

# Implementing C++ Servlet Containers

John Hinke

**By combining the performance benefits of C++ and the power of the Java servlet API, you can more easily create robust web applications.**

John is a software architect for Rogue Wave Software. He can be contacted at [hinke@roguewave.com](mailto:hinke@roguewave.com).

---

Creating applications for the Internet with C++ can be difficult, mostly due to the lack of pure C++ technologies that can be used. Consequently, most C++ developers are required to use another language for interfacing their C++ code with the Internet. They might use a Java servlet combined with Java Native Interface (JNI) for integrating their C++ code, or they may write [web server](#) adapters such as an Apache module or an ISAPI filter.

What if the full power of the Java servlet API could be harnessed in C++ to create a pure C++ web application by using the same techniques used when creating Java [web applications](#)? C++ developers would then be able to create C++ web applications without needing to use multiple languages, or using a slower technology such as Java or CGI.

In this article, I examine alternatives to C++ and some of the necessary details for creating a 100 percent pure C++ servlet container. Working in C++ is much more viable than the other alternatives because it is faster, more efficient, native to the application, and easier for the developer to learn and implement.

## C++ Alternatives

Today, C++ programmers have limited options for bringing their applications to the Web: Java servlets combined with JNI, CGI, or a proprietary web server's API.

A pure C++ servlet container provides a highly efficient, scalable, robust, and very portable servlet API. By creating a C++ servlet container, you have the necessary technology to quickly and easily create 100 percent C++ web applications.

**JNI.** One approach is to use Java servlets combined with JNI for access to legacy C/C++ code. The downside to this approach is that it requires using two languages and a good understanding of the complicated JNI interface. Additionally, there is an associated performance cost. While there are techniques for reducing the complexity of the JNI code, the performance and multilanguage issues are still prevalent. As you can see in [Figure 1](#), the pure C++ approach reduces many of these performance problems.

[Listing One](#) is a HelloWorld Java servlet that calls to a C++ method that executes the Hello World request. While that is a simple example, the JNI code is quite complex. [Listing Two](#) is the generated JNI layer.

Providing an implementation for this simple method is rather difficult and error prone. Also, JNI provides access to C methods, not C++ classes and methods. Therefore, you must manage accessing an instance of the C++ object and call the appropriate method.

**CGI.** While coding a C++ application to be used with CGI can be similar to coding a Java servlet, the C++ servlet gives much greater performance. Every request for a CGI application spawns a new process. This is not only resource intensive but prevents any of the CGI programs from efficiently

saving state. One single browser request can spawn several different CGI processes; multiply this by thousands of simultaneous requests and the performance of the web server is severely affected.

The C++ servlet container holds a servlet in memory and all requests go through that one particular servlet. The only process created is for the servlet container itself. By actively holding a servlet in memory, the C++ servlet container also allows the code to maintain state (variable values, paths of execution, and so on) while it executes. This allows for better usage of resources on a web server. With CGI, the state is reinitialized each time a program executes.

**Web Server Adapters.** Another option for C++ programmers is to use a proprietary API such as Netscape NSAPI or Microsoft ISAPI. While these allow a more managed approach to integrating an application to the Web than CGI, they lock developers into a single vendor and platform. In addition, these APIs often only allow access to C code and not C++, which can be very limiting.

## C++ Servlets: A Better Option

A pure C++ servlet container provides a highly efficient, scalable, robust, and very portable servlet API. A C++ servlet container solves the C++ developer's dilemma, providing a better way of integrating C++ with the Web than CGI. In addition, because they are compiled, C++ servlets provide better performance than Java, integrating easily with current C++ implementations with minimal effort and training at a significant cost savings. By following the Java servlet specification, you can quickly and easily learn this new technology.

## Creating a C++ API: The Design

First, some goals need to be established for creating a C++ servlet container to help guide the design and implementation.

- **Consistent API.** The C++ API should be nearly identical to Java, without any surprises. If parts of the Java API do not make sense in C++, they should either be removed or replaced with an alternate API. Users should be able to read any Java servlet book and be able to apply that information to a C++ version.
- **Performance.** The C++ version must be faster than Java. As this is an alternative to Java servlets, it should not introduce any unnecessary overhead.
- **Use C++ features where possible.** Examples might include templates, the Standard Library, and even destructors for cleaning up memory. C++ features should only be used if they aid in the usability of the API.

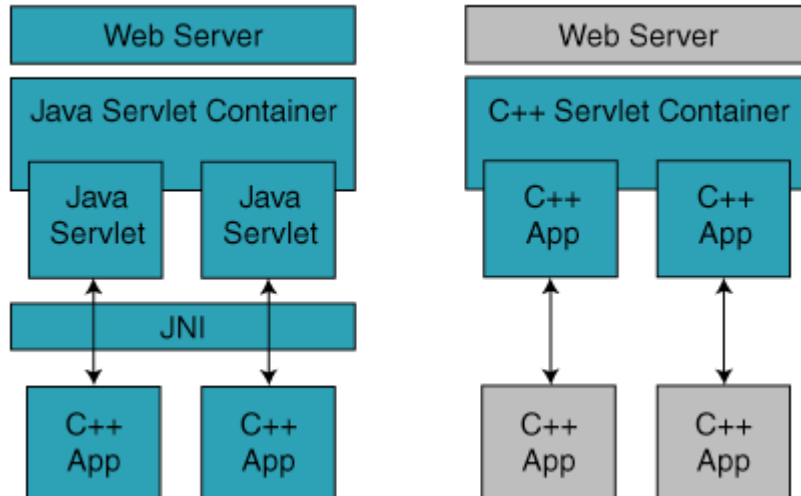
Second, port some sample Java servlets to C++. This is an exercise designed to help understand what the C++ API should look like. While the C++ API should be as close to the Java API as possible, it also should take advantage of C++-specific features.

While most of the Java API is fairly straightforward, there are a few places in the API that need special attention. (Issues such as connecting to the servlet container are beyond the scope of this article.)

## Conclusion

By porting the Java servlet API to C++, you finally have the power and flexibility of the servlet specification that Java developers have enjoyed for so long. Creating pure C++ web applications then becomes a matter of writing a C++ servlet and deploying that servlet into a pure C++ servlet container. You no longer need to use a multilanguage approach for their web applications. By combining the performance benefits of C++ and the power of the servlet API, C++ developers can easily create powerful, robust web applications.

For more information on C++ servlet containers and to download an evaluation version, see <http://www.roguewave.com/developer/tac/bobcat/>.



**Figure 1: The pure C++ approach reduces many performance problems.**

## DDJ

### Listing One

```
class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) {
        response.setContentType("text/plain");
        try {
            PrintWriter out = response.getWriter();
            out.println(helloWorld("Hello C++"));
        } catch(IOException e) { /* ignore for now. */ }
    }
    private native String helloWorld(String in);
}
```

[Back to Article](#)

### Listing Two

```
extern "C" {
    JNIEXPORT jstring JNICALL Java_HelloWorld_helloWorld
        (JNIEnv *, jobject, jstring);
}
```

[Back to Article](#)