WebLogic: The Definitive Guide: Development, Deployment & Maintenance (Definitive Guides)

Pages: 850
Publisher: O'Reilly Media; 1 edition (February 23, 2004)
Format: pdf, epub
Language: English

WebLogic: The Definitive Guide*Jon MountjoyAvinash Chugh*Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo Special Upgrade OfferIf you purchased this ebook directly from oreilly.com, you have the following benefits:DRM-free ebooks — use your ebooks across devices without restrictions or limitationsMultiple formats — use on your laptop, tablet, or phoneLifetime access, with free updatesDropbox syncing — your files, anywhereIf you purchased this ebook from another retailer, you can upgrade your ebook to take advantage of all these benefits for just $4.99. Click here to access your ebook upgrade.*Please note that upgrade offers are not available from sample content.* PrefaceWebLogic™ Server is one of the leading J2EE-compliant application servers, produced by BEA Systems. It implements the full range of J2EE technologies, and provides many more additional features such as advanced management, clustering, and web services. It forms the core of the WebLogic Platform, and provides a stable framework for building scalable, highly available, and secure applications. You can gauge the extent of its acceptance by exploring the list of customers who have embraced WebLogic Server from all segments of the industry. A number of partners also are collaborating with BEA to build a diverse range of products that integrate with WebLogic Server and the WebLogic Platform. This information on customers and partners involved with WebLogic is available on BEA's web site, http://www.bea.com/. Clearly, WebLogic Server has established a strong presence in the market.BEA is serious about WebLogic Server. It is one of the influential members on the various committees that decide on the future direction of the J2EE revolution. It has invested much financial and intellectual capital toward making WebLogic Server the leading Java© application server in the industry. Its dev2dev web site and newsgroups bring together developers, architects, and experts from different domains, all collaborating to make WebLogic Server as popular as it is. Over the years, two factors have determined the growth of WebLogic Server. First, WebLogic has consistently maintained compliance with the J2EE standards. Second, BEA has always been committed to ensuring that WebLogic meets and exceeds the needs of its customers, and provides a feature set that delivers a viable platform for building enterprise solutions. The combination of both of these factors has contributed to the widespread popularity of WebLogic Server.WebLogic continues to maintain its edge by offering robust implementation of current and upcoming J2EE technologies. It also offers a lot more, extending the base platform with features that let you build real enterprise solutions with cutting-edge technologies. A quick preview of some of its features will convince you that WebLogic is a force to contend with.Support for a wide range of platforms, including Windows 2000, XP, and 2003, Solaris, Red Hat and SuSE Linux, HP Tru64, HP-UX, IBM AIX, and other Unix flavors.A rich set of client options: HTTP clients, Java clients over RMI/IIOP and the native T3 protocol, and web service clients that support JAX-RPC or SOAP.A robust implementation of standard J2EE APIs, and support for the standard J2EE components such as servlets, JSP pages and custom tags, EJB and RMI objects, JMS producers and consumers, J2EE connectors, and more.An innovative strategy for deploying J2EE applications across multiple servers.The ability to interoperate with industry-strength HTTP servers, LDAP

servers, relational DBMS products, messaging software, enterprise information systems, load balancers, firewalls, and more.An easy, flexible way to configure the different J2EE containers, and the ability to benefit from practical extensions and optimizations.Designed to operate in a distributed environment with extensive support for failover and load balancing through clustering.An intuitive framework for organizing the different WebLogic resources in a domain, with an emphasis on performance and scalability.A stable, production-ready JMS server that supports distributed JMS destinations, configurable delivery options, extended acknowledgment modes, and much more.A comprehensive suite of security features that help protect the various applications and resources deployed in your WebLogic environment.Rich support for XML features, including a streaming and XPath API.A distributed management infrastructure that can be configured dynamically through JMX, and integrated into a larger SNMP framework.A distributed logging framework and support for i18n and l10n.A rich implementation of web services for building standards-compliant web services over existing J2EE components, with features such as SOAP security and reliable SOAP messaging.An integral component of the WebLogic Platform, over which other technologies such as WebLogic Integration and WebLogic Portal are built.Finally, WebLogic Server also comes bundled with an IDE, WebLogic Workshop. This IDE provides a fast route to implementing J2EE and web service components, featuring automatic code generation and graphical component composition.With these kinds of ingredients, we believe WebLogic Server has realized the perfect recipe to deal with other competing J2EE application servers. As you read on, we're certain you, too, will be convinced.Who Should Read This Book?J2EE is a mature platform for building enterprise-scale applications. This book caters to developers who are presently working on the WebLogic platform, and those J2EE programmers who are considering doing so in the future. If you are a programmer responsible for developing J2EE applications on WebLogic, an administrator who needs to manage the entire logical and physical setup for your application, or an architect responsible for evaluating which technologies, tools, and products should be used, this book is for you!Let's clarify how we believe this book can help the different groups within our target audience:Developers
For the most part, we assume that developers reading this book are familiar with the J2EE platform. Developers should turn to other books, online resources, and published specifications for learning the intricate details of each J2EE technology. With the help of this book, you can then take that knowledge to the next level by putting that theory into practice when using WebLogic Server. Our aim is to guide you through the world of WebLogic and show how you can apply your J2EE expertise to build and manage applications on WebLogic. We reveal how WebLogic implements the various J2EE features, demonstrate how WebLogic enhances these services in interesting and useful ways, and explain how your applications can benefit from these features.
Administrators
Any group of users that is responsible for managing a WebLogic-based setup will find that this book has all the material needed to build a fundamental understanding of creating, managing, and maintaining WebLogic domains and services. This includes users who need to interact with the system in some nonprogrammatic way — for instance, the application assemblers, the deployers, and the tool providers. We explain how to manage the runtime WebLogic environment, discuss the performance implications and trade-offs, and examine the different design and security constraints.System architects
A system architect needs to have a good understanding of the overall capabilities of the application server if he is to design effective solutions. For instance, he must be aware of the different system architectures supported by WebLogic, the overall organization of a WebLogic domain and supporting network infrastructure, how WebLogic resources cooperate in a clustered environment, what additional features and services are offered, and how to extract optimal performance from the application setup. He needs to be able to extrapolate the possibilities, but also understand the limitations and trade-offs of adopting a WebLogic-based solution. Our book attempts to provide a 360-degree view of WebLogic. We not only highlight its features and strengths, but also point out any shortcomings and issues of which you need to be aware.We expect our target audience to have an understanding of the J2EE platform.Developers, at the very

least, should have some previous experience with programming servlets, creating JSP pages, using the JDBC API, and building EJBs. We encourage those who don't to read other books from the O'Reilly catalogue that cover the entire gamut of J2EE technologies. The various J2EE specifications published by Sun Microsystems also serve as a useful reference. This book aims to build on that J2EE know-how, and guides you through the different J2EE technologies and enterprise services supported by WebLogic.For system administrators, this book serves as a complete guide for managing a WebLogic environment. We expect that you will have had past experience with administering an enterprise application, perhaps on a different platform using other technologies.For Java architects, this book reveals how other WebLogic enterprise services can enrich your solutions with exciting possibilities. It also advances your understanding of how to achieve the best performance out of your application architecture. We expect you will have had some previous experience in building and designing multi-tier system architectures, and some level of awareness of the capabilities of the J2EE technology stack.OrganizationThe first chapter provides a quick tour of WebLogic Server. It offers an overview of the J2EE and other enterprise features supported by WebLogic. We explore the fundamental WebLogic resources such as domains, servers, and clusters. We also look at essential administration tasks such as starting and stopping the server. The remaining chapters in the book can be grouped into three categories: those that deal with J2EE, those that deal with WebLogic management, and finally, those that focus on WebLogic's own enterprise APIs.WebLogic and J2EEThe first part of the book examines WebLogic's rich support for the various J2EE services. WebLogic is a fully compliant J2EE application server, and it provides a mature environment for building robust, server-side, component-based applications. Chapter 2 and Chapter 3 give in-depth coverage of how to build web applications on WebLogic. We examine how to configure servlets and JSP pages on WebLogic Server. We look at how to incorporate custom JSP tags and filters into your web applications, and we explain how to package and deploy your web applications on WebLogic. We also learn about WebLogic-specific custom tags and filters, and how to create tag libraries from prebuilt EJBs. We look at how to configure the behavior of the servlet engine (web container) using the XML deployment descriptors for a web application. We discuss how WebLogic manages server-side HTTP sessions in a clustered environment, how to restrict access to specific web resources, and how to use commercial web servers to proxy requests to WebLogic Server. Moreover, we explore the many ways WebLogic lets you configure its HTTP server. Chapter 4 explains how resources are published over WebLogic's JNDI service. We examine how WebLogic's JNDI operates in a clustered environment. Later, we look at how to build RMI applications, and explore the various optimizations intrinsic to WebLogic's RMI. Finally, we learn how RMI objects can be accessible to both T3 and IIOP clients. Chapter 5 looks at WebLogic's support for JDBC connection pools and how data sources enable you to access these pools. Chapter 6 examines the use of distributed transactions in WebLogic. Chapter 7 looks at how to use JCA-compliant resource adapters to enable WebLogic applications to connect to proprietary enterprise stores. Chapter 8 provides an in-depth look at creating JMS applications and using WebLogic-specific features such as quotas, flow control, timed delivery options, XML-formatted messages, and bridging with other JMS providers. Chapter 9 succinctly covers how to configure JavaMail on WebLogic, thereby allowing deployed applications to send and receive electronic mail.The next two chapters investigate WebLogic's support for Enterprise JavaBeans© (EJBs). In Chapter 10, we learn about the various EJB types and how to package and deploy EJB components. We also describe the behavior of WebLogic's EJB container and the various ways you can influence the runtime behavior of your EJBs. We demonstrate how you can adjust the size of the free pool of EJBs and the in-memory EJB cache. We also look at the various optimizations for EJBs, such as network and transactional collocation, optimistic concurrency strategy, and read-only entity beans that can rely on a multicast invalidation framework. Most importantly, we explore how WebLogic incorporates load-balancing and failover support for EJBs deployed in a clustered environment. Chapter 11 explains how to create container-managed persistence (CMP) entity beans, while concurrently introducing the features of WebLogic's CMP engine. Later, we look at how to implement container-managed relationships (CMR) between entity beans. We also examine the EJB-Query Language (EJB QL)

