Etude préalable :

1) Langage Java/JEE :

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C. Ses caractéristiques ainsi que la richesse de son écosystème et de sa communauté lui ont permis d'être très largement utilisé pour le développement d'applications de types très disparates. Java est notamment largement utilisé pour le développement d'applications d'entreprises et mobiles.

Java Micro Edition (JME)

• développement d'applications embarquées

Java Standard Edition (JSE)

• développement d'applications classiques

Java Enterprise Edition (JEE)

• développement d'applications d'entrepris

JEE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :

un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme serveur d'applications. un ensemble d'API qui peuvent être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.

Sun (qui a été racheté par oracle en 2009) propose une implémentation minimale des spécifications de JEE : le JEE SDK. Cette implémentation permet de développer des applications respectant les spécifications mais n'est pas prévue pour être utilisée dans un environnement de production. Ces spécifications doivent être respectées par les outils développés par des éditeurs tiers.

L'utilisation de JEE pour développer et exécuter une application offre plusieurs avantages :

une architecture d'applications basée sur les composants qui permet un découpage de l'application et donc une séparation des rôles lors du développement la possibilité de s'interfacer avec le système d'information existant grâce à de nombreuses API : JDBC, JNDI, JMS, JCA ... la possibilité de choisir les outils de développement et le ou les serveurs d'applications utilisés qu'ils soient commerciaux ou libres

JEE permet une grande flexibilité dans le choix de l'architecture de l'application en combinant les différents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de JEE.

L'architecture d'une application se découpe idéalement en au moins trois tiers :

la partie cliente : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composée d'une application standalone, d'une application web ou d'applets

la partie métier : c'est la partie qui encapsule les traitements (dans des EJB ou des JavaBeans)

la partie données : c'est la partie qui stocke les données .

2) MODELE MVC :

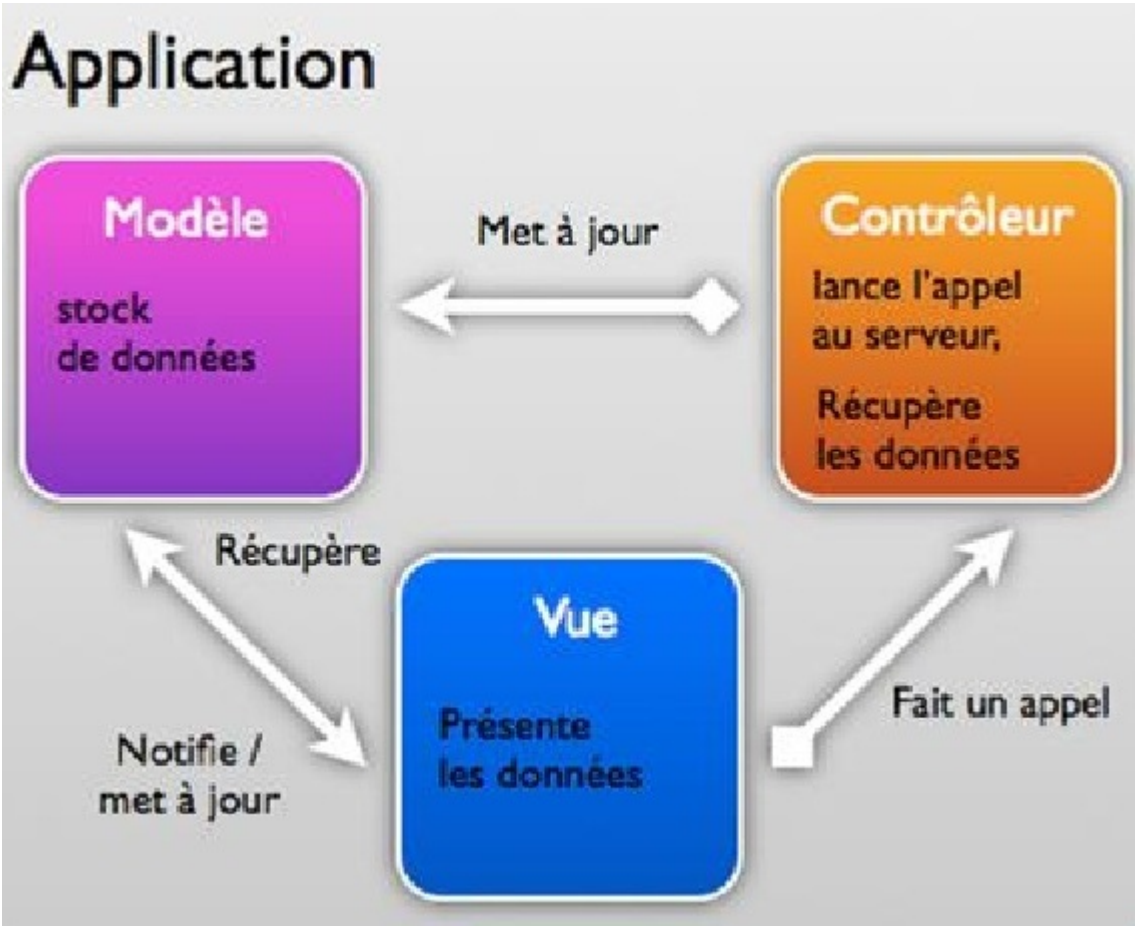
Le modèle MVC (Model View Controler) a été initialement développé pour le langage Smalltalk dans le but de mieux structurer une application avec une interface graphique.

