Wicket Wish List

striken through means:
1) already happened
2) already happened elsewhere
3) never gonna happen

Most items fall in categories 1 and 2, see each item for the details.

Remove redundant Java component-hierarchy

The component hierarchy has to be defined twice; it has to be specified in HTML, and then it has to be matched in Java. This second hierarchy therefore is redundant.

There has been much discussion whether this is really a problem. The consensus seems to be this is not an issue, although some strongly object. See, for instance
Wicket has unnecessary binding between wicket:id and component hierarchy and here.

Also see JIRA ticket.

==> You can easily achieve that by registering an application wide IComponentResolver with app.getPageSettings().addComponentResolver(). In addition we'll support "queuing" components in the near future, which is kind of what you are looking at but not exactly. You can queue a component with a parent container but eventually it'll be added to the container providing the child markup. That way the hierarchy remains in sync.

Wicket should have native support for multiple homepages, or "multi homing"

Quite often your application has multipe home pages: front page (not logged in), login page, and possible mobile login page and other "home pages" for various purposes / use cases. The problem is that wicket 1.4.12 does not currently support multiple home pages "out-of-the-box". You can achieve support for multiple homepage, errorpage, etc. with quite lot of work, but it would be beneficial to have it out-of-the-box.

MartinG: isn't that already supported ?

```
MyApp#getHomePage() {

 User user = MySession.get().getUser();
 if (user != null) {
   return user.getHomePageClass();
 } else if (...) {
 ...
 } else {
   return DefaultHomePage.class;
 }

}
```

and in 1.5 you can use a request mapper to intercept the root url and return whatever page you want.

MartinM: Multi homing is not properly supported in many cases, for example assuming user's session timeouts -> what homepage to redirect him to?

Modal window built-in

Modal window capability should be built in such that coder don't need to insert <div wicket:id=modal-window-placeholder/>. It seems very futile. It should be enough to just override "isUsingModalWindow" in component hierarchy and the Wicket will provide one. Modal window implementation can be some interface, abstract one, or current one.

~~h5. Model refactoring~~

~~Hollywood principle into ajax refresh. When a certain component is refreshed, now we must manually call target.addComponent on all components that we want to refresh. Instead there should be a hollywood principle: the dependent components themselves can sign up for refresh.~~

this will be possible using an event bus that will be built at some point in the future and will not be limited just to descendant components.

create interface generation for .html markup files in order to ensure validity of wicket:ids.

```
public HelloWorld()
    {
        add(new Label("message", "Hello World!"));
    }
```

The wicket:id list is known at compile time for every .html. The java code could have been safer:

```
public HelloWorld() implements HelloWorldHtml
    {
        add(new Label(MESSAGE, "Hello World!"));
    }
```

Constant MESSAGE might have been defined in the interface extracted from HelloWorld.html.

Perhaps id generation could be done via Project Lombok, Bindgen, or similar to allow pre-compile auto completion etc.

~~Wicket MashUp: Provide built-in support for modular pluggable components~~ Replaced by "Free Wicket" proposal

~~Sometimes on pages that simply put together self-contained (mash-up) components, markup is partly or completely unnecessary. Css styling can be used to position modular components so long as their order can be controlled.~~

-Perhaps a new <wicket:mashup-tag> for placing components on page at given tag location in a "mashup fashion", along the lines of "Wicket MashUp". Also something along the lines of RenderersList or APanel can be used as a starting point. -

~~create abstraction for content types~~

currently, wicket has a concept of "markup type" that is actually the markup extension. in doing work on wml support for "mobile wicket", i'm discovering a need for full mime type support. if it doesn't break too many people, it would ideally be best to introduce a formal MarkupType class with a getExtension() and getContentType(), where ContentType includes mime type information. there quite a bit to think through here in terms of request/response markup type mappings, but i'd be up for working on this. this problem is definitely generic to problems outside mobile wicket. – jonathan

Fixed in 1.5

~~h5. multiply child/extend inheritance~~

https://issues.apache.org/jira/browse/WICKET-1134

improve ajax support of javascript

i still think we should be doing something more object oriented in terms of composing javascript for ajax purposes. i'd like to see us adopt something like that JavaScript class i was promoting a while back. one thing that occurs to me today is that not only could you easily compose javascript this way, but there might be a whole host of reusable javascript behaviors we could all leverage. for example, instead of having bits of javascript code hacked into strings all over projects to accomplish deletion confirmation on link/button clicks, we could have ConfirmJavaScript extends JavaScript and take care of the gory details in there (string resources for localization, whatever). as hunks of javascript get more complex and prevalent, creating this abstraction could really serve us nicely as a community. i personally would have loved to reuse things like FadeComponentJavaScript or DisableComponentJavaScript. the subclasses will all take a lot of work to create, but let's make an abstraction for JS code so we can do this work at all. – jonathan