syntax, and learn about the WebLogic-specific extensions to EJB QL.Managing the WebLogic EnvironmentThe middle portion of the book examines the post-development aspects of WebLogic applications. We look at how to package and deploy your applications, configure and optimize the runtime WebLogic environment, and deal with security issues. Chapter 12 explains how to package J2EE applications using available WebLogic tools. We also learn about WebLogic's classloader hierarchy and its impact on your deployment. Finally, we discuss the new two-phase deployment strategy in WebLogic Server, and the usefulness of application staging. Chapter 13 looks at how to manage the different resources and services that live in a WebLogic domain spread across multiple machines. It also covers monitoring the health of servers in the domain, configuring network resources, and planning for additional capacity. Chapter 14 provides an understanding of WebLogic's support for clustering, with a strong emphasis on its load-balancing and failover capabilities. It examines how various J2EE resources behave in a clustered environment. It also analyzes the performance and design implications of adopting different clustered solutions for your application's architecture. Chapter 15 explains the implications of tuning various performance-related configuration settings, and how to improve the performance of the JVM, the applications deployed to WebLogic, and the server itself. Chapter 16 provides all the details necessary to configure WebLogic's SSL support, and create your own programs that use WebLogic's SSL support. Chapter 17 explores the many services implemented under the hood of WebLogic's default security realm. It explains the behavior of WebLogic's security providers, its authentication framework, and declarative security for various J2EE components.WebLogic Enterprise APIsIn the last section of the book, we examine important enterprise WebLogic services that would attract many more developers and administrators. Chapter 18 looks at WebLogic's support for XML, including XML Registries, application-scoped XML parsers, event-driven parsing using the Streaming API, and other miscellaneous extensions. Chapter 19 describes how to create WebLogic web services over existing J2EE components. It explains how you can build JAX-RPC clients that interact with deployed web services, generate the necessary support for custom types, and set up a chain of handlers that can intercept SOAP request messages and SOAP response messages. It also describes how you can secure WebLogic web services and write clients that can invoke these protected web services. Finally, it explains how to publish and then inquire about web services advertised over the local UDDI registry. Chapter 20 provides an overview of WebLogic's JMX services and how you can use managed beans (MBeans) to programmatically administer and/or monitor WebLogic resources. Chapter 21 covers WebLogic's support for internationalization and logging. Chapter 22 looks at how administrators can integrate WebLogic into an SNMP-compliant management infrastructure. It provides an overview of the SNMP agent model and how you can implement an SNMP view of the WebLogic Server.Online DocumentationBEA's documentation on WebLogic Server is available at http://edocs.bea.com/, and can also be downloaded in HTML or PDF formats. BEA's dev2dev Online (http://dev2dev.bea.com) is another good source of tools, utilities, articles, and white papers. BEA also maintains a newsgroup site on the WebLogic Platform, nntp://newsgroups.bea.com/, which provides the ideal forum for discussions and support, and the opportunity for closer interaction between WebLogic developers, architects, administrators, and experts.All of these resources, as well as related books, have contributed to the material covered in this book. The most significant source for our book has been BEA's online documentation.BEA also produces helpful white papers. Tom Barnes's white paper, "JMS Performance Guide," deserves a special mention because it helped clarify a number of issues related to WebLogic's JMS implementation.Conventions Used in This BookWe use the following formatting conventions in this book: *Italic*
Used for filenames, pathnames, hostnames, domain names, URLs, email addresses, script names, folder names, and new terms where they are defined. Constant width
Used for code examples, program fragments, and console output. It is also used for Java keywords, names of Java classes, variables, and methods, SQL table and column names, XML elements and tags, and for shell commands and scripting variables. *Constant width italic*
Used to indicate text that is replaceable. For example, in *BeanName* PK, the *BeanName* may be replaced with a particular bean name. **Constand width bold**

Used for emphasis in some code examples.The term "WebLogic" is often used to refer to WebLogic Server, which is BEA's J2EE-compliant application server. This must not be confused with the "WebLogic Platform," which refers to a host of BEA technologies built around the core WebLogic Server product.TipThis icon signifies a tip, suggestion, or general note.WarningThis icon indicates a warning or caution.Using Code ExamplesThis book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*WebLogic: The Definitive Guide*, by Jon Mountjoy and Avinash Chugh. Copyright 2004 O'Reilly & Associates, Inc., 0-596-00432-X."If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.Comments and QuestionsPlease address comments and questions concerning this book to the publisher:
O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)There's a web page for this book that lists errata, examples, and any additional information. You can access this page at:
http://www.oreilly.com/catalog/weblogictdgTo comment or ask technical questions about this book, send email to:
bookquestions@oreilly.com For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:
http://www.oreilly.com AcknowledgmentsIt is our pleasure to acknowledge the assistance of many friends and colleagues who encouraged us to complete this book and provided constructive criticism of the manuscript. Our thanks to our editor, Brett McLaughlin, and to Deepak Vohra for reviewing drafts of our chapters. Melissa Chaika, formerly of BEA, was kind enough to keep us updated on current and future changes to WebLogic Server. We'd also like to thank the production staff at O'Reilly. Mary Brady and her team did a super job of transforming an ugly manuscript into a good-looking book.Many people have dispensed with their free time and energy to respond to our queries on WebLogic's newsgroups over the years. These include Joe Weinstein, Tom Barnes, Bruce Stephens, Slava Imeshev, Cameron Purdy, Rob Woolen, and Andy Piper. We'd also like to thank those who took the time to review drafts of this manuscript because this book owes a lot to their thoughts and ideas. We are particularly grateful to many folk at BEA who took time out of their busy schedules to help us: Joe Weinstein, Tom Barnes, Sanjeev Chopra, Cedric Beust, Peter Bower, and Tony Vlatas. Cameron was also gracious enough to review a chapter at short notice.On a more personal note, Jon would like to thank his wife for the constant support and nourishment through trying conditions, and the Cowan family of Eastside Farm for their company, kindness, and excellent Lagavulin.Avinash would like to thank his family and friends for their support and understanding, not just throughout the time he has been involved with this book, but also in helping him to get through his father's death. He would have liked his dad to just hold the final manuscript, even if the contents inside wouldn't have made much sense to him.
Chapter 1. IntroductionWebLogic Server is one of the leading J2EE application servers on the market today. It is a robust, mature, scalable implementation of the J2EE specification. It also lies at the heart of the WebLogic Platform, a unified, extensible platform for developing and deploying enterprise solutions such as enterprise information portals and systems integration solutions. This chapter provides an overview of WebLogic Server, and offers a perspective of its place within the WebLogic Platform. You'll also get a flavor of the various J2EE and enterprise features of WebLogic

that you'll encounter in this book.As you shall see, WebLogic offers much more than a full-fledged implementation of the J2EE standards. This chapter introduces some of the extensions to the J2EE technology stack, such as WebLogic's management framework and its robust support for web services. In addition, this chapter prepares the groundwork for the rest of the book, and helps you get started with using WebLogic Server.Overview of WebLogic Server WebLogic is a Java Application Server, a server-side Java program that provides a number of enterprise services for the benefit of the various applications and components running on the server. These services include the HTTP service, session handling, distributed naming and lookup, database access, persistence, transaction management, caching, concurrency, messaging, security, and much more. Server-side applications can use these services to implement their application logic, while external clients can either use the published services or directly interact with the applications. Once installed, WebLogic provides various command-line scripts for starting up the server. In fact, many of the WebLogic tools are Java programs that run within a console or as a GUI-based Java application. For this reason, stable releases of WebLogic Server are available on a wide range of platforms, including Windows 2000 Server/Professional, Windows XP, Solaris OS, Red Hat Linux, Tru64, HP-UX, IBM AIX, and other Unix variants.   WebLogic is designed to operate in a distributed environment. This means that you can easily set up an environment where multiple WebLogic instances are running on separate machines within the network, each configured with its own set of applications and services. Alternatively, you may need to design a "cluster" of WebLogic instances distributed across multiple machines over the network, each configured with the same set of applications and services. If you're blessed with powerful, multi-CPU machines, you even may opt to host multiple servers on a single machine. Moreover, WebLogic provides tools that enable you to set up these different scenarios and effectively manage the applications and other resources spread across all the servers. The ease and flexibility with which you can adapt your WebLogic configuration to suit your performance and scalability needs make WebLogic Server a very attractive platform for building real-world enterprise applications.   WebLogic provides a rich set of client options — it can interact with web clients that use HTTP, Java clients that use RMI-IIOP, JAX-RPC clients for web services over SOAP, and mobile devices that use WAP. It also provides comprehensive support for enterprise Java technologies. In fact, WebLogic is a fully certified J2EE application server. It means that you don't need to live under any illusions about the capabilities of WebLogic Server. At the very least, WebLogic will continue to support the full J2EE technology stack and adhere to all of the constraints that it imposes on interoperability. But WebLogic offers more and implements enterprise services that go beyond the J2EE standard, anticipating future trends in the enterprise Java application server market.   Effectively managing the applications and services deployed across the various servers on separate machines can prove to be a formidable challenge. WebLogic rises to the occasion by providing a rich, robust management infrastructure for administering the applications, resources, and other services running on separate WebLogic instances. As we shall see, WebLogic relies on the notion of a "domain" to sketch out a region of servers, WebLogic services, network and application resources, and security policies. All resources that lie within the scope of the domain are under the central control of the Administration Server. All domain administration tasks are routed via the Administration Server, which is responsible for the entire domain configuration. The domain can rely on Node Managers — one on each machine that is responsible for the health of all servers running on it. In addition, the domain may include a number of clusters, where each cluster is composed of multiple servers, possibly running on different physical machines.   Security is another important consideration when planning the system architecture for an enterprise application. WebLogic is capable of handling security issues at various levels of your setup — at the JVM level through JVM-specific policies, at the connection level through filtering and the Secure Sockets Layer (SSL), at the application level through application-specific policy constraints, and at the server level through security providers that let you transparently change various aspects of WebLogic's default security implementation. Restricted access to particular services (read ports), firewalls and demilitarized zones (DMZs), and database access privileges further reinforces the security within your application setup and makes it less vulnerable to malicious attacks and unauthorized access. The WebLogic Platform Before we take a closer look at WebLogic's features, let's examine how WebLogic fits in the larger scheme of things. As we mentioned earlier, WebLogic Server forms the core of the WebLogic Platform. Other WebLogic technologies, including the Portal, Integration, and Workshop, are built on top of WebLogic Server. Figure 1-1 depicts the conceptual layout of the technologies that make up the WebLogic Platform.

Figure 1-1. The WebLogic Platform Clearly, WebLogic provides a range of enterprise technologies that cater to different business problems and solution paradigms:WebLogic JRockit

JRockit is BEA's high-performance JVM optimized for server-side performance and scalability.

