Controls Containment

Table of contents

1 Overview....2
2 Basic Architecture....2
2.1 The Foundation: JavaBeans Containment....2
2.2 Building Up: Controls Containment....4
3 Control Container Services....7

1. Overview

This document describes the basic architecture for how Beehive Controls interact with the runtime container they are executing within. Examples of runtime containers for Controls include:

- Servlet Container
- EJB Container
- Web Services (WSM)
- Client JVM
- JUnit Test Container (standalone testing)
- ...

The base runtime comes with a sample container integration for the servlet container, but the container integration model is flexible enough to support any and all of the above containers, as well as enabling the list above to be extended or customized in new and interesting ways. The model makes it possible to author controls that run in a wide variety of containers, as well as ones that expect and leverage the capabilities of a specific container (where desirable).

This is possible because there is a basic architecture for how Controls will interact with their container. This includes the interfaces for how a new type of container can be constructed, or for how an existing runtime environment can be extended to act as a container of controls.

There are two target audiences for this document:

- A Control author who wants a deeper understanding of how controls interact with their runtime environment for resource management, configuration, contextual services, etc.
- A Control container developer who wants to define a new type of Control container to integrate support for Beehive Controls into an existing environment.

2. Basic Architecture

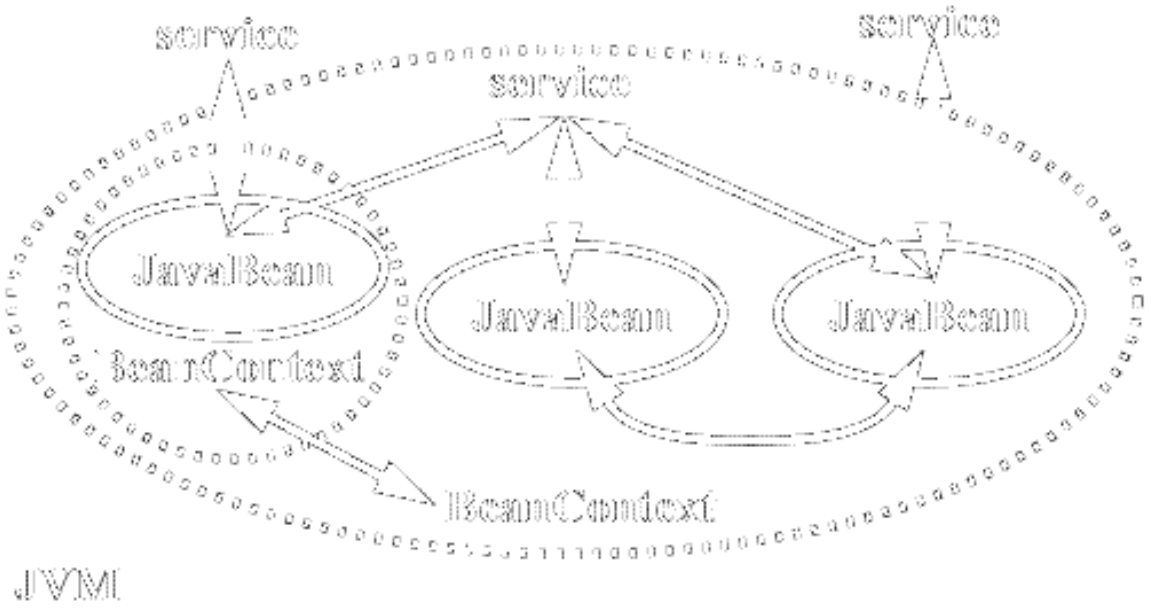
This section outlines the basic implementation architecture for Control containment. Containment is based upon an existing JavaBean standard for bean composition and services, and build atop this to provide additional features that are unique to Controls.

2.1. The Foundation: JavaBeans Containment

The basic foundation of Control Containment is the Extensible Runtime Containment and Services Protocol For JavaBeans", a little known but useful containment model for JavaBeans that has been part of J2SE since 1.2. The intent of the protocol (actually, a set of

interfaces and supporting implementation classes) was to add a simple containment model for JavaBeans, as well as a mechanism for allowing beans to discover, request, and use services provided by their container. All of the APIs defined by the protocol live in the java.beans.beancontext package.

This diagram shows the basic architecture for this protocol:



The basic concepts shown in the diagram are:

- A JavaBean can be nested within a BeanContext that acts as a container for one (or more) beans.
- A BeanContext can itself be nested within another BeanContext, enabling hierarchical composition.
- A BeanContext can provide the JavaBean with access to services. These services may be directly implemented by the BeanContext, or the BeanContext may simply act as a discovery mechanism/bridge to services provided by the runtime environment of the BeanContext.

Some of the key classes are described in the following sections:

2.1.1. BeanContext

The java.beans.beancontext.BeanContext interface defines the basic interface for a container of JavaBeans. It derives from the java.util.Collection interface, so the standard Collection APIs can be used to add, remove, and iterate over the JavaBeans contained within the context. It also extends the java.beans.beancontext.BeanContextChild interface (see next section), meaning it is possible for one BeanContext to be nested within another BeanContext, forming a hierarchical structure.

Whenever you see word "*BeanContext*" throughout this document or in API names, mentally replace it with "*JavaBeans Container*".

2.1.2. BeanContextChild

The java.beans.beancontext.BeanContextChild interface defines the basic interface that will be implemented (directly or indirectly via java.beans.beancontext.BeanContextProxy) by a JavaBean that wants to be contained within/access the services of a BeanContext. It defines the basic mechanism for setting/retrieving the parent BeanContext for a JavaBean, as well as the APIs for listening to / vetoing property changes on the nested bean.