A few advantages to making our javascript more OO:

- conform to Yahoo recommendations on page rendering performance by putting javascript just before </body>
- more component oriented development through object literals (JSON)
- separate javascript from markup

```
ListView users = new ListView("users", usersModel);
users.setOutputMarkupId(true);
// DomQuery-esque selector (http://extjs.com/learn/Tutorial:DomQuery_v1.1_Basics)...
//selector could be turned into a Builder object instead of string
Event e = new ClickEvent("a[class=deleteLink]");
e.addHandler(new ConfirmDialog("Are you sure you want to delete?"));

users.addEvent(e); // can be shortened to new ClickEvent("...", new ConfirmDialog("..."))
```

serializes to:

```
{ element: "{usersMarkupId}:", // generated by Component
  events: [
      { event: "click", [  // generated by ClickEvent
          { selector: "a[class=deleteLink]",
            handlers: [
              function() { // generated by ConfirmDialog
                if(confirm("Are you sure you want to delete?") {
                  // Wicket.ajax...
                }
              }
            ]
          }
        ]
      }
    ]
}
```

Wrap the above in an 'ondomready' event and we have unobtrusive javascript that should be easier to abstract going forward.

– brian

~~h5. New Inspectors~~

~~Add inspector to look for empty src="" attrs which cause extra requests to the server~~ see WICKET-2412

~~h5. More flexible enable visible using "behaviors"~~

-Often we have, say 3 components that depend on the same enabling condition and out of these components maybe 1 depends on one more condition. It would be nice to do something like:

```
public interface ConditionInterface {
  public boolean isTrue();
}

ConditionInterface generalCondition = ....;
ConditionInterface specialCondition = ....;

FormComponent<String> textField1 = new TextField("field_1").addEnablers(ConditionFactory.and(generalCondition,
specialCondition));
FormComponent<String> textField2 = new TextField("field_2").addEnablers(generalCondition);
FormComponent<String> textField3 = new TextField("field_3").addEnablers(generalCondition);
```

-

See Component#onConfigure(). There you can do whatever is needed to decide the visibility/enable-ability of the component.

~~h5. Remove Change and possibly VersionManager~~

no longer needed since we serialize the entire page. Downside - makes HttpSessionStore not as space efficient
(possibly move these implementation details down into the session store package? - jonathan)
(this is already pushed to the store package i guess, version manager and pagemap are created by the httpsession store already,
the problem with removing this is that it will break all stuff that cant depend on the file based store are really need to be in mem,
the disk based store already doesn't do anything with Change and also the version manager, it only uses the version number from the vm, johan)

Done in 1.5

Cleanup component visibility

Currently we have Component.set/isVisible, Component.isRenderAllowed, Component.set/isVisibilityAllowed

The reason for Component.set/isVisible is simple - allow users to control visibility of their component

The reason for Component.isRenderAllowed is for security strategy to be able to force component to be invisible

Now if i am a user and want to test if the component will be rendered or not I need to do

```
boolean visible=component.isVisible()&&component.isRenderAllowed()
```

This is ugly as I have to remember both checks all the time. Because of this I have introduced Component.determineVisibility() in 1.3.3 which encapsulates all necessary checks.
Now, what if I want to write a component that controls visibility of its children or other components. It can call child.setVisible(false), but that is not guaranteed to work because the child might have its isVisible() overridden and thus any call to setVisible() will be ignored. For this reason I have introduced Component.set/isVisibilityAllowed() in 1.3.3. Both setter and getter are final so it is impossible for the user to break the contract.

So now to check for visibility I need to call component.determineVisibility() which is very kludgy. I also have two sets of getters and setters that seemingly control visibility. Bad.

I think we should rename/refactor some of this to make it cleaner. The best solution would be to have isVisible() be final and equivalent of what determineVisibility() does now because that is the most natural way to query a component for its visibility. Also whatever the equivalent of current isVisible ends up being (an overridable getter) it should not have a setter.

~~Remove PageMap~~

PageMap is a misnomer and isnt really needed anymore - not with second level session store around.
(again, move into httpsessionstore package for backwards compat?)
(see above points, johan)

Done in 1.5

~~Transparent clustering support out of the box and enabled by default~~

Coming in Wicket 1.3.1 WICKET-1272

~~Window scope~~

~~this will basically replace PageMap and allow users to store per browser tab/window attributes~~

Pagemaps/windows were dropped from 1.5+ in favor of simpler page versioning mechanics.