Ce modèle est un concept d'architecture qui propose une séparation en trois entités des données, des traitements et de l'interface :

le Modèle représente les données de l'application généralement stockées dans une base de données la Vue correspond à l'IHM (Interface Homme Machine) le Contrôleur assure les échanges entre la vue et le modèle notamment grâce à des composants métiers

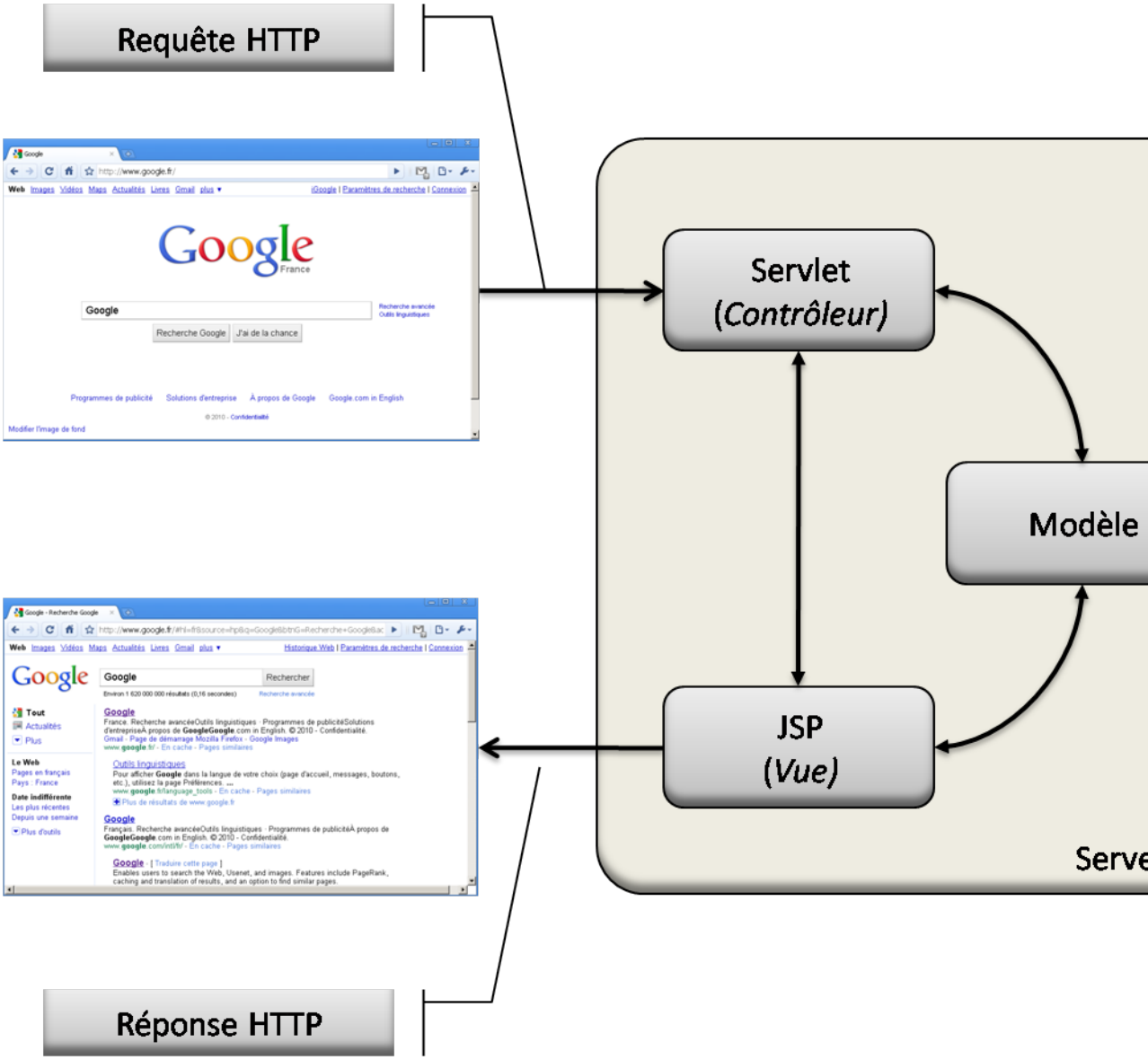
Initialement utilisé pour le développement des interfaces graphiques, ce modèle peut se transposer pour les applications web sous la forme d'une architecture dite 3-tiers : la vue est mise en oeuvre par des JSP, le contrôleur est mis en oeuvre par des servlets et des Javabeans. Différents mécanismes peuvent être utilisés pour accéder aux données.

