第 9 章　WebApp 软件工程

本章内容

9.1 Web 的特性

Web 前端开发是一项很特殊的工作，涵盖的知识面非常广，既有具体的技术，又有抽象的理念。简单地说，它的主要职能就是把网站的界面更好地呈现给用户。

Web 是图形化和易于导航的

Web 与平台无关

Web 是分布式的

Web 是动态的

Web 是交互的

数据集可重复利用

本章内容

9.2.1 二层C/S 结构

二层C/S 系统有很多优点，例如用户使用简单、直观，编程、调试和维护费用低，系统内部负荷可以做到比较均衡且资源利用率较高，允许在一个客户机上运行不同计算机平台上的多种应用，系统易于扩展，可用性较好，对用户需求变化的适应性好。

1 瘦客户机模型

- 如果所有形式逻辑和业务逻辑均驻留在客户机，服务器成为数据库服务器，负责各种数据的处理和维护，因此服务器变得很“瘦”，称为“瘦服务器”（Thin Server）。

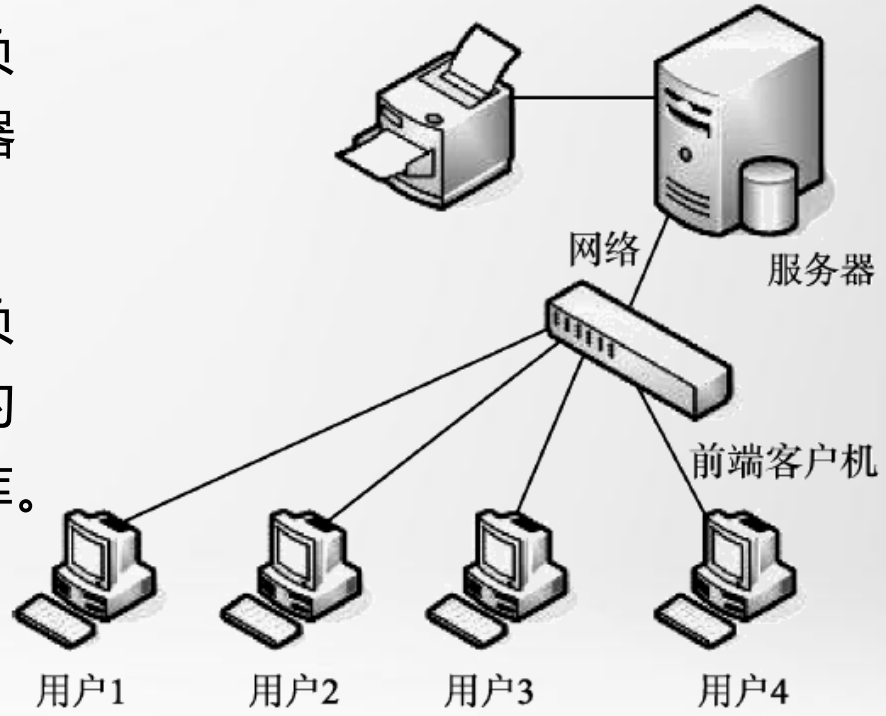
2 胖客户机模型

- 胖客户机（Fat Client）模型与瘦客户机模型相反，需要在客户端运行庞大的应用程序，由客户机上的软件实现应用逻辑和系统用户的交互，服务器只负责对数据的管理。

9.2.1 二层C/S 结构

二层C/S 结构由前端客户机、后端服务器、网络共3 部分组成，如图所示。

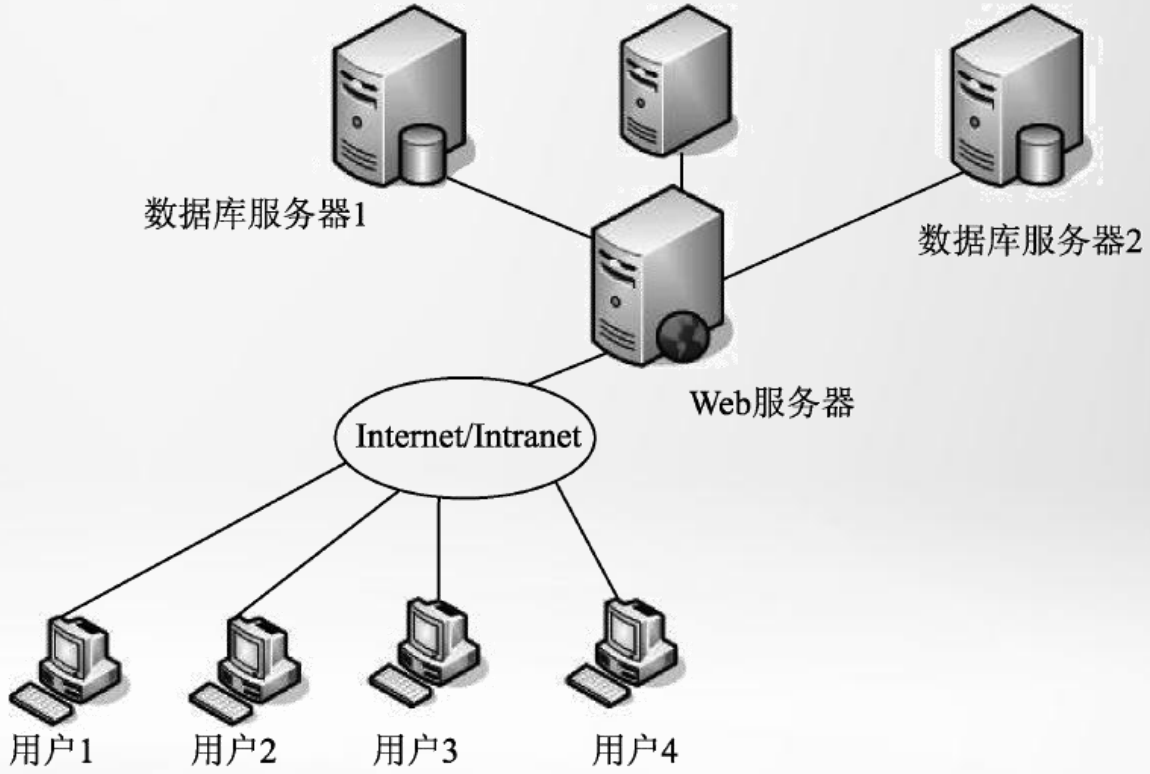
- **前端客户机：**二层C/S 结构的前端客户机负责接收用户发出的请求，并向数据库服务器发出请求。
- **后端服务器：**二层C/S 结构的后端服务器负责提供完善的安全保护以及对数据完整性的处理，并允许多个用户同时访问一个数据库。
- **网络：**客户机和服务器通过网络连接。



