانک اطلاعاتی استفاده می شود. بارگذاری درایور بانک اطلاعاتی در متد init() سرولت انجام می شود :

```
public void init() throws ServletException {
      String driver = getServletConfig().getInitParameter("driver");
      protocol = getServletConfig().getInitParameter("protocol");

      if (driver == null || protocol == null) {
         throw new UnavailableException("Driver not specified.");
      }

      try {
         Class.forName(driver);
      } catch (ClassNotFoundException cnfe) {
         throw new UnavailableException("Driver <" + driver
                              + "> not found in the classpath.");
      }
   }
```

در متد init() دو پارامتر اولیه سرولت بنام های driver و protocol را از فایل توصیف گر توسعه (web.xml) می خواند و درایور بانک اطلاعاتی را بارگذاری می کند. اگر این دو پارامتر اولیه در فایل web.xml تعریف نشده باشد یا کلاس درایور JDBC در دسترس نباشد ، استثناء UnavailableException رخ می دهد.

استخراج اطلاعات

برای گرفتن اطلاعات وارد شده توسط کاربر در techsupp.html از متد getParameter() شیء HttpServletRequest استفاده می شود. آرگومان این متد از نوع String و نام پارامتر درخواست HTTP را مشخص می کند. قطعه کد زیر اطلاعات کاربر را از پارامترهای درخواست استخراج می کند :

```
protected void doPost(HttpServletRequest req, HttpServletResponse res)
         throws ServletException, IOException {

      String firstName = req.getParameter("firstName");
      String lastName = req.getParameter("lastName");
      String email = req.getParameter("email");
      String phone = req.getParameter("phone");
      String software = req.getParameter("software");
      String os = req.getParameter("os");
      String problem = req.getParameter("problem");
```

خواندن مقدار پارامترهای درخواست HTTP بسیار آسان است. اگر پارامتر مشخص شده در درخولست HTTP وجود داشته باشد ، متد getParameter() یک شیء از نوع String حاوی مقدار پارامتر و در غیر اینصورت مقدارnull را بر می گرداند. بررسی null بودن مقدار پارامترهای HTTP خطاهای زمان اجرا را به حداقل می رساند .

درج درخواست پشتیبانی

پس از استخراج پارامترهای درخواست مرحله بعدی ، اجرای دستورات SQL برای درج رکورد جدید در بانک اطلاعاتی است. این فرآیند به یک شماره سریال جدید نیاز دارد که توسط کلاس Sequencer تولید می شود:

```
int requestId = 0;
Connection connection = null;
String insertStatementStr =
      "INSERT INTO SUPP-REQUESTS VALUES(?,?,?,?,?,?,?,?)";
try {
   connection = DriverManager.getConnection(protocol);
   PreparedStatement insertStatement =
         connection.prepareStatement(insertStatementStr);

   requestId = Sequencer.getNextNumber(protocol);
   insertStatement.setInt(1, requestId);
   insertStatement.setString(2, firstName);
   insertStatement.setString(3, lastName);
   insertStatement.setString(4, email);
   insertStatement.setString(5, phone);
   insertStatement.setString(6, software);
   insertStatement.setString(7, os);
   insertStatement.setString(8, problem);
   insertStatement.executeUpdate();
} catch (SQLException sqle) {
   throw new ServletException("Database error", sqle);
} finally {
   if (connection != null) {
      try {
         connection.close();
      } catch (SQLException sqle) {}
   }
}
```

- تعریف یک اتصال به بانک اطلاعاتی با استفاده از DriverManager
- آماده سازی یک شیء PreparedStatement برای دستور INSERT

- تولید یک شماره سریال جدید با استفاده از متد `getNextNumber()` کلاس `Sequencer`
- تعیین پارامترهای شیء `PreparedStatement`
- اجرای دستور `SQL`
- بستن شیء `Connection` در بلاک `finally` برای اطمینان از بسته شدن اتصال بانک اطلاعاتی