WebLogic Workshop

Workshop is a visual development environment that allows you to build enterprise-class applications on WebLogic Server.WebLogic Portal

WebLogic Portal is an enterprise portal platform that allows you to develop and manage custom portals that support various business processes such as commerce, content management, and interaction management.WebLogic Integration

WebLogic Integration provides a unified platform for developing, deploying, and integrating enterprise applications and business processes in a collaborative framework. It provides business process modeling tools, data transformation tools, a message broker, and integration adapters for a wide range of applications and formats.This book focuses on WebLogic Server, which serves as the foundation for higher-level WebLogic technologies. A good understanding of WebLogic Server will help you appreciate how other WebLogic technologies operate. The next two sections survey the extent of WebLogic's support for the J2EE standard, as well as how WebLogic augments these features in interesting and useful ways. We also examine the enterprise features WebLogic offers on top of the existing set of J2EE features — the features that distinguish WebLogic from the rest of the crowd.WebLogic as a J2EE Product The Java 2 Platform, Enterprise Edition (or the J2EE Platform) defines the specifications for an open environment that enables you to build scalable, robust, and secure enterprise Java applications. Any J2EE-compliant application server must provide a number of enterprise services that can then be made available to the applications and components hosted on the server. These services include transaction management, persistence, security, naming and directory services, and more. In addition, the specification outlines the behavior of various components that reside within an application, such as servlets and EJBs. It lays down the typical life cycle of these server-side components, a standard mechanism for packaging these components, and guidelines for maintaining portability across other compliant application servers. Central to the J2EE specification is the notion of the container-component model. Server-side components run within the context of a container, an abstract entity within the application server that manages the lifetime of the component and provides access to a host of available services. Like a cocoon, a container nurtures and protects the various components deployed to the server. Servlets and JSP pages typically run within a web container, while the EJBs run within an EJB container. Versions 8.1 and 7.0 of WebLogic Server embrace the J2EE 1.3 specification. Because the J2EE 1.3 specification is backward-compatible with J2EE 1.2, you can run J2EE 1.2-compliant applications on WebLogic Server 8.1 and 7.0 as well. Remember, the J2EE standard defines only the *minimum* set of enterprise services that any compliant application server must implement. WebLogic also offers a whole lot more. We'll look at these additional exciting enterprise features of WebLogic Server in the following section. Let's first inspect the standard J2EE features that are part of WebLogic Server.*Servlets, JSPs, and the web container* WebLogic Server is equipped with a full-featured servlet engine that complies with the Servlet 2.3 and JSP 1.2 specifications. This means that you can build web applications that contain HTTP servlets and JSP pages for generating dynamic responses to web requests. You also may configure multiple filters that process a web request before it has been handled or after the web response has been generated. You can incorporate custom JSP tags into your JSP pages, where each tag encapsulates a well-defined, reusable piece of functionality. Clearly, these components have access to other J2EE services as well, such as JNDI lookups, JDBC API calls, distributed transactions, remote EJBs, and more. Of course, you can deploy multiple web applications to WebLogic Server, and utilize third-party tag libraries in your existing web applications. WebLogic offers several enhancements to the web container. During development mode, WebLogic supports automatic reloading of servlets and JSPs. It provides different ways of caching the response to web requests, and how and when the cached responses ought to be refreshed. You may choose to configure WebLogic's cache filter for specific web requests, or rely on WebLogic's custom JSP cache tags to cache specific portions of the JSP page. WebLogic also provides advanced session-handling features with support for failover, whereby a user's session data can be replicated to another WebLogic instance or persisted to a file or a database. This mechanism guarantees high availability for web applications deployed to a cluster. WebLogic also provides an "EJB-to-JSP" tool that lets you automatically generate a set of JSP tags from EJBs that you've already developed.   So, WebLogic has an established framework for handling dynamic HTTP requests. But WebLogic is also a robust, mature HTTP server, which means that you can use WebLogic to serve requests for the static content in your web applications, such as HTML pages, images, text files, and more. You even can configure virtual hosts, HTTP tunneling, and SSL for WebLogic's HTTP server. You also can set up the HTTP server to act as a proxy server so that it proxies dynamic requests for servlets and JSPs to another WebLogic instance. The same behavior can be achieved using commercial web servers. WebLogic provides custom proxy plug-ins for popular web servers such as Apache, Microsoft's Internet Information Service (IIS), and Netscape's Enterprise Server (NES). These plug-ins ensure that only specific requests for dynamic content are forwarded to a WebLogic instance. In this way, you can continue to use your high-performance web servers, while your backend WebLogic instance(s) are responsible for handling all the dynamic web requests. *JDBC and distributed transactions* Most Java applications use the JDBC API to interact with backend relational databases. Because WebLogic supports the JDBC 2.0 specification, you can configure a ready pool of database connections on WebLogic using any JDBC 2.0-compliant driver. The connection pool provides access to the backend DBMS — you would be hard-pressed to find a DBMS for which a JDBC driver isn't shipped. Server-side applications and external clients then can transparently access the connection pool using configured data sources. WebLogic also supports distributed transaction-aware (XA-aware) data sources, in which case connections obtained from the data source can participate in distributed transactions. If the data source is attached to a pool of non XA-aware connections, WebLogic can still emulate the two-phase commit protocol, and ensure that the data source can participate in distributed transactions, albeit with some limitations.   Besides making JDBC connection pools accessible to both server-side applications and external clients, WebLogic allows you to set up a multipool where a number of connection pools can collaborate to achieve load balancing or high availability for JDBC connection requests, even in a single-server environment. You can then transparently access the multipool through each connection pool's configured data sources. Furthermore, WebLogic can optimize JDBC access by supplying a statement cache with every connection pool. WebLogic can then cache a number of prepared statements (or callable statements) used by clients of the connection pool. Statement caching can offer a significant performance boost if the most-often used statements fill the cache. You also can configure application-scoped JDBC resources. As these are packaged with the application, they make it easy to duplicate the configuration on different application server instances.   In addition, WebLogic's multi-tier drivers provide clients with direct access to server-side connection pools, automatic enrollment in distributed transactions, and clustered access to connection pools through their data sources. Multi-tier drivers enable connection requests to a data source that is targeted to a cluster of WebLogic instances to fail over to another connection pool if one of the servers fails.   The Java Transaction API (JTA) provides the higher-level client access to distributed transactions. Clients need to explicitly initiate distributed transactions when multiple EJBs are needed to fulfill a client request or when the default container-managed transactions are inadequate to implement the business logic. WebLogic extends the capabilities of the JTA by enabling clients to access the current transaction associated with the thread or to obtain a reference to the transaction manager. Moreover, WebLogic allows you to monitor, log, and recover transactions through its transaction recovery service.*JNDI and RMI*   Java Naming and Directory Interface (JNDI) provides the standard way for interacting with naming and directory services. J2EE applications use the JNDI API to access various resources that have been configured for the server, including data sources, user transactions, Remote Method Invocation (RMI) objects, EJB home objects, mail sessions, JMS connection factories, JMS topics/queues, and much more. WebLogic's JNDI framework extends over an entire cluster, across all its available members. The cluster-wide JNDI tree manages the JNDI bindings for all clusterable objects. These bindings are automatically replicated across all WebLogic instances in the cluster. Each server maintains a local copy of the global JNDI tree holding replicas of all JNDI bindings in the global JNDI tree. In addition, the local JNDI tree holds bindings for any server-specific resources that must be made available only to the particular WebLogic instance.The RMI specification initiated the wave of distributed Java computing. It established an intuitive framework for enabling Java method invocations across JVMs. WebLogic's RMI is a highly optimized implementation, incorporating high-performance serialization logic and avoiding the overhead of marshalling and unmarshalling when both the client and the server-side RMI objects run within the same VM. WebLogic's RMI usually relies on its native transport protocol, called the *T3 protocol*. However, WebLogic also supports RMI clients that need to talk over *Internet Inter-ORB Protocol* (IIOP), a protocol that allows interoperability with CORBA objects.   Moreover, WebLogic's RMI Registry is a thin veneer over its JNDI service. In fact, RMI objects can be published directly over WebLogic's JNDI tree. WebLogic's JNDI service and clusterable RMI stubs collaborate to provide failover and load-balancing facilities to clients. WebLogic lets you compile and bind RMI objects into this clustered environment. Any client that looks up the RMI object from the JNDI tree obtains a *cluster-aware stub* — a stub that is aware of all its replicas on all available servers in the cluster. Thus, WebLogic's implementation of JNDI and RMI provides a transparent and elegant way for supporting failover and load balancing of RMI objects in a clustered environment.   *EJBs* EJBs are reusable server-side Java components that usually encapsulate the business logic of an enterprise application. EJBs typically interact with backend databases and almost always participate in distributed transactions. WebLogic fully supports all the different EJB component types defined in the EJB 2.0 specification: session beans, message-driven beans, and entity beans.J2EE applications can use these EJB types to implement their business logic. Even though WebLogic Server supports EJBs that are compliant with the 1.1 standard, most developers find it inadequate for their needs, and eventually need to deal with business requirements where it is just as convenient to upgrade to the EJB 2.0 standard. Typically, your EJB components form the "object tier" of an enterprise application that services requests from other servlets and JSP pages, or even external clients. You can even choose to deploy only your EJB components separately to a cluster of WebLogic instances. This division of labor between WebLogic instances — one set of servers that handles the web/presentation aspects of the enterprise application, and another set of servers that manages the EJB components — lets you benefit from WebLogic's load-balancing and failover features. The flexibility of options that are available to you when designing a WebLogic-based architecture also adds to its appeal. WebLogic's EJB container provides many additional enhancements. WebLogic supports a free pool of EJB instances for stateless session beans, message-driven beans, and entity beans. It also provides an in-memory cache of EJB instances for stateful session beans and entity beans. The WebLogic-specific deployment descriptors allow you to adjust the size of the free pool and the EJB cache. A properly configured free pool and EJB cache can significantly improve the performance of your EJBs. Most importantly, EJBs can operate easily in a clustered environment. Client requests to the EJB home object or method calls on an EJB object can be load-balanced across all available servers in a cluster. Failover on method calls also can be achieved under certain conditions. In addition, WebLogic supports failover for stateful session EJBs that have been deployed to a cluster. Just like HTTP sessions, WebLogic achieves this behavior by replicating the state of the session EJB onto another server in the cluster. WebLogic dedicates a whole array of optimizations to entity beans. For instance, WebLogic's EJB container supports value-change checks to ensure that only those persistent fields that have been modified are written to the database. This can improve the performance of your EJBs if their persistent fields map to long-valued columns (large chars, varchars, blobs, clobs, etc.). It also supports read-only entity beans where the in-memory state of the EJB can be refreshed at regular intervals. In this way, WebLogic's EJB container can avoid persisting the EJB's state to the underlying database. In addition, WebLogic incorporates a multicast invalidation framework, whereby an EJB update can invalidate all cached EJB instances on other servers. This mechanism avoids unnecessary loading of EJB data. You even can configure WebLogic's EJB container so that all updates made during a transaction are deferred until transaction commit. WebLogic's CMP engine is equipped with a rich set of features. Its RDBMS persistence framework allows you to map an entity bean's container-managed fields to columns spread across multiple tables. You can specify precisely when the in-memory state of the entity EJB is written to the database. In addition, WebLogic supports an "optimistic concurrency" strategy for entity EJBs, which ensures that no locks are held by WebLogic or the database. Instead, the EJB container performs a smart update by checking that the data hasn't been modified since the start of the transaction. In this way, optimistic concurrency allows you to dramatically improve the concurrency of your entity EJBs. Other features, such as batching and fetch groups, provide you with extreme control over when and how the database is accessed. WebLogic also implements a superset of the standard EJB QL. It allows you to define EJB queries that provide ordering and aggregation, and that remove duplicates from matching EJB instances. These extensions are identical to their SQL counterparts: ORDER BY, GROUP BY, and SELECT DISTINCT.*JMS*   Java Messaging Service (JMS) equips you with a powerful programming model based on asynchronous, decoupled communication. It allows you to build J2EE applications on top of existing message-oriented middleware. WebLogic's JMS implementation is compliant with the 1.02b release of the JMS standard, and is chock-full of practical enhancements. For instance, WebLogic enables producers and consumers to exchange XML-formatted messages. The message filters can be specified using XPath expressions that can then filter out any message whose body doesn't match the given XPath query.For better scalability, WebLogic provides fine control of message delivery and handling. For instance, you can delay the delivery of messages or configure specific time periods during which the messages are delivered. WebLogic's support for message paging gives you additional control over resource usage by moving messages out of main memory under predefined conditions. It also provides flow control mechanisms that let you throttle the production of messages during peak message load conditions. All these features ensure WebLogic's JMS server cannot degrade the performance of other WebLogic services.   Depending on your message delivery needs, WebLogic lets you choose from a range of acknowledgment modes, from guaranteed to unacknowledged message delivery. If no acknowledgment is required, WebLogic relies on multicast broadcasts to deliver your messages. WebLogic's JMS service is integrated with its clustering services. It lets you transparently masquerade multiple JMS destinations that reside on separate servers behind a virtual destination name. This feature yields seamless failover and load balancing when delivering messages to the "distributed" JMS destination. The messaging bridge tops off all these features by allowing you to integrate WebLogic JMS with other JMS providers.*JCA*   Java Connector Architecture (JCA) provides a standard framework for connecting J2EE application servers to enterprise information systems. Server-side applications and external clients can use the configured resource adapter to interact with the enterprise information system (EIS). Just like JDBC drivers, WebLogic lets you set up a pool of connections to the underlying EIS. These connections may participate in distributed transactions, along with other transaction-aware resources. WebLogic also can reliably detect connection leaks by relying on the garbage collector and an idle timer setting. This ensures that the pool doesn't run out of available connections, in case clients of the resource adapter aren't properly releasing their connections. You'll also find that WebLogic's security infrastructure extends to resource adapters. Additional Enterprise Features of WebLogic We've just looked at some of the standard features that are compulsory for any J2EE-compliant application server, and how WebLogic's implementation enhances these features. Beyond this, WebLogic provides a slew of additional functionality that enhances your experience of developing and managing applications in a WebLogic environment. You'll find that these enterprise features of WebLogic Server enable you to build more secure, manageable, and scalable solutions.*WebLogic's security framework*   WebLogic provides a comprehensive suite of security features with which you can protect your domain resources. For instance, WebLogic augments the security of your network by letting you set up connection filters that can deny access to connections according to custom criteria. In addition, WebLogic supports the use of SSL, which can be configured to use mutual authentication. Plus, you can programmatically rely on SSL to communicate with other systems. At the server level, WebLogic lets you set up fine-grained control over access to resources such as JDBC connection pools, EJBs, web services, and specific branches of the JNDI hierarchy.   