2.1.3. BeanContextServices

The java.beans.beancontext.BeanContextServices interface derives from the BeanContext interface and defines a BeanContext that is capable of providing services to the JavaBeans contained within it. It defines a model for how services can be discovered and used by the contained JavaBeans, as well as a model for how service providers can register themselves with the context so their services will be available.

Service discovery is hierarchical; if a particular BeanContext does not implement a service requested by a contained JavaBean, but is itself contained within another BeanContext, it will delegate the request upwards to see if any parent context can provide the requested service.

2.2. Building Up: Controls Containment

The Beehive Controls runtime builds atop the base JavaBeans BeanContext model by adding a set of interfaces and support classes that provide containment and composition services that are unique to Controls. This section provides an overview of this functionality, and a subsequent section on *Control Container Services* will describe them in more detail.

2.2.1. ControlBeanContext

The org.apache.beehive.controls.api.context.ControlBeanContext interface extends the base java.bean.BeanContextServices interface to add the unique services available to JavaBeans that are Beehive Controls. These include access to control property values bound by

annotations, external configuration, or client invocation of property accessors, as well as a unique set of lifecycle events.

Every Control is guaranteed to have an associated peer ControlBeanContext that can be used to query Control properties, nest other controls (either declaratively or programmatically), and to receive lifecycle events. This is true even if the Control is not itself nested within a parent context.

This peer ControlBeanContext can be obtained by declaring:

```
@Context ControlBeanContext myContext;
```

within the Control Implementation class, or by calling the org.apache.beehive.controls.api.bean.ControlBean.getControlBeanContext() API on a Control bean instance.

2.2.1.1. Control Identifiers

The ControlBeanContext interface goes beyond the simple java.util.Collection collection capabilities of the base BeanContext class to also manage a unique identifier associated with each contained Control. This identifier can come from a number of different sources:

- An argument to the bean constructor
- The field name, for an instance field annotated with @Control. In the example above, the Control ID would be "myContext"
- The ControlBeanContext will autogenerate a unique one, if none is provided

Because Controls can be hierarchically nested inside one another, a given Control instance actually has two IDs: the local (or BeanContext relative) ID that was provided by one of the mechanisms above and a full (or absolute) ID that is built by concatenating the IDs of all Controls from the root BeanContext down to the control, using a forward slash ('/') as a separator.

For example, the Control ID "foo/bar" refers to the Control with a local ID of "bar" that is nested inside the control with a local ID of "foo" in the root context.

An absolute Control ID is effectively a unique address that allows a control within a BeanContext hierarchy to be located by traversing the ownership path of Controls defined by this composite identifier.

This is useful in a number of contexts:

- To enable external configuration based upon identifier (usage context) rather than just a type.

- To locate a specific control instance within a hiearchy by navigating the BeanContext tree based upon the ID. This can be useful in scenarios such as an external event dispatch.

The org.apache.beehive.controls.runtime.bean.ControlBeanContext class provides a concrete implementation of the ControlBeanContext interface for the Controls runtime.

This class is used:

- To provide the basic set of services for Controls. Every instantiated Control will have an associated ControlBeanContext to provide access to properties, or to contain nested Controls.
- As a base class for other types of Control containers. These common services are also available for other types of containers that want to support controls. A later section describes this in more detail.

2.2.2. ControlBean

The org.apache.beehive.controls.api.bean.ControlBean interface defines a base interface implemented by all Controls. It provides accessors for:

- The parent BeanContext of the Control
- The peer ControlBeanContext providing property access and containment for nested controls
- The (absolute) Control ID of the Control
- The public interface (@ControlInterface or @ControlExtension) implemented by the Control.

The org.apache.beehive.controls.runtime.bean.ControlBean class provides a concrete implementation of the ControlBean interface, and is used as the base class for all code-generated Control JavaBeans.

2.2.3. ControlContainerContext

The org.apache.beehive.controls.runtime.bean.ControlContainerContext class extends the base ControlBeanContext class to define a base integration model (and default implementation, where appropriate) of containment and services to integrate an external container type with the Controls runtime. Examples of existing external containers for controls are the Servlet container, the EJB container, the Spring bean container, ...

An external container can provide additional services to Controls that are running within its scope, such as a definition of how long it is OK for Controls to acquire and hold resources, the integration of a native container configuration model, or contextual services that are unique and specific to the container.

For any given container, a custom subclass of the ControlContainerContext class can be provided that defines the unique attributes and semantics of the container for controls executing within it. For example, the ServletBeanContext class provided as part of the Controls runtime provides control containment for the web tier. The ServletBeanContext defines the resource scope for Controls such that any control instance can hold a resource (connection, session, ...) for the lifetime of a single http request (but no longer). Additionally, it exposes web-tier-specific contextual services, such as access to the current ServletContext or active HttpServletRequest instance.

The following section on Control Container Services describes many of the services and behaviors that can be customized by a ControlContainerContext subclass.

A ControlContainer context is intended to be the root context which will contain (either directly, or indirectly via nested BeanContexts) all Controls used within the scope of a container instance. Generally speaking, the relationship between container instances and ControlContainerContext instances will be one-to-one.

Impl Note: there really should be an org.apache.beehive.controls.spi.context.ControlContainerContext interface that defines the basic interface for Controls containment, with the above class acting as the concrete implementation thereof. This follows the pattern used everywhere else, and decouples the declaration of control container requirements from its implementation.

3. Control Container Services

The interactions between a control and its container are best expressed in terms of the set of functional services that the container provides to the control. This provides a basic framework for understanding what happens at runtime when a Control uses those services (for the Control author) as well as the effort required to integrate these services into a specific container (for the Control container developer).