L'utilisation du modèle MVC rend un peu plus compliqué le développement de l'application qui le met en oeuvre mais il permet une meilleure structuration de celle-ci.



Modele MVC type 1 :

Dans ce modèle, chaque requête est traitée par un contrôleur sous la forme d'une servlet. Celle-ci traite la requête, fait appel aux éléments du model si nécessaire et redirige la requête vers une JSP qui se charge de créer la réponse à l'utilisateur



L'inconvénient est donc une multiplication du nombre de servlets nécessaires à l'application : l'implémentation de plusieurs servlets nécessite beaucoup de code à produire d'autant que chaque servlet doit être déclarée dans le fichier web.xml.

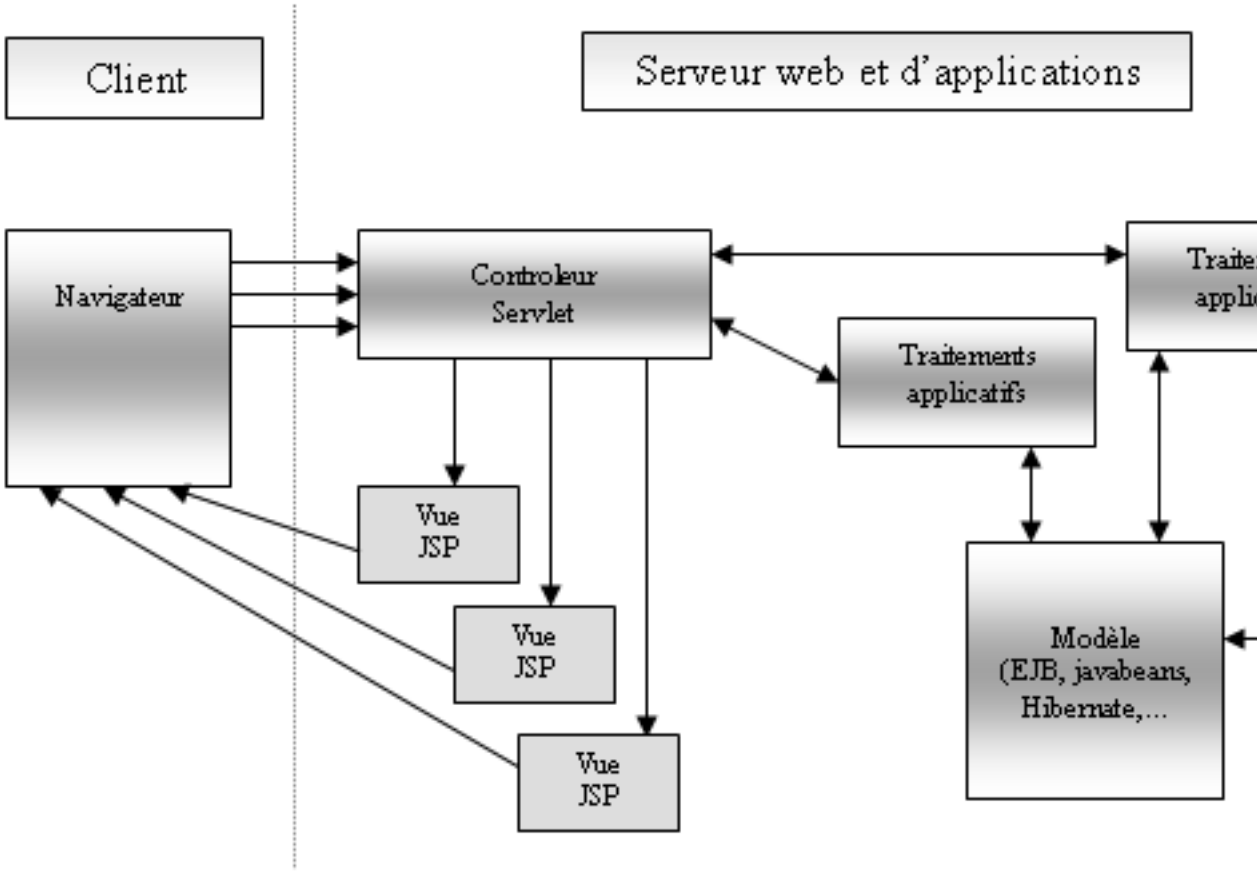
Modele MVC type 2 :