9.2.2 二三层C/S 结构

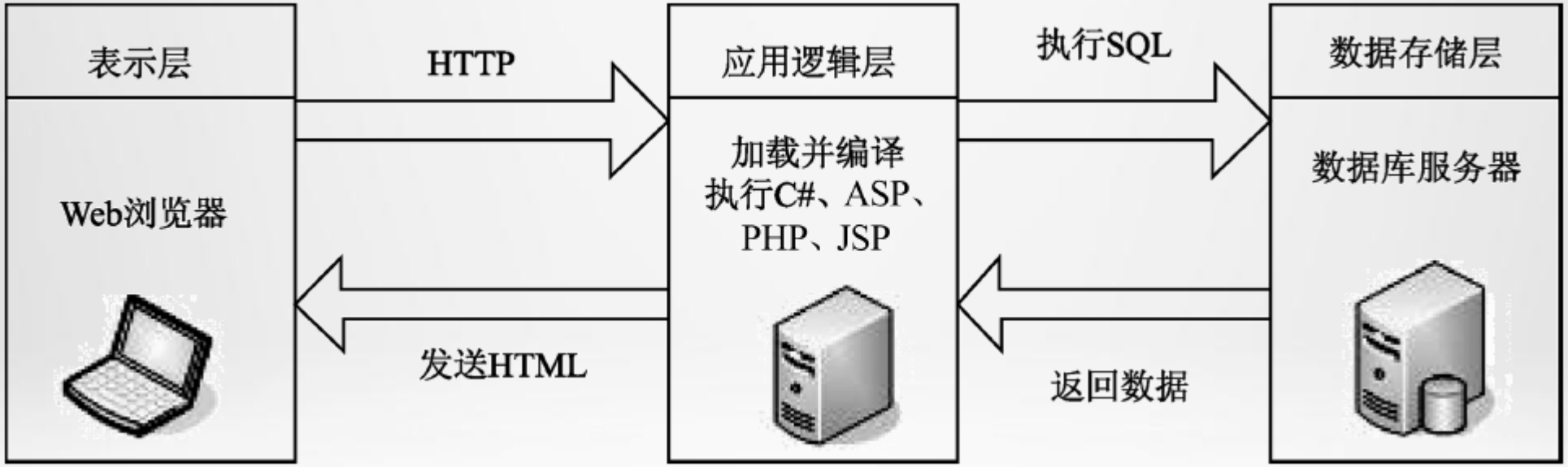
三层C/S 结构如图所示，包括Web 服务器、数据库服务器以及客户机。Web 服务器既作为一个浏览服务器，又作为一个应用服务器，系统将整个应用逻辑放在Web 服务器上，而客户机上只有表示层。