تولید پاسخ

آخرین مرحله در سرولت `TechSupportServlet` ، تولید پاسخ برای کاربر است :

```
    // Prepare the response
    PrintWriter out = res.getWriter();

    res.setContentType("text/html");
    out.println("<HTML><HEAD><TITLE>");
    out.println("Tech support:Request Confirmation");
    out.println("<TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<H1>Tech Support: Request Confirmation</H1>");
    out.println("<P>Thank you for your request. your request with the following"
            + "reference number has been received.</P>");
    out.println("<P>Request Reference: " + requestId + "</P>");
    out.println("<P>Please note this number for future references.</P>");
    out.println("<P>Your request will be attended to within 24 hours.</P>");
    out.println("<P>Administrator <br>Techsupport team. </P>");
    out.println("</BODY></HTML>");
    out.close();
  }
```

کامپایل کردن برنامه

فایل های `TechSupportServlet.java` و `Sequencer.java` را کامپایل و کلاس ها را در دایرکتوری `C:\JavaPro\Chapter14\techSupport\WEB-INF\classes` قرار دهید.

توصیف گر توسعه

سرولت TechSupportServlet و پارامترهای اولیه آن بصورت زیر در فایل web.xml تعریف می شود:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app-2.2.dtd">

<web-app>
   <welcome-file-list>
      <welcome-file>techsupp.html</welcome-file>
   </welcome-file-list>

   <servlet>
      <servlet-name>techSupport</servlet-name>
      <servlet-class>TechSupportServlet</servlet-class>

      <init-param>
         <param-name>driver</param-name>
         <param-value>COM.cloudscape.core.JDBCDriver</param-value>
      </init-param>

      <init-param>
         <param-name>protocol</param-name>
         <param-value>
          jdbc:cloudscape:C:/JavaPro/Chapter14/techSupport/WEB-INF/db;autocommit=true
         </param-value>
      </init-param>
   </servlet>
</web-app>
```

پارامترهای driver و protocol را با توجه به نوع بانک اطلاعاتی و مشخصات آن تغییر دهید.

درفایل web.xml بالا دو نوع پیکربندی تعریف شده است :

- صفحه پیش فرض

در توصیف گر توسعه این مثال ، فایل techsupp.html بعنوان صفحه پیش فرض تعریف شده است.بنابراین بجای آدرس

http://localhost:8080/techSupport/techsupp.html

می توان آدرس زیر را وارد کرد :

http://localhost:8080/techSupport/

- پارامترهای اولیه سرولت

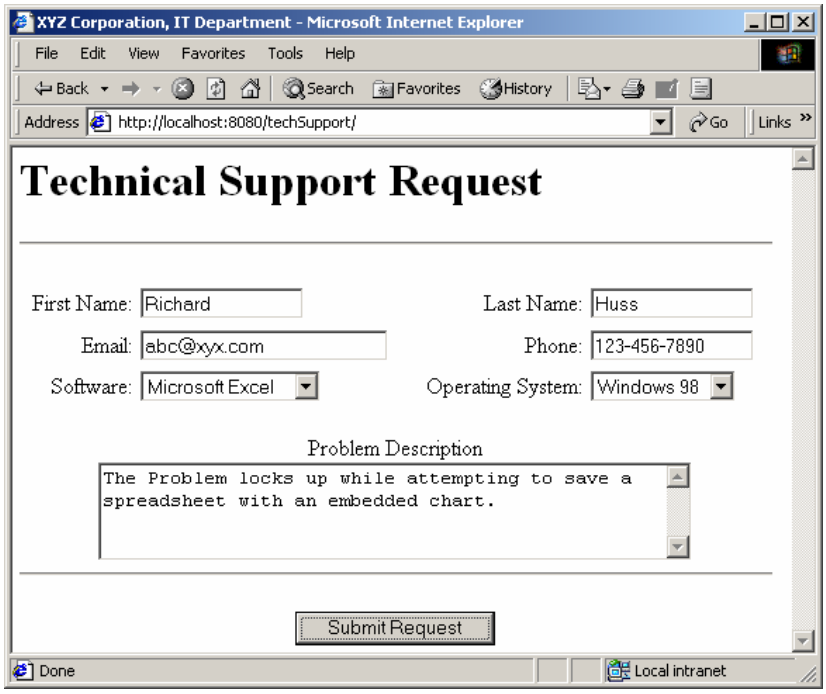
نام درایور و پروتکل JDBC برای اتصال به بانک اطلاعاتی بعنوان پارامترهای اولیه سرولت TechSupportServlet تعریف شده اند.

راه اندازی سیستم

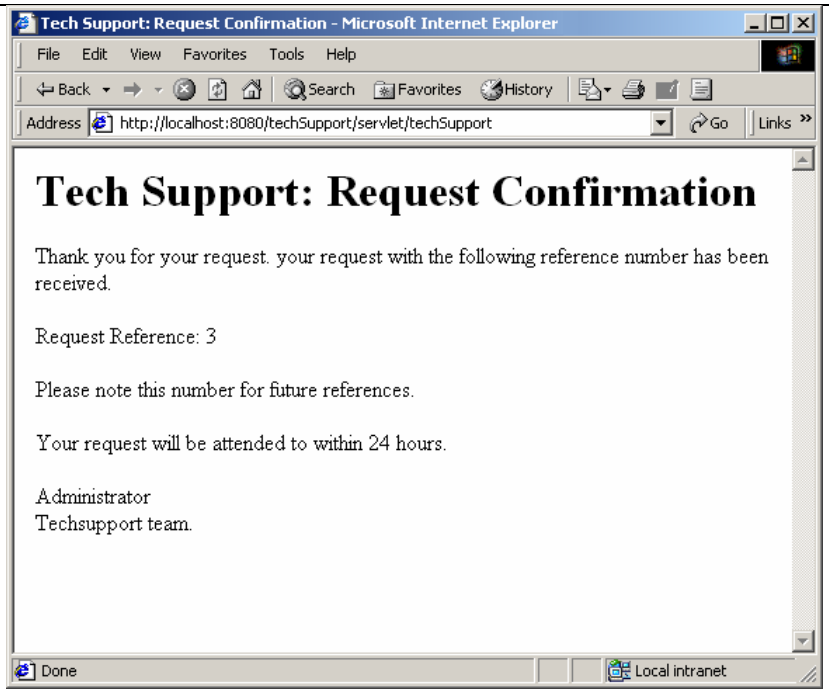
آخرین مرحله قبل از تست برنامه ، اطمینان از پیکربندی و تنظیم صحیح سرویس دهنده Tomcat و نیز معرفی بانک اطلاعاتی است. بدین منظور فایل%TOMCAT_HOME%\bin\tomcat.bat طبق الگوی (خطوط پررنگ شده) زیر اصلاح شود :