Le principal défaut du modèle MVC est le nombre de servlets à développer pour une application.

Pour simplifier les choses, le modèle MVC model 2 ou MVC2 de Sun propose

de n'utiliser qu'une seule et unique servlet comme contrôleur. Cette servlet se charge d'assurer le workflow des traitements en fonction des requêtes http reçues.

Le modèle MVC 2 est donc une évolution du modèle 1 : une unique servlet fait office de contrôleur et gère toutes les requêtes à traiter en fonction d'un paramétrage généralement sous la forme d'un fichier au format XML.



Le modèle MVC II conserve les principes du modèle MVC, mais il divise le contrôleur en deux parties en imposant un point d'entrée unique à toute l'application (première partie du contrôleur) qui déterminera à chaque requête reçue les traitements applicatifs à invoquer dynamiquement (seconde partie du contrôleur).

Une application web implémentant le modèle MVC de type II utilise une servlet comme contrôleur traitant les requêtes. En fonction de celles-ci, elle appelle les traitements dédiés généralement encapsulés dans une classe.

Dans ce modèle, le cycle de vie d'une requête est le suivant :

-Le client envoie une requête à l'application, requête est prise en charge par la servlet faisant office de contrôleur.

-La servlet analyse la requête et appelle la classe dédiée contenant les traitements Cette classe exécute les traitements nécessaires en fonction de la requête, notamment, en faisant appel aux objets métiers.

-En fonction du résultat, la servlet redirige la requête vers la page JSP La JSP génère la réponse qui est renvoyée au client .

3) Servlets :

Une servlet est un composant logiciel, utilisé dans un serveur web, tel que Tomcat, qui peut être invoqué par les navigateurs clients via une URL. Le protocole de communication est dans ce cas HTTP. Même si le web dynamique est l'utilisation majeure de l'API Servlet, elle permet théoriquement de couvrir d'autres domaines d'application.

Le principe de fonctionnement est très simple : ce composant logiciel reçoit une requête, et il envoie une réponse. Cette réponse est transmise au client, qui l'interprète enfin.

Techniquement, l'API Servlet est un ensemble d'interfaces et de classes Java, rangées dans les packages javax.servlet et javax.servlet.http. Le protocole HTTP a bien sûr un statut particulier parmi les protocoles utilisés par les servlets.

Dans la mesure où les servlets sont des composants programmés entièrement en Java, elles sont portables sur toutes les architectures munies d'une machine Java. Il est donc parfaitement possible de développer ces servlets dans un environnement Windows pour les déployer ensuite, c'est à dire les mettre en production, dans un environnement Unix.

. Concepts, cycle de vie

. .Requête :

Une requête HTTP provient d'un client, le plus souvent un navigateur web, et porte avec elle des paramètres. Ces paramètres peuvent être techniques (les paramètres standard HTTP), ou applicatifs, comme le contenu d'un formulaire.