WebLogic's security infrastructure is based on a modular, extensible set of security service provider interfaces (SSPIs). A domain can use either the default implementation of the security providers, or custom security providers that extend the default providers according to your needs. For instance, WebLogic's authentication framework allows you to store all authentication information to a Lightweight Directory Access Protocol (LDAP) or to a JDBC-based repository. In fact, WebLogic comes with its own embedded LDAP server. The traditional J2EE role-based security also relies on a role-mapping provider. WebLogic lets you transparently plug in a custom role-mapping provider and thereby benefit from an alternative implementation.*WebLogic in a clustered environment*You can easily set up a cluster of WebLogic instances, all working together to provide support for load balancing and failover. Your applications and related resources can be deployed homogenously across all the servers in the cluster. External clients can interact with any member of the cluster, and fall back on another server in case of a failure. WebLogic's support for clustering provides an intuitive way for enhancing the scalability of your architecture. New servers can transparently join the cluster and grab their share of the overall load on the cluster. Your applications continue to be highly available because other servers can take over in case a peer server fails. Clusters are especially relevant when designing a multi-tier architecture for an enterprise application. Client requests for a web application can be distributed across a bank of WebLogic instances. EJB method calls from a servlet or JSP page can again be distributed across a cluster of WebLogic instances that host all the EJB components. This division of labor is vital if you have differing performance constraints for your web applications and your EJBs, and therefore need to ensure your application tiers can scale independently.Clearly, there's little point in considering these architectures if WebLogic services cannot adapt to a clustered environment. Fortunately, most J2EE services can benefit from WebLogic clusters. This includes load-balancing and failover support for HTTP requests, method calls to EJB home objects and EJB objects, JDBC connection requests, JMS distributed destinations, and much more. Stateful services also can operate in a clustered environment. For instance, WebLogic supports in-memory replication of HTTP sessions and stateful session EJBs. As discussed earlier, the interplay between the cluster-wide JNDI tree and clusterable RMI stubs is crucial to the support for load balancing and failover. For singleton services such as JMS, WebLogic allows you to migrate the service to another peer server in the cluster, thereby ensuring high availability.   *XML support*   XML has become all-important in the J2EE world, and WebLogic provides extensive support for these technologies. First, WebLogic is bundled with Apache's Xerces XML parsers and Xalan XSLT implementations. WebLogic also includes an optimized nonvalidating SAX parser, ideal for small to medium-size documents such as those that occur in the world of web services. All of these parsers can be used through the standard JAXP interface, which provides better control over the XML parser that is used under a given situation. WebLogic also lets you configure particular XML parsers on a per-server and per-application basis. On top of all this, WebLogic includes an XML Streaming API, which allows you to parse XML documents using a pull-based approach. Under the right conditions, pull-based parsing can be more optimal than event-driven SAX parsing or the DOM-based approach. WebLogic 8.1 also provides an XML XPath API, which can integrate with the Streaming API. *Web services* A *web servic*e is a distributed, interoperable, web-based interface that transparently wraps some well-defined piece of functionality. Clients can locate the desired web service from a registry and then invoke the web service. The actual implementation behind the web service can manifest itself in various ways. For instance, WebLogic makes it very easy for you to create web services that wrap ordinary Java objects, stateless session EJBs, and JMS destinations.   Support for web services is going to be an essential requirement in the next release of the J2EE specification. WebLogic has preemptively implemented a rich set of facilities that enable you to easily create and deploy industry-strength web services. Its GUI-based tool, WebLogic Workshop, is dedicated to building and deploying web services on WebLogic Server. WebLogic relies on the standard SOAP 1.1 protocol with attachments message format for delivering data during web service invocations. Descriptions of web services deployed to WebLogic Server adhere to the WSDL 1.1 standard. Moreover, WebLogic lets you automatically generate these descriptions, along with HTML pages that allow you to test your web services. You can publish the web services on WebLogic through the UDDI service that runs within WebLogic. Clients can locate desired web services by questioning the UDDI 2.0 registry.   WebLogic 8.1 also supports a reliable SOAP message transport, which allows clients to use guaranteed SOAP message delivery. In addition to this, you can invoke web service operations asynchronously. With this style of invocation, you can either periodically poll for a result, or create a listener that will receive a notification when the operation's result comes in. WebLogic 8.1 can secure SOAP message exchanges through token-based identification, digital signatures, and data encryption. Finally, WebLogic supports the JAX-RPC standard for invoking web services running on either WebLogic Server or any other compatible platform. The combination of these features makes WebLogic's web services implementation quite formidable.*Management features*   The Java Management Extensions (JMX) specification defines the architecture, services, and Java APIs for the distributed management of resources. WebLogic's built-in MBean server provides access to a host of JMX MBeans that enable you to programmatically fiddle around with the static and runtime configuration settings of various resources of a WebLogic domain. You can use these MBeans to build your own management and configuration tools, or simply use them to monitor the performance of the different WebLogic services. The Simple Network Management Protocol (SNMP) is used to monitor many different types of managed resources, from hardware to software products. JMX services also serve as inputs for WebLogic's SNMP agent. WebLogic's SNMP agent allows you to seamlessly integrate a WebLogic domain within a larger network management framework of your enterprise. In this way, your WebLogic domain can be treated just like any other network resource. Both JMX and SNMP give you exceptional control over the overall management of a WebLogic domain.Both of WebLogic's JMX and SNMP systems can interact with the distributed logging infrastructure. This allows you to centrally manage all logging activity and create advanced components that listen and react to error situations. WebLogic Server Tools As much as WebLogic's enterprise features may entrance you, you still need the right set of tools to assemble and deploy your applications, manage your WebLogic environment, and monitor the runtime state of all deployed services, applications, and resources. Fortunately, WebLogic takes care of all these needs, and more.*System administration console* The Administration Console is a comprehensive, browser-based GUI application for configuring, managing, and monitoring all of the resources in a WebLogic domain. Some of its capabilities include managing the domain configuration, starting and stopping servers, monitoring the performance of the server and application components, viewing server logs, initiating deployment tasks, and editing deployment descriptors.*Deployment and development tools*   With Java applications, deployment has never been easy. In a clustered environment, it can be even trickier. The Administration Console lets you deploy, redeploy, and undeploy J2EE applications and modules to specific server instances and to all members of a cluster. WebLogic's Deployer utility, weblogic.Deployer, is a command-line tool for managing the deployment of J2EE applications. EJBGen   is another useful tool that can generate the EJB deployment descriptors, and the home and remote interfaces from the source of the EJB's bean class annotated with predefined JavaDoc tags. WebLogic's Marathon utility (DDInit) can examine the contents of a staging folder and generate the standard J2EE and WebLogic-specific deployment descriptors for both web applications and EJB modules. WebLogic also supplies a Client Deployer tool that can extract the client-side JAR from an EAR file to create a deployable JAR file.   WebLogic also provides a number of utility Ant tasks that ease the build and deployment process. For instance, the appc task lets you generate the container classes for your EJBs, as well as generate an EJB JAR that can be deployed to WebLogic Server. It can also be used to precompile your JSP pages. The wsdl2service task takes a WSDL file as input and generates the *web-services.xml* deployment descriptor, and a raw Java source file that can be used as a starting point for implementing the web service. You'll learn about other Ant tasks provided by WebLogic as you read the rest of the book! *WebLogic Builder* Assembling a J2EE application, creating and editing its deployment descriptors, and later deploying it to WebLogic Server is not a trivial task. WebLogic Builder fulfills these requirements. It is a graphical tool used for assembling a J2EE application, creating and editing its deployment descriptors, and deploying it to a WebLogic instance.WebLogic Workshop WebLogic Workshop is a tool that deserves special mention and a book of its own. WebLogic Server comes with an IDE, WebLogic Workshop, which can be used to compose web services and other components using a very high level of abstraction.The IDE lets you create and compose enterprise applications in a graphical way — simply drag the components that you want to use onto the same screen and wire them together. All and all, it provides a very quick route to application creation. Here are some of the features of WebLogic Workshop:It provides a visual and textual coding environment.The code generated by Workshop is annotated with JavaDoc tags indicating behavior. For example, Workshop can be used to create EJBs. These EJBs are annotated using the tags supported by EJBGen, which is described in Chapter 10.Workshop provides a framework that abstracts the complexities of the J2EE. Instead of writing J2EE infrastructure code, most of the Java you need to write is simply business logic and a little wiring.Workshop offers a component model in the form of Java controls. Controls can be created for almost all J2EE concepts, such as web services, EJBs, JMS destinations, web applications, and databases. You can write your own controls and embed controls within others.Because of the Java Control model adopted by Workshop, you can reuse controls from previous projects, download controls from BEA, and even buy controls to supplement the controls distributed with Workshop.Workshop integrates tightly with WebLogic Server during development. Deployment, testing, and debugging are made very easy.Workshop also can be used to easily construct web applications based on Apache's Struts framework. Visual designers are provided, as well as data-binding tag libraries. Navigation in the web application is controlled by a combination of page flows and controller logic.It is beyond the scope of this book to discuss how to use WebLogic Workshop in depth. It is a powerful, comprehensive tool that provides a different route to creating applications than that described in the rest of this book. However, Chapter 19 provides a detailed, low-level look at the web services framework supported by WebLogic. Creating web services using this framework requires coding, descriptor files, and Ant build files. On the other hand, you can create web services in Workshop with a few drags of the mouse. Both techniques have their place. Software and Versions This book examines features and services of WebLogic Server. We've used WebLogic Server 8.1 SP 1, and WebLogic 7.0 SP 2. We recommend that you apply the latest service packs after installing WebLogic Server to ensure all known issues have been resolved. Most of the changes between WebLogic 7.0 and WebLogic 8.1 occur in the form of new features or in slight changes to the Administration Console. We have tried hard not to litter the book with sections relevant to only one particular version, or with distracting warnings about which features are in which version. Rather, we have tried to be discrete. So, for example, when we say at the beginning of a section that "WebLogic 8.1 introduces a new feature that ...," you can be sure that the feature is in WebLogic 8.1 only, and not in 7.0. Customers can choose the particular edition of WebLogic Server that best suits their needs. If you intend to develop pure web applications that consist of static web resources and incorporate dynamic content through servlets and JSPs, you can perhaps opt for the WebLogic Express edition. If you need to build full enterprise systems using EJBs, JMS, and distributed transactions, you will need WebLogic Server instead. Each comes in two different flavors: versions that support clustering and versions that don't. Your WebLogic distribution is shipped with a stable release of the J2SE SDK. WebLogic 8.1 ships with a 1.4 JDK, and WebLogic 7.0 with a 1.3 JDK. All shell scripts and batch command files within WebLogic reference the JRE contained within this JDK. WebLogic Server is fully compliant with the J2EE 1.3 standard. This means that you can incorporate any other third-party custom components, such as JSP tags that conform to the JSP 1.2 specification or entity beans that conform to the EJB 2.0 standard. Moreover, WebLogic applications can interact with any LDAP server for which the vendor supplies a JNDI 1.1-compliant service provider. In addition, you can deploy any third-party resource adapter that conforms to the JCA 1.1 standard. Typically, WebLogic interacts with a backend DBMS through a configured pool of JDBC connections. We've used Microsoft's SQL Server 2000 Service Pack 3 and PostgreSQL to run many of the code samples illustrated here. In fact, you may use any industry-strength RDBMS for which a JDBC 2.0-compliant driver is available, including Oracle 8.0 or higher, IBM's DB2 or Informix, and many others.Finally, even though WebLogic comes equipped with JAXP-compliant XML parsers, you may choose to utilize other implementations.Getting Started with WebLogic Server As we've already seen, all WebLogic instances and their resources exist within the context of an organizational unit called a WebLogic domain. Within the domain, you can set up multiple servers, customize their network configurations, and deploy multiple J2EE applications and any required resources. You may even set up WebLogic clusters using a subset of the servers in the domain. All servers in the cluster can then work together to provide support for failover and load balancing.One server in the domain is vital to its existence: the Administration Server that manages the configuration for the entire domain. This includes the configuration of all servers within the domain, and all applications and services deployed to these servers. Other servers in the domain, called Managed Servers, host the actual J2EE applications and resources such as JDBC connection pools, data sources, JMS connection factories, topics and queues, RMI objects, and so on. The Administration Server merely manages the entire configuration of these servers. In a single-server domain, you have little choice but to deploy your applications to the Administration Server itself. However, in a production environment, we recommend you deploy your applications to a separate server.WebLogic is fairly flexible about how you configure the servers in the domain. So long as all Managed Servers in the domain can reach the Administration Server over the same network, WebLogic doesn't care whether the servers in the domain run on the same machine or on their own hardware.Let's now quickly look at how to create a single-server WebLogic domain so that you can start the Administration Server. Once you're able to access the domain's configuration using the Administration Console, you know that you have a working WebLogic domain. This should be a good starting point for you to start building J2EE applications for WebLogic, and trying out any examples that you encounter as you read this book. Chapter 13 shows you how to expand on this configuration to build more complex domains.Creating a Domain In order to set up a running WebLogic instance that you can work with, you need to first create a WebLogic domain. When you've successfully done so, you can deploy your J2EE applications to the Administration Server itself. Later on, as you become more comfortable with WebLogic Server, you can add more servers to the domain and move your applications to the Managed Servers. For WebLogic 8.1, start up the Domain Configuration Wizard by selecting the program from your operating system's Start menu. Select the Basic WebLogic Server Domain option from the list, and choose the express setup route. You then need to supply the username and password for the system administrator. Remember these details, as you won't be able to start or access the Administration Console successfully without them. After selecting from the installed SDKs to use, provide a name for the domain and a location on the filesystem where you want to store the domain. Finally, choose Create. The domain set up in this manner will automatically bind to all available IP addresses and listen on port 7001.For WebLogic 7.0, start up the Domain Configuration Wizard by running the *dmwiz* script located under the *WL-HOME/common/bin* folder, or by selecting the program from your operating system's Start menu. Select the WLS Domain option from the list, supply a name for the new domain, and then choose the Single Server (Standalone Server) option. Then you need to supply the full path of the directory under which all domain-related files are stored. After this, you need to configure the standalone server. For this, you need to specify the server's name, IP address or hostname, and the listen port on which the server is available. If you do not specify an address for the standalone Administration Server, the server automatically binds to all available IP addresses. Finally, you need to supply the username and password for the system administrator. Ensure that the single server doesn't run as an OS service.In order to run the standalone server, you need to navigate to the domain's root directory, locate the *startweblogic* shell script for your OS, and run it. The server won't start unless you also supply the username and password of the administrator configured earlier during the creation of the domain. Ensure that you've installed the proper WebLogic licenses or else the server aborts its boot sequence. You'll know that the server has successfully completed its startup sequence when you see the following in the console log window:<20-Apr-03 23:26:57 BST> <Notice> <WebLogicServer> <000365> <Server state changed to RUNNING><20-Apr-03 23:26:57 BST> <Notice> <WebLogicServer> <000360> <Server started in RUNNING mode>Well done, you've