```
set CLOUDSCAPE_CP=%CLOUDSCAPE_INSTALL%\lib\cloudscape.jar
set CLASSPATH=.
set CLASSPATH=%TOMCAT_HOME%\classes
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\webserver.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\jasper.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\xml.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\servlet.jar
set CLASSPATH=%CLASSPATH%;%TOMCAT_HOME%\lib\tools.jar
set CLASSPATH=%CLASSPATH%;% CLOUDSCAPE_CP%
```

سرویس دهنده Tomcat را اجرا و آدرس http://localhost:8080/techSupport را در مرورگر وب وارد و طبق شکل زیر اطلاعات فرم را وارد نمایید:



با ارسال فرم توسط دکمه `Submit` ، خروجی زیر نمایش داده می شود :



خلاصه

در این فصل سرولت ، چرخه حیات سرولت ، مدل برنامه نویسی سرولت های و Java Servlet API مرور گردید.

- برای نوشتن یک سرولت ، رابط Servlet باید پیاد سازی شود. این پیاده سازی توسط GenericServlet و HttpServlet صورت می گیرد.
- سرولت و پارامترهای اولیه آن در فایل توصیف گر توسعه (web.xml) تعریف می شود.
- چرخه حیات یک سرولت با متدهای init() ، service() و destroy() اداره می شود.
- کلاس HttpServlet از کلاس GenericServlet برای کنترل درخواست های HTTP ارث می برد.

- با استفاده از توصیف گر توسعه می توان بسیاری از تنظیمات برنامه را از داخل کد برنامه خارج و بدون نیاز به تغییر کد ، این تنظیمات را دوباره تعریف کرد.
- با استفاده از صفحات کنترل خطا می توان استثنات رخ داده در برنامه را بصورت اتوماتیک به این صفحات هدایت کرد و پیغام مناسب تر نمایش داد.