Session

`Session` provides a global object on the client that you can use to store an arbitrary set of key-value pairs. Use it to store things like the currently selected item in a list.

What's special about `Session` is that it's reactive. If you call `Session.get('currentList')` from inside a template, the template will automatically be rerendered whenever `Session.set('currentList', x)` is called.

To add `Session` to your application, run this command in your terminal:

```
meteor add session
```

{% apibox "Session.set" %}

Example:

```
Tracker.autorun(() => {
  Meteor.subscribe('chatHistory', { room: Session.get('currentRoomId') });
});

// Causes the function passed to `Tracker.autorun` to be rerun, so that the
// 'chatHistory' subscription is moved to the room 'home'.
Session.set('currentRoomId', 'home');
```

`Session.set` can also be called with an object of keys and values, which is equivalent to calling `Session.set` individually on each key/value pair.

```
Session.set({
  a: 'foo',
  b: 'bar'
});
```

{% apibox "Session.setDefault" %}

This is useful in initialization code, to avoid re-initializing a session variable every time a new version of your app is loaded.

{% apibox "Session.get" %}

Example:

```
<!-- main.html -->
<template name="main">
  <p>We've always been at war with {{theEnemy}}.</p>
</template>
```

```
// main.js
Template.main.helpers({
  theEnemy() {
    return Session.get('enemy');
  }
});

Session.set('enemy', 'Eastasia');
// Page will say "We've always been at war with Eastasia"

Session.set('enemy', 'Eurasia');
// Page will change to say "We've always been at war with Eurasia"
```

{% apibox “Session.equals” %}

If value is a scalar, then these two expressions do the same thing:

```
Session.get('key') === value
Session.equals('key', value)
```

. . . but the second one is always better. It triggers fewer invalidations (template redraws), making your program more efficient.

Example:

```
<template name="postsView">
  {{! Show a dynamically updating list of items. Let the user click on an item
      to select it. The selected item is given a CSS class, so it can be
      rendered differently. }}

  {{#each posts}}
    {{> postItem}}
  {{/each}}
</template>

<template name="postItem">
  <div class="{{postClass}}">{{title}}</div>
</template>
```

```
Template.postsView.helpers({
  posts() {
    return Posts.find();
  }
});
```

```
Template.postItem.helpers({
  postClass() {
    return Session.equals('selectedPost', this._id)
      ? 'selected'
      : '';
  }
});

Template.postItem.events({
  'click'() {
    Session.set('selectedPost', this._id);
  }
});
```

Using Session.equals here means that when the user clicks on an item and changes the selection, only the newly selected and the newly unselected items are re-rendered.

If Session.get had been used instead of Session.equals, then when the selection changed, all the items would be re-rendered.

For object and array session values, you cannot use `Session.equals`; instead, you need to use the `underscore` package and write `_.isEqual(Session.get(key), value)`.