~~JDK 1.5 support~~

- ~~Generify models~~
- ~~Annotation for mounting a bookmarkable page~~ - done in wicketstuff-annotation
- Use varargs, for instance (a bad example) MarkupContainer#add(Component) -> MarkupContainer#add(Component...)
- Take advantage of new classes (java.util.concurrent, LinkedHashMap that can be used as an LRUMap, etc) to remove some of the wicket.util classes

Done in 1.4

Make WicketTester a first class citizen

Currently WicketTester is one of the less mature parts of the core project. It really is useful and great, but the API is not thought through and there are lots of areas you still can't test.

Migrating the API to a Junit 4.4 + Hamcrest style of programming would be a very nice addition.

IDataProvider#size() should return long

The JPA spec has count queries return long instead of int. This change has been attempted earlier, and it was not found to be trivial. It is also a major break in API so we need to consider this carefully.

Rewrite repeaters to make it easier to work with

- boundless datasets
  - unknown size or too expensive to calculate every request
  - provide pager with first/pref/next only?

- data stores that return count and window at the same time
  - need to know the window boundaries without querying the size first (no culling? fallback?)

(see WICKET-1784)

Take a good look at PagingNavigator and AjaxPagingNavigator

The PagingNavigator has been with us since 1.0 and might need some cleaning up.

~~h5. Replace ImageMap with a better one~~

The ImageMap has been with us since 1.0 and might need some cleaning up.

WICKET-1456: rewrite existing ImageMap or create a new one?
WICKET-1936: Use the ClientSideImageMap instead?
Wicket Wish List - Apache Wicket - Apache Software Foundation
ClientSideImageMap is in use in 1.5.

~~Establish Wicket Security strategies sub project~~

- ~~Make Swarm part of wicket core - Giving users easier access to a security framework.~~ was voted against
- ~~Move auth-roles into sub project~~ no need since there's no security subproject

Improve Tree model

the tree can use a more web-friendly model then swing's tree model

~~Add Rapid Prototyping Forms~~

~~The Idea of Alistair Maw and Jonathan Locke (ROR? No. Wicket on Wheels.) should be included in next Wicket. A set of nice looking Defaultformcomponents and a Beanresolver to automatically let create forms from just the plain bean but still allow to customize it later by overiding parts of it. Beside nice looking (to show/ even may go live with it in first place) and fast it should have enabled possibility to integrate security on a Form as well as Field level.~~

See Wicket Web Beans.

~~Clean up/ Have a look at Models~~

~~With now having a CompoundPropertyModel that has the same abilities as BoundCompundPropertyModel it might be nice to have someone overwork all models and decide if we **really** need all. Stiping out/ Migrating ino one e.g.: BoundCompoundProperty and CompoundProperty model as well as others (if possible) would really ease wicket for beginners as well as save experienced users some thoughts about what model to use.~~

Already done in 1.3, there is not much that needs to be improved other than apply generics.

~~Postprocessing Response~~

~~Can be interesting for me have a feature (configurable in application) that apply a xsl postprocessig for each response. Now i can use a Border o XslBehaivor to have same feature but is not transparent for page.~~

see IResponseFilter

~~Package Level Properties / resources~~

done in 1.4, look for package.properties, see PackageStringResourceLoader

~~Add support for SSL pages/components out of the box~~

~~The wiki describes a nice way (though still diamond in the rough) to switch from/to SSL connections using an annotation. This could/should go into core.~~

implemented in 1.4, see WICKET-2229

~~Add possibility to use interface implementation instead of extending Panel~~

Allow components (for example formComponents) have their own markup without the need to extend Panel. The change should not change/brake any existing stuff, only add additional possibility how to solve some problems easier then currently. For example instead of wrapping TextField in Panel just subclass the TextField and implement the given interface.

You may use MarkupComponentBorder already to surround any Component with additional markup. And in 1.5 any component can provide it own markup independently from the parent markup via getMarkup() and getMarkup(child component).

~~Mounting with "nicer URLs"~~

Ability to mount page so it will have one instance per session which will be always accessable by mounted URL.

Done in 1.5

~~Normalize page URLs~~

so when user hits a page /my/page/a/b/c//// redirect immediately to /my/page/a/b/c

Done in 1.5

~~Improve url generation~~

-Instead of urlFor returning a CharSequence it should return a UrlBuilder that contains methods to append parameters and would encode them automatically.