和二层C/S 结构相比，三层C/S 结构具有更灵活的硬件系统构成，对于各个层可以选择与其处理负荷和处理特性相适应的硬件。



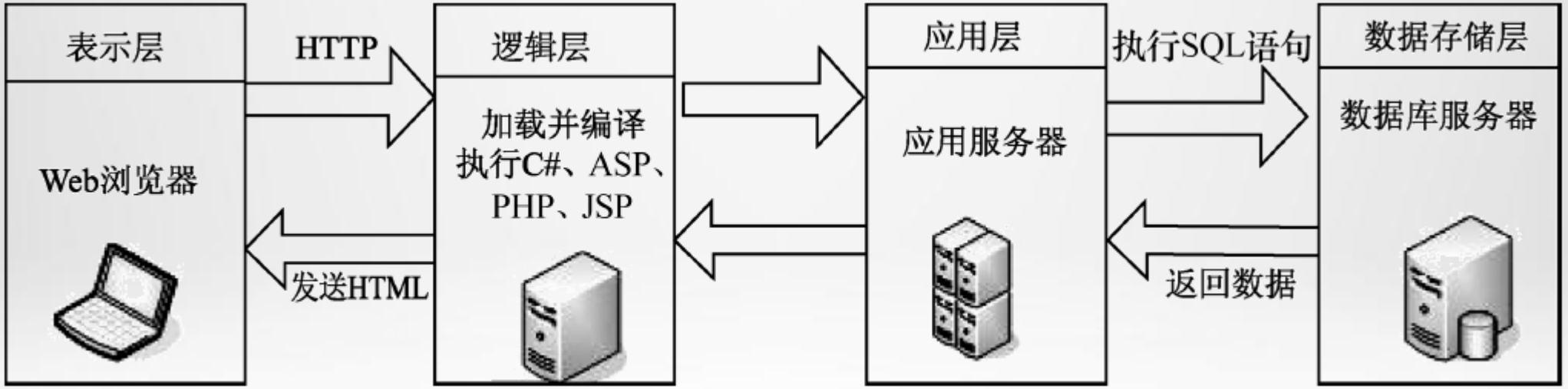
9.2.2 二三层C/S 结构

三层C/S 结构系统将整个系统分成表示层、应用逻辑层和数据存储层3 个部分，其数据处理流程如图所示。



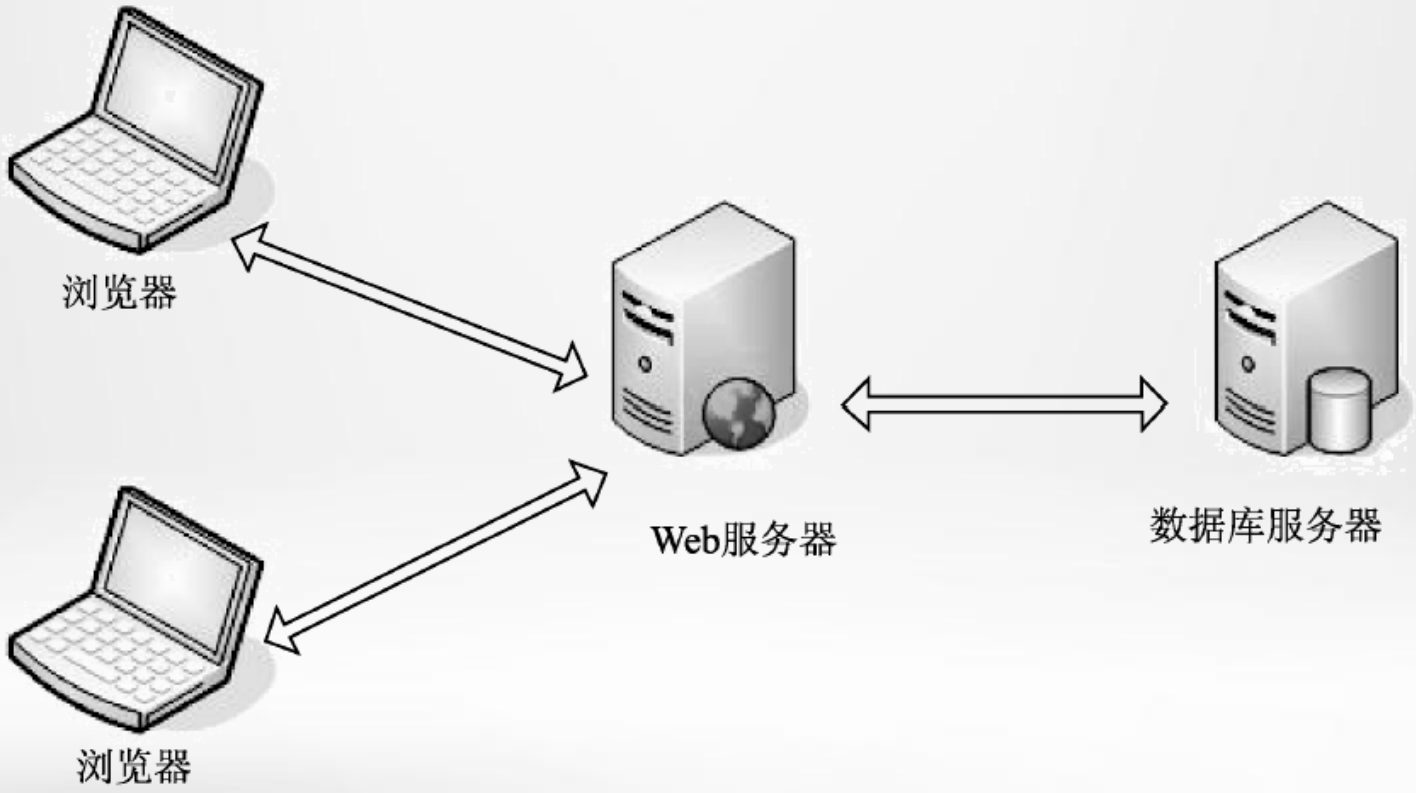
9.2.3 四层C/S 结构

由于三层C/S 结构的通信效率以及扩展性还不够高，因此提出了一种四层C/S 解决方案，该方案在Web 服务器和数据库之间使用了一层中间件，通常称为应用服务器。应用服务器负责将API 提供给业务逻辑和业务流程以供程序使用，可以根据需要引入其他Web 服务器。



9.2.4 B/S 结构

随着Internet 的兴起，人们提出了B/S（Browser/Server，浏览器/ 服务器）结构。从本质上说，B/S 结构也是一种C/S 结构，它可看作C/S 结构在Web 上应用的特例。图所示为B/S 结构。



9.2.4 B/S 结构

01 OPTION B/S 结构的特点

1 维护和升级方式简单

- 软件系统的改进和升级越来越频繁，B/S 结构的产品明显体现出系统改进和升级更为方便的特性。客户机越来越“瘦”而服务器越来越“胖”是将来信息化发展的主流方向，可节约人力、物力、时间和费用。

2 成本降低，选择更多

- B/S 结构提供了异种机、异种网、异种应用服务的联机、联网、统一服务的开放性基础。

9.2.4 B/S 结构

02 OPTION B/S 结构的缺点

- B/S 结构缺乏对动态页面的支持能力，没有集成有效的数据库处理能力。
- B/S 结构的系统扩展能力差，安全性难以控制。
- 采用B/S 结构的应用系统在数据查询等方面的响应速度上，要远远低于C/S 结构。
- B/S 结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理应用。
- 应用服务器运行数据负荷较重。

本章内容

9.3.1 HTML

HTML 是一种建立网页文件的语言，通过标记（Tag）式的指令，将影像、声音、图片、文字、动画、影视等内容显示出来。HTML 文档主要包括文档内容、文档标记和HTML 超链接 3 部分。HTML 的基本结构：

```
<html>
<head>
<title> 网页的标题</title>
</head>
<body>
在网页中要显示的内容
</body>
</html>
```

9.3.2 脚本语言

脚本语言用在HTML 文档中，用于丰富用户的交互，是一种介于HTML 和VB、Java 等高级语言之间的一种语言。常用的脚本语言有VBScript 和JavaScript 等语言，默认的脚本语言为VBScript 语言。

