## Design goals:

- Support all addEventListener options
- Avoid the need for a `jQuery.Event` wrapper
- Eliminate manual `.trigger()` bubbling and method calls
- Remove special events hooks
- Provide backcompat through Migrate 4.0

## Support all addEventListener options

Standards consensus is emerging for additional options to DOM `addEventListener` for things like passive listeners and delegated events. Users should have access to those features while using jQuery.

Some of these newly defined features might supplant jQuery's own functionality, such as delegated events. We should work with the standards orgs to ensure that they don't define an implementation that contradicts ours without a strong reason. The jQuery implementation could act as a polyfill and we could defer to the native implementation when it is present.

Ideally the majority of this could be done in a "future-proof" way so that if a new option is defined it is not *always* necessary for jQuery code to change in order to support it. That depends on the extent to which jQuery's event implementation interprets and implements semantics; I hope jQuery 4.0 will need to do that a lot less.

Potential issues:

- Multiplexing: We can no longer use a single `addEventListener` call for each elem+type combination. The easiest way to deal with that is by using 1:1 jQuery handler to native handler.
- Compatibility: Lots of code looks at `jQuery._data("events")` including Firefox dev tools and we are sure to break it to some extent when we change data structures to demultiplex the handlers.
- Premature spec: If we do need to do option-specific work we may be screwed if they change semantics after we've shipped 4.0 and we have to support "the wrong way" for a while.
- Plain objects: It may be hard to support event triggering on things that are not DOM elements and still delegate more of the work (propagation, default action) to the native layer.
- Bookkeeping: We still need to ensure that events are properly handled in `.clone()` and `jQuery.cleanData()`.

## Avoid the need for a `jQuery.Event` wrapper

jQuery passes its own `jQuery.Event` object to the handler instead of the native event object. This creates no end of confusion when people find that *some* but not *all* properties are in `event` and that they have to use `event.originalEvent` to get the rest. jQuery has the ability to copy more properties but there is a size and performance tradeoff in making the list longer, even with the improvements in jQuery 3.0.

We will never be able to maintain a complete list of all the properties people need on every event type. An `Event` is a host object and can't be extended for example with `Object.create( event )` to use it as the prototype for our own object. The new ES6 `Proxy` could be used but it will be several years before jQuery's supported browsers all have it and it is much slower to access which could be a problem for high-frequency events. None of these approaches seem viable, at least for 4.0.

It would be better to pass in the native DOM `Event` object. Modern browsers seem to allow adding new properties but don't allow overwriting existing ones, so we have at least a bit of wiggle room in making a native object look like `jQuery.Event` as long as we don't need to overwrite. For example, we could add an `originalEvent` property to the native event so existing code that uses `event.originalEvent.data` would work without modification.

This would also mean that manual event triggering would use a native event. For compatibility reasons we could keep the `jQuery.Event` constructor but have it generate and return a native `CustomEvent` that we use to trigger. Users who want to trigger a specific native event such as `MouseEvent` can construct it themselves and `.trigger( eventObject )`. Potential problems:

- Nested events: jQuery UI in particular creates deep chains of their own custom events that use `originalEvent` and it's unclear if this approach will break that code.
- Duck typing: Code may be checking for `event instanceof jQuery.Event` or `event.originalEvent` to distinguish between a jQuery or native event, but we also want to fill `event.originalEvent` on a native object.
- Recursion: Adding `event.originalEvent = event` might cause some code to hang from a circular reference.

## Eliminate manual `.trigger()` bubbling and method calls

jQuery currently implements its own bubbling algorithm for `.trigger()` and `.triggerHandler()`. It tracks the state of `stopPropagation` and `preventDefault`, then decides whether to perform the default action by calling the DOM method. Two notable exceptions here are `click` on a link and `submit` on a form, which *don't* call the corresponding DOM method; `click` acts that way for consistency with old Webkit browsers and `submit` does so because the DOM method also is not called when the event is triggered.

jQuery's logic is sometimes very different from the approach the browser takes. For example, a call to a DOM method like `.click()` on a checkbox changes the state of the check, calls the handler, and reverts the change if a handler prevents the default action. A call to the DOM `.focus()` method triggers `blur` and `focusout` on the previously focused element and then a `focusin` and `focus` on the new element. jQuery's existing trigger code and special events hooks try to make this less wrong but the event order is still not right in all cases.

The conflation of manual event dispatch and DOM method invocation in `.trigger()` creates a strange situation where jQuery has to ignore the generation of some (but not all) events that a DOM method creates by its invocation. For example, if the user calls `.trigger( "focus" )` jQuery must first call all the jQuery and inline focus handlers and then ignore any `focus` events that occur when it invokes the DOM `.focus()` method. Note that these issues have also confused the spec writers so it's not just jQuery.

Focus management in particular is messy because calling the DOM `.focus()` method on an element will not fire the `focus` event handlers on that element if the element is *already* in focus, is hidden, or is not currently in a document. Users sometimes use `.trigger( "focus" )` for those cases to run the event handlers, either not caring that setting focus will fail or being oblivious to this restriction.

Ideally we'd be able to reuse the pretty names in the jQuery API namespace so that `$("#myDiv").click()` executes the DOM `.click()` method on the element and `$("#myDiv").trigger("click")` dispatches a `click` event on it. That would require deprecating the event shortcut methods immediately though and the difference in behavior across versions might be too confusing. It might also make Migrate filling hard or impossible, it's hard to tell at the moment. We could introduce a new method like `.invoke()` or `.call()` to call the method if needed.

Potential issues:

- User code may be calling `.trigger()` only to get the handlers to run (e.g., at initialization) or primarily to get the DOM method to run (e.g. setting focus). We can't tell which from the code but might be able to make some guesses based on event type and the presence of a same-name method on the element.

- Using DOM `.dispatchEvent()` will mean that jQuery 4.0 `.trigger()` invokes native event handlers where previous versions did not. This is either a great new feature or an unexpected incompatible behavior. :imp:
- Plain objects: It may be hard to support event triggering on things that are not DOM elements and still delegate more of the work (propagation, default action) to the native layer.

## Remove special events hooks

There is a significant amount of plumbing in the event system to support special events hooks. Most of it was introduced so that we could make old IE look sane, or to work around places where our own simulation of native event behavior creates problems. The hooks also have a lot of inside knowledge about the implementation of the event subsystem that will change with this rewrite. What makes it worse is that this infrastructure is exposed, although lightly documented.

Although it seems less elegant, it may be much smaller and simpler to just include any specific workarounds directly in the code and not expose any external interface to allow monkeying with the event subsystem at various stages.

Potential issues:

- Plugins that define special events will break without Migrate or a rewrite (which may be easier).
- https://github.com/jquery/jquery-mousewheel
- https://github.com/search?q=jQuery.event.special&ref=simplesearch

## Provide backcompat through Migrate 4.0

This is a lot of significant changes and no matter how soon or how loudly we talk about this there will be a lot of code affected. We can put most of the old behavior into Migrate 4.0 and issue warnings for things that we know to be deprecated or removed such as events on plain objects. We can put the special event hooks back and honor them, using all the old manual event triggering and plumbing. We can issue warnings for any access to `jQuery._data("events")` and even create a compatible structure on the fly when it's fetched.

Potential issues:

- If 4.0 changes the behavior of existing APIs, it may be difficult for Migrate to tell whether the old or the new behavior is desired. That will make Migrate less of a just-fix-it tool and more of a use-once-and-remove-it tool. However we've dealt with this in the past too (e.g. attroperties).