. .Réponse :

La réponse provient du serveur, et est destinée le plus souvent à être affichée dans un navigateur. Ce peut être une page HTML, dynamique ou non, une image, ou bien un code d'erreur HTTP, accompagné d'un message.

. .Session :

De nombreuses applications web ont besoin de reconnaître un utilisateur donné de requête en requête. Malheureusement, le protocole HTTP ne permet pas cela : rien n'est prévu pour garder la mémoire de ce client. La façon la plus répandue de pallier ce problème, est d'utiliser des cookies. Le principe est le suivant : le serveur envoie au navigateur un cookie , qui, techniquement, est un petit fichier texte. Ce fichier contient des informations telles que le nom du serveur qui en est à l'origine, quel serveur a le droit de le lire, une date de péremption, et bien sûr, une donnée d'identification. Une fois le cookie accepté (il peut être refusé), le navigateur le renvoie systématiquement avec chaque requête. Charge donc au serveur de se souvenir de la personne à qui il a envoyé ce cookie, de façon à la reconnaître. L'API Servlet définit un cookie : JSESSIONID, ce qui lui permet de définir la notion de session.

Une session est simplement un ensemble de cycles requêtes / réponses avec

un utilisateur donné. Une session est reconnue grâce à un cookie nommé JSESSIONID.

. .Contexte d'exécution :

Le standard Servlet définit trois contextes d'exécution pour une servlet. Ces trois contextes sont :

La requête : l'exécution d'une servlet se déroule dans le contexte d'une requête unique.

La session : de la même façon, une servlet est exécutée dans le contexte d'une unique session. Une session est définie par un ensemble de requêtes successives en provenance d'un même client. Elle peut durer les quelques minutes du temps d'une visite sur un site de commerce électronique, ou beaucoup plus longtemps, et conserver les données d'identification d'un internaute d'une journée à l'autre.

L'application : enfin une servlet appartient à une application web unique.

Ces trois contextes exposent les mêmes méthodes setAttribute(String, Object), getAttribute(String) et getAttributeNames(). Ces méthodes permettent d'attacher des objets Java quelconques à ces contextes, et de les récupérer. En fonction de la durée de vie de ces objets (une requête, une session ou toute l'application), on choisira le bon contexte pour sauvegarder les informations dont on a besoin.

On notera cependant que l'entretien du contexte session peut être coûteux, notamment en mémoire. Certaines applications choisissent d'ailleurs de ne pas l'utiliser, pour des raisons de perfomance. D'une façon générale, on évitera de stocker trop d'information dans ce contexte.

. Cycle de vie

La notion de cycle de vie est une notion qui existe pour tous les types de composant qu'un serveur d'application peut gérer. Entre le moment où un serveur d'application démarre, et le moment où les composants qu'il gère sont prêts à être utilisés, chaque composant passe par différents états, définis dans les standards. Ces états, les transitions qui permettent de passer de l'un à l'autre, et les conditions qui permettent d'activer ces transitions forment un ensemble que l'on appelle le cycle de vie d'un composant.

À chaque changement d'état d'un composant, l'application qui possède ce composant est notifiée par le serveur d'application. Techniquement, cette notification consiste en l'appel d'une méthode d'un objet particulier, défini dans le standard. Cet objet est fourni par l'application, s'il n'est pas déclaré, alors la notification n'a pas lieu. On appelle ces objets particuliers des listeners .

En tant que composant standard, les servlets ont un cycle de vie, sur lequel il est possible de poser des listeners . Lors de ce cycle de vie, certaines méthodes particulières des servlets sont invoquées, nous verrons ce point dans la suite.

La classe HttpServlet

Définit les méthodes doGet et doPut pour répondre respectivement aux requêtes GET et POST d'un client. Ces méthodes sont appelées par la méthode service de la classe HttpServlet , lorsqu'une requête arrive au serveur. La méthode service détermine d'abord le type de la requête, puis appelle la méthode appropriée. La classe propose aussi d'autres types de requêtes moins usuels, qui

sortent du cadre de ce livre. Pour plus d'informations sur le protocole HTTP, visitez le site :

http ://www.w3.org/Protocols/

La figure 19.2 reprend les méthodes de la classe HttpServlet qui répondent aux autres types de requêtes. Elles reçoivent toutes des paramètres des types HttpServletRequest et HttpServletResponse et renvoient un void . Les méthodes de la figure 19.2 sont d'un usage plutôt rare. Les méthodes doGet et doPost reçoivent en arguments un objet HttpServletRequest et un objet HttpServletResponse qui assurent l'interaction entre le client et le serveur.

