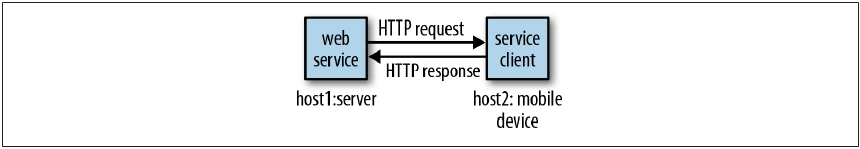
**Web Services Quickstart**

Although the term *web service* has various, imprecise, and evolving meanings, a working definition should be enough for the upcoming code example, which consists of a service and a client, also known as a consumer or requester. As the name suggests, a web service is a kind of webified application – an application typically delivered over HTTP (HyperText Transport Protocol). HTTPS (HTTP Secure) adds a security layer to HTTP; hence, a service delivered over HTTPS likewise counts as a web service. Until the main topic of interest is web service security, HTTP should be understood to include HTTPS.

Amazon, a pioneer in web services, is well known for its various websites, among which is the E-Commerce site for shopping. Amazon has other popular websites as well. Of interest here is that the data and functionality available at Amazon websites are likewise available as Amazon web services. For example, someone can use a browser to shop interactively at the Amazon E-Commerce site, but this person also could write a program, as later examples show, to do the shopping through the corresponding Amazon E-Commerce web service. Amazon is particularly good at pairing off its websites with web services.

Web services can be programmed in a variety of languages, old and new. The obvious way to publish a web service is with a web server; a web service client needs to execute on a machine that has network access, usually over HTTP, to the web server. In more technical terms, **a web service is a distributed software system whose components** **can be deployed and executed on physically distinct devices**. Consider, for example, a web server *host1* that hosts a web service and a mobile device *host2* that holds an application issuing requests against the service on *host1.* Web services may be more architecturally complicated that this, of course; for one thing, a service may have many clients issuing request against it, and the service itself may be composed of other services. For instance, a stock-picking web service might consist of several code components, each hosted on a separate commercial-grade web server, and mix of PCs, handhelds, and other networked devices might host programs that consume the service. Although the building blocks of web services are relatively simple, the web services themselves can be arbitrarily complex.

****

An HTTP request goes, by definition, from client to server, and an HTTP response goes, also by definition, from server to client. For web services over HTTP, the HTTP messages are the infrastructure, and these HTTP messages can be combined into basic conversational patterns that characterize a web service. For example, if the web service conversation starts with an HTTP request that expects an HTTP response, the pattern is the familiar ***request/response*** conversation. By contrast, if the conversation starts with an HTTP message from the server a message that expects a message from the client in return, then the pattern is ***solicit/response*.** Richer conversational pattern can be composed of even more primitive ones: a message from client to server without a response is a pattern known as ***one-way,*** and the reverse pattern, from server to client without a client response, is known as ***notification.*** Web services tend to be simple in structure. The four conversational patterns enumerated just now cover most modern web services, and request/response is the pattern that still dominates.

Web Services come in two popular flavors: SOAP- based and REST-style. SOAP is an XML dialect with a grammar that specifies the structure that a document must have in order to count as SOAP. In a typical SOAP-based service, the client sends SOAP messages to the service and the service responds in kind, with SOAP messages. REST-style services are hard to characterize in a sentence or two, but with respect to pattern, these services tend to be request/response; the same holds for SOAP-based services. For now, a REST-style service is one that treats HTTP not only as transport infrastructure but also as a set of guidelines for designing service requests and service responses. In a REST- style service, HTTP itself can be seen as an API. SOAP has standards, toolkits, and bountiful software libraries. REST has no official standards, comparatively few toolkits, and uneven software libraries among programming languages. Yet there is growing support for REST – style services across programming languages; hence, it seems only a matter of time until toolkits and libraries for REEST- style services mature.

SOAP originally stood for **Simple Object Access Protocol** and then, by serendipity but never officially, might have stood **for Service-Oriented Architecture (SOA) protocol.**