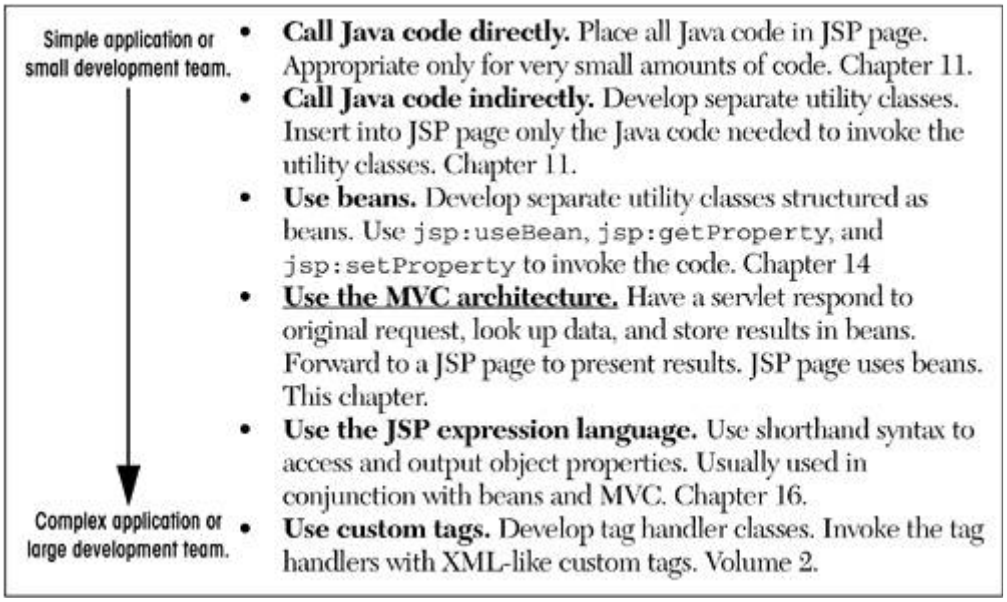
15.1 Understanding the Need for MVC

Servlets are great when your application requires a lot of real programming to accomplish its task. As illustrated earlier in this book, servlets can manipulate HTTP status codes and headers, use cookies, track sessions, save information between requests, compress pages, access databases, generate JPEG images on-the-fly, and perform many other tasks flexibly and efficiently. But, generating HTML with servlets can be tedious and can yield a result that is hard to modify.

That's where JSP comes in: as illustrated in Figure 15-1, JSP lets you separate much of the presentation from the dynamic content. That way, you can write the HTML in the normal manner, even using HTML-specific tools and putting your Web content developers to work on your JSP documents. JSP expressions, scriptlets, and declarations let you insert simple Java code into the servlet that results from the JSP page, and directives let you control the overall layout of the page. For more complex requirements, you can wrap Java code inside beans or even define your own JSP tags.

Figure 15-1. Strategies for invoking dynamic code from JSP.



Great. We have everything we need, right? Well, no, not quite. The assumption behind a JSP document is that it provides a *single* overall presentation. What if you want to give totally different results depending on the data that you receive? Scripting expressions, beans, and custom tags, although extremely powerful and flexible, don't overcome the limitation that the JSP page defines a relatively fixed, top-level page appearance. Similarly, what if you need complex reasoning just to determine the type of data that applies to the current situation? JSP is poor at this type of business logic.

The solution is to use *both* servlets and JavaServer Pages. In this approach, known as the Model View Controller (MVC) or Model 2 architecture, you let each technology concentrate on what it excels at. The original request is handled by a servlet. The servlet invokes the business-logic and data-access code and creates beans to represent the results (that's the *model*). Then, the servlet decides which JSP page is appropriate to present those particular results and forwards the request there (the JSP page is the *view*). The servlet decides what business logic code applies and which JSP page should present the results (the servlet is the *controller*).

MVC Frameworks

The key motivation behind the MVC approach is the desire to separate the code that creates and manipulates the data from the code that presents the data. The basic tools needed to implement this presentation-layer separation are standard in the servlet API and are the topic of this chapter. However, in very complex applications, a more elaborate MVC framework is sometimes beneficial. The most popular of these frameworks is Apache Struts; it is discussed at length in Volume 2 of this book. Although Struts is useful and widely used, you should not feel that you must use Struts in order to apply the MVC approach. For simple and moderately complex applications, implementing MVC from scratch with `RequestDispatcher` is straightforward and flexible. Do not be intimidated: go ahead and start with the basic approach. In many situations, you will stick with the basic approach for the entire life of your application. Even if you decide to use Struts or another MVC framework later, you will recoup much of your investment because most of your work will also apply to the elaborate frameworks.

Architecture or Approach?

The term "architecture" often connotes "overall system design." Although many systems are indeed designed with MVC at their core, it is not necessary to redesign your overall system just to make use of the MVC approach. Not at all. It is quite common for applications to handle some requests with servlets, other requests with JSP pages, and still others with servlets and JSP acting in conjunction as described in this chapter. Do not feel that you have to rework your entire system architecture just to use the MVC approach: go ahead and start applying it in the parts of your application where it fits best.