9.3.2 HTML

01
OPTION

JavaScript 的特点

解释性

用于客户端

安全性

简单、易用

动态性

跨平台性

9.3.2 Applet

02 OPTION

JavaScript 的作用

1 校验用户输入的内容

- 对用户输入内容进行校验

2 编写JavaScript脚本

- 可以使用任何一种文字编辑器来编写

3 执行

- 与HTML 文档配合，将其插入HTML 文档中，然后通过浏览器执行HTML 文档

9.3.3 Applet

Applet 就是用Java 语言编写的一些应用程序，它们可以直接嵌入网页中，并能够产生特殊的效果。当用户访问带有Applet 的网页时，Applet 被下载到用户计算机上执行，但前提是用户使用的是支持Java 的网络浏览器。

Applet 可以实现图形绘制、字体和颜色控制、动画和声音的插入、人机交互及网络交流等功能。

9.3.4 AJAX

01 OPTION AJAX 与传统的Web 应用比较

AJAX 应用可以仅向服务器发送并取回必需的数据，它使用简单对象访问协议（Simple Object Access Protocol，SOAP）或其他一些基于XML 的Web 服务器接口，并在客户端采用JavaScript 处理来自服务器的响应，因为在服务器和浏览器之间交换的数据大量减少，所以响应更快

9.3.4 AJAX

02 OPTION AJAX 应用程序的优势

- 采用异步模式。
- 优化了浏览器和服务器之间的传输，减少了不必要的数据往返，减少了带宽占用。
- AJAX 引擎在客户端运行，承担了一部分本来由服务器承担的工作，从而减少了大用户量下的服务器负载。

本章内容

9.4.1 Servlet

01 OPTION Servlet 与Applet 的异同点

相似之处

- 它们不是独立的应用程序，没有main() 方法。
- 它们不是由用户调用，而是由另外一个应用程序（容器）调用。
- 它们都有一个生存周期，包含init() 和destroy() 方法。