Les méthodes de HttpServletRequest facilitent l'accès aux données fournies au sein de la requête. Les méthodes HttpServletResponse facilitent le retour au client Web des résultats du servlet au format HTML. Les deux sections suivantes étudient les interfaces HttpServletRequest et HttpServletResponse .

Interface HttpServletRequest

Chaque appel à doGet ou doPost sur un HttpServlet reçoit un objet qui implémente l'interface HttpServletRequest . Le serveur Web qui exécute le servlet crée un objet HttpServletRequest et le passe à la méthode service du servlet, qui, à son tour, le passe à doGet ou doPost . Cet objet contient la requête du client. Diverses méthodes permettent au servlet de traiter la requête du client. Quelques-unes de ces méthodes proviennent de l'interface ServletRequest , l'interface qu'étend HttpServletRequest . La figure 19.3 décrit quelques-unes des méthodes-clés utilisées dans ce chapitre .

Méthode	Description
doDelete	Appelée en réponse à une requête HTTP ***DELETE***. Une telle requête sert nor à supprimer un fichier sur le serveur. Il se peut qu'elle ne soit pas disponible certains serveurs, à cause des risques évidents auxquels elle l'expose, au nive la sécurité.
doOptions	Appelée en réponse à une requête HTTP ***OPTIONS***. Elle renvoit des informa au client, indiquant les options HTTP acceptées par le serveur.
doPut	Appelée en réponse à une requête HTTP ***PUT***. Une telle requête est utilisée normalement pour stocker un fichier sur le serveur. Il se peut qu'elle ne soit p disponible sur certains serveurs étant donné les risques auxquels elles l'expo au niveau de la sécurité.
doTrace	Appelée en réponse à une requête HTTP ***TRACE***. Une telle requête sert norm au débogage. L'implémentation de cette méthode renvoie automatiquement un document HTML au client, contenant les informations d'en-tête de la req (soit les données envoyées par le fureteur au sein de la requête).

Interface HttpServletResponse

Chaque appel à doGet ou doPost pour un HttpServlet reçoit un objet qui implémente l'interface HttpServletResponse . Le serveur Web qui exécute le servlet crée un objet HttpServletResponse et le passe à la méthode service du servlet (qui, à son tour, le passe à doGet ou doPost). Cet objet contient la réponse au client. Diverses méthodes sont également fournies pour permettre au servlet de formuler sa réponse au client. Certaines de ces méthodes proviennent de l'interface ServletResponse , l'interface que HttpServletResponse étend. La figure 19.4 décrit quelques-unes de ces méthodes, exploitées dans ce chapitre.

Méthode	Description
`void addCookie( Cookie cookie )`	Ajoute un `Cookie` à l'en-tête de la réponse au client. L'âge maximum que peut atteindre le `Cookie` et, si le client autorise l'enregistrement des `Cookie`, détermine si oui ou non les `Cookie` seront stockés sur le client.
`ServletOutputStream getOutputStream()`	Obtient un flux de sortie en octets qui permet l'envoi de données binaires au client.
`PrintWriter getWriter()`	Obtient un flux de sortie de caractères qui permet l'envoi de données du genre texte au client.
`void setContentType( String type )`	Spécifie le type MIME pour les réponses au navigateur. Le type MIME permet au navigateur de déterminer la manière dont il doit afficher les données (ou éventuellement quelle autre application il lui faut exécuter pour traiter ces données). Par exemple, le type MIME "`text/html`" indique que la réponse est un document HTML, de sorte que le navigateur affiche la page HTML.

Modelisation et réalisation :

1)Spécification fonctionnelles :

Pour donner un exemple à la mise en service des servlets j'ai opté pour réaliser le concepte de la boite à idées en application web en utilisant la technologie Java/JEE

Presentation et outils utilisés :

2)Spécification techniques et mise en oeuvre :

Partie Base de données :

Partie

3)Conclusion et perspectives :