The problem currently this: suppose you want to generate a callback, it should be pretty simple:

```
String url=urlfor(ilinklistener.interface)+"&myparam="+value;
```

However, if the page is bookmarkable+stateless the encoding is different - /param/value/ by default or whatever the coding strategy is, so simply doing +"&myparam="+value will not work.-

MountedMapper supports all of this. See MountedMapperTest for many examples.

Investigate JDK-7 properties support/libraries

Java properties are probably something we will have to deal with sooner or later. We should try to look at what is proposed, and what is readily available. The following blogs have links to sources:

- http://freddy33.blogspot.com/2008/01/wild-java-properties-road-so-far.html
- http://tech.puredanger.com/java7#property

~~h5. Portlet 2.0 (jsr 286)~~

~~Support jsr 286 portlet specification:~~

- -* http://jcp.org/aboutJava/communityprocess/pfd/jsr286/index.html-

-See this issue for Portlet 2.0 related issues: -
-https://issues.apache.org/jira/browse/WICKET-1620-
Update: Portlet support has been dropped for Wicket 1.5 WICKET-2976

Previous page / Back button support

Using the pagemap isn't very elegant and passing in the current page isn't always convenient.

~~Make Validators more flexible~~

- ~~let validators be a behavior to contribute to the markup (eg. maxlength attribute for <input>)~~ Done

- ~~let validators use IModel instead/in addition to fixed values~~ (in 1.5 see ivalidatable#getmodel() in 1.4 see imodelawarevalidatable to which validtable in validators can be cast)

~~Make JavascriptStripper more flexible~~

Currently, stripping of comments and whitespaces is implemented as static method. Therefore, the default implementation cannot be extended (e.g. with an implementation that strips Firebug's logging, i.e. console.log(), consoloe.debug(), ...) or replaced (e.g. by a JS compressor like http://developer.yahoo.com/yui/compressor/).

Fixed in 1.5

Support JSR 303 Bean Validation

Validation in the presentation layer, like in Wicket, conflicts with the don't repeat yourself principle. The same validation rule may be needed in different (Wicket) forms, in web services, in the domain layer and so on. The constraints should therefore be concentrated in one place - the domain class (take a look at Grails http://www.grails.org/Validation). Bean Validation is an easy and smart way to validate input.

A couple of thoughts from Jonathan on conversion and validation:

Although JSR-303 is an emerging standard and we could surely support just to play nice, it does not look general enough to be the be-all end-all of validation frameworks and so there is still room to do better (I'd actually like to see us create a solution that's workable for most if not all problems across all tiers and domains). Looking at JSR-303, even the basic idea of a validator is not something I like very much as it is not as OO as possible. To leak fewer details about internal object state (better encapsulation), it would make more sense for objects to implement a Validatable interface directly instead (you could, of course, create external validators using the same interface, but who would want to?). Who better to know if the object is in a valid state than the object itself?

Anyway, I think the right interface is this:

```
public interface Validatable
{
    boolean isValid(ProblemListener listener);
}
```

This is quite beautiful in practice . Since Validatable objects are fully OO, validating sub-objects is elegantly recursive and preserves encapsulation all the way down. The implementation of isValid() can be made easy with an AbstractValidatator class that takes care of talking to problem listeners, validating sub-objects or dealing with domain-specific situations such as the those addressed by features like JSR-303's groups (which over-specify their contract and therefore lack generality). Also, JSR-303 specifies ConstraintViolation as an interface, which seems to me to be a fatal flaw if you're looking to solve validation at all levels of the stack and for all problems. By contrast, the simple and very general ProblemListener interface (which is optional if you only care about true/false) is fully extensible and creates an independent solution domain which could solve many other not-yet-known problems:

```
public interface ProblemListener
{
    void onProblem(Problem problem);
}

public class Problem
{
    String toString(); // Message value of problem
}

public abstract class AbstractValidator implements Validatable
{
    public boolean isValid(ProblemListener listener) { ... }
    protected ensure(boolean condition, Problem problem);
    protected ensureValid(Object... objects);
    protected abstract onValidate();
}
```

Implementation details of a particular validation system such as bean paths and roots and so forth from the JSR303 spec would depend on the specific Problem instances being generated by validators (which would depend on the AbstractValidator subclass being used internally). In many cases, a lighter weight solution will be nice. In other cases, JSR303 won't go far enough. By instead picking a simpler set of interfaces and specifying a lighter-weight and more extensible system (via AbstractValidator), we could solve a FAR wider variety of problems. In general, the interfaces I've suggested are certainly more future-proof than JSR-303 because they specify LESS. To express an implementation detail like bean validation involving groups, we could implement and evolve specific AbstractValidator classes without ever having to change the basic plumbing (the public Validatable interface on the object remains a consistent contract with no need for evolution). The external contract is also dead-simple so the API is bare and discoverable and the learning curve is almost zero. It probably goes without saying that I don't like declarative constraints via annotations.