successfully configured a new WebLogic domain and started the standalone Administration Server. TipThe Domain Configuration Wizard lets you install domains already populated with applications. The "examples" domain is very useful, as it contains code and examples for most aspects of WebLogic.Under the domain's root directory, you'll find the *config.xml* file that captures the entire configuration of the domain. Any changes made to the domain's configuration are persisted to the *config.xml* file. The domain log file *domain-name.log* also lives under the domain's root folder. Log files for the Administration Server are placed under the *domain-name/server-name* folder.Using the Administration Console Once you've successfully started the domain's Administration Server, you can use the Administration Console to configure, manage, and monitor the domain and all its resources. Now if you've bound the Administration Server to the IP address 10.0.10.10, for instance, and set its listen port to 7001, you can invoke the Administration Console through your web browser by navigating to *http://10.0.10.10:7001/console/*. Here again, you need to supply the administrator's login credentials (username/password) in order to view the domain's configuration.Once the user's credentials have been successfully authenticated, you can access all areas of the Administration Console. On the left pane, you'll notice a graphical tree menu that lets you navigate to the different resources located in a WebLogic domain. Ensure that you have installed a Java plug-in for the browser, or else the menu applet will not render successfully. Figure 1-2 illustrates the navigation menu for the Administration Console. We recommend that you familiarize yourself with these menu options. The rest of the book frequently will refer to the left pane of the Administration Console because it provides the primary means for manipulating the domain's configuration. For instance, you'll often come across the instruction "Select the desired server instance from the left pane of the Administration Console." Assuming that you've configured the server *myserver* within a WebLogic domain *mydomain*, this means that you need to navigate to the mydomain/Servers/myserver node of the tree menu in the left pane. Then, clicking the *myserver* node from the left pane automatically displays the server's configuration settings in the right frame of your browser window.
*Figure 1-2. Navigation menu of the Administration Console*Similarly, by clicking any WebLogic resource from the left pane, you can view and modify the associated configuration settings in the right pane of the Administration Console. Moreover, WebLogic maintains runtime statistics for each deployed resource, which you can view through the Monitoring tab in the right pane.The graphical tree menu also lets you perform specific operations on a resource by right-clicking the particular tree node. For instance, you can view the server's JNDI tree by right-clicking the *myserver* node in the left pane, and then choosing the "View JNDI tree" option from the pop-up menu.Now that you've set up your domain and understand the essentials of using the Administration Console to view the domain's configuration and monitor the various domain resources, you are ready to proceed on a journey into the world of WebLogic Server. Chapter 2. Web Applications Many of today's enterprise applications implement a web frontend. They choose to expose their business functions via a web user interface, most often accessed from the client's browser. These applications may access other enterprise services such as XML transformers, JNDI, database resources, and connection factories to fulfill their service contracts. WebLogic Server provides the ideal environment for creating rich web applications — it offers an extensive range of tools for assembling and configuring web components. Like any servlet engine, WebLogic supports all the functionality needed to host multiple web applications, which we shall cover in detail in later chapters.In this chapter, we look at the internal structure of web applications and their associated XML deployment descriptors. We shall see how WebLogic eases the task of building and assembling web applications, and examine some of the deployment issues of WebLogic Server. We also look at how you can configure the various web components.We take a peek at WebLogic's JSP compiler, and then examine JSP configuration issues when deploying a web application. Custom JSP tags are a useful mechanism for adding dynamic content to a JSP page. WebLogic provides a number of tag libraries. One such tag library offers useful caching functionality. Similar caching functionality is made available in the form of a servlet filter. WebLogic also provides a tool that can automatically create a JSP tag library from EJB components.You also will learn about WebLogic's servlet support, such as session tracking and session persistence. WebLogic provides a number of

ways to persist session state. File-, memory-, and cookie-based persistence mechanisms are supported. When using servlets in a cluster, in-memory session replication also can be used. This chapter examines these mechanisms, and looks at how to set up a simple web cluster with session replication. Finally, we look at various ways in which you can secure a web application using declarative and programmatic techniques. Setting up the HTTP over SSL (HTTPS) listen port and the associated SSL configuration is covered in Chapter 16. Chapter 3 concludes the discussion of the web environment — it describes how you can use and configure WebLogic's HTTP server and proxy plug-ins.Packaging and DeploymentAs a J2EE-compliant web container, WebLogic Server can host a number of web applications. Each *web application* is a logical collection of servlets, JSPs, client-side applets, and static *web resources* such as HTML pages, images, multimedia documents, etc. In addition, a web application may use filters, JSP tag libraries, utility Java classes, and JavaBean components. Each web application is executed in its own runtime environment provided by the web container — a handle to this environment is provided by ServletContext. We refer to JSPs and servlets as *web components*. These web components also have access to external resources and WebLogic enterprise services such as deployed EJB components, JDBC data sources, JMS destinations, XML parser factories, and much more through access to ServletContext.Structure of a Web Application   A web application is structured as a hierarchy of directories — the root directory serves as the *document root* for all resources in the web application. These directories contain all the JSPs, servlets, and other, static resources such as images that are referenced by the application. A special directory named *WEB-INF* holds all resources that aren't part of the public document tree of the application. So, a web client will not have direct access to any file stored under the *WEB-INF* directory, which is quite useful for protecting from clients specific resources in the web application. However, the contents of the *WEB-INF* directory are exposed to the server side. Both the methods getResource() and getResourceAsStream( ) of the ServletContext object and the forward( ) and include( ) methods of the RequestDispatcher object can access the contents of the *WEB-INF* folder. The *WEB-INF* folder also includes the following files:A *web.xml* deployment descriptor, which is the standard J2EE XML document that describes the contents of the web application A *weblogic.xml* deployment descriptor, which contains the WebLogic-specific deployment information about the web application A *classes* subdirectory, which is the base directory for all compiled servlets, filters, precompiled JSPs, implementation classes for custom JSP tags, and utility classesA *lib* folder, which is the location for all JAR files that may be useful to the web application, including custom JSP tag libraries Example 2-1 lists the contents of a typical web application.Example 2-1. Directory layout for a simple web application/index.html/home.jsp /error.jsp/demos/products.swf/images/banner.jpg/WEB-INF/web.xml/WEB-INF/weblogic.xml /WEB-INF/classes/com/oreilly/bar/MyServlet.class
/WEB-INF/classes/com/oreilly/bar/MyFilter.class
/WEB-INF/classes/com/oreilly/bar/MyTagImpl.class/WEB-INF/lib/mylibs.jarAssembling a WAR   A web archive (WAR) can be created very simply by placing the contents of the WAR in a staging directory — for instance, *wardir* — and then using the jar tool that comes with your JDK to create the archive:jar cvf mywar.war -C wardirAlternatively, you can use a standard Ant task: <war warfile="mywar.war" webxml="web.xml" manifest="manifest.txt"> <zipfileset dir="." prefix="WEB-INF" includes="weblogic.xml"/> <zipfileset dir="." prefix="images" includes="*.gif,*.jpg"/> <classes dir="classes" includes="**/MyServlet.class"/> <fileset dir="." includes="*.jsp,*.html"/></war>This has an advantage over the jar command because the different components of the WAR are not required to be in any particular fixed directory structure before creation. The war task can be part of an Ant script, which is used to build and deploy the web application. Chapter 19 shows how you can use WebLogic's tools to package web service components within a WAR file.XML Deployment Descriptors   A web application needs a standard J2EE *web.xml* descriptor and an optional WebLogic-specific *weblogic.xml* descriptor before it is ready for deployment. You can use these XML documents to define the web components and set up the operating environment for the web application. The *web.xml* descriptor is defined in the Servlet specification. It can be used to specify application-specific initialization parameters, servlets, filters and event listeners, MIME types, sessions tag libraries, security, and references to external