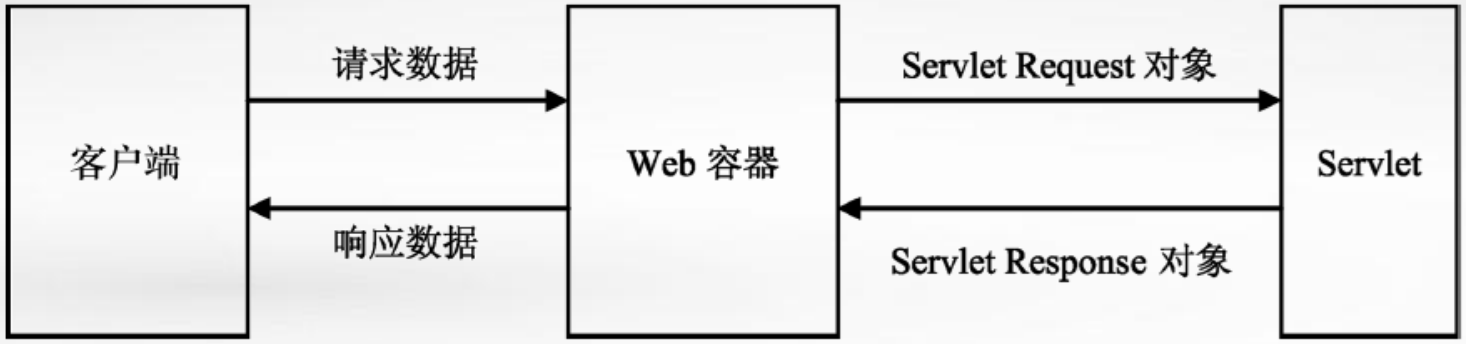
不同之处

Applet 运行在客户端，具有丰富的图形界面；Servlet 运行在服务器端，没有图形界面

9.4.1 Servlet

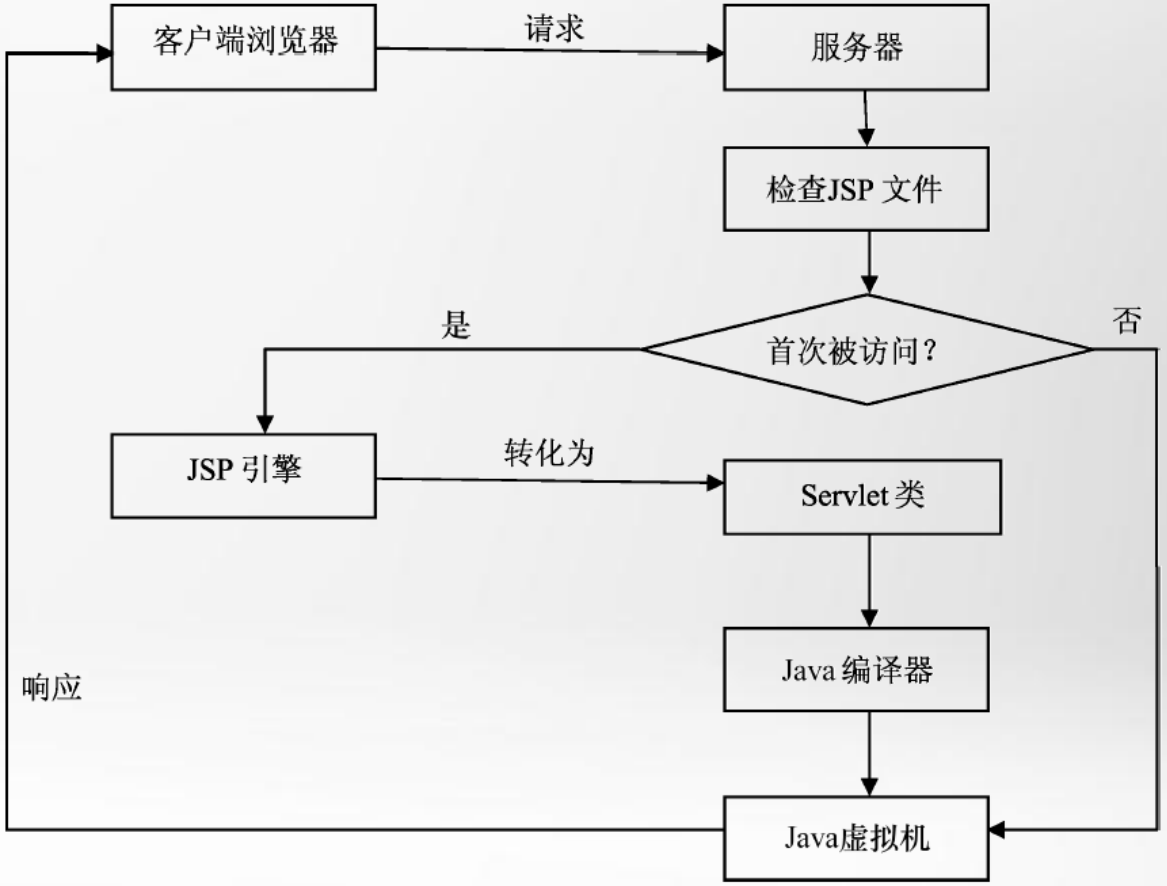
02 OPTION Servlet 的工作原理

- 客户端将请求发送到服务器端。
- 服务器上的Web 容器实例化（装入）Servlet。
- Web 容器将请求信息发送到Servlet。
- Servlet 创建一个响应，并将其返回Web 容器。
- Web 容器将响应发回客户端。
- 服务器关闭或Servlet 空闲时间超过一定限度时，调用destory() 方法退出。



9.4.2 JSP

JSP 是Servlet 技术的变形，其调用过程如图所示。JSP 本质是一个Servlet，它的运行需要容器的支持。在 JSP 和 Servlet 文件中都可以编写Java 和HTML 代码，不同的是，Servlet 虽然可以动态生成页面内容，但更偏向于逻辑控制，而JSP 更加偏向于页面视图的展现。



本章内容

9.5.1 观察者模式

01 OPTION 观察者模式的组成

抽象主题角色

把所有对观察者对象的引用保存在一个集合中，每个抽象主题角色都可以有任意数量的观察者

抽象观察者角色

为所有具体的观察者定义一个接口，在得到主题的通知时更新自己

具体主题角色

在具体主题内部状态改变时，给所有登记过的观察者发出通知。

具体观察者角色

具体观察者角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态相协调

9.5.1 观察者模式

02 OPTION 观察者模式的实现方式

观察者模式的优点

- 观察者模式在被观察者和观察者之间建立一个抽象的耦合
- 观察者模式支持广播通信，被观察者会向所有登记过的观察者发出通知

观察者模式的缺点

- 如果一个被观察者有很多直接和间接的观察者，将所有观察者都通知到需花费很多时间。
- 如果在被观察者之间有循环依赖，被观察者会触发它们之间的循环调用，从而导致系统崩溃。在使用观察者模式时要特别注意这一点。
- 如果对观察者的通知是通过另外的线程进行异步投递的，系统必须保证投递的可达性，也就是要保证所有观察者都能收到通知。
- 虽然观察者模式可以使观察者随时知道所观察的对象发生了变化，但是观察者模式没有相应的机制使观察者知道所观察的对象是怎么发生变化的。

9.5.1 观察者模式

03 OPTION 观察者模式的形式

实现观察者模式有很多形式，比较直观的一种是“注册 -通知-撤销注册”的形式。

1 注册

- 观察者将自己注册到被观察者中，被观察者将观察者存放在一个容器里

2 通知

- 如果被观察者发生了某种变化，要从容器中得到所有注册过的观察者，将变化通知注册过的观察者

3 撤销注册

- 观察者告诉被观察者要撤销观察，被观察者从容器中将观察者移除

9.5.1 观察者模式

04 OPTION 观察者模式的注意事项

实现观察者模式的时候要注意，观察者和被观察者之间的互动关系不能体现成类之间的直接调用，否则将使观察者和被观察者之间紧密耦合，从根本上违反面向对象设计的原则。

9.5.2 组合模式

组合模式是指一个对象可以由其他对象组合而成，这是一种对象的树形结构形式。组合模式对单个对象（叶子对象）和组合对象（容器对象）的使用具有一致性。组合模式又可以称为整体- 部分（Part-Whole）模式，它是一种对象结构型模式。

1 抽象构件

- 抽象构件可以是接口或抽象类，为叶子构件和容器构件对象声明接口，抽象构件中可以包含所有子类共有行为的声明和实现

2 叶子构件

- 在组合模式中表示叶子结点对象，叶子结点没有子结点，它实现了在抽象构件中定义的行为

3 容器构件

- 容器构件在组合模式中表示容器结点对象，容器结点包含子结点，其子结点可以是叶子结点，也可以是容器结点

9.5.3 工厂方法模式和策略模式

01 OPTION 工厂方法模式

实现观察者模式有很多形式，比较直观的一种是“ 注册 -通知-撤销注册”的形式。

1 工厂方法模式的主要优点

- 可以使代码结构清晰，有效地封装变化
- 对调用者屏蔽具体的产品类
- 降低耦合度

2 工厂方法模式的要素

- 工厂接口
- 工厂实现
- 产品接口
- 产品实现

9.5.3 工厂方法模式和策略模式

02 OPTION 策略模式

策略模式是工厂方法模式的一种变形，它提供了一种用多个行为中的一个行为来配置一个类的方法，即实现某一个功能有多种算法或者策略，根据环境或者条件的不同，可以选择不同的算法或者策略来完成该功能。