I've also done some thinking about conversion and I think I was on the right track originally concerning conversion. This is the best interface for that:

```
public interface Converter<A, B>
{
    B convert(A a);
}
```

It's hard to argue with this as an interface for converting one object into another!

The trouble people run into is thinking about String conversion which is actually a separate problem from type conversion. I've implemented this more recently than the converters that got into Wicket and I think the solution really should be more like this:

```
public interface StringConverter<T>
{
    Converter<T, String> toStringConverter();
    Converter<String, T> toObjectConverter();
}
```

Sticking to simplicity in conversion and validation would create solutions with much more shelf-life and generality than we can get from existing solutions or even JSR-303.

– Jonathan

~~Converters should benefit from Generics~~

-It's unbelievable that during the age of generics, one must cast the result of a conversion:

```
  /**
   * @param <DataType>
   * @param formComponent
   * @return DataType
   */
  public static <DataType> DataType getFreshValue(FormComponent<DataType> formComponent) {
    return (DataType) formComponent.getConverter(formComponent.getType()).convertToObject(formComponent.
getValue(), formComponent.getLocale());
  }
```

-
Done in 1.5.

~~Integrate more widgets~~

~~A component framework like Wicket lives on the compents it provides. A rich set of easy to use gui components would increase Wickets popularity.~~

See https://github.com/wicketstuff/core

~~Add LabelLink~~

~~Dynamic setting of a link labels is a very common usecase and should be supported by wicket directly. LinkLabel discussion~~

See AbstractLink#setBody(IModel)

~~Accelerate Serialization~~

~~Serialization is heavy used in wicket, therefore it would be good to accelerate it with JBossSerialization. It should be at least as twice as fast as default serialization.~~

See org.apache.wicket.serialize.ISerializer and IFrameworkSettings.setSerializer(). With these tools you are welcome to implement any custom serialization strategy.

Scala Support

Event handlers are an important part of Wicket applications. Today they are implemented usually with anonymous inner classes. Functional programming would be more concise and elegant. Scala is an object oriented and functional programming language, which would be a good choice to write Wicket Applications. A small Scala layer on top of wicket should be sufficient to use the benefits of Scala.

Effort to this effect has already been started, and proved working and very useful:https://wicket-stuff.svn.sourceforge.net/svnroot/wicket-stuff/trunk/wicketstuff-core/scala-extensions-parent/

Separate Examples

The examples blow up the API documentation and may lead to confusion. It would be better, to have a separate documentation for the examples. Maybe even a separate examples package.

Make flags more OOD

```
	/** Component flags. See FLAG_* for possible non-exclusive flag values. */
	private int flags = FLAG_VISIBLE | FLAG_ESCAPE_MODEL_STRINGS | FLAG_VERSIONED | FLAG_ENABLED |
		FLAG_IS_RENDER_ALLOWED | FLAG_VISIBILITY_ALLOWED;
```

Consider using enumerations etc.

Also ~~`IVisitor.CONTINUE_TRAVERSAL`~~ and `String NO_RAW_INPUT = "-NO-RAW-INPUT-"` etc. could be more OOD.

~~Add onBind/onAttach event handler method for components.~~

~~Add onBind/onAttach event handler method for components.~~
The method will be invoked when components are added to page/parent (meaning, you can call getPage() and not get an exception).-
~~Useful for initializing components once (when using methods like getString(), findParent(), etc...), when added, instead of doing it before every render.~~
~~For example, form components can find parent once onBind, instead of on every render.~~
~~An onUnbind/onDetach will also be needed.~~
See o.a.w.Component.onInitialize()

~~Serve multiple javascript/css as one resource~~

~~Would be nice if something like done in Jawr or wro4j will be integrated into wicket core: compressing and bundling all page's resources into once javascript /css file.~~

~~You may want to look at wicketstuff-merged-resources.~~

See http://www.wicket-library.com/wicket-examples/resourceaggregation

~~Sprites/~~ImageBundle ~~for wicket.~~

~~Bundling several images into one image resource.~~
See http://github.com/ananthakumaran/imagebundler-wicket

~~Support of html5~~

~~Html5 is being used more and more. Wicket should support all new tags, but at least the new attributes of form input fields. Such as date, number, email, phone (13 in all).~~

The currently supported ones by the modern browsers are in - NumberTextField, RangeTextField, UrlTextField and EmailTextField. All other can be easily created by the user application. They will be added in Wicket when the browsers add support for them.