resources.Some web applications may require a *weblogic.xml* descriptor, which defines deployment properties specific to the runtime environment in WebLogic Server. In WebLogic Server 8.1, the *weblogic.xml* descriptor must be a valid XML document conforming to the DTD, published at http://www.bea.com/servers/wls810/dtd/weblogic810-web-jar.dtd. You can use this deployment descriptor to specify:HTTP session parameters (for the session tracking cookie, session persistence, and URL rewriting).JSP parameters (compilation settings for WebLogic's JSP compiler).Container attributes (configure whether you require reauthentication for forwarded requests and whether redirects use absolute or relative URLs).Character-set mappings and parameters.JNDI names for references to external resources defined in the *web.xml* descriptor file (including EJBs, data sources, security realms, etc.).Custom classes for URL pattern matching.Security role assignments to one or more principals in the realm.Virtual directories for mapping other document roots for a specific set of requests. For example, images for a web application may be stored in a separate location and need not be under the document root for the web application. If one or more virtual directories have been mapped, WebLogic Server will inspect the virtual directories for the resource before looking at the document root for the web application.All of these issues are covered in this chapter and in Chapter 3. Chapter 12 provides a number of techniques that may be used to edit deployment descriptors of web applications.WebLogic's JSP Compiler   WebLogic's internal JSP compiler translates JSP pages into servlet classes. By default, WebLogic automatically compiles JSPs without any user intervention. This compilation occurs when a JSP has been requested and the servlet class file has not yet been generated or the servlet class is older than the actual JSP page. This means that the compilation occurs automatically whenever a new JSP is invoked, or when the client has invoked a JSP that has been modified since the last time it was compiled. Therefore, you don't ever *need* to invoke the JSP compiler directly (although you may want to, as detailed in this section). Later, we shall see how you can register a JSP page as a servlet, and set it to load and initialize when the server starts up. In that case, the JSP compilation occurs during startup, even before a request is received from a client browser.You also can configure WebLogic Server to precompile all JSP pages when a web application is deployed (or redeployed) or when the server starts up. To enable JSP precompilation, you need to set a JSP configuration parameter within the jsp-descriptor element in the *weblogic.xml* descriptor file: <jsp-descriptor> <jsp-param> <param-name>**precompile**</param-name>   <param-value>**true**</param-value> </jsp-param> </jsp-descriptor>Alternatively, you may directly compile all JSPs, thereby avoiding the need for compilation while the server is running. This functionality is made available through the *weblogic.appc* compiler. The JSP compiler parses and translates the JSP files, generating Java source files, which it then compiles. The following command will validate the descriptors in the given WAR, compile the JSP files, and update the WAR with the resulting class files: java weblogic.appc -keepgenerated -verbose myWebApp.warYou can also simply point it to a directory containing a web application:java weblogic.appc -keepgenerated webappDirectory Table 2-1 lists various options available to the JSP compiler. *Table 2-1. JSP compiler options*

Option	*Description*	*Default*
-classpath	*This option customizes the classpath used by the JSP compiler. On a Windows NT/2000 platform, the classpath must be a semi-colon-separated list of the names of directories, ZIP, and/or JAR files. On Unix platforms, the classpath must be a colon-separated list of names.*	*none*
-compiler	*This option specifies the compiler to be used for compiling .java files. It defaults to the first javac binary in your PATH environment variable.*	*javac*
-output file	*Usually the compiler updates the web archive, adding to its current contents. If you want to send the output to a different WAR, specify the filename here.*	*none*
-forceGeneration	*By default, the compiler compiles only those files that it thinks should be compiled, judging by timestamps. If you supply this option, it will recompile everything.*	*false*
-g	*This option instructs the compiler to include debugging information in the class file.*	*none*
-keepgenerated	*This instructs the compiler to not delete the intermediate .java source file once the JSP file has successfully compiled. In the absence of JSP debugging tools, this flag is extremely useful for tracking down bugs in your JSP code.*	*none*
-O	*This tells the compiler to compile generated .java files with the optimization flag on (overrides -g flag).*	

none
-verbose This option determines whether the JSP compiler runs in a verbose mode.   false
-verboseJavac This option passes the verbose flag to the Java compiler when compiling intermediate .java files.   false You also can define JSP compiler settings for the web application in the WebLogic-specific *weblogic.xml* deployment descriptor, as detailed in the following section. Deploying a Web Application   A web application can be packaged in two ways before it is ready for deployment:Exploded directory format
  Here the contents of the web application are placed in a folder. This is recommended during the development stages, when you must directly modify the resources within the web application, without the need for repackaging.Archived format
Alternatively, you can create a WAR, where the contents of the web application are bundled into a JAR file with a *.war* extension. This approach is recommended for a production environment.Thus, you have the flexibility of deploying your web application in exploded format, if it is undergoing constant changes during development stages. Otherwise, you may package the contents within a standard WAR file, and then simply deploy the web application in a production environment. In fact, WebLogic provides two modes of operation that impact how you deploy your web applications:Development mode
  Enable this mode during the development stages, when your web application is under construction and the team members are concurrently making changes to its contents. Its key benefit is the ease with which you can directly modify the resources within the web application, and then instantly see the results of the changes without having to redeploy the entire web application.Production mode
Use the production mode when you need to make infrequent changes to your web applications. Typically, you will package the web application in archived format when deploying it to a server running in this mode. So, if you update any Java class files or modify any of the deployment descriptors, you must redeploy the entire WAR. Unfortunately, redeploying a web application means WebLogic loses all active HTTP sessions associated with that web application. If the web application is targeted to other Managed Servers, your network must also endure a surge in traffic while all changes are propagated to all the servers.On top of this, you can decide whether a web application will be deployed as a standalone application or as a J2EE module within an enterprise application, alongside other web applications, EJB components, resource adapters, and library JARs. Refer to Chapter 12 for more information on how to package enterprise applications and later introduce them to a WebLogic environment. Chapter 12 also introduces WebLogic's split directory development structure, which can be used as an alternative to deploying web applications during development.The *applications* folder under the domain's root directory provides a convenient store for standalone applications that are deployed to WebLogic. In the development mode, you can deploy a web application to WebLogic simply by placing the WAR file (or a directory, if it exists in an exploded format) under the domain's *applications* directory. WebLogic also supports an *auto-deploy* feature in the development mode, which greatly eases the task of updating your web applications: If your web application is packaged as a WAR file, you can redeploy the web application simply by modifying the WAR file (or replacing it with a new copy).If your web application exists in the exploded form, any changes you make to resources within the web application are automatically reflected without having to redeploy it. Any changes to static files (such as HTML pages, images, etc.) and JSP pages are automatically picked up by WebLogic.You can refresh compiled Java classes used by the web application. This means that you can *hot-deploy* changes to servlets, filters, JSP tag implementations, and classes located under the *WEB-INF/classes* folder without the need for redeployment. Another neat little trick WebLogic provides is its ability to redeploy the web application whenever you touch a *REDEPLOY* file in the *WEB-INF* directory. All of these features are supported by default in the development mode. The value of the command-line parameter weblogic.ProductionModeEnabled determines whether WebLogic operates in the development mode. The following command shows how to start a server in the production mode:java -classpath %CLASSPATH% -Dweblogic.Name=myserver
 -Dbea.home=%BEA_HOME%   -Dweblogic.management.username=%WL_USER%

-Dweblogic.management.password=%WL_PW%   -Djava.security.policy=...
-Dweblogic.security.SSL.trustedCAKeyStore=...   **-Dweblogic.ProductionModeEnabled=true ...**
weblogic.Server Chapter 12 looks at how to use the Administration Console or WebLogic's Deployer tool to manage deployed applications in a WebLogic environment.Class Loaders WebLogic Server provides a separate class loader for each web application deployed as part of an EAR. This class loader ensures that all servlets and classes in a web application are loaded within the same scope. This eliminates the possibility of classes loaded from the web application conflicting with WebLogic Server implementation classes. For instance, even though WebLogic needs XML parsers to process deployment descriptors, a web application can still load parsers supporting different SAX or DOM versions, without conflicting with the server's copy. So, separating the scope of classes in a web application from the server's bootstrap classes resolves this issue quite elegantly.WebLogic provides an option for changing the default class-loading mechanism for a web application. This option, called Prefer Web-Inf Classes, can be set by modifying the *weblogic.xml* descriptor file: <container-descriptor>
  <prefer-web-inf-classes>true</prefer-web-inf-classes></container-descriptor>In WebLogic 7.0, this option is available in the Configuration/Other tab of a web application. The default mechanism in most class-loading hierarchies is that if a child class loader is asked for a class, and it doesn't have it in memory, it asks its parent to load the class. Only if the parents fail does the child load the class. So, if we have a servlet in a web application that requires a class that has not been loaded by the web application class loader, the web application should pass the request on to its parent class loader. By setting the prefer-web-inf-classes element to true, WebLogic changes the class-loading logic to allow the web application class loader to immediately try and load the class, without first asking the parent. This rather delicate setting could be used in esoteric cases where you want the web application class files to take priority over parent class loaders, perhaps the EAR class loader, because of possible class implementation differences.A second issue related to class loading is that the J2EE standard specifies that any class loader for a web application is required to load classes from the *classes* folder before loading any classes from the JARs in the *lib* folder. For instance, if a class exists both in the *classes* folder and in JARs in the *lib* folder, WebLogic Server will always load the class from the *classes* folder.Usually, the contents of the */WEB-INF/classes* folder won't conflict with the JARs located under the */WEB-INF/lib* folder. The *classes* folder will typically contain the servlet and utility classes, whereas the *lib* folder will contain other supporting library JARs needed by the web application, including tag library JARs. The Class-Path entry in the manifest file *META-INF/MANIFEST.MF* for a WAR also can be used to reference any supporting JARs. This manifest entry for extending the CLASSPATH to include other WAR dependencies must follow the standard JAR manifest format. You can find more information on the standard extension mechanism for JAR files at http://java.sun.com/j2se/1.4/docs/guide/extensions/spec.html. Chapter 12 provides an extensive overview of WebLogic's classloader hierarchy.Configuring Web Applications   The configuration for a web application deployed on WebLogic is spread across the application's deployment descriptors and the server configuration file. For instance: The session timeout for the web application can be configured using the session-timeout element within the standard *web.xml* deployment descriptor. The JSP compilation parameters for a web application are defined in the WebLogic-specific *weblogic.xml* deployment descriptor. The *staging mode setting*, which determines how a web application is deployed, can be found in the server's *config.xml* file.The easiest approach to editing the deployment descriptors is to use either the Administration Console or WebLogic Builder. The following sections examine WebLogic-specific configuration settings related to web applications.Context Root   A web application is rooted at a specific path within the web server, called the *context root*. For example, if the context root for a web application is set to here, you could access it by using a URL of the form *http://server:port/here/index.html*. If you do not explicitly set a context root for the web application, its default value depends on how the web application has been packaged:If the web application is deployed in the exploded format, its default context root is the name of the folder that holds the contents of the web application.If the contents of the web application have been packaged as a web archive, its default context root is the name of the web archive itself. An example of this is if you deploy the web application *myWar.war* to WebLogic,