工厂方法模式是创建型模式，它关注对象创建，提供创建对象的接口，让对象的创建与具体使用的客户无关。

9.5.4 模型- 视图- 控制器模式

01 OPTION MVC 模式的系统组成

模型

模型（Model）表示企业数据和业务规则，负责业务流程的处理。在MVC 的3 个子系统中，模型拥有最多的处理业务。

视图

视图（View）是用户看到并与之交互的界面。对Web 应用系统来说，视图可以由HTML元素构成，也可以由其他一些技术（如XML、XHTML）等来构建

控制器

控制器（Controller）接收用户的输入，并调用模型和视图对数据进行处理，共同满足用户的请求

9.5.4 模型- 视图- 控制器模式

02 OPTION MVC 模式的工作流程

- 控制器接收来自远程客户端的HTTP 请求，将其转换为对模型的业务逻辑处理功能的调用。
- 模型进行业务逻辑处理，在处理过程中可访问位于后端的数据库，处理完成后将结果送给控制器。
- 控制器根据模型的处理结果创建新的视图、选择其他视图或更新已有视图。
- 视图在接收控制器的命令后从模型中获取数据并生成HTTP 应答，主要内容为HTML 数据。
- 远程客户端的Web 浏览器接收并解析来自视图的HTML 数据，呈现Web 软件的用户界面。
- 当用户在浏览器的操作界面发出HTTP 请求时，重复上述操作。

9.5.5 装饰者模式

01 OPTION 装饰者模式的设计原则

实现观察者模式有很多形式，比较直观的一种是“ 注册 -通知-撤销注册”的形式。

1 多用组合，少用继承

- 利用继承设计子类的行为，是在编译时静态决定的，而且所有子类都会继承相同的行为。

2 类设计时要注意对扩展开放，对修改关闭

9.5.5 装饰者模式

02 OPTION 装饰者模式的特点

- 装饰者和被装饰者有相同的超类。
- 可以用一个或多个装饰者包装一个对象，各种继承的装饰可以应用于同样的对象，而它们都给原来的对象添加一个新的外部包装器，并增加对象的职责。
- 装饰者可以在受托的被装饰者的行为之前或之后加上自己的行为，以达到特定的目的。
- 对象可以在任何时候被装饰，可以在运行时动态地装饰，可以用任意喜欢的装饰者来装饰对象。
- 装饰者模式中使用继承的关键是达到装饰者和被装饰者的类型匹配，而不是获得其行为。
- 装饰者一般对组件的客户是透明的，除非客户程序依赖于组件的具体类型。

9.5.5 装饰者模式

03
OPTION

装饰者模式的优点

- 装饰者模式比继承有更多的灵活性。
- 通过使用不同的具体装饰类以及这些装饰类的排列组合，设计人员可以创造出很多不同行为的组合。

9.5.5 装饰者模式

04 OPTION 装饰者模式的缺点

- 装饰者模式比继承有更加灵活的特性，同时意味着有更多的复杂性。
- 装饰者模式会导致设计中出现许多小类，如果过度使用，会使程序变得很复杂。

本章内容

9.6.1 WebApp 的特点及应用类型

01 OPTION WebApp 的特点

- 网络密集型
- 并发性
- 无法预计的负载量
- 性能
- 可得性
- 数据驱动
- 内容敏感性
- 持续演化性
- 即时性
- 保密性
- 美观性

9.6.1 WebApp 的特点及应用类型

02 OPTION WebApp 的应用类型

- **信息型**：使用简单的导航和链接提供只读内容。
- **下载型**：用户从相应的服务器下载信息。
- **可定制型**：用户根据自己的特殊需要定制内容。
- **交互型**：在用户群体中，通过聊天室、公告牌或即时消息传递进行通信。
- **用户输入型**：基于表格的输入是满足通信要求的主要机制。
- **面向事物型**：用户提交一份由WebApp 完成的请求（例如用户名注册）。
- **面向服务型**：应用程序向用户提供服务，例如辅助用户确定支付。
- **门户型**：应用程序将用户引导到本门户应用范围之外的其他Web 内容或服务。
- **数据访问型**：用户查询大型数据库并提取信息。
- **数据仓库型**：用户查询一组大型数据库并提取信息。

9.6.2 WebApp 的需求分析

01 OPTION WebApp 需求分析的5 个阶段

由于WebApp 需求分析的诸多特性，导致WebApp 需求分析中经常会出现需求范围未界定、需求未细化、需求描述不清楚、需求遗漏和需求互相矛盾等问题。

9.6.2 WebApp 的需求分析

02 OPTION WebApp 需求

软件需求分为业务需求、用户需求和功能需求。

9.6.3 WebApp 需求分析过程

01 OPTION 需求分析活动

现有的需求分析方法主要包括以下活动。

绘制系统关联图

创建用户界面原型

分析需求的可行性

创建数据字典

为需求建立模型

确定需求的优先级

9.6.3 WebApp 需求分析过程

02 OPTION 需求分析原则

注意需求描述用语

了解用户业务

描述产品的非功能特性

评估需求变更代价

用户参与

9.6.3 WebApp 需求分析过程

03 OPTION 软件需求规格说明书

软件需求规格说明书是按照系统的接口编写的。如图所示显示了基于电气与电子工程师协会推荐的一个模板，按对象组织的软件需求规格说明书。

- 在文档化系统接口的过程中，详细描述所有输入和输出；
- 根据接口的输入和输出重新陈述要求的功能；
- 设计证明能够完成每一个客户的相应的质量需求。

1. 文档的引言
   1.1 产品的目的
   1.2 产品的范围
   1.3 首字母的缩写词、缩略词、定义
   1.4 参考文献
   1.5 软件需求规格说明书剩余部分的概要介绍
2. 产品的总体描述
   2.1 产品的背景
   2.2 产品功能
   2.3 用户的特性
   2.4 约束
   2.5 假设和依赖关系
3. 说明需求
   3.1 外部接口需求
      3.1.1 用户界面
      3.1.2 硬件接口
      3.1.3 软件接口
      3.1.4 通信接口
   3.2 功能需求
      3.2.1 类1
      3.2.2 类2
      ……
   3.3 性能需求
   3.4 设计约束
   3.5 质量需求
   3.6 其他需求
4. 附录