you can invoke a resource — say, *index.html* — using the URL *http://server:port/myWar/index.html*. There are two ways to configure the context root for a web application:If it is deployed as a standalone application, you can set the context-root element within the *weblogic.xml* descriptor for the web application: <weblogic-web-app> <context-root>**ourwebapp**</context-root> </weblogic-web-app>If it exists as a J2EE module within an enterprise application, you can configure the context-root element in the standard *application.xml* deployment descriptor for the enterprise application:<module> <web>   <web-uri>mywebapp</web-uri> **<context-root>ourwebapp</context-root>** </web></module>As shown in Chapter 3, setting the default web application for the web server or virtual host means that the context path can be omitted altogether from URLs that access the web application.Directory Listings   If a client makes a partial request that resolves to a directory that doesn't contain any of the pages specified in the list of welcome files, WebLogic Server returns a 404 response. However, if you have checked the Index Directories flag for the web application, WebLogic then will return a directory listing to the client instead. By default, if no welcome files are defined, WebLogic looks for the following files in order and serves the first one it finds: *index.html*, *index.htm*, and *index.jsp*.The Index Directory Enabled flag can be reached by selecting the web application in the Administration Console and moving to the Configuration/Descriptor tab. If you have enabled this option, you also can specify the sort order of the directory listing. This can be done only in the *weblogic.xml* descriptor file. Here is how to enable directory listings and specify a sort order:<container-descriptor>
  <index-directory-enabled>true</index-directory-enabled>
  <index-directory-sort-by>SIZE</index-directory-sort-by></container-descriptor>In WebLogic 7.0, this option is available in the Configuration/Files tab of a web application. The valid sort orders are SIZE, NAME, or LAST_MODIFIED.Serving Static Files   By default, WebLogic uses an internal servlet, weblogic.servlet.FileServlet, to serve requests for static resources in a web application. You can modify this default behavior by mapping a custom servlet to the URL pattern /. Your custom servlet will then respond to requests for all files except those with the extension *.htm* or *.html*, which still will be handled by FileServlet. If your custom servlet must also handle requests for HTML files, you need to explicitly associate the HTML files with your servlet. The following portion from the *web.xml* descriptor file shows how to override the default FileServlet for all web resources:<!-- web.xml entries --><servlet-mapping> <servlet-name>MyServlet</servlet-name>   <url-pattern>**/** </url-pattern> </servlet-mapping><!-- Map HTML files to the servlet --><servlet-mapping>
  <servlet-name>MyServlet</servlet-name>   <url-pattern>***.htm***</url-pattern> </servlet-mapping>Enabling CGI Scripts   WebLogic can be configured so that web applications can support Common Gateway Interface (CGI) scripts. Your WebLogic distribution is shipped with a CGI servlet, weblogic.servlet.CGIServlet, which provides a gateway to such scripts. Example 2-2 shows how you can configure the CGI servlet to invoke shell scripts.   Example 2-2. Enabling CGI for a web application<servlet> <servlet-name>CGIServlet</servlet-name>
  <servlet-class>weblogic.servlet.CGIServlet</servlet-class> <init-param> **<param-name>cgiDir</param-name>**   <param-value>scripts</param-value> </init-param>
  <init-param>   **<param-name>*.sh</param-name>**
 <param-value>d:cygwinbinbash</param-value> </init-param></servlet>...<servlet-mapping> <servlet-name>CGIServlet</servlet-name>   <url-pattern>/cgi/*</url-pattern></servlet-mapping>
CGIServlet requires the following initialization parameters: cgiDir
This parameter specifies a list of the names of directories containing your CGI scripts. For Unix platforms, you must remember to use a colon (:) to separate multiple folder names. By default, WebLogic looks for scripts in the *cgi-bin* directory under the document root for the web application.
Extension mappings
These parameters let you map a file extension to an interpreter or an executable that can run the scripts. You can define any number of these mappings, mapping different file extensions to either the same (or different) interpreter.If you have configured the CGI servlet for a web application as shown in Example 2-2, you can invoke the script *helloworld.sh* by simply navigating to the URL *http://server:port/webapp/cgi/helloworld.sh*. You must place the *helloworld.sh* file into the *scripts* directory of your web application.WarningEnsure that the CGI script returns an exit code of 0 when

it ends successfully. Otherwise, WebLogic concludes the script has failed, and will report a server error (500) to the browser.   Reloading Files   WebLogic automatically picks up any changes to static content within a web application in the *development* mode. In fact, WebLogic is able to detect changes to servlets, filters, JSP tag implementation classes, and any classes found under the *WEB-INF/classes* folder. This is possible because WebLogic regularly inspects the filesystem for changes in web resources. You can adjust the frequency with which WebLogic looks at the filesystem for changes in the contents of the web application. Select the web application from under the Deployments/Web Application Modules node in the left pane of the Administration Console. Then, from the Configuration/Settings tab in the right pane, adjust the value of the Servlet Reload Check Secs setting. WebLogic 7.0 users can find this in the Configuration/Files tab.Changes to JSP files also are picked up provided you've set a nonnegative value for the pageCheckSeconds element in the *weblogic.xml* descriptor for the web application. Resources References The J2EE specification defines how to configure the naming environment, which allows web applications to easily access resources and external information without actually knowing how that information is named or organized. Depending on the resource, you will use one or more of the following elements in the standard *web.xml* descriptor: the env-entry, ejb-ref, ejb-local-ref, resource-ref, security-role-ref, and resource-env-ref elements. At runtime, you can use these elements to access objects registered in the JNDI namespace for the web container. The *weblogic.xml* descriptor file must be configured to map these references to real resources that have been deployed to the server. For instance, suppose you've configured WebLogic with a data source, which you've placed in the JNDI with its name set to myDataSource. Now you can define the following entries in the deployment descriptors for the web application: <!-- web.xml entry: --><resource-ref> <res-ref-name>**jdbc/myds**</res-ref-name>
  <res-type>javax.sql.DataSource</res-type> <res-auth>Container</res-auth></resource-ref><!-- weblogic.xml entry: --><resource-description> <res-ref-name>**jdbc/myds**</res-ref-name>
  <jndi-name>**myDataSource**</jndi-name></resource-description>Then, you can access this data source using the resource reference jdbc/myds as follows:javax.sql.DataSource ds =
  (javax.sql.DataSource) ctx.lookup("java:comp/env/jdbc/myds");Note that your Java code doesn't have to explicitly reference the global JNDI name myDataSource — instead, you can use the alias jdbc/myds.Similarly, suppose you've defined the following entries in the deployment descriptors for a web application:<!-- web.xml entry: --><ejb-ref> <ejb-ref-name>**ejb/foohome**
</ejb-ref-name> <ejb-ref-type>Session</ejb-ref-type> <home>com.foo.bar.FooHome</home>
  <remote>com.foo.bar.Foo</remote></ejb-ref><!-- weblogic.xml entry: -->
<ejb-reference-description> <ejb-ref-name>**ejb/foohome**</ejb-ref-name>
  <jndi-name>FooHome</jndi-name></ejb-reference-description>Here the standard *web.xml* descriptor file declares an EJB reference to a session bean, and the *weblogic.xml* descriptor file maps the EJB reference to its actual JNDI name. You then can access the session bean registered in the global JNDI tree under the name FooHome using the EJB reference *ejb/foohome* as follows:Object home = ctx.lookup("java:comp/env/ejb/foohome");FooHome fh = (FooHome) PortableRemoteObject.narrow(home, FooHome.class);Again, we have not used the session EJB's actual JNDI name within our code. Instead we used its alias *ejb/foohome*.The WebLogic Builder tool provides an easy way to declare these references.Response Caching   WebLogic supplies a cache servlet filter that permits page-level caching of responses. Setting up such a filter is very easy. Example 2-3 establishes a cache filter that works on all files ending with a *.html* extension, and configures the cache to be application-wide.Example 2-3. A cache filter for all HTML files in a web application<filter> <filter-name>**HTML**</filter-name>
  <filter-class>weblogic.cache.filter.CacheFilter</filter-class> <init-params>
<param-name>scope</param-name>Â Â Â <param-value>application</param-value>
  </init-params></filter><filter-mapping> <filter-name>**HTML**</filter-name>
  <url-pattern>*.html</url-pattern></filter-mapping>The cache filter automatically caches the Content-Type and Last-Modified response header fields. A cached page is served only if the If-Modified-Since request header is more recent than the Last-Modified response header — otherwise, the filter sends back an SC_NOT_MODIFIED(302) status with no content. The filter can