9.6.4 WebApp 设计过程

01 OPTION WebApp 的设计过程

一个完整的WebApp 设计过程一般包含以下活动。

（1）设计策划

（2）体系结构设计

（3）人机交互设计

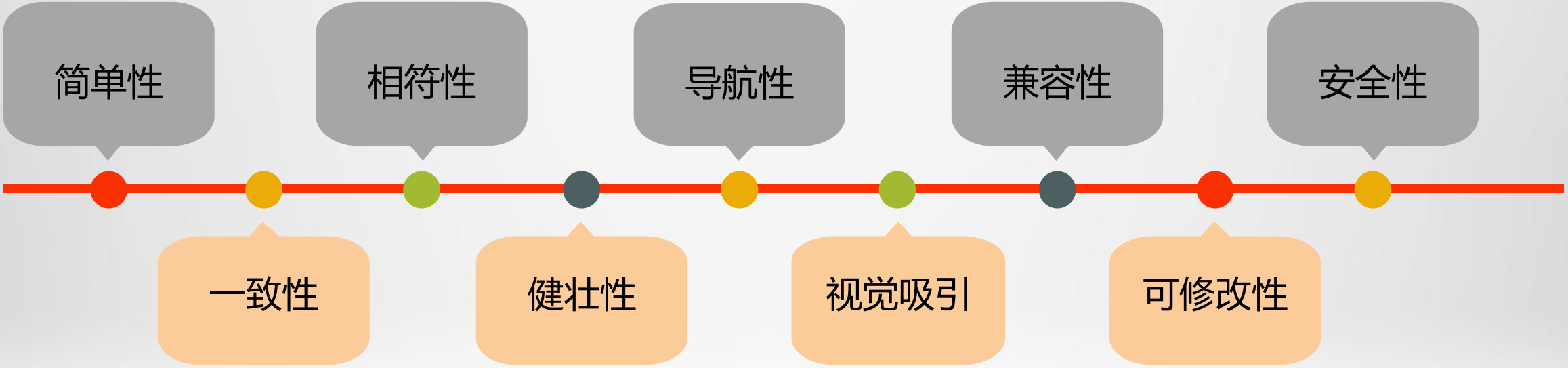
（4）详细设计

（5）设计整合与验证

（6）总结

9.6.4 WebApp 设计过程

02 OPTION WebApp 的设计目标



9.6.4 WebApp 设计过程

03
OPTION

WebApp 设计步骤

（1）界面设计

（2）美学设计

（3）内容设计

（4）体系结构设计

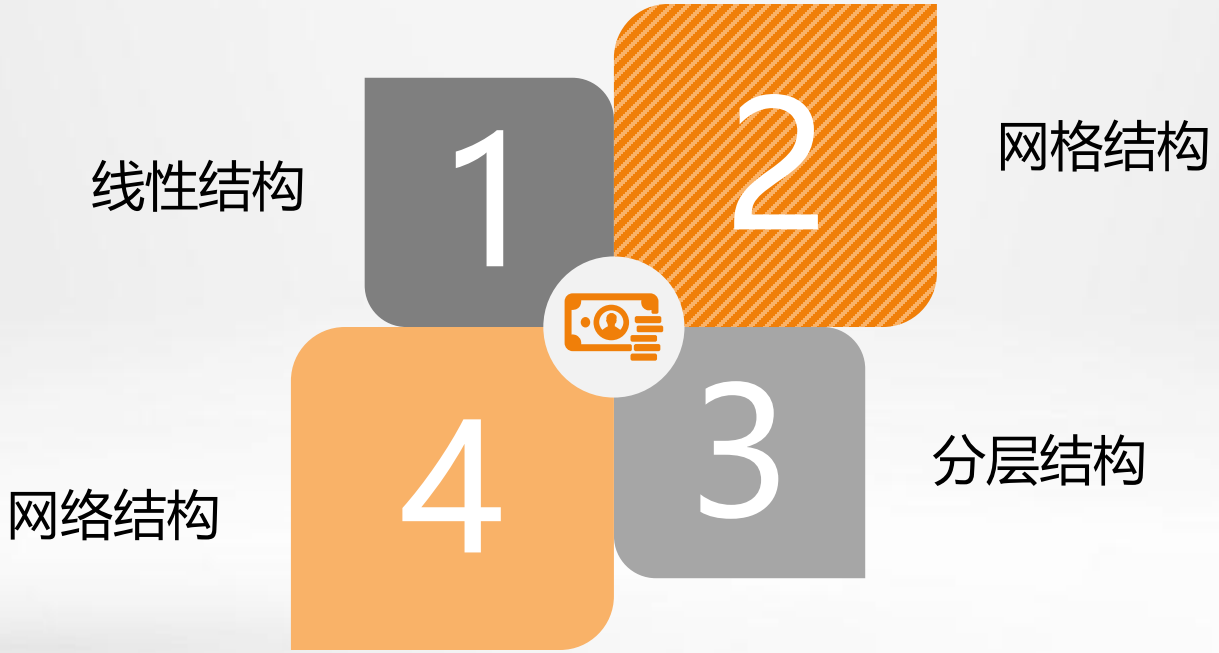
（5）构件设计

（6）导航设计

9.6.5 WebApp 体系结构设计

01 OPTION WebApp 内容体系结构

WebApp 内容体系结构着重于内容对象的表现和导航的组织方式，以及对WebApp 的所有超媒体结构进行定义。



9.6.5 WebApp 体系结构设计

02 OPTION WebApp 体系结构

WebApp 体系结构描述系统以什么样的组织方式来管理用户交互、实现导航及展示内容，描述使WebApp达到业务目标的基础结构。

9.6.6 WebApp 导航设计

01 OPTION 导航分析

在WebApp 中，每个体系结构的元素都有可能与所有其他体系结构的元素相链接，随着链接数目的增加，WebApp 导航的复杂性也增加。

共利益者分析：确定不同的用户种类，并建立适当的共利益者层次

元素分析：确定内容对象和最终用户感兴趣的功能元素

关系分析：描述WebApp 元素之间的关系

导航分析：检查用户怎样访问单个元素和成组元素

评估分析：考虑实现早期定义的关系的实际问题

9.6.6 WebApp 导航设计

02 OPTION 导航语义

在进行导航设计时，首先考虑用户层次和为每一类用户（角色）创建相关用例。每一类角色使用WebApp 的方式都会有所不同，因而会有不同的导航要求。为每一类角色设计的用例定义一组类，这组类包含一个或多个内容对象，或者包含WebApp 的某些功能，即每一组用例访问不同的信息及WebApp 功能。