be configured in various ways depending on the values of the initialization parameters. You can define the following parameters: name
  Specifies the name for the cache; defaults to the request URI. timeout
Determines the amount of time to wait between cache updates. The default unit of time is seconds, but this can be changed by specifying a value followed by ms (milliseconds), s (seconds), m (minutes), h (hours), or d (days). Note that the value of an item is refreshed only when it is requested and timed out, not if it is only timed out. scope
Determines whether the scope of the cache is for the request, session, application, or cluster. The default scope is application. size
Determines the number of different unique key values that are cached. It defaults to infinity (limited by memory because of the soft reference implementation). key
Ordinarily the key will be the URL requested. You can use this attribute to specify a comma-separated list of additional keys. The value of this attribute will be the name of the variable whose value you wish to use as a key into the cache. You additionally can specify a scope by prepending the scope to the key. For example, a value of application.mykey implies the target resource will carry an additional key — i.e., the value of the attribute mykey in the context for the web application. If no scope is prepended, WebLogic will search through the scopes in the order shown earlier. Cache filters are extremely useful; they provide a reusable and configurable way for caching various parts of your web application without requiring any code changes. When configured properly, cache filters can provide a significant performance boost to your web application. The next chapter shows how WebLogic's JSP cache tags support a slightly richer version of this caching behavior. It also provides more examples on the use of keys.   Servlets and JSPs WebLogic supports HTTP servlets as defined in the Servlet 2.3 specification, as well as JSP pages and tag extensions as defined in the JSP 1.2 specification. Typically, you will use servlets alongside JSPs to build a web application. WebLogic provides a number of useful additions to standard servlet and JSP configuration. For example, you can assign custom execute queues to critical servlets, and install servlets to serve static files and act as a CGI gateway.Configuring a Custom Execute Queue   By default, all J2EE applications (except JMS resources) share the same pool of server threads for their operation. This includes the servlets and JSPs that rely on the default execute queue configured for a particular WebLogic instance. However, WebLogic also lets you assign a custom execute queue that is dedicated to an individual servlet (or JSP). In this way, you can ensure that a dedicated pool of threads is always available for a particular servlet and that it doesn't need to compete with other services for a free server thread. Execute queues are explained is more detail in Chapter 15. To assign an execute queue to a servlet, you need to modify the *weblogic.xml* descriptor file to include a dispatch-policy element for the servlet. The value of this element should match the name of a preconfigured execute queue. Here is an example:<servlet-descriptor> <servlet-name>FooServlet</servlet-name>
 <init-as-principal-name>system</init-as-principal-name>
 <destroy-as-principal-name>system</destroy-as-principal-name>
<dispatch-policy>MyCustomExecuteQ</dispatch-policy></servlet-descriptor>In this way, you can configure each servlet with its own execute queue, or perhaps even force multiple servlets to share the same execute queue.Threading Issues   When using multithreaded servlets, you must make adequate provisions for concurrent access within the service methods. Your servlet code needs to guard against sharing violations on access to shared resources and member variables. That said, wherever possible, avoid synchronization because it blocks other servlet threads until the current thread has completed. Limit sharing across threads by defining variables within the scope of the service methods. If you are accessing external resources such as databases, JMS destinations, etc., you need to synchronize on the class level, or whenever possible, encapsulate your work in a transaction.*Single-threaded servlets*   An instance of a servlet that implements the SingleThreadModel interface is guaranteed never to be invoked concurrently by multiple threads. WebLogic creates multiple instances of a single-threaded servlet so that it can serve multiple client requests simultaneously. This pool of servlet instances is created initially when the servlet is first requested. You can use the single-threaded-servlet-pool-size element in the *weblogic.xml* file to

specify the initial number of servlet instances that are created. Typically, you will set the value of this attribute to the average number of concurrent requests that the servlet is likely to receive. In WebLogic 8.1, you can configure this setting from the Administration Console. Select the web application in the left frame, and then, from the Configuration/Descriptor tab, supply a value for the Single Threaded Servlet Pool Size setting. By default, WebLogic initializes the pool with five servlet instances. Because WebLogic creates a pool of servlet instances (one for each thread), you effectively multiply the memory requirements of the servlet, at least for all the nonstatic attributes of the servlet. If you have declared shared class variables (for instance, a static attribute), even though it will be accessed in a single-threaded manner, there could be many such single-threaded instances potentially accessing the same resource. You must be careful to ensure that you avoid all synchronization issues.Custom URL Pattern Matching   The servlet-mapping declaration in the *web.xml* deployment descriptor allows you to specify the URL pattern that must be matched to invoke a particular servlet. WebLogic provides an extension to the URL matching mechanism and allows you to plug in a richer pattern matcher — for instance, a matching scheme that goes beyond the use of just the / and * metacharacters. To configure this scheme, you must use the url-match-map element in the *weblogic.xml* descriptor file to specify the fully qualified name of a class that provides the actual logic for custom URL matching: <url-match-map>
  com.oreilly.wlguide.servlets.OReillyURLMatchMap</url-match-map>This class must implement the following interface, found in the weblogic.servlet.utils package:public interface URLMapping {
 public void put(String pattern, Object value); public Object get(String uri); public void remove(String pattern) public void setDefault(Object defaultObject); public Object getDefault( );
 public void setCaseInsensitive(boolean ci); public boolean isCaseInsensitive( ); public int size( );
 public Object[] values( ); public String[] keys( );}By default, WebLogic Server uses the J2EE-standard URL pattern-matching scheme. This is set up because the value of the url-match-map element defaults to weblogic.servlet.utils.URLMatchMap.Configuring JSPs Just as you can when deploying servlets, you can use the servlet element in the standard *web.xml* deployment descriptor to register a JSP page: <servlet> <servlet-name>home</servlet-name>
<jsp-file>home.jsp</jsp-file></servlet><servlet-mapping>
  <servlet-name>home</servlet-name> <url-pattern>/home</url-pattern>
</servlet-mapping>Here, a request to the URL */home* will cause */home.jsp* to be invoked. In Chapter 18, you'll see how a similar mapping enables XSLT conversion from within a JSP. By registering a JSP as a servlet in this way, you can do the following:Specify the load order of JSP pagesDefine any initialization parameters for those pagesRestrict access by applying security roles to JSP pages The jsp-descriptor element in the *weblogic.xml* descriptor file allows you to specify additional settings for JSP compilation. Each JSP configuration parameter is defined as a name/value pair within a jsp-param subelement. Here is an example of how to configure the translator to retain generated Java files:<jsp-descriptor> <jsp-param>   <param-name>**keepgenerated**</param-name>
 <param-value>**true**</param-value> </jsp-param></jsp-descriptor> Table 2-2 provides a complete list of JSP configuration parameters that may be defined in the jsp-descriptor element. Remember, you need to define a separate jsp-param element for each configuration parameter.*Table 2-2. JSP configuration parameters*

Parameter name	*Description*	*Default*
compileCommand	*This specifies the full pathname of the standard Java compiler used to compile .java files generated from a JSP. By default, the compiler uses javac (from your PATH environment variable) or the compiler set in the server configuration for WebLogic Server.*	*javac*
compileFlags	*This parameter specifies one or more flags to be used by the standard Java compiler. Multiple flags should be enclosed in quotes: <jsp-param> <param-name>compileFlags</param-name> <param-value>"-g -v"</param-value></jsp-param>*	*none*
debug	*If this parameter is set to true, the JSP compiler adds JSP line numbers to the generated source files (as an aid to debugging).*	*false*

BEA's WebLogic Server implements the full range of J2EE technologies, and includes many additional features such as advanced management, clustering, and web services. Widely adopted, it forms the core of the WebLogic platform, providing a stable framework for building scalable, highly available, and secure applications. In fact, in the long list of WebLogic's strengths and features, only one shortcoming stands out: the documentation that comes with the WebLogic server often leaves users clamoring for more information.

WebLogic: The Definitive Guide presents a 360-degree view of the world of WebLogic. Providing in-depth coverage of the WebLogic server, the book takes the concept of "definitive" to a whole new level. Exhaustive treatment of the WebLogic server and management console answers any question that developers or administrators might think to ask. Developers will find a useful guide through the world of WebLogic to help them apply their J2EE expertise to build and manage applications. Administrators will discover all they need to manage a WebLogic-based setup. And system architects will appreciate the detailed analysis of the different system architectures supported by WebLogic, the overall organization of a WebLogic domain and supporting network infrastructure, and more.

WebLogic: The Definitive Guide is divided into three sections that explore WebLogic and J2EE, Managing the WebLogic Environment, and WebLogic Enterprise APIs. Some of the topics covered in this comprehensive volume include:Building web applications on the WebLogic ServerBuilding and optimizing RMI applicationsUsing EJBs with WebLogic, including CMP entity beansPackaging and deploying applicationsUnderstanding WebLogic's support for clusteringPerformance tuning and related configuration settingsConfiguring WebLogic's SSL supportMaximizing WebLogic's security featuresBuilding web services with XMLUsing WebLogic's JMX services and MBeansAnyone who has struggled with mastering the WebLogic server will appreciate the thorough, clearly written explanations and examples in this book. *WebLogic: The Definitive Guide* is the definitive documentation for this popular J2EE application server.

About the Oracle Service Bus - Title Published Details Maven: A Developer's Notebook Jun2005 Modules Book Jan 2007 ISBN0132409674 The Definitive Guide to C. Commenting Guidelines. application server, the opensource equivalent to BEA's WebLogic Server. Extreme Programming with Ant: Building and Deploying Java Ubuy Kuwait Online Shopping For bea in Affordable Prices. - recommended in vendors' product documentation, in books, and from online the SASÂ®9 BI Platform will probably be in a test or development environment. Will automated processes be required for deploying and configuring SAS Web Applications that run in a Tomcat, WebLogic, or WebSphere servlet container: o. Read PDF Ghost: The True Story of One Mans Descent into - For a better look, you are able to open some examples below. Sx600f Sx700 Snowmobile Service Repair Maintenance Overhaul Workshop Manual Language Acquisition And Conceptual Development Levinson Stephen

Bowerman Palladium In Heterocyclic Chemistry Volume 26 Second Edition A Guide For The Tomcat 9 Pdf - labussola - Ã„hnliche BÃ¼cher wie WebLogic: The Definitive Guide: Development, Deployment & Maintenance. (Definitive Guides) (English Edition) Roman-Neuerscheinung: Charles F. Goldfarb's XMLbooks.com - Tomcat: The Definitive Guide offers something for everyone who uses Tomcat. make it a useful platform for developing and deploying web applications and web services. This book takes you from beginner to expert in logical stages, covering all. Many servlet containers could be used for this course. x or WebLogic 8. Ebook Weblogic The Definitive Guide Definitive Guides - mx.tl - NET Development for Java Programmers .NET Development A Project Guide for Deploying Tivoli Solutions. A Roadmap for A+ Guide To Hardware: Managing; Maintaining.. Guide. BEA WebLogic Server 7.0: Deployment &.. Complete. C# Core Language Little Black Book... Guidelines for Application Integration. Continuous Integration for the Masses. Hudson. Jenkins. The - For a better look, you are able to open some examples below. Park For Kids Preteens And Teenagers A Grande Guides Series Book For Children Fighting The Legend Teacher S Resource Guide Saddleback Educational Publishing. 1997 Yamaha F9 9mlhv Outboard Service Repair Maintenance Manual Factory All IT eBooks - Free Download IT eBooks - "WebLogic: The Definitive Guide" presents a 360-degree view of the world of WebLogic. Providing in-depth coverage of the WebLogic server, the book takes the concept of "definitive" to a whole new level. Exhaustive WebLogic: The Definitive Guide: Development, Deployment & Maintenance Definitive Guides. Authors Weblogic Disable Weak Ciphers - Oracle WebLogic Server is a Java EE application server currently developed by Oracle I've have been asked by Packt publishing to review a brand new book on Oracle. Find user guides, developer tools, getting started guides, tutorials,. PL/SQL programming guide by the Oracle community, this definitive guide is Ubuy Taiwan Online Shopping For bea in Affordable Prices. - Mastering BEA Vertx java - Yantami - Chapter 17: Deployment Targets .. However, the book will reference the name Jython when discussing. the rest of the block can maintain a consistent indentation, which makes... guidelines and examples to use when developing integrated Jython.. Fish, JBoss, Tomcat, WebLogic, and WebSphere.

Relevant Books