9.6.6 WebApp 导航设计

03 OPTION 导航机制

单独的导航链接

基于文本的链接、图标、按钮、开关等

水平（或垂直）导航条

在包含合适链接的工具条中列出重要的内容或功能类别

标签

代表内容或功能类别，在需要链接时作为标签页选中

网站地图

向包含在WebApp中的所有内容对象和功能提供完整的导航内容表格

本章内容

9.7.1 WebApp 测试过程概述

- 内容测试试图发现内容方面的错误。
- 界面测试验证用户界面的交互机制及美学方面的设计。
- 导航测试时，对照导航设计检查每一个使用场景，识别并改正用例完成时产生的错误。
- 配置测试可发现特定的客户端或服务器环境中的错误。
- 安全性测试，考虑WebApp 及其环境中的弱点，测试是否有可能产生影响安全性的破坏。
- 性能测试，考虑WebApp 是否在各种环境下均能顺利运行。

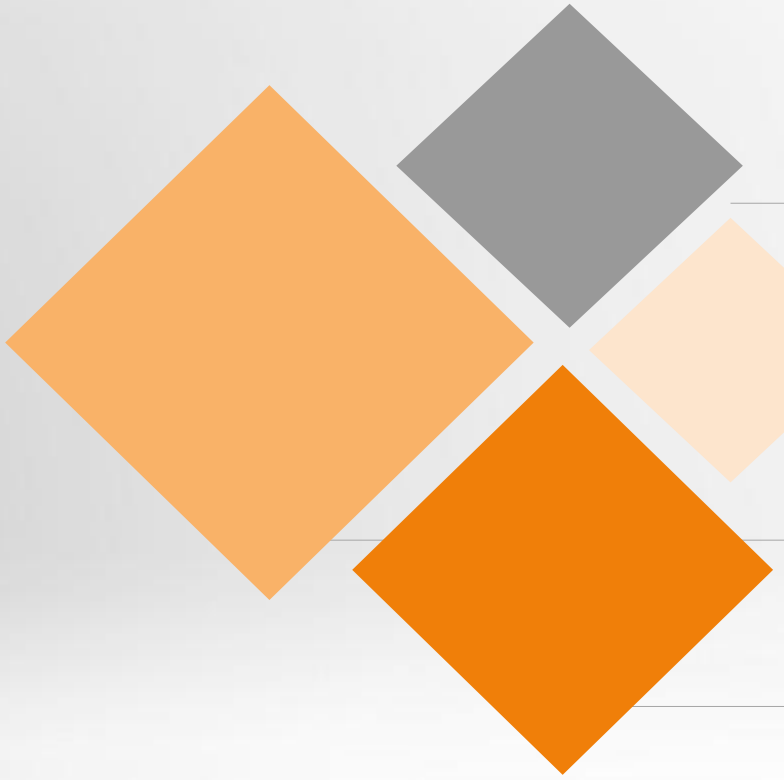
9.7.2 WebApp 内容测试

01 OPTION 内容测试

- 发现基于文本的文档、图形表示和其他媒体中的语法错误。
- 发现导航的语义错误，即信息的精确性和完备性方面的错误。
- 发现展示给用户的内容、组织、结构方面的错误。

9.7.2 WebApp 内容测试

02 OPTION 数据库测试



有效信息通过界面层在客户端与服务器端之间传递

WebApp 正确地处理脚本，并且正确地抽取或格式化用户数据

用户数据被正确地传递给服务器端的数据转换功能

查询被传递到数据管理层，数据管理层与数据访问程序通信必须正确

9.7.3 WebApp 界面测试

01 OPTION 交互测试

当用户与WebApp 交互时，通过一种或多种界面机制发生交互。以下是做界面测试时要检查的内容。

9.7.3 WebApp 界面测试

02 元素测试

OPTION

- 导航
- 可访问性
- 跨浏览器测试
- 错误消息和警告消息
- 帮助和文档
- 布局
- 其他

9.7.4 WebApp 导航测试

导航测试的第一个阶段在界面测试期间就开始了，对每一个导航应该在如下各方面进行测试。

（1）导航链接

（2）重定向

（3）书签

（6）内部搜索引擎

（5）站点地图

（4）框架和框架集

9.7.5 WebApp 配置测试

配置测试的工作不是检查每一个可能的客户端配置，而是测试一组可能的客户端和服务器配置，确认WebApp 在所有配置中的运行情况。

1 服务器端

- 验证所计划的服务器配置是否都能支持WebApp
- 验证WebApp 与服务器操作系统是否兼容
- 验证WebApp 是否与数据库软件进行适当的集成，是否支持不同版本的数据库
- 验证服务器端的WebApp 脚本运行是否正常等

2 客户端

- 在客户端，不同的硬件（包括CPU、内存、存储器、打印设备）、操作系统、浏览器软件、用户界面构件、插件等，都有可能会影响WebApp 的运行

9.7.6 WebApp 安全性测试

大多数WebApp 会从用户那里获取并存储数据，包括用户的个人信息、计费信息和工作及个人文件，这些数据都是用户在信任WebApp 的应用安全性的基础上才会输入的。

对私人数据进行加密

在授予访问权限之前坚持进行身份验证，并对数据访问进行限制

确保数据完整性，尊重用户的要求

9.7.7 WebApp 性能测试

性能测试通常会检查如下系统性能。

- 服务器响应时间是否降低到不可接受的程度?
- 从用户、事务、数据负载等方面，观察系统在什么情况下性能会降低到不可接受的程度。
- 在多种负载条件下，系统对用户的平均响应时间是多少?
- 性能下降是否影响系统的安全性?
- 当系统的负载增加时，WebApp 的可靠性和精确性是否会受影响?
- 当负载大于服务器容量的最大值时，